O‘ZBEKISTON RESPUBLIKASI
OLIY VA O‘RTA MAXSUS TA‘LIM VAZIRLIGI

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT
AXBOROT TEXNOLOGIYALARI UNIVERSITETI

B.B.MO‘MINOV

DASTURLASH II

O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta‘lim vazirligi tomonidan oliy o‘quv yurtlarining talabalari uchun darslik sifatida tavsiya etilgan
(28.12.2020 yil 676 sonli buyrug‘i)

TOSHKENT – 2021

UO‘K: 000(000)
KBK 00.00
M00

Mo‘minov B.B., Dasturlash II. (Darslik). – T.: «Nihol print» OK, 2021. – 616 b.

ISBN 978–9943–

Darslikda noma’lum tiplar va noma’lum nomlar fazosi, STL kutubxonalari, oddiy, assosiativ va tartiblanmagan assosiativ konteynerlar hamda konteynerlar adapterlari tushunchalari, standart algoritmlar, iteratorlar va sonli sinflar bilan ishlash, sintaktik tahlil qilishni tashkil etishning samarali usullari va murakkab saralash algoritmlaridan unumli foydalanish, Visual C++ muhitida menyular va uskunalar paneli, komponenta tushunchasi va xususiyatlari, muloqot oynalari, Visual C++ ning grafik imkoniyatlari, OLE va MFC texnologiyalari hamda kichik loyihalarni yaratishning nazariy va amaliy asoslari keltirilgan.

UO‘K: 000(000)
KBK 00.00

Taqrizchilar:

Nuraliyev F.M. – Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti, Televizion texnologiyalari fakulteti dekani, t.f.d., dotsent;

Raximov N.O. – O‘zbekiston Milliy universiteti Jizzax filiali, o‘quv ishlari bo‘yicha direktor o‘rinbosari, t.f.d.

ISBN 987–9943–

KIRISH

O‘zbekiston Respublikasi Prezidentining bir qator Qaror va Farmonlari Axborot-kommunikatsiya texnologiyalarini yanada jadal ruvojlantirish maqsadlarida qabul qilinmoqda. Ushbu qonunlardan “Raqamli iqtisodiyot va elektron hukumatni keng joriy etish chora-tadbirlari to‘g‘risi”dagi PQ-4699-son 28.04.2020 va “«Raqamli O‘zbekiston — 2030» strategiyasini tasdiqlash va uni samarali amalga oshirish chora-tadbirlari to‘g‘risi”dagi PF-6079-son 05.10.2020 yilgi qaror va farmonlarida takidlanganidek «Bir million dasturchi» loyihasi doirasida 500 ming nafar yoshlarni qamrab olish orqali kompyuter dasturlash asoslariga o‘qitish tashkillashtirish maqsadida Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universitetining barcha yo‘nalish talabalariga mo‘ljallangan “Dasturlash II” fani uchun darslikni yaratish dolzarb masala hisoblanadi.

Darslikning birinchi bobida noma’lum tiplar va noma’lum nomlar fazosi, STL kutubxonalari, oddiy konteynerlar, assosiativ va tartiblanmagan assosiativ konteynerlar hamda konteynerlar adapterlari tushunchlari va ulardan dasturlashda unumli foydalanish o‘rgatiladi.

Darslikning ikkinchi bobida standart algoritmlar, iteratorlar va sonli sinflar bilan ishlash hamda sintaktik tahlil qilishni tashkil etishning samarali usullari va murakkab saralash algoritmlaridan unumli foydalanish o‘rgatiladi.

Darslikning uchinchi bobida Visual C++ muhitida menyular va uskunalar paneli, komponenta tushunchasi va xususiyatlari, muloqot oynalari, Visual C++ning grafik imkoniyatlari, OLE va MFC texnologiyalari hamda kichik loyihalarni yaratish o‘rgatiladi.

Darslik bo‘yicha takliflar, undagi aniqlangan kamchiliklar bo‘yicha fikr-mulohazalaringizni mbbahodir@gmail.com elektron pochtasiga yuborishingizni so‘raymiz. Sizning bu bildirgan fikr-mulohazalaringiz darslikning keyingi nashrlarida uni yanada mazmunliroq va kamchiliklardan holi tarzda chop etishimizga albatta asqotadi.

Muallif

1-BOB. KONTEYNERLAR

1.1. Noma’lum tiplar va noma’lum nomlar fazosi.

📖 Tiplarni dinamik tarzda aniqlashning ba‘zi usullari, typeid operatori, typeinfo kutubxonasi, type_info sinfi a’zolaridan foydalanish, dastur bajarilish davomida ko‘rsatkichli va oddiy o‘zgaruvchilarning tiplarini almashtrish operatorlari dynamic_cast, const_cast, static_cast, reinterpret_cast operatorlaridan foydalanish, noma’lum va foydalanuvchining nomlar fazolarini yaratish va dasturda ulardan foydalanish usullari, buferlashtirilgan kirish va chiqishni yaratish, IO amallarida formatlashning identifikatorlari va manipulyatorlarini o‘rnatish va o‘chirish usullari keltirilgan bo‘lib, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni mustahkamlash uchun 25 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 4 ta assisment topshirig‘i va har assismentda 7 ta topshiriq, jami 28 ta topshiriq berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

🖋 ***Kalit so‘zlar.*** Tipni aniqlash, tipni almashtrish, nomlar fazosi, buferlashtirilgan kirish va chiqish, typeid, typeinfo, type_info sinfi, dynamic_cast, const_cast, static_cast, reinterpret_cast, namespace, istrstream, oqim, formatlash identifikatori, manipulyatorli formatlash.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Tip tushunchasi, sinf va sinf obyekti, polimorf sinf, kasting amali, dastur fragmenti va sohasi, kutubxona, sinf a’zolari, funksiya va ko‘rsatkich, indentifikator, ma’lumotlarni kirish va chiqishi, oqim, dasturlashga oid dastlabki tushunchalar va C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Ixtiyoriy ko‘rsatkichli o‘zgaruvchining tipni aniqlash va tipni almashtrish usullari, typeid, typeinfo, type_info sinfi, dynamic_cast, const_cast, static_cast, reinterpret_cast yangi operatorlar, kalit so‘z namespace va nomlar fazosini yaratish, ulardan dasturda foydalanish usullari va texnologiyalari, buferlashtirilgan kirish va chiqish orqali ma’lumotlarni massivga joylatirish va istrstream oqimi, formatlash identifikatori, manipulyatorli formatlash usullari orqali lokalizatsiyalashgan IO operatorlarini yaratish va foydalanish usullarini o‘rganishingiz mumkin.

REJA

1. Tiplarni dinamik tarzda aniqlash.
2. Tiplarni almashtirish operatorlari.
3. Yangi nomlar fazosi yaratish.

4. Buferlashtirilgan kiritish va chiqarish.

KIRISH

Noma’lum tiplar va noma’lum fazolar uchun dinamik tipni aniqlash va tipini almashtirish operatorlari zamonaviy obyektga yo‘naltirilgan dasturlashni qo‘llab-quvvatlovchi C++ dasturlash tilining ikkita vositalari: dinamik tipni aniqlash (runtime type identification - RTTI) va qo‘shimcha tip qo’yish operatorlari bilan bog‘liq. Bu vositalarning biri C++ spesifikatsiyasi qismi emasdi, lekin har ikki runtime polimorfizm uchun qo‘llab-quvvatlash uchun qo‘shilgan. RTTI dastur bajarish davomida obyekt turini aniqlash imkoniyati mavjudligini bildiradi. Bunda muhokama qilinayotgan tip turi tashlash operatorlari - bu amalni bajarish uchun dasturchiga xavfsiz yo‘llarini taklif qiladi. Ulardan biri, dynamic_cast, bevosita RTTI aniqlash bilan bog‘liq.

Tiplarni dinamik tarzda aniqlash. Ushbu vosita polimorf bo‘lmagan tillarda (masalan, C) mavjud emasligi sababli dinamik turdagi identifikatsiyalash bilan tanish bo‘lmasligingiz mumkin, chunki har bir obyektning tipi kompilyatsiya vaqtida (ya‘ni, dasturni bajarishda) ma’lum bo‘lgani uchun dastur bajarilishida ma’lumot olishning hojati yo‘q. Biroq, C++ kabi polimorf tillarda obyekt tipi kompilyatsiya vaqtida noma’lum bo‘lgan holatlar bo‘lishi mumkin, chunki dastur bajarila boshlagunga qadar bu obyektning aniq tipi (tiplari) aniqlanmaydi. Ma’lumki, C++ sinf ierarxiyasi, virtual funksiyalar va ko‘rsatkichlarni (pointer) asos sinflarga ishlatib polimorfizmni amalga oshiradi. Tayanch sinfga ko‘rsatkich bu tayanch sinf a’zolariga yoki undan olingan har qanday obyekt a’zolariga murojaat qilish uchun foydalanish mumkin. Shuning uchun asos sinfga ko’rsatkichning istalgan vaqtda qanday turdagi obyektga murojaat qilishi har doim ham oldindan ma’lum emas. Bu faqat dastur ishlayotganida — dinamik turdagi identifikatsiya vositalaridan biri yordamida aniqlanadi.

Dastur bajarilishi davomida obyekt tipini olish uchun *typeid* operatoridan foydalaniladi.

Dastur bajarilayotgan vaqtda obyekt tipini olish uchun <typeinfo> kutubxonasini kiritish va *typeid* operatoridan foydalanish mumkin. typeid operatoridan foydalanishning eng keng tarqalgan yozilish sintaktisi quyidagicha:

```
typeid(object)
```

Bunda, *object* elementi tipini olish uchun obyektni anglatadi. Shuningdek, nafaqat ichki tipini, balki dasturchi tomonidan yaratilgan

sinf tipini ham aniqlash mumkin. Buning uchun typeid operatori obyekt tipini tasvirlaydigan type_info obyektiga murojaat qiladi.

type_info sinfda public modifikatori bilan quyidagi statik operatorlar aniqlangan bo‘ldi.

```
bool operator == (const type_info &ob);
bool operator !=(const type_info &ob);

bool before(const type_info &ob);

const char *name();
```

Bunda qayta aniqlangan (overloaded) operatorlari [==] va [!=] tiplarini taqqoslash uchun ishlatiladi. Chaqiruvchi obyekt parametr sifatida ishlatiladigan obyekt (element ob) oldida ketma-ketlik tartibida bo‘lsa before() funksiyasi true qaytaradi. Bu funksiya asosan ichki foydalanish uchun mo‘ljallangan. Uning natija qiymati merosxo‘r yoki sinf iyerarxik bilan hech qanday aloqasi yo‘q. name() funksiyasi tip nomiga ko‘rsatkich qaytadi.

typeid operatoridan foydalanishga bir misol keltiramiz.

1.1-dastur. Tiplarni taqqoslash[1]

```
#include "stdafx.h"
#include <iostream>
#include <typeinfo>
using namespace std;
class myclass {
// . . .
};
int main(){
int i, j;
float f;
myclass ob;
cout << "i o‘zgaruvchi tipi: " <<
typeid(i).name()<<endl;
cout << "f o‘zgaruvchi tipi: " <<
typeid(f).name()<<endl;
cout << "ob o‘zgaruvchi tipi: " <<
typeid(ob).name()<<endl<<endl;
```

[1] Dastur Visual C++ muhitida tuzilgan.

```
if(typeid(i) == typeid(j))
    cout << " i va j tiplar bir xil.\n";
if(typeid(i) != typeid(f))
    cout << " i va f tiplar bir xil emas.\n";
system("PAUSE");
return 0;
}
```

1.1 – dastur natijasi. Output

```
 i o‘zgaruvchi tipi: int
 f o‘zgaruvchi tipi: float
 ob o‘zgaruvchi tipi: class myclass
 i va j tiplar bir xil.
 i va f tiplar bir xil emas
```

1.1-dastur tahlili. Dasturda int tipiga oid i va j, float tipida f, myclass tipida ob o‘zgaruvchilari aniqlangan. typeinfo kutubxonasining typeid().name() metodi bilan ularning tiplari aniqlangan. Taqqsolash operatorlari bilan o‘zraguvchilarning tiplari tekshirilgan va natijalar ekranga chiqarilgan. Bu esa tiplar bo‘yicha dastlabki ishlov berish uchun lozim.

1.1-jadval. type_info sinf a’zolari va vazifalari

№	Funksiya nomi	vazifasi
1	(constructor)	Sinfda standart va nusxa konstruktori mavjud emas.
2	(destructor) [virtual]	xavfsiz tayanch sinf uchun pointer orqali obyektni o‘chirish mumkin (virtual funksiyasi)
3	operator=	O‘zlashtirish amalidan foydalanib ko‘chirib bo‘lmaydi
4	operator== operator!=	Obyektlar bir xil tipgaligini taqqoslaydi (public funksiya)
5	before	Joriy tipdagi boshqa type_index obyekt tomonidan nusxasi tipini bor yoki yo‘qligini tekshiradi. Obyektlar, aniqrog‘i tiplari, amalga oshirilishi belgilangan tartibda tashkil etiladi. (public funksiya)
6	hash_code	Joriy tipga mos keladigan tiplarni qaytaradi (public funksiya)
7	name	Aniqlangan joriy tip nomini qaytaradi. (public funksiya)

typeid operatori polimorf tayanch sinf uchun ko‘rsatkich sifatida qo‘llaniladigan bo‘lsa (eslabtib o‘tamiz, polimorf sinf kamida bitta virtual funksiyani o‘z ichiga olgan sinfga aytiladi), u avtomatik ravishda real obyekt tipini qaytaradi va uning asosidan olingan sinf tayanch sinf obyekt yoki obyekt emasligini ham ko‘rsatadi.

Shuning uchun typeid operatoridan asos sinfga ko’rsatkich bilan murojaat qilingan obyekt tipini dinamik ravishda aniqlash uchun foydalanish mumkin. Bu xususiyat quyidagi dasturda ko‘rsatilgan.

1.2-dastur. typeid operatorini polimorf sinflar ierarxiyasiga qo‘llanilishi.

```
#include "stdafx.h"
#include <iostream>
#include <typeinfo>
using namespace std;
class Base {
virtual void f() {}; // Asos sinf uchun polimorf
// . . .
};
class myClass_one: public Base {
// . . .
};
class myClass_two: public Base {
// ...
};
int main(){
Base *p, baseob;
myClass_one ob1;
myClass_two ob2;
 p = &baseob;
 cout << " O'zgaruvchi p tipdagi obyektga ishora:
";
 cout << typeid(*p).name() << endl;
 p = &ob1;
 cout << " O'zgaruvchi p tipdagi obyektga ishora:
";
 cout << typeid(*p).name() << endl;
 p = &ob2;
 cout << " O'zgaruvchi p tipdagi obyektga ishora:
```

```
";
 cout << typeid(*p).name() << endl;
 system("pause");
 return 0;}
```

1.2 – dastur natijasi. Output

```
 O'zgaruvchi p tipdagi obyektga ishora: class Base
 O'zgaruvchi p tipdagi obyektga ishora: class
myClass_one
 O'zgaruvchi p tipdagi obyektga ishora: class
myClass_two
```

1.2-dastur tahlili. Dasturda dinamik ko‘rsatkichli `Base` tipli p o‘zgaruvchisi yaratilgan va 3 ta, Base tipiga baseob, myClass_one tipiga ob1, myClass_two tipiga ob2 mos o‘zgaruvchilar yaratilgan. Dastur davomida dinamik o‘zgaruvchiga oddatiy o‘zgaruvchilar qiymat qilib berilgan. Natija shu ma’lum bo‘ldiki dinamik o‘zgaruvchining tipi dinamik ko‘rinishda o‘zgardi.

typeid operatori polimorf turdagi tayanch sinfga ko’rsatkich sifatida qo‘llanilsa, haqiqiy manzillangan obyektning tipini dastur bajarilayotgan vaqtda aniqlaydi.

Typeid operatori polimorf bo‘lmagan iyerarxik sinflar uchun ko‘rsatkich qo‘llaniladigan barcha hollarda, tayanch tipi uchun ko‘rsatikich bo‘ladi. Boshqa so‘zlar bilan aytganda, bu ko‘rsatkich aslida tipni aniqlash mumkin emas. Tajriba sifatida 1.2-dasturdagi asos sinfdagi f() virtual funksiyani izohga oling va natijani tekshirib ko‘ring. Dasturda bu o‘zgarishni amalga oshirgandan so‘ng har bir turli tipli obyektning tipi Base deb aniqlanishini ko‘rasiz, chunki bu p ko’rsatkichga ega bo‘lgan tipdir.

typeid operatori odatda kengqirrali ko‘rsatkich (dereferenced pointer ya‘ni, [*] operatoriga ega bo‘lgan ko‘rsatkich) uchun qo‘llanilganligi sababli, bu ko‘rsatkich null bo‘lib chiqqanda vaziyatni boshqarish uchun maxsus istisno yaratiladi. Bu holda, typeid operatori bad_typeid istisnosiga o‘tadi.

Polimorf sinflar ierarxiyasidagi obyektlarga havolalar ko’rsatkichlar kabi ishlaydi. typeid operatori polimorf sinfning mos yozuvlar uchun qo‘llaniladigan bo‘lsa, u aslida obyekt tipini qaytaradi va u asosiy tipi o‘rniga olingan tipdagi obyekt bo‘lishi mumkin. Bu usul o‘zgaruvchilarni funksiyalarga murojaat qilib o‘tishda eng ko‘p

qo‘llaniladi. Masalan, quyidagi (1.3-dastur) dasturda WhatType() funksiyasi asosiy tipdagi obyektlarga mos yozuvlar parametrini e‘lon qiladi. Demak, WhatType() funksiyasi bazaviy tipdagi obyektlarga murojaatlarni yoki asos sinfdan olingan har qanday sinf obyektlariga murojaatlarni keltirib chiqaradi. Bunday parametrlarga qo‘llaniladigan typeid operatori funksiyaga o‘tgan obyektning haqiqiy tipini qaytaradi.

1.3-dastur. typeid operatorini havolali parametriga qo‘llash.

```
#include "stdafx.h"
#include <iostream>
#include <typeinfo>
using namespace std;
class Base {
virtual void f() {}; // Asos sinf uchun polimorf
// . . .
};

class myClass_one: public Base {
// . . .
};
class myClass_two: public Base {
// ...
};
void WhatType(Base &ob){
cout << " ob parametri obyekt tipiga havolasi: ";
cout << typeid(ob).name() << endl;
}
int main(){
Base baseob;
myClass_one ob1;
myClass_two ob2;
WhatType(baseob);
WhatType(ob1);
WhatType(ob2);
 system("pause");
 return 0;
}
```

1.3 – dastur natijasi. Output

```
 ob parametri obyekt tipiga havolasi: class Base
 ob parametri obyekt tipiga havolasi: class
myClass_one
 ob parametri obyekt tipiga havolasi: class
myClass_two
```

1.3-dastur tahlili. Dasturda `WhatType` funksiyasi yaratilib, unga asos sinf tipidpgi havolali parametr belgilandi. Asosiy funksiyada 3 ta, Base tipiga baseob, myClass_one tipiga ob1, myClass_two tipiga ob2 mos o‘zgaruvchilar yaratilgan. Dastur davomida `WhatType` funksiyasining havolali parametrga oddatiy o‘zgaruvchilar qiymat qilib berilgan. Natijadan shu ma’lum bo‘ldiki, dinamik o‘zgaruvchining tipi danamik ko‘rinishda o‘zgardi.

Dasturlash jarayonlarida o‘zgaruvchilarni tiplarini real tip bilan taqqoslashga to‘g‘ri keladi. Buning uchun typeid operatori agrgument sifatida tipi nomini olish yordamida amalga oshiriladi. Uning yozilishi quyidagicha:

```
tyreid(tip nomi)
```

Masalan, quyidagi listing amal qiladi.

```
Cout<<typeid(int).name();
```

Typeid operatorini ushbu variantda ishlatilishining maqsadi tipni taqqoslash uchun foydalanishdir, (belgilangan ma’lumotlar tipini ta‘riflaydi) ammo, type_info obyektini olish uchun emas.

Dinamik tipni aniqlashni (RTTI) qo‘llash. Dasturda dinamik tipni aniqlash (identifikatsiya) vositasi (RTTI) imkoniyati va foydali bo‘lishi mumkinlingini ko‘rsatib o‘tamiz. Masalan, geometrik shakllar uchun iyerarxik sinf berilgan bo‘lsin, unda doira, uchburchak va to‘rtburchak maydoni hisoblash usullari mavjud. Ushbu dasturda doira, uchburchak yoki to‘rtburchakni yaratish uchun mo‘ljallangan funksiyasi factory() bo‘lsin. Ushbu funksiya yaratilgan obyektga bir nusxani qaytadi. Obyektlarni hosil qiluvchi funksiya ba‘zan obyekt fabrikasi deb ataladi. Yaratilayotgan obyektning o‘ziga xos tipi C++ tasodifiy sonlar generatorining Rand() funksiyasiga kirish orqali aniqlanadi. Demak, qanday turdagi obyekt hosil bo‘lishini oldindan bilolmaymiz. 1.4-dasturda o‘nta obyekt yaratadi va har bir turdagi yaratilgan obyektlar sonini hisoblaydi. factory() funksiyasini chaqirishda har qanday shakl turini hosil qilish mumkinligi sababli, dastur aslida yaratilgan obyekt tipini aniqlash uchun typeid operatoridan foydalanadi.

1.4-dastur. Dinamik tipdagi identifikatsiya vositasidan foydalanish.

```
#include "stdafx.h"
#include <iostream>
#include <typeinfo>
#include <cstdlib>
using namespace std;
class figure {
protected:
    double x, y;
public:
    figure(double i, double j) {
        x = i;
        y = j;
    }
virtual double area() = 0;
};
class triangle : public figure {
public:
    triangle(double i, double j) : figure(i, j) {}
    double area() {
        return x * 0.5 * y;
    }
};
class rectangle : public figure {
public:
    rectangle(double i, double j) : figure (i, j)
{}
    double area() { return x * y;}
};
class circle : public figure {
public:
    circle(double i, double j=0) : figure(i, j) {}
    double area() {return 3.14 * x * x;}
};
// figure sinf obyektlarini yaratish.
figure *factory(){
    switch(rand() % 3 ) {
    case 0: return new circle (rand()%15);
```

```
    case 1: return new triangle
(rand()%15,rand()%15);
    case 2: return new rectangle (rand()%15,
rand()%15);
    }
return 0;
};
int main(){
    figure *p; // asos sinfga ko‘rsatkich
    int i;
    int t=0, r=0, c=0;
        // 5 ta obyektlarni hosil qilsh va sanash
    for(i=0; i<15; i++) {
        p = factory(); // obyektni hosil qilish
        cout << "Obyektning tipi: " <<
typeid(*p).name();
        cout << ". ";
        // obyektni tekshirish orqali sanasi
        if(typeid(*p) == typeid(triangle)) t++;
        if(typeid(*p) == typeid(rectangle)) r++;
        if(typeid(*p) == typeid(circle)) c++;
        // Yuzasini hisoblash
        cout << " S= " << p->area() << endl;
        }
cout << endl;
cout << "Quyidagi obyektlar hosil qilindi:\n";
cout << " Uchburchaklar: " << t << endl;
cout << " To‘rtburchaklar: " << r << endl;
cout << " Doiralar: " << c << endl;
system("pause");
return 0;
}
```

1.4 – dastur natijasi. Output

```
Obyektning tipi: class rectangle.  S= 8
Obyektning tipi: class triangle.  S= 28
Obyektning tipi: class circle.  S= 28.26
Obyektning tipi: class triangle.  S= 35
```

```
Obyektning tipi: class rectangle.  S= 12
Obyektning tipi: class triangle.  S= 55
Obyektning tipi: class rectangle.  S= 72
Obyektning tipi: class circle.  S= 254.34
Obyektning tipi: class rectangle.  S= 21
Obyektning tipi: class triangle.  S= 33
Obyektning tipi: class rectangle.  S= 10
Obyektning tipi: class circle.  S= 379.94
Obyektning tipi: class circle.  S= 254.34
Obyektning tipi: class triangle.  S= 4
Obyektning tipi: class triangle.  S= 36

Quyidagi obyektlar hosil qilindi:
 Uchburchaklar: 6
 To‘rtburchaklar: 5
 Doiralar: 4
```

typeid operatorini shablon sinflariga qo‘llash. typeid operatorini shablon sinflariga ham qo‘llash mumkin albatta. Shablon sinfi asosida hosil bo‘lgan obyekt tipi obyektni amalga oshirishda uning umumlashgan ma’lumotlari uchun qanday ma’lumotlardan foydalanilganligi asosida qisman aniqlanadi. Shuning uchun, turli ma’lumotlar yordamida yaratilgan bir xil shablon sinfga tegishli bo‘lgan ikki obyektlar tip xil bo‘ladi.

1.5-dastur. shablon sinflari bilan typeid operatoridan foydalanish.

```
#include "stdafx.h"
#include <iostream>
#include <typeinfo>
using namespace std;
template <class T>
class myclass {
    T a;
    public:
        myclass(T i) { a = i; }
    // . . .
};
int main(){
    myclass<int> o1(28), o2(1);
```

```
    myclass<double> o3(19.80);
    cout << " o1 obyekt tipi: ";
    cout << typeid(o1).name() << endl;
    cout << " o2 obyekt tipi: ";
    cout << typeid(o2).name() << endl;
    cout << " o3 obyekt tipi: ";
    cout << typeid(o3).name() << endl;
    cout << endl;

    if(typeid(o1) == typeid(o2))
        cout << " o1 va o2 obyektlar bir xil
tip.\n";
    if(typeid(o1) == typeid(o3)) cout <<
"Xatolik\n";
    else cout << "o1 va o3 obyektlar bir xil tip
emas.\n";
    system("pause");
return 0;
}
```

1.5 – dastur natijasi. Output

```
 o1 obyekt tipi: class myclass<int>
 o2 obyekt tipi: class myclass<int>
 o3 obyekt tipi: class myclass<double>

 o1 va o2 obyektlar bir xil tip.
 o1 va o3 obyektlar bir xil tip emas.
```

1.5-dasturdan ko‘rinib turibdiki, ikki obyektlar bir xil shablon sinf obyektlarida bo‘lsa-da, ularning parametrli ma’lumotlar mos bo‘lmasa, ular turi teng emas. Bu dasturda o1 obyekti myclass <int > tipida, o3 obyekti esa myclass <double>tipida bo‘ladi. Shunday qilib, bu obyektlar turli xilligini ko‘rsatadi.

typeid operatorini shablon sinflariga qo‘llashning yana bir misolini, ya‘ni 1.4-dastur geometrik shaklni aniqlash dasturining o‘zgartirilgan variantini ko‘rib chiqamiz. Bu safar figure sinfini shablon sinfiga aylantiriladi.

1.6-dastur. Figure ierarxiyasining shablon varianti.

```
#include "stdafx.h"
#include <iostream>
#include <typeinfo>
#include <cstdlib>
using namespace std;
template <class T>
class figure{
    protected:
        T x, y;
    public:
        figure(T i, T j) {
            x = i;
            y = j;
            }
        virtual T area() = 0;
    };
template <class T>
class triangle : public figure<T>{
    public:
        triangle(T i, T j) : figure<T>(i, j) {}
        T area() {
            return x * 0.5 * y;
            }
};
template <class T>
class rectangle : public figure<T>{
    public:
        rectangle(T i, T j) : figure<T>(i, j) {}
        T area() {
            return x * y;
            }
};
template <class T>
class circle : public figure<T>{
    public:
        circle(T i, T j=0) : figure<T>(i, j) {}
        T area() {
            return 3.14 * x * x;
```

```
        }
};
figure<double> *generator(){
    switch(rand() % 3 ) {
        case 0: return new circle<double> (rand() %
28);
        case 1: return new triangle<double>(rand()
% 28, rand() % 28);
        case 2: return new rectangle<double>
(rand() % 28, rand() % 28);
    }
    return 0;
}
int main(){
    figure<double> *p;
    int i;
    int t=0, c=0, r=0;
    for(i=0; i<15; i++) {
        p = generator();
        cout << "Obyekt tipi: " <<
typeid(*p).name();
        cout << ". ";
        if(typeid(*p) == typeid(triangle<double>))
t++;
        if(typeid(*p) == typeid(rectangle<double>))
r++;
        if(typeid(*p) == typeid(circle<double>))
c++;
        cout << " S= " << p->area() << endl;
        }
    cout << endl;
    cout << "Obyektlarni sanash:\n";
    cout << " uchburshaklar: " << t << endl;
    cout << " to‘rtburchaklar: " << r << endl;
    cout << " doiralar: " << c << endl;
    system("pause");
    return 0;
}
```

1.6 – dastur natijasi. Output

```
Obyekt tipi: class rectangle<double>.  S= 90
Obyekt tipi: class triangle<double>.  S= 136
Obyekt tipi: class circle<double>.  S= 615.44
Obyekt tipi: class triangle<double>.  S= 210
Obyekt tipi: class rectangle<double>.  S= 351
Obyekt tipi: class triangle<double>.  S= 202.5
Obyekt tipi: class rectangle<double>.  S= 44
Obyekt tipi: class circle<double>.  S= 803.84
Obyekt tipi: class rectangle<double>.  S= 156
Obyekt tipi: class triangle<double>.  S= 30
Obyekt tipi: class rectangle<double>.  S= 225
Obyekt tipi: class circle<double>.  S= 706.5
Obyekt tipi: class circle<double>.  S= 1384.74
Obyekt tipi: class triangle<double>.  S= 66.5
Obyekt tipi: class triangle<double>.  S= 15

 Obyektlarni sanash:
 uchburshaklar: 6
 to‘rtburchaklar: 5
 doiralar: 4
```

Dinamik tipni aniqlash har bir dasturda ishlatilmaydi. Ammo, dastur bajarilish vaqtida, polimorf turlari bilan ishlashda, undagi obyektlarning har qanday tipini aniqlashga imkonini beradi.

Tiplarni almashtirish operatorlari. C++ tilida beshta tipni almatirish operatorlari mavjud.

Ulardan birinchisi bu odatdagi (an‘anaviy) uslubda ishlatiladigan [()](cast operatori) azaldan C++ga qurilgan. U quyidagicha ishlatiladi:

```
    float f = (float)5;
bool b = (bool)5;
```

Tipni almatirish to‘rtta operatori (dynamic_cast, const_cast, reinterpret_cast va static_cast) 10 yil oldin C++ tilga qo‘shildi. Bu operatorlar tiplar bilan amallarni bajarganda tur xil imkoniyatlarni yaratib beradi Ularning har birini alohida ko‘rib chiqaylik.

Dynamic_cast operatori. Dynamic_cast operatori dastur bajarilishi davomida polimorf tipdagi keltirish amalini bajaradi. Yangi operatorlardan eng muhimi dynamic_cast tipidagi tip almashtirish

(kasting, casting) operatoridir. Dasturni bajarish davomida taklif qilinayotgan amalning bajarilishini tekshiradi. Agar belgilangan amal chaqirilganda yaroqsiz bo‘lsa, hech qanday turdagi kasting amalga oshirilmaydi. Dynamic_cast operatorini qo‘llashning umumiy formati quyidagicha:

```
dynamic_cast<type> (expr)
```

Bunda type elementi bu amalning maqsadi bo‘lgan yangi tipni, expr elementi esa bu yangi tipga almashadigan ifodani bildiradi. Tip turi ko‘rsatkich yoki mos yozuvlar bilan almashishi kerak va expr ifoda ko‘rsatkich yoki mos yozuvlar uchun tashlanishi kerak. Shu tarzda dynamic_cast operatoridan bir turdagi ko’rsatkichni boshqa turdagi ko’rsatkichga yoki bir turdagi murojaatni boshqa turdagi murojaatga aylantirish uchun foydalanish mumkin.

Bu operator asosan polimorf tiplar orasida dinamik tipni keltirish amallari uchun ishlatiladi. masalan, agar polimorf sinflar B va D bo‘lsin. sinf D va sinf V olingach, dynamic_cast operatori yordamida, har doim ko‘rsatkichni Dga aylantirish mumkin. Chunki *in pointer* tayanch sinfga ko‘rsatkich har doim bazasidan olingan sinf obyektini ko‘rsatish uchun foydalanish mumkin. Biroq, dynamic_cast operatori faqat ko‘rsatkich aylantirishi mumkin. D obyekt ko‘rsatkich uchun murojaat qilinganda, albatta, sinf D obyekti bo‘ladi. Umuman olganda, dynamic_cast operatori faqat polimorf tipdagi kastingga ruxsat berilsa, muvaffaqiyatli bajariladi, ya‘ni yangi tipdagi ko‘rsatikch ushbu yangi tipdagi obyektga yoki undan olingan obyektga havola qilishi mumkin. Aks holda, agar belgilangan tipdagi almashish amalini bajarib bo‘lmasa, bu amalda ko’rsatkichlar ishtirok esa dinamic_cast operatorining natijasi null qiymatga aylanadi. Agar ushbu amalni bajarilishi muvaffaqitsiz bo‘lsa, uzilishlar ishtirok etgan bo‘lsa, bad_cast istisnosi avtomatik bajariladi.

Oddiy misolni ko‘rib chiqamiz. Base asos sinfi polimorf sinf bo‘lsin va myClass sinf esa Base sinfidan olingan deb faraz qilaylik.

```
Base *bp, b_ob;
myClass *dp, d_ob;
bp = &d_ob; // myClass sinf obyektga Base sinf uchun ko‘rsatkich.
dp = dynamic_cast< myClass *> (bp); // myClass sinf ko’rsatkichga
almashtirish
if(dp) cout << " Tipni almashtirish muvaffaqiyatli bo‘ldi!";
```

Bu yerda bp ko‘rsatkichini (asos sinf) dp ko‘rsatkichiga (myClass sinf) almashtirish muvaffaqiyatli bo‘ladi, chunki bp aslida sinf obyektiga ishora qiladi.

myClass shuning uchun ushbu misolni bajarishda tipni almashtirish muvaffaqiyatli chiqdi. bp aslida sinf bazasini obyektga ishora qilgandi. Quyidagi misolda tipni almashtirishda myClass sinf obyekti bo‘lsa, ko’rsatkich tayanch sinfga obyektni tashlash uchun muvaffaqiyatsiz va noto‘g‘ri olingan hisoblanadi.

```
Base *bp, b_ob;
myClass *dp, d_ob;
bp = &b_ob; // Base sinfga ko‘rsatkich sifatida Base sinf obyekt
berilgan.
dp = dynamic_cast< myClass *> (bp); // xatolik
if(!dp) cout << " Tipni almashtirish muvaffaqiyatsiz ";
```

Ushbu dastur fragmentida tipni almashtirish amali muvaffaqiyatsiz bo‘ldi va tipni almashtirish uchun muvaffaqiyatsiz bo‘lganligini ekranga chiqaradi.

dynamic_cast operatoridan foydalanishda quyidagi 1.7-dasturda tipni almashtirish holatlari aniqlaymiz.

1.7-dastur. dynamic_cast operatoridan foydalanish.

```
#include "stdafx.h"
#include <iostream>
#include <typeinfo>
using namespace std;
class Base {
    public:
        virtual void f() { cout << " Base sinfi
orqali.\n"; }
// . . .
};
class myClass : public Base {
    public:
        void f() { cout << "myClass sinfi
orqali.\n"; }
};
int main(){
    Base *bp, b_ob;
    myClass *dp, d_ob;
```

```
    dp = dynamic_cast<myClass *> (&d_ob);

    if(dp) {
        cout << " tipni almashtirish" <<" (myClass*
=> myClass *) bajarildi. ";
             dp->f();
    } else cout <<"Xatolik\n";
    cout << endl;
    bp = dynamic_cast<Base *> (&d_ob);

    if(bp) {
        cout << " tipni almashtirish " <<"(myClass*
=> Base *) bajarildi. ";
        bp->f();
    } else cout << "Xatolik\n";
    cout << endl;

    bp = dynamic_cast<Base *> (&b_ob);
    if(bp) {
        cout << " tipni almashtirish " <<"(Base* =>
Base *) bajarildi. ";
        bp->f();
    } else cout << "Xatolik\n";
    cout << endl;
    dp = dynamic_cast<myClass *> (&b_ob);
    if(dp) cout <<"Xatolik\n";
    else
    cout <<" tipni almashtirish "<<"(Base*=>
myClass*) bajarilmadi.\n";
    cout << endl;

    bp = &d_ob; // bp myClass ob'eyktiga
koʻrsatkich
    dp = dynamic_cast<myClass *> (bp);
    if(dp) {
        cout << " bp ni myClass tipiga almashtirish
bajarildi. ";
        dp->f();
```

```
    } else cout << "Xatolik\n";
    cout << endl;

    bp = &b_ob; // br Base ob'eyktiga ko‘rsatkich
    dp = dynamic_cast<myClass *> (bp);
    if(dp) cout << "Xatolik";
    else cout <<" bp myClass tipiga almashtirish
bajarilmadi.\n";
    cout << endl;
    dp = &d_ob; // dp myClass ob'eyktiga
ko‘rsatkich
    bp = dynamic_cast<Base *> (dp);

    if(bp) {
        cout <<" dp ni myClass tipiga almashtirish
bajarildi. ";
        bp->f();
    }
    else cout <<"Xatolik\n";
    system("pause");
return 0;
}
```

1.7 – dastur natijasi. Output

```
tipni almashtirish (myClass* => myClass *)
bajarildi. myClass sinfi orqali.
tipni almashtirish (myClass* => Base *) bajarildi.
myClass sinfi orqali.
tipni almashtirish (Base* => Base *) bajarildi.
Base sinfi orqali.
tipni almashtirish (Base*=> myClass*) bajarilmadi.
bp ni myClass tipiga almashtirish bajarildi.
myClass sinfi orqali.
bp myClass tipiga almashtirish bajarilmadi.
dp ni myClass tipiga almashtirish bajarildi.
myClass sinfi orqali.
```

Dynamic_cast operatori ba‘zan typeid operatori o‘rniga ishlatilishi mumkin. Masalan, Base sinf polimorf va myClass sinf uchun asos sinf,

deb faraz qilaylik. Quyidagi dastur fragmentini bajarishda, obyekt aslida myClass sinf obyekti bo‘lsada dp ko‘rsatkich bp ko‘rsatkich tomonidan obyekt manzili beriladi.

```
Base *bp;
Derived *dp;
// . . .
if(typeid(*bp) == typeid(Derived))
dp = (Derived *) bp;
```

Bu holda, odatiy tipni almashtirish ishlatildi. Bu juda xavfsiz, chunki if operatori typeid operatori yordamida typeid amalining haqiqiyligini bajarilishidan oldin tekshiradi. typeid operatori va if operatori bilan almashtirish orqali yanada samaraliroq qilish mumkin.

```
dynamic_cast:
dp = dynamic_cast<myClass *> (bp);
```

Dynamic_cast operatori faqat obyektning belgilangan tipi yoki belgilangan tipi almashtirilgan tipi bilan bir obyekt bo‘lsa muvaffaqiyatli natija beradi, bu almashtirish tugagandan so‘ng, dp ko‘rsatkich olingan obyektga null qiymatini yoki ko‘rsatkichni ham o‘z ichiga oladi. Belgilangan tipni tashlash amali to‘g‘ri bajarilgan bo‘lsa dynamic_cast operatori faqat muvaffaqiyatli bajariladi, uning almashtirish mantig‘ini muayyan vaziyatlarda soddalashtirilgan mumkin. Quyidagi 1.8-dasturda typeid operatorini dynamic_cast operatori bilan almashtirish mumkinligini ko‘rsatilgan. Bu yerda bir xil amallar to‘plami ikki marta bajariladi, avval typeid operatori, so‘ngra dynamic_cast operatori yordamida.

1.8-dastur. typeid operatori o‘rniga dynamic_cast operatoridan foydalanish.

```
#include "stdafx.h"
#include <iostream>
#include <typeinfo>
using namespace std;
class Base {
    public:
        virtual void f() {}
};

class MyClass : public Base {
    public:
```

```
        void derivedOnly() {
            cout << "Bu MyClass obyekti.\n";
        }
};
int main(){
    Base *bp, b_ob;
    MyClass *dp, d_ob;
        //  typeid operatoridan foydalanish
    bp = &b_ob;
    if(typeid(*bp) == typeid(MyClass)) {
        dp = (MyClass *) bp;
        dp->derivedOnly();
    } else
    cout <<" Base tipidagi obyekt MyClass
almashtirilmadi. \n";
    bp = &d_ob;
    if(typeid(*bp) == typeid(MyClass)) {
        dp = (MyClass *) bp;
        dp->derivedOnly();
    } else
    cout <<"Xatolik, tip almashtirish yozilmagan!
\n";
    // dynamic_cast operatoridan foydalanish
    bp = &b_ob;
    dp = dynamic_cast<MyClass *> (bp);
    if(dp) dp->derivedOnly();
        else cout << " Base tipidagi obyekt MyClass
almashtirilmadi. \n";
        bp = &d_ob;
        dp = dynamic_cast<MyClass *> (bp);
        if(dp) dp->derivedOnly();
            else cout << "Xatolik, tip almashtirish
yozilmagan! \n";
    system("pause");
return 0;
}
```

1.8 – dastur natijasi. Output

```
 Base tipidagi obyekt MyClass almashtirilmadi.
Bu MyClass obyekti.
 Base tipidagi obyekt MyClass almashtirilmadi.
Bu MyClass obyekti.
```

1.8-dasturdan ko‘rinadiki, asos sinfning ko‘rasatkichini MyClass ko‘rsatkichiga almashtirishda dynamic_cast operatorini qo‘llash dastur mantig‘ini qulayroq qiladi. Yuqoridagi dastur natijasiga qarang. Shuningdek, dynamic_cast operatorini shablon sinflarga ham qo‘llash mumkin.

const_cast operatori. const_cast operatori const va/yoki volatile o‘zgartirgichlarni qayta aniqlash uchun xizmat qiladi.

const_cast operatori const va / yoki volatile o‘zgartirgichlarni ochiq qayta aniqlash/joriy qilish uchun ishlatiladi. const yoki volatile ning atrubutlariga istesno tariqasida, yangi tipi joriy tipi bilan mos bo‘lishi kerak. Ko‘pincha const_cast operatori const atributini olib tashlash uchun ishlatiladi. Uning umumiy formati quyidagicha:

```
const_cast<type> (expr)
```

Bu yerda type elementi yangi tipga almatirish tipini o‘rnatadi, expr elementi esa yangi tipga tipi almatirilishi kerak bo‘lgan o‘zgaruvchi(ifoda)ni bildiradi.

1.9-dastur. const_cast operatoridan foydalanish.

```
#include "stdafx.h"

#include <iostream>
#include <typeinfo>

using namespace std;

void f (const int *p){
    int *v;
    v = const_cast<int *> (p); //  const-
o‘zgaruvchini qayta almashtirish.
    *v = 1980; // yangi qiymat berish,
modifikatsiyalash
}

int main(){
```

```
    int x = 2004;
    cout <<" f() funksiya chaqirilmasdan oldin x ni
qiymati: " << x<< endl;
    f (&x);
    cout <<" f() funksiya chaqirilgandan so‘ng x ni
qiymati: " << x<< endl;
    system("pause");
return 0;
}
```

1.9 – dastur natijasi. Output

```
f() funksiya chaqirilmasdan oldin x ni qiymati:
2004
f() funksiya chaqirilgandan so‘ng x ni qiymati:
1980
```

Ko‘rib turganingizdek, x o‘zgaruvchi qabul qilgan parametr const ko’rsatkichi sifatida o‘rnatildi va f() funksiyasi tomonidan o‘zgartirildi.

const atributini o‘chirish uchun const_cast operatoridan foydalanish xavfli darajadagi vosita ekanligini alohida ta‘kidlash lozim. Shuning uchun, juda diqqat bilan foydalanishni tavsiya qilamiz.

Faqat const_cast operatori const atributini o‘chirishi mumkinligini bilib oldingiz. Boshqa so‘zlar bilan aytganda, na dynamic_cast, static_cast, yoki reinterpret_cast obyekt const atributini o‘zgartirish uchun foydalanish mumkin emas.

static_cast operatori. Static_cast operatori polimorf bo‘lmagan tipdagi ko‘rsatkichlarni almashtirish uchun foydalaniladi. Har qanday tipni almashtirish uchun foydalanish mumkin. Biroq, dastur bajarilatgan vaqtda hech qanday tekshirish amalga oshirilmaydi. Static_cast operatori quyidagi umumiy yozuv formatiga ega:

```
static_cast<type> (expr)
```

Bu yerda type elementi yangi tipdagi almashtirish tipini belgilaydi va expr elementi bu yangi tipga tipi almashtirilishi kerak bo‘lgan ifodani bildiradi.

Static_cast operatori tipni almashtirishning orginal operatoridir. U faqat polimorf bo‘lmagan tiplar uchun amalga oshiradi. Masalan, quyidagi 1.10-dasturni ishlaganda float tipidagi o‘zgaruvchi int tipiga aylantiradi.

1.10-dastur. Static_cast operatoridan foydalanish.

```
#include "stdafx.h"
```

```
#include <iostream>

using namespace std;

int main(){
    int i;
    float f;
    f = 199.22F;
    i = static_cast<int> (f);
    cout << i;
    return 0;
  system("pause");
}
```

reinterpret_cast operatori. reinterpret_cast operatori fundamental tipni almashtirishni amalga oshiradi.

reinterpret_cast operatori tipni tubdan farq qiladigan tipga aylantiradi. Masalan, ko‘rsatkichni int tipiga hamda int qiymati uchun ko‘rsatkichga aylantirish uchun foydalanishingiz mumkin. Bundan tashqari, tabiatan mos kelmaydigan ko‘rsatkich tiplarni tashlash uchun foydalanish mumkin. Ushbu operator quyidagi umumiy formatga ega:

```
reinterpret_cast<type> (expr)
```

Bu yerda type elementi yangi tipdagi tipni almashtirishni belgilaydi va expr elementi bu yangi tipga almatiriladigan ifodani bildiradi.

reinterpret_cast operatoridan foydalanish quyidagi 1.11-dasturda ko‘rsatilgan.

1.11-dastur. reinterpret_cast operatoridan foydalanish.

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(){
    int i;
    char *p = "Salom Buxoro";
    i = reinterpret_cast<int> (p); // ko‘rsatkichni
int tipiga almashtirish.
```

```
    cout << i<<endl;
    system("pause");
return 0;
}
```

Bu yerda reinterpret_cast operatori p ko‘rsatkichni butun son int qiymatiga aylantiradi. Bu o‘zgartirish asosiy tipni o‘zgarish ifodalaydi. Bu tip almashtirishni oddiy kasting bilan ham amalga oshirish mumkin.

Yangi to‘rtta cast operatorlari bilan an‘anaviy tipni almashtirish amali (kasting) qanday farq mavjud. Ba‘zi dasturchilar to‘rt cast operatorlari yuqorida tasvirlangan butunlay an‘anaviy tipni almashtirish amalining o‘rniga deb o‘ylashlari mumkin. Ularning o‘rniga har doim odatiy tipni almashtirishning yangi vositalaridan foydalanish kerakmi? – degan savol qiziqtirishi mumkin. Aslida barcha dasturchilar uchun umumiy qoida yo‘q. Yangi operatorlar bir ma’lumot tipini boshqasiga almashtirishni ancha xavfli amallar xavfsizligini yaxshilash uchun yaratilganligi sababli, ko‘pgina C++ dasturchilari faqat shu maqsadda ishlatilishi kerakligiga ishonch hosil qilgan. Bunga e‘tiroz qilib bo‘lmaydi. Boshqa dasturchilar an‘anaviy tipi almashtirish amali ko‘p yillar davomida ularga "sadoqat" bilan xizmat qilib bergan, undan shunday voz kechish oson emas. Masalan, oddiy va nisbatan xavfsiz tipdagi kasting amallarini bajarish uchun read() va write() funksiyalarini chaqirganda talab qilinadi.

Ammo shunday dasturlashda jarayonlar borki, ularda cast operatorlaridan foydalanish maqsadga muvofiqdir, masalan, polimorf tiplar uchun cast amallarini bajarishda albatta dynamic_cast operatoridan foydalanishga to‘g‘ri keladi.

Yangi nomlar fazosi yaratish. Nomlar fazosi haqida qisqacha ma’lumotga ega bo‘lsangiz kerak. Endi bu tushunchani batafsilroq ko‘rib chiqamiz va foydalanuvchining nomlar fazosi qanday yaratilishini ko‘rib chiqamiz. Namespaces tushunchasi C++ dasturlash tilida nisbatan yaqinda paydo bo‘lgan tushunchalarda biridir. Ular nomlarning o‘zaro bir xilligini oldini olish (kolliziya saqlanish) uchun identifikatorlarning nomlarini lokalizatsiya qilish uchun mo‘ljallangan. Nomlar fazosi tushunchasi kiritilishidan oldin, bu nomlarning barchasi bir xil global nomlar bo‘lib, ko‘plab nizolarni (xatoliklarni, o‘zaro bir xilliklarni) keltirib chiqargan. Agar dasturda dasturchi o‘zining toupper() funksiyasini yaratgan bo‘lsa, u holda standart kutubxona toupper()

funksiyasini (uning parametrlari ro‘yxatiga qarab) almashtirishi mumkin. Ammo, ikkala funksiyaning nomlari bir xil global namespaceda saqlangan bo‘ladi. Bir xil dasturlarni turli ishlab chiqaruvchilarning funksiyalari va sinflar, kutubxonalar foydalanganda nomlar bir xil bo‘lgan funksiyalar dastur bajarilishida ziddiyatlarga duch keladi. Bunday holda, bir kutubxonada aniqlangan nomlar boshqa kutubxonada aniqlangan nomlar bilan to‘qnashishi mumkin.

Barcha muammolar nomlar fazosi tushunchasi va namespace kalit so‘z tomonidan hal qilindi. Bu kalit so‘z yangi nomlar fazosi e‘lon qilish orqali barcha funksiya, sinf va kutubxonalarni mahalliylashtirish imkonini beradi. Bu nomlar fazosi turli kontekstlarda bir xil nomdagi funksiya va sinflar ziddiyatga olib kelmasdan ishlatish imkonini beradi. std standart kutubxona (std -Standart Library) juda ko‘p foydalanilgan bo‘lishi mumkin. Tilning dastlabki variantlarida C++ kutubxonasi global nomlar fazosida belgilangan. Sezilarli darajadagi nomlar ziddiyatlari ehtimolini kamaytirish uchun std namespace nomlar fazosi yaratilgan. Bundan tashqari, ziddiyatlarni oldini olish uchun nomlar fazasi ko‘lamini mahalliylashtirish uchun dasturda o‘z nom fazongizni yaratishingiz mumkin. Bu shaxsiy sinf yoki funksiya kutubxonalar yaratishda muhim ahamiyatga ega.

Namespace kalit so‘zi nomlar fazosi sohalarini qo‘shib global namespacedan ajratish imkonini beradi. U aslida nomlar fazosi doirasini belgilaydi. Namespace kalit so‘zidan foydalanishning asosiy shakli quyida ko‘rsatilgan:

```
namespace Space{
    //....
}
```

Namespace kontentida aniqlangan ixtiyoriy har bir element, bu namespace doirasida bajariladi.

1.12-dastur. myNameSpace nomli namespace e‘lon namuna.

```
namespace MyNameSpace {
    int i,k;
    void myfunc (int j) { cout <<j; }
    class myclass {
        public:
            void set(int x) { i = x; }
            int get() { return i; }
    };
};
```

Mynamespace nomli namespace da *i* va *k* o‘zgaruvchilarning nomlari, myfunc() funksiyasi va myclass sinfi yaratilgan.

Bu namespace ichida o‘zgaruvchilarga, funksiya va sinflarga to‘g‘ridan-to‘g‘ri murojaat qilish va yangisini kirish mumkin. Biroq, namespace kalit so‘zi ma’lum bir dastur sohasini belgilaydi, chunki, namespace tashqaridan namespace ichida e‘lon qilingan obyektlardan foydalanish imkoniyatini faqat ziddiyatlarni oldini olgan holda foydalanishga ruxsat beradi. Masalan, Mynamespace nomli namespace tarkibiga kirmagan dasturning bir qismida *i* o‘zgaruvchiga 28 qiymat berish uchun quyidagicha ifoda yozish lozim:

```
MyNameSpace::i = 28;
```

Mynamespace nomli namespace sohasiga kiradigan Myclass tipidagi obyektni odatdagidek e‘lon qilishimiz mumkin, ammo kirmagan Myclass tipidagi obyektni quyidagicha e‘lon qilish lozim:

```
MyNameSpace::myclass ob;
```

Shunday qilib, bu soha tashqarida nomlar fazosining nomi bilan kengaytirish operatori (::) bilan murojaat qilish mumkin.

Agar dasturda namespace a’zolariga tez-tez murojaat bo‘ladigan bo‘lsa, har doim uning nomi va kengaytirish operatori yozish dasturning ko‘rinishiga va dasturchining ko‘p yozishiga sabab bo‘lishi mumkin. Ushbu muammoni hal qilish uchun foydalanuvchining nomlar fazosidan foydalanish fragmenti ishlab chiqilgan. Ushbu fragment ikkita asosiy shaklga ega:

```
using namespace MyNameSpace;

using MyNameSpace::myclass;
```

Fragmentning birinchi qatorida keltirilgan shaklda a’zolar kirmoqchi bo‘lgan nomlar fazosining nomini bildiradi. Foydalanish fragmentida bu shakl yordamida yozilgan nomlar fazosidagi barcha a’zolarga joriy nom mavjud deb olinadi va to‘g‘ridan-to‘g‘ri murojaat qilish mumkin, fazo nomi va kengaytirish operatori har safar yozish kerak bo‘lmaydi.

Fragmentning ikkinchi qatorida using fragmentining ikkinchi shaklidan foydalanilganda faqat fragmentda ko‘rsatilgan nomdosh a’zosi ko‘rinadi. Masalan, yuqorida ta‘riflangan mynamespace nomli namespace bo‘lsa, unda quyidagi fragmentlar va operatorlaridan foydalanish to‘g‘ri:

```
using MyNameSpace::k;
```

```
int main(){
    k = 10;
return 0;
}
```

Fragment asosida faqat k o‘zgaruvchi ko‘rinadi. Demak, ko‘rsatma to‘g‘ri.

```
using namespace MyNameSpace;

int main(){
    k = 10;
    i = 10;
    myclass ob;
    myfunc(25);
return 0;
}
```

Bu fragmentda MyNameSpace fazoning barcha a’zolari ko‘rinadi demak, ko‘rsatma to‘g‘ri.

Shuningdek, bir xil nom bilan bir nechta fazoni e‘lon qilish imkoniyati mavjud. Bu bir necha fayllar ichiga fazolarni ajratish imkonini beradi, yoki hatto bitta fayl ichida fazolarni ajratish imkonini. Buni dasturchining dasturga ishlatayotgan funksiya va sinflarini ajratish uchun qo‘llash mumkin. Quyidagi misolni ko‘rib chiqamiz:

```
namespace NS { int a;}
namespace NS { int b;}

int main(){
    NS::a = 10;
    NS::b = 10;
return 0;
}
```

Bu yerda NS namespace ikki qismga bo‘lingan. Shunga qaramay, har bir qismi a’zolariga ham bir xil nom bilan murojaat qilingan, ya‘ni NS fazo.

Bir dastur faylida bir nomli fazolarni e‘lon qilish maqsadga muvofiq. Ammo, dasturda 2 va undan ortiq dastur fayllari ishlatilsa va fazolarni birlashtirish kerak bo‘lsa, shunda fazoni ikkiga ajratish usulidan foydalanish mumkin. Boshqa so‘zlar bilan aytganda, bir namespace faqat boshqa namespace bo‘lishi mumkin emas. Bu bir nomli

fazo e‘lon qilinishi mumkin emas degan ma‘noni anglatadi, masalan, bir funksiya ichida.

Nomsiz fazo - fazoning maxsus turidir. Muayyan fayl ichida noyob identifikatorlar, a’zolarni yaratish imkonini beradi. Nomsiz fazoning asosiy shakli quyidagicha bo‘ladi:

```
namespace {
    const float f = 3.1415;
    int myIf(int a, int b){return a>b?1:0;}
}
```

Fazolarni nomsiz nomlashlar faqat bitta fayl doirasida ma’lum bo‘lgan noyob identifikatorlar va funksiya, sinflarni o‘rnatish imkonini beradi. Shu tarzda, noma’lum nomli fazoni o‘z ichiga olgan fayl ichida, ushbu fazoning a’zolariga to‘g‘ridan-to‘g‘ri, hech qanday tushuntirishga muhtoj bo‘lmasdan foydalanish mumkin. Ammo bu fayldan tashqarida bunday identifikatorlar noma’lum bo‘lib qoladi.

Siz kichik va o‘rta dasturlar uchun o‘z fazolaringizni yaratishingiz shart emas. Ammo, agar qayta foydalanish uchun mo‘ljallangan fnuksiyalar, sinflar va/yoki sinflar kutubxonalar yaratish uchun ba‘zi namespace ichida sizning kodlaringiz joylashtirish yozgan dasturingizni tushunarliroq qiladi.

1.13-dasturda nomlar fazosini yaratish va foydalanishga misol keltirilgan.

1.13-dastur. Nomlar fazosini yaratish va foydalanish.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
// birinchi nomlar fazosi
namespace NS1{
    class demo{
        int i;
        public:
            demo(int x) { i=x; }
            void set(int x) { i=x; }
            int get() { return i; }
    };
    char str[] = "Nomlar fazolarini o‘rnatish\n";
    int counter;
}
// ikkinchi nomlar fazosi
```

```
namespace NS2 { int x, u; }

int main(){
    NS1::demo ob(10);
    cout <<" ob = " << ob.get() << endl;
    ob.set(99);
    cout << "Endi ob = "<< ob.get() << endl;

    using NS1::str; cout << str;

    using namespace NS1;
    for(counter = 1; counter<=10; counter++)
        cout << counter << ends;
    cout << endl;

    NS2::x = 10; NS2::u = 20;
    cout << "o‘zgaruvchilar x, y = "<< NS2::x<<", "
<< NS2::u << endl;

    using namespace NS2;
       demo x_ob(x), y_ob(u);
       cout << "x_ob, y_ob obyektlarning qiymati:
"<< x_ob.get(); cout<< " " << y_ob.get() << endl;
system("pause");
return 0;
}
```

1.13 – dastur natijasi. Output

```
ob = 10
Endi ob = 99
Nomlar fazolarini o‘rnatish
1 2 3 4 5 6 7 8 9 10
o‘zgaruvchilar x, y = 10, 20
x_ob, y_ob obyektlarning qiymati: 10 20
```

Dasturda bir muhim nuqta ko‘rsatilgan: bir nechta nomlarni birgalikda ishlatganda, bir nomni boshqasi almashtirmaydi. Hozirgi fragmentda ba‘zi nomlarni kiritganingizda, uning nomlari o‘sha paytda boshqa nomdagi nomlarning o‘zida bo‘lishidan qat‘i nazar, bu

fragmentda qo‘shiladi. Shunday qilib, dastur bajarilayotgan vaqtda, std, NS1 va NS2 namespaces global namespace bo‘lib qo‘shildi.

Yuqorida aytib o‘tilganidek, fayllar orasida yoki bitta fayl ichida nomlar fazosini ajratish mumkin, keyin esa bu nomlar fazosining a’zolarini mazmunan birlashtiriladi. Birgalikda nomlar fazosini birlashtirish misolini ko‘rib chiqamiz.

1.14-dastur. Namespace larni birlashtirish.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
namespace Demo {
    int a; // Demo fazoda a o‘zgaruvchi
}
int x; // Global fazoda x o‘zgaruvchi
namespace Demo {
    int b; // Demo fazoda b o‘zgaruvchi
}
using namespace Demo;
int main(){
    a = b = x = 101;
    cout << a << " " << a << " " << x << endl;
system("pause");
return 0;
}
```

1.14 – dastur natijasi. Output

```
101 101 101
```

Bu misolda a va b o‘zgaruvchilari ham bir xil fazoda - Demo fazoda va x o‘zgaruvchisi global fazoda e‘lon qilingan. Ammo, using namespace Demo; fragmenti bilan, a va b o‘zgaruvchilarini va global o‘zgaruvchini ishlatish mumkin.

Yuqorida aytib o‘tilganidek, C++ standart o‘z std fazosiga ega va barcha kutubxonalarni qamrab oladi. Shuning uchun ushbu kitobdagi barcha dasturlar quyidagi ko‘rsatmaga ega:

```
using namespace std;
```

Bu fragment std namespace joriy qiladi va bevosita C++ standart kutubxonasida belgilangan funksiyalari va sinflar nomlarini kirish imkonini beradi. std namespace kengaytirish operatori yordamida har doim ishlatish maqsadga muvofiq emas.

Biroq, agar xohlasangiz, har bir identifikatordan oldin std namespace nomini va kengaytirish operatorini qo‘yishingiz mumkin, xato bo‘lmaydi. Masalan, quyidagi 1.15-dasturda standart std kutubxonasi global doirada kiritilmagan.

1.15-dastur. Ochiq foydalanish uchun nom ko‘rsatilgan.

```
#include "stdafx.h"
#include <iostream>
int main(){
    double val;
    std::cout << "Son kiriting:";
    std::cin >> val;
    std::cout << "Son:";
    std::cout << val << std::endl;
system("pause");
return 0;
}
```

Ushbu dasturda ko‘rsatilganidek, standart kirish va chiqish oqimlari cin va cout operatorlaridan foydalanish uchun nomlaridan oldin ularning qaysi fazodanligini belgilash kerak.

Agar dasturingiz C++ standart kutubxonasidan keng foydalanish zarur bo‘lmasa, std namespaceni global doirada kiritish shart emas. Ammo, agar sizning dasturingiz standart kutubxona a’zolari uchun minglab murojaatlarni o‘z ichiga olsa, har a’zo uchun std fazosini chaqirish juda ham oson kechmaydi. Shuning uchun uni global kiritish nisbatan dasturchi oson va std barcha standart identifikatorlarni o‘z ichiga oladi.

Agar dasturingizda standart kutubxonadan faqat bir necha a’zodan foydalansangiz, bu bir necha a’zolarni alohida belgilash uchun using fragmentidan foydalanish qulayroq bo‘lishi mumkin. Ushbu yondashuvning afzalligi shundaki, siz dasturda ko‘rsatilgan a’zolarni fazosini ko‘rsatmasdan kiritishingiz mumkin va ayni paytda butun standart kutubxonani global nomda kiritishingiz shart emas. Masalan:

```
#include "stdafx.h"
#include <iostream>

using std::cin;
using std::cout;
int main(){
    double val;
```

```
    cout << "Son kiriting:";
    cin >> val;
    cout << "Son:";
    cout << val << std::endl;
system("pause");
return 0;
}
```

Bu yerda standart kirish va chiqish oqimlari cin va cout operatorlari to‘g‘ridan-to‘g‘ri ishlatilishi mumkin, ammo ayni paytda std fazoning boshqa a’zolar joriy sohadan tashqarida qoldiriladi. Maslan, endl;

Avvalroq, C - tilining kutubxonasi global nomda belgilanganligi haqida aytib o‘tilgan edi. Agar eski C dasturini yangilashingiz kerak bo‘lsa, siz ham unda foydalanish uchun namespace std foydalanish kerak bo‘ladi, yoki kengaytirish operatori va std namespace nomini kiritish lozim. Agar yangi uslub sarlavhalari bilan eski header fayllarni o‘rniga, ayniqsa, muhim ahamiyatga ega (*.h). Eski header fayllar global namespace o‘z mazmunini joylashtiradi va yangi uslub sarlavhalari uchun std namespace o‘z mazmunini joylashtiradi.

Agar, nomlar fazosini global sifatida bitta faylga e‘lon qilib, cheklamoqchi bo‘lsangiz, S tilida statik deb e‘lon qilish kerak edi, ya‘ni, statik identifikator bilan. Masalan, quyidagi ikki fayl bir xil dasturning bir qismi deb faraz qilaylik:

Birinchi fayl	Ikkinchi fayl

`static int counter;`
`void f1(){`
`    counter == 99; //`
`to‘g‘ri`
`}`
	`extern int counter;`
	`void f2() {`
	`    counter = 10; // Xato`
	`}`

counter o‘zgaruvchisi birinchi faylda aniqlanganligi uchun ham uni birinchi faylda ishlatishingiz mumkin. Ikkinchi faylda counter o‘zgaruvchisi bilan fragmentdagi extern identifikatorini ko‘rsatishiga qaramay, bu o‘zgaruvchini ishlatishga urinish xatolikka olib keladi. Birinchi fayldagi counter o‘zgaruvchisini statik deb e‘lon qilib, uning ko‘lamini shu faylga cheklangan.

Statik identifikator bilan global o‘zgaruvchilar deklaratsiyalari C++ hali ham mavjud bo‘lsa-da, bu qochish maqsadda eng yaxshi yo‘l Nomlar fazosidan foydalanish hisoblanadi, quyidagi misolda ko‘rsatilgandek.

Birinchi fayl	Ikkinchi fayl

namespace {
int counter;
}
void f1(){
counter == 99; //
to‘g‘ri
}
	extern int counter;
	void f2() {
	counter = 10; //
	to‘g‘ri
	}

Buferlashtirilgan kiritish va chiqarish. C++ tilida ekran va faylga IO dan tashqari kiritish va chiqarish vazifalarini mahalliylash uchun massivlardan foydalanadigan qator funksiyalarni qo‘llab-quvvatlaydi. Massivlarga asoslangan kiritish/chiqarish konseptual (array-based I/O) yechimga o‘xshasada, C tilining IO funksiyalari bilan analog(bir-birini o’rnini bosuvchi)dir (ayniqsa, sscanf() va sprintf() funksiyalari). Massivlar yordamida kiritish/chiqarish C++ tilida juda moslashuvchan va foydalidir, chunki unda foydalanuvchi tomonidan yaratilgan o‘zgaruvchilarni kiritish va chiqarishda ham foydalaniladi. Massivlar bilan kiritish/chiqarishning barcha jihatlarini qamrab olish mumkin emas, ammo, eng muhim va tez-tez ishlatiladigan xususiyatlari amalga oshirish mumkin.

Massivlar yordamida IOni amalga oshirish uchun ham oqimlarni talab qilishini tushunish muhimdir. Oldingi ko‘nikmalaringizdan C++ da IO amalarini bilishingiz mumkin. Biroq, kiritish/chiqishda parametrlar sifatida obyektlardan foydalanishning barcha afzalliklari haqida dasturchilarning ko‘nikmalari muhimdir. Shuningdek, ma’lumot olish uchun, siz bir necha yangi xususiyatlarni boshqaradigan funksiyalar bilan tanishishingiz kerak. Bu funksiyalar istalgan oqimni ma’lum xotira maydoniga bog‘lash uchun mo‘ljallangan. Bu amallar tugagandan so‘ng, barcha IO xusisiyatlari allaqachon dasturchi biladigan IO vazifalari yordamida amalga oshiriladi degan ko‘nikmaga olib keladi.

Kiritish/chiqarish obyektlari sifatida massivdan foydalanishni boshlashdan oldin, <strstream> kutubxonasini dasturingizga kiritilgan bo‘lishi kerak. Bu kutubxonadagi istrstream, ostrstream, va strstream sinflardan foydalanish mumkin. Bu sinflar, o‘z navbatida, kiritish, chiqish va kirish/chiqish uchun oqimlardan foydalanish asoslangan. Ios sinfi bu sinflar uchun tayanch sinfdir, shuning uchun istream, ostream va iostream sinflarining barcha funksiyalari va manipulyatorlari istrstream, ostrstream

va strstream sinflarida ham mavjud. Satr massivini chiqish uchun ostrstream sinf konstruktordan quyidagi asosiy shaklda foydalaniladi:

```
ostrstream chiqish_oqimi(char *bufer, streamsize size, openmode mode = ios::out);
```

Bu yerda, chiqish_oqimi massivli satr bilan bog‘laydigan oqim bo‘lib, buffer pointer (buferlangan ko‘rsatkich) orqali beriladi. size parametri massivning o‘lchamini belgilaydi. mode - bu standart rejim parametri, odatda normal chiqish holatiga o‘rnatiladi, lekin siz ios sinfda belgilangan har qanday chiqish rejimi indentifikatori o‘rnatishingiz mumkin.

Chiqish uchun massiv ochilganda, simvolar massivga sig‘guncha joylashtiriladi.To‘lgan massiv bo‘lishi mumkin emas. Ixtiyoriy to‘lgan massiv kiritish/chiqarishni xatolikka olib kelishi mumkin. Masssivdagi belgilar sonnini aniqlash uchun pcount() funksiya-a’zosi ishlatiladi.

```
int pcount()
```

Funksiya faqat bir oqim bilan aloqada bo‘lib, shu munosabati bilan chaqiriladi va u massivga yozilgan belgilar sonini qaytaradi. Shu jumladan null tugatishni bildiradi.

Undan massivga kiritish uchun istrstream sinf konstruktorining quyidagi shaklidan foydalaniladi.

```
istrstream kirish_oqimi (const char * bufer) ;
```

Bu yerda, bufer belgilar kiritiladigan massivga ko’rsatkich hisoblanadi va kirish_oqimi kirish oqimi bilan ko‘rsatiladi. Biror massivdan kirish ma’lumotlarini o‘qishda eof() funksiyasidan foydalaniladi va massivning oxiriga yetganida true qaytaradi.

Kirish/chiqish uchun massiv ochish uchun strstream sinf konstruktor quyidagicha shakldan foydalaniladi:

```
strstream kch_oqimi(char * bufer, streamsize size, openmoda mode = ios : : in | ios: : out) ;
```

Bu yerda, kch_oqimi – bu kiritish/chiqish oqimi, belgilar o‘lchamida belgilangan uzunligidagi massivga bufer ko‘rsatkich orqali kirish/chiqish obyekti sifatida ishlatiladi.

Yuqorida keltirilgan barcha IO vazifalari, shuningdek, oqim bilan ishlash, ikkilik, shu jumladan tasodifiy kirish uchun muhimligini tushinish lozim.

Massivlar uchun oqim sinflaridan foydalanish C++ standarti tomonidan kam qo‘llab-quvvatlanadi. Kelajakda C++ versiyalarida qo‘llab-quvvatlanishi bo‘lmasligi ham mumkin. Belgilarni massivlar

oqimlari uchun mo‘ljallangan bir xil vazifalar C++da sinf-konteyner asosida takomillashtirilgan.

1. Massivga ma’lumotlarni yozish va chiqish uchun oddiy misol keltiramiz.

1.16-dastur. Massivga chiqarish.

```
#include "stdafx.h"

#include <iostream>
#include <strstream>
using namespace std;
int main(){
char buf[255]; // chiqarish uchun bufer
ostrstream ostr(buf, sizeof buf); // chiqarish
uchun massiv
ostr << "kirish/chiqish massiv oqimi bilan
ishlaydi.\n";
ostr << "oddiy kirish/chiqish kabi: \n " << 100;
ostr <<' ' << 123.23 <<'\n';
// manipulyatordan ham foydalanish mumkin
ostr << hex << 100 << ' ';
// identifikatordam ham
ostr.setf(ios::scientific);
ostr << 123.45 << ' ';
ostr << ends;
// oxirgi natijani chiqarish
cout << buf;
system("pause");
return 0;
}
```

1.16 – dastur natijasi. Output

```
kirish/chiqish massiv oqimi bilan ishlaydi.
oddiy kirish/chiqish kabi:
 100 123.23 64 1.234500e+002
```

Dasturda IO operatorlari qayta aniqlangan hamda kiritish/chiqarish uchun ajralmas manipulyatorlar a’zo funksiyalari va massivdan foydalanishda to‘liq format indentifikatorlari oqimlar uchun

foydalanilgan. Bu barcha manipulyatorlar uchun amal qiladi va lokal sinflar uchun IO operatorlari yaratish imkonini beradi.

Yuqoridagi dasturda ends manipulyatori massivga tugatish null belgisini qo‘shish uchun ishlatiladi. Null belgi avtomatik ravishda massivga kiritilgan yoki yo‘qligini amalga oshirish dasturchining o‘ziga bog‘liq. Shuning uchun, bu dasturchi uchun muhim, chunki massiv qatorining oxiriga null belgi yozish yaxshidir.

1.17-dastur. Massivga ma’lumotlarni kiritish.

```
#include "stdafx.h"

#include <iostream>
#include <strstream>
using namespace std;
int main(){
char buf[] = "SalomBuxoro 1980 28.01 B";
istrstream istr(buf);
char str[80]; float f; char c; int i;
istr >> str >> i >> f >> c;
cout << str << ' ' << i << ' ' << f << ' ' << c <<
'\n';
system("pause");
return 0;
}
```

1.17 – dastur natijasi. Output

```
SalomBuxoro 1980 28.01 B
```

Bu dastur bufer ko‘rsatkichida massivda mavjud ma’lumotlarni o‘qiydi va qayta ishlab ekranga chiqaradi.

Agar massiv oqim bilan bog‘langan bo‘lsa, fayl kabi ishlatiladi. Masalan, massiv ma’lumotlarni bufer ko‘rsatkichi bilan o‘qish va dasturda eof() va get() foydalanishni ifodalaydi

1.18-dastur. Massivlar asosida eof() va get() funksiyalarning IO da foydalanish.

```
#include "stdafx.h"
#include <iostream>
#include <strstream>
using namespace std;
int main(){
    char buf[] = "SalomBuxoro 1980 28.01 B";
```

```
    istrstream istr(buf);
    char c;
    while(!istr.eof()) {
        istr.get(c);
        if (!istr.eof()) cout << c;
    }
    cout<<endl;
    system("pause");
return 0;
}
```

1.18 – dastur natijasi. Output

```
SalomBuxoro 1980 28.01 B
```

Quyidagi dastur massiv ma’lumotlarni kiritish va chiqarishga oid.

1.19-dastur. Massivlar yordamida kirish/chiqish.

```
#include "stdafx.h"
#include <iostream>
#include <strstream>
using namespace std;
int main(){
    char iobuf[255];
    strstream iostr (iobuf, sizeof iobuf) ;
    iostr << "tekshrish \n";
    iostr << 100 << hex << ' ' << 100 << ends;
    char str[80]; int i ;
    iostr.getline(str, 79);
    iostr >> dec >> i;
    cout << str << ' ' << i << ' ';
    iostr >> hex >> i;
    cout << hex << i << endl;
    system("pause");
return 0;
}
```

1.19 – dastur natijasi. Output

```
tekshrish  100 64
```

Buferlashtirilgan kiritish va chiqarishda C++ da IO ni boshqarish vazifalari uchun a’zo funksiyalarini, identifikatorlarni va manipulyatorlarni

formatlash orqali ta‘minlanadi. Indentifikatorlar, funksiyalar va manipulyatorlar bir xil vazifani bajaradi - ular axborotni kiritish/chiqarish uchun maxsus formatni o‘rnatadi. C++ da ekrandan/ga kiritish/chiqarish mos ravishda cin va cout operatorlari yordamida amalga oshiriladi, ya‘ni formatlash manipulyatorlari ushbu IO operatorlari bilan birgalikda ishlatiladi. Indentifikator vazifalari va formatlash manipulyatorlar o‘rtasidagi farq ularga qo‘llaniladigan usuldir. Endi formatlash obektlaridan foydalanish yo‘llarini ko‘rib chiqamiz.

Asosiy formatlash a’zo funksiyalari:

cout.fill('symbol'); to‘ldiruvchi belgini o‘rnatadi. Bunda symbol to‘ldiruvchi belgi va tirnoq ichida beriladi.

cout.width(width_field); maydon uzunligini berish. width_field – pozitsiyalar soni, har bir pozitsiyaga bitta simvol sig‘adi.

cout.precision(number); haqiqiy sonlarda nuqtadan keyingi belgilar sonini berish. number - haqiqiy sonlarda nuqtadan keyingi belgilar soni.

Funksiyalarga nuqta amali orqali kirishiladi va qavslar ichida kerakli argument yoziladi. Fill() funksiyasiga argumentni bitta qo‘shtirnoq bilan belgi sifatida yoki son(belgi kodi) sifatida yozish mumkin. kirish/chiqish oqimlarni formatlash uchun bitta funksiya yetarli emas. C++da indentifikatorlar bilan formatlash usullarini ko‘ramiz.

Indentifikatorlar orqali formatlash kirish/chiqish parametrlaridan birini yoqish yoki o‘chirish imkonini beradi. IO nindentifikatorni o‘rnatish uchun setf() funksiyasini ishlatamiz va agar chiqish indentifikatorni o‘chirmoqchi bo‘lsangiz, unsetf() funksiyasidan foydalaniladi. Quyidagi indentifikatorni o‘rnatish va o‘chirishga doir instruksiya ko‘rsatilgan.

Chiqish indentifikatorni o‘rnatish

```
cout.setf( ios::name_flag );
```

name_flag - indentifikator nomi. Chiqarish operatorlariga nuqta orqali ruxsat beriladi. setf() funksiyasi bitta argument qabul qiladi va indentifikator nomidir. Indentifikatorlar ios sinfiga tegishli bo‘lganligi uchun oldin shu sinf nomi yoziladi va kengaytirish amali orqali kerakli indentifikator tanlanadi.

Chiqish indentifikatorni o‘chirish

```
cout.unsetf( ios::name_flag);
```

name_flag - indentifikator nomi.

Agar kirish/chiqishda bir nechta identifikatorlarni ishlatish kerak bo‘lsa, yoki [|] mantiqiy amalidan foydalanish mumkin.

Bir nechta identifikatorlarni o‘rnatish

```
cout.setf( ios::name_flag1 | ios:: name_flag2 | ios:: name_flag_n );
```

Bir nechta identifikatorlarni o‘chirish

```
cout.unsetf( ios::name_flag1 | ios:: name_flag2 | ios:: name_flag_n );
```

1.1-jadval asosiy formatlash identifikatorlarini keltirilgan va ulardan foydalanishga doir misollar ko‘rsatilgan.

1.1-jadval. C++ da formatlash identifikatorlari.

Identifi-kator	Vazifasi	misol	Misol natija-si

boolalpha	Mant ko‘rinishida mantiqiy kattalikni kiritish (true, false)	cout.setf(ios::boolalpha);
		bool log_false = 0,
		log_true = 1;
		cout << log_false << endl
		<< log_true << endl;
			false
			true

oct	Sakkizlik sanoq sistemasida kattaliklarni kiritish (avval dec identifikatorni o‘chiramiz va oct identifikator o‘rnatamiz)	cout.unsetf(ios::dec);
		cout.setf(ios::oct);
		int value;
		cin >> value;
		cout << value << endl;
			kirish:
			9910
			chiqish:
			1438

dec	O‘nlik sanoq sistemasida kattaliklarni kiritish/chiqarish (identifikator joriy o‘rnatilgan)	cout.setf(ios::dec);
		int value = 148;
		cout << value << endl;
			148

hex	O‘n oltilik sanoq sistemasida kattaliklarni kiritish (avval dec identifikatorni o‘chiramiz va hex identifikator o‘rnatamiz)	cout.unsetf(ios::dec);
		cout.setf(ios::hex);
		int value;
		cin >> value;
		cout << value << endl;
			kirish:
			9910
			chiqish:
			6316

showbase	Sanoq sistemaga	cout.unsetf(ios::dec);	kirish:

	asoslangan indikatorni chiqarish	cout.setf(ios::oct | ios::showbase); int value; cin >> value; cout << value << endl;	9910 chiqish: 01438

uppercase	16 lik SSda katta harflarni chiqarish (joriy holatda kichik harflar o‘rnatilgan)	cout.unsetf(ios::dec);
		cout.setf(ios::hex | ios::uppercase);
		int value;
		cin >> value;
		cout << value << endl;
			kirish: 25510 chiqish: FF16

showpos	Musbat sonlar uchun + ishorasini chiqarish	cout.setf(ios::showpos);
		int value = 15;
		cout << value << endl;
			+15

scientific	Eksponensial sonlarni chiqarish	cout.setf(ios::scientific);
		double value = 1024.165;
		cout << value << endl;
			1.024165e+003

fixed	Fiksirlangan holda haqiqiy sonlarni chiqarish (joriy holatda)	double value = 1024.165; cout << value << endl;	1024.165
right	O‘ng tomondan tekislash	cout.width(40); cout << «cppstudio.com» << endl;	__cppstudio.com

left	Chap tomondan tekislash	cout.setf(ios::left);
		cout.width(40);
		cout << «cppstudio.com» << endl;
			cppstudio.com__

Shuningdek, manipulyatorlar bilan formatlashning usullari ham bor. manipulyator – maxsus tip bo‘lib, oqimlarga o‘tgan ma’lumotlarni formatlash uchun kiritish/chiqish oqimlarni nazorat qiladi. Lekin, ko‘pchilik manipulatorlarining funksiyalari identifikatorlar bilan bir xil formatlash amallarini bajaradi. Ba‘zida, identifikatorli yoki formatlash funksiyalaridan foydalanish osonroq bo‘lsada, ba‘zan formatlashda manipulyatorlaridan foydalanish qulayroq. Shuning uchun C++da kirish

/ chiqishni formatlashning bir necha vositalari bilan to‘ldirilgan. 1.2-jadvalda formatlashning manipulyatorlari va misollari keltirilgan.

1.2-jadval. Formatlashning manipulyatorlari.

Manipulyator	vazifasi	misob	natijasi
endl	Chiqarishda yangi qatorga o‘tish	cout << «Salom:» << endl << «mbbGroup»;	website: cppstudio.com
boolalpha	Mantiqiy kattalikni matn ko‘rinishida chiqarish (true, false)	bool log_true = 1; cout << boolalpha << log_true << endl;	true
noboolalpha	Mantiqiy kattalikni son ko‘rinishida chiqarish (true, false)	bool log_true = true; cout << noboolalpha << log_true << endl;	1
oct	8 SSda sonlarni chiqarish	int value = 64; cout << oct << value << endl;	1008
dec	10 SSda sonlarni chiqarish	int value = 64; cout << dec << value << endl;	6410
hex	16 SSda sonlarni chiqarish	int value = 64; cout << hex << value << endl;	408
showbase	SS uchun asosiy indikatorni chiqarish	int value = 64; cout << showbase << hex << value << endl;	0x40
noshowbase	SS uchun asosiy indikatorni chiqarmaslik (joriy holat).	int value = 64; cout << noshowbase << hex << value << endl;	40

uppercase	16 SS da katta harflarni chiqarish (joriy holatda kichik harflar)	int value = 255; cout << uppercase << hex << value << endl;	FF16
nouppercase	16 SS da kichik harflarni chiqarish (joriy holat)	int value = 255; cout << nouppercase << hex << value << endl;	ff16
showpos	Sonlarni + belgisi bilan chiqarish	int value = 255; cout << showpos<< value << endl;	+255
noshowpos	Sonlarni + belgisiz chiqarish (joriy holat)	int value = 255; cout <<noshowpos<< value << endl;	255
scientific	Sonlarni eksponensial ko‘rinishda chiqarish	double value = 1024.165; cout << scientific << value << endl;	1.024165e+003
fixed	Fiksirlash holda sonlarni chiqarish (joriy holat).	double value = 1024.165; cout << fixed << value << endl;	1024.165
setw(int number)	Maydon uzunligini o‘rnatish, number — simvollarning pozitsiyalari soni, parameterli manipulyator	cout << setw(40) << “mbbGroup” << endl;	__ mbbGroup
right	O‘ngdan tekslash	cout << setw(40) << right << « mbbGroup » << endl;	__ mbbGroup

left	Chapdan tekislash	cout << setw(40) << left << « mbbGroup » << endl;	mbbGroup __
setprecision(int count)	Verguldan keyi kalit belgi berish, count — verguldan keyingi belgilar soni	cout << fixed << setprecision(3) << (13.5 / 2) << endl;	6.750
setfill(int symbol)	Qo‘shimcha belgilarni o‘rnatish	cout << setfill(‘0′) << setw(4) << 15 << ends << endl;	0015

C++ da buferlashtirilgan formatlangan kirish/chiqish dasturlashdagi eng oddiy mavzulardan biri hisoblanadi. Ma’lum formatlash vositalaridan qanday foydalanish yuqoridagi jadvallarda ko‘rsatilgan, shuning uchun bu mavzuda hech qanday qiyinchilik bo‘lmasligi kerak.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Polimorf sinf qanday xususiyatga ega bo‘lishi lozim?
2. Dastur bajarilishi davomida obyekt turini olish uchun qanday operatordan foydalaniladi?
3. type_info sinfda public modifikatori bilan qaysi statik operatorlar aniqlangan?
4. Type_info sinf a’zolaridagi qanday operatordan foydalanish taqiqlangan?
5. Typeid operatorini havolali parametriga qo‘llash mumkinmi?
6. C++ tilida necha tipni almatirish operatorlari mavjud va ularni nomlari, ishlatish uslublarini ayting.
7. Dynamic_cast operatori dastur bajarilishi davomida qaysi turdagi quyish amalini bajaradi?
8. Dynamic_cast va const_cast tip almashtirish operatorlarining vazifalari, kamchiliklari va yutuqlari eslang.
9. O‘zgarmas e‘lon qilingan identifikatorning tipini almashtirish mumkinmi. Mumkin bo‘lsa qanday amalga oshiriladi. Mumkin bo‘lmasa, nima sababdan izohlayu bering.
10. Polimorf bo‘lmagan va bo‘lgan sinf obyektining tipini almashtirish uchun qaysi operatorlardan foydalaniladi, har birini izog‘layu bering.
11. To‘rtta cast operatorlari dynamic_cast, const_cast, static_cast, reinterpret_cast bilan an‘anaviy tipni almashtirish amali (kasting) qanday farq mavjud.
12. Yangi nomlar fazosi yaratish uchun kerak, misollar bilan eslab tushuntirib bering.
13. Jamoaviy dasturlash vaqtida dasturdaga nomlarning ziddiyatlarni oldini olish uchun nima vazifani amalga oshrish kerak va ziddiyatlarni bekor qilishning eng oddiy va professonal usulini tushuntiring.
14. Nomlar fazosini 2 ga ajratish mumkinmi, mumkin bo‘lsa, qanday va nima uchun nomlar fazosini 2 ga ajratish kerak bo‘ladi.
15. Using kalit so‘zni ishlatmasdan turib, nomlar fazosiga murojaat qilish mumkinmi? Mumkin bo‘lsa, misol bilan asoslab bering.
16. C++ tilida ekran va faylga IO dan tashqari kiritish va chiqarish vazifalarini lokalizatsiyalash sifatida qanday vositadan foydalanadigan qator funksiyalarni qo‘llab-quvvatlanadi.
17. Kiritish/chiqarish obyektlari sifatida massivdan foydalanish uchun qanday header fayldan foydalanish lozim.

18. Massivga ma’lumotlarni yozish va chiqish uchun qanday tipdan foydalaniladi.
19. Massivlar asosida eof() va get() funksiyalarning IO da foydalanish nima uchun zarur.
20. Kirish/Chiqish indentifikatorni o‘rnatish va o‘chirish usullari aytib bering va misollar orqali asoslang.
21. Formatlash identifikatorlari nima uchun kerak va u qanday ishlatiladi.
22. Sanoq sistemaga asoslangan indikatorni chiqarish uchun qaysi formatlash identifikatoridan foydalanamiz.
23. Chap tomondan tekislab formatlashda qaysi formatlash identifikatoridan foydalanamiz.
24. Manipulyatorlar bilan formatlashning usullari nima uchun kerak va u qanday ishlatiladi.
25. Sanoq sistemaga asoslangan indikatorni chiqarish uchun qaysi manipulyatorlar bilan formatlashning usuldan foydalanamiz.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I	
	Tipni aniqlashga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`#include <iostream>`
`#include <typeinfo>`
`#include <cstdlib>`
 	
`using namespace std;`
 	
`class figure {`
`protected:`
`  double x, y;`
`public:`
	1. Figure sinfdagi xatolikni aniqlang. Dasturda tuzatish kiriting.
	_____
	_____
	_____
	_____
	_____
	_____
	_____
	_____
	_____

```
    figure(int i,
double j) {
        x = i;
        y = j;
    }
virtual double
area()=0;
};

class triangle : public
figure {
public:
    triangle(double i,
double j) : figure(i,
j) {}
    double area() {
        return x * 0.5
* y;
    }
};

class rectangle :
public figure {
public:
    rectangle(double i,
double j) : figure (i,
j) {}
    double area() {
return x * y;}
};

class circle : public
figure {
public:
    circle(double i,
double j=0) : figure(i,
```

2. Figure sinfi uchun kub nomli merosxщr sinf yarating va uning ham tasodifiy tanlash operatoridan foydalanib, yarating va dasturda yaratilgan obyektlarnining tiplarini kubga nisbatini topuvchi dastur tuzing.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_

3. Dasturda obyektlarni yaratish sonini dinamik almashtirish uchun o‘zgartirish kiriting va fragment aniq yozing.

_____
_____
_____
_____
_____
_____
_____
_____

```
j) {}
    double area()
{return 3.14 * x * x;}
};

// figure sinf
obyektlarini yaratish.

figure *factory(){
    switch(rand() % 3 )
{
    case 0: return new
circle (rand()%15);
    case 1: return new
triangle
(rand()%15,rand()%15);
    case 2: return new
rectangle (rand()%15,
rand()%15);
    }
return 0;
};

int main(){
    figure *p; // asos
sinfga ko‘rsatkich
    int i;
    int t=0, r=0, c=0;
// 15 ta obyektlarni
hosil qilsh va sanash
    for(i=0; i<15; i++)
{
        p = factory();
// obyektni hosil
qilish
        cout <<
"Obyektning tipi: " <<
typeid(*p).name();
```

4. typeid(*p) == typeid(triangle) fragmentga almatirish variantlarini aniqlang va dasturga yozib, tekshirib ko‘ring.

_____
_____
____

5. dasturda tipni almashtirish ishlatilganmi yoki tiplarni taqqoslashmi? Va bu amal dasturning qaysi fragmentlarida o‘z aksini topgan.

_____
_____
_____
_____
_____
_____

```
        cout << ". ";
        // obyektni
tekshirish orqali
sanasj
        if(typeid(*p)
== typeid(triangle))
t++;
        if(typeid(*p)
== typeid(rectangle))
r++;
        if(typeid(*p)
== typeid(circle)) c++;
    // Yuzasini
hisoblash
        cout << " S= "
<< p->area() << endl;
        }
cout << endl;
cout << "Quyidagi
obyektlar hosil
qilindi:\n";
cout << "
Uchburchaklar: " << t
<< endl;
cout << "
To‘rtburchaklar: " << r
<< endl;
cout << " Doiralar: "
<< c << endl;
system("pause");
return 0;
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

IKKINCHI ASSISMENT TOPSHIRIG‘I

Tipni almashtrishga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.
☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur

```
 #include "stdafx.h"
#include <iostream>
#include <typeinfo>
using namespace std;
class Base {
    public:
    virtual void f()
{ cout << " Base
sinfi orqali.\n"; }
// . . .
};
class myClass :
public Base {
    public:
        void f() {
cout << "myClass
sinfi orqali.\n"; }
};
void funk (const int
*p){
    int *v;
    v =
const_cast<int *>
(p);
    *v = 1980;
};
int main(){
    Base *bp, b_ob;
    myClass *dp,
d_ob;

    bp =
```

topshiriqlar

1. Dynamic_cast tip almatirishni ishlamasligi uchun dasturning qaysi fragmentini o‘chirish kerak.

_____
_____
_____

2. int berilgan o‘zgarmasni tipi almatrish dastur fragmentining qaysi qismida berilgan. Shu fragmentni float tipdan boshqa tipga almashtirish fragmentini tuzing.

_____
_____
_____
_____
_____
_____

3. static_cast<bool>(x); fragmentining qiymatini 0 chiqaradigan qilib dasturni o‘zgartiring.

_____
_____
_____
_____

4. Dastur natijasida ekranga 2 marta Xatolik ni chiqarish uchun dasturning kerakli fragmentlariga o‘zgartirishlarni yozing.

_____

```
dynamic_cast<Base *>
(&d_ob);

    if(bp) {
        cout << "
tipni almashtirish"
<<" (myClass* => Base
*) bajarildi. ";
            bp->f();
    } else cout
<<"Xatolik\n";
    cout << endl;
    dp =
dynamic_cast<myClass
*> (&b_ob);
    if(dp) cout
<<"Xatolik\n";
    else
    cout <<" tipni
almashtirish
"<<"(Base*=>
myClass*)
bajarilmadi.\n";
    cout << endl;
    int x = 2004;
    cout <<" x = " <<
x << endl << endl;
    funk(&x);
    cout <<" x = " <<
x << endl;
    cout << endl;
    double d;
    d =
static_cast<bool>(x);
    cout <<" x = " <<
dec << d << endl;
    cout << endl;
    char *p = "Salom
```

_____
_____

5. Dasturda reinterpret_cast<int> ning natijasi 65 chiqishi uchun qaysi fragmentni o‘zgartirish kerak.

_____
_____
_____

```
Buxoro";
    x  =
reinterpret_cast<int>
(p);
    cout <<" x = " <<
x << endl;
system("pause");
return 0;
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

UCHINCHI ASSISMENT TOPSHIRIG‘I

Nomlar fazosidan foydalanishga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`#include "stdafx.h"`
`#include <iostream>`
`using namespace std;`
 	
`namespace NS1{`
`class demo{`
`int i;`
`public:`
`demo(int x) { i=x; }`
`void set(int x) { i=x; }`
`int get() { return i; }`
`};`
	1. Dasturdagi std fazosining elementlarilar using namespace std; fragmentidan foydalanmasdan qanday o‘zgarishlar kiritilishini aniqlang.
	_____
	_____
	_____
	_____
	
	2. NS1 fazodagi str ko‘rsatkichining qiymatini o‘zgartirib chiqarish fragmentini yozing.
	_____
	_____
	_____
	_____

```
    char str[] =
"Nomlar fazolarini
o‘rnatish\n";
    int counter;
}

namespace NS2 { int
x, u; }

int main(){
    NS1::demo ob(10);
    cout <<" ob = "
<< ob.get() << endl;
    ob.set(99);
    cout << "Endi ob
= "<< ob.get() <<
endl;

    using NS1::str;
cout << str;

    using namespace
NS1;
    for(counter = 1;
counter<=10;
counter++)
        cout <<
counter << ends;
    cout << endl;

    NS2::x = 10;
NS2::u = 20;
    cout <<
"o‘zgaruvchilar x, y
= "<< NS2::x<<", " <<
NS2::u << endl;

    using namespace
```

3. Dasturda yaratilgan fazolardan yopiq holda foydalnish uchun barcha fragmentlarni o‘zgartiring.

_____

_____

_____

_____

_____

_____

_____

4. Dasturda 2 ga ajratilgan fazo yarating va tekshirib ko‘ring.

_____

_____

_____

_____

5. Dasturdagi takrorlanish jarayoni kamayish tartibda saralanib chiqishi uchun qanday o‘zgartirish kiritish kerak.

_____

_____

_____

_____

```
NS2;
       demo x_ob(x),
y_ob(u);
       cout << "x_ob,
y_ob obyektlarning
qiymati: "<<
x_ob.get(); cout<< "
" << y_ob.get() <<
endl;
system("pause");
return 0;
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

TO‘RTINCHI ASSISMENT TOPSHIRIG‘I

🎓 Buferlashtirilgan kirish va chiqarishdan foydalanishga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`#include "stdafx.h"`
 	
`#include <iostream>`
`using namespace std;`
`int main(){`
`char buf[] =`
`"SalomBuxoro 1980`
`28.01 B";`
`istrstream istr(buf);`
`char c;`
`    while(!istr.eof())`
`{`
`    istr.get(c);`
`    if (!istr.eof())`
`cout << c;`
	1. Dasturdagi buferlashtirilgan kirish/chiqish operatorlarning ishlamasligini aniqlang va xatosini tuzatib, kerakli joyga fragmentni yozing.
	_____
	_____
	_____
	2. Birinchi ekranga chiqarish operatorini nimani ekranga chiqaradi va birinchi buferlashtirilgan chiqarish massivga nimani yozadi.
	_____
	_____
	_____

```
    }
    cout<<endl;

char str[80]; float f;
char ch; int i;
istr >> str >> i >> f
>> ch;
cout << str << ' ' <<
i << ' ' << f << ' '
<< ch << '\n';

strstream iostr
(iobuf, sizeof iobuf)
;
iostr << 100 << hex <<
' ' << 100 << ends;
    char str[80]; int
j ;
    iostr.getline(str,
79);
    iostr >> dec >> j;
    cout << str << ' '
<< j << ' ';
    iostr >> hex >> j;
    cout << hex << j
<< endl;
cout.setf(ios::left);
cout.width(28);
cout << «mbbGroup» <<
endl;
cout.setf(ios::showpos
);
int value = 15;
cout << value << endl;
system("pause");
return 0;
}
```

_____
_____
_____
_____
_____

3. Dasturda iostr.getline(str, 79); fragmentning vazifasini aniqlang.

_____
_____
_____

4. Dasturda matni tekislash va sonlarni ishorasi bilan chiqarish fragmentini ajrating va tekshirib ko‘ring.

_____
_____
_____
_____
_____

5. Dasturdagi takrorlanish jarayoni qanday vazifani bajaradi.

_____
_____
_____
_____
_____
_____

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

1.2. Konteynerlar (Kollektsiyalar).

📖 STL (Standard Template Library) kutubxonalari bilan tanishib, uning to‘plamlari va har to‘plamning vazifasi, ishlatish maqsadi, maxsus usullari va algoritmlarini, iterator va dinamik struktura xususiyatlarini, iteratorlarning turlarini, iteratorlarning usullaridan foydalanishni, barcha to‘plamlarda mavjud bo‘lgan asosiy usullarni, STL kutubxona algoritmlarini, barcha to‘plam elementlari tanlash va ularga ishlov berish usullari, saralash usullari, to‘plam a’zolari ustida ma’lum arifmetik amallarni bajarish usullari, predikatlar, oqim xavfsizligi, konteyner sinflar, ketma-ket konteynerlar, assotsiativ konteynerlar, konteynerlarning xususiyatlari, konteyner sinflarning umumiy usullari, chiziqli konreynerlar (array, vector, deque, list, forward_list), konteynerlar bilan ishlash funksiyalari, allocator ajratuvchining vazifasi, bir aloqali ro‘yxat <forward_list>, ikki aloqali ro‘yxat (ikkilangan ro‘yxat) <list>, ikki tomonlama navbat <deque>, dinamik massiv (vektor) <vector>, statik massiv < array> kabi to‘plamlarning vazifalari va usullari keltirilgan bo‘lib, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni mustahkamlash uchun 20 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 3 ta assisment topshirig‘i va har assismentda 7 ta topshiriq, jami 21ta topshiriq berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

🖋 ***Kalit so‘zlar.*** STL kutubxonasi, to‘plam, shablon, vector, list, map, set, multimap, multiset, string, wstring, strstream, satrli oqim, iterator, dinamik ma’lumotlar tuzilmasi, predikat, konteyner sinflar, ketma-ket konteynerlar, chiziqli konreynerlar (array, vector, deque, list, forward_list), bir aloqali ro‘yxat <forward_list>, ikki aloqali ro‘yxat (ikkilangan ro‘yxat) <list>, ikki tomonlama navbat <deque>, dinamik massiv (vektor) <vector>, statik massiv < array>, allocator.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Tip tushunchasi, sinf va sinf obyekti, to‘plam, massiv, elementga murojaat, funksiya va ko‘rsatkich, ma’lumotlarni kirish va chiqishi, oqim, dasturlashga oid dastlabki tushunchalar va C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

☝ ***Bilib olasiz***. STL kutubxonasining imkoniyati, vector, list, map, set, multimap, multiset, string, wstring, strstream, satrli oqim kabi to‘lplamlarni ishlatish usullari va vazifalar, iterator va dinamik ma’lumotlar tuzilmasi ishlatish va turlari, predikat, konteyner sinflar, ketma-ket konteynerlar, chiziqli konteynerlar (array, vector, deque, list, forward_list) xususiyatlarni va funksiyalarini ishlash tamoyllari, bir aloqali ro‘yxat <forward_list>, ikki aloqali ro‘yxat (ikkilangan ro‘yxat) <list>, ikki tomonlama navbat <deque>, dinamik massiv (vektor) <vector>, statik massiv < array> kabi konteynerlarni dasturlashdagi o‘rni va allocator orqali yaratish va foydalanish usullarini o‘rganishingiz mumkin.

REJA

1.STL kutubxonalari.
2.Konteyner sinflar.
3.Chiziqli konteynerlar (array, vector, deque, list, forward_list).

KIRISH

Dasturlash texnologiyalarining rivojlanishi dasturlash tillarini ishlab chiqaruvchilar oldiga juda jiddiy masallarni paydo qilmoqda. Shulardan biri bu dasturlash tillari uchun turli xil to‘plamlar bilan ishlashdir. Masalan, xayotdan olib qaraydigan bo‘lsak, barcha narsalar qandaydir to‘plam, ammo qonuniyati har xil. Bu to‘plamlarni qanday dasturlash kerak degan muammo paydo bo‘ladi. Shuning uchun barcha obyektga yo‘naltirilgan dasturlash tillarida STL, ya‘ni standart shablonlar kutubxonasi tushunchasi kiritilgan. STL (Standard Template Library) kutubxonalari, to‘plam turlari, xususiyatlari, usullari va funksiyalar, konteyner va iteratorlar bo‘yicha nazariy materiallarni keltiramiz.

STL (Standard Template Library) kutubxonalari. Shablon mexanizmlari C++ kompilyatoriga moslab qurilgan bo‘lib, dasturchilarga umumiy dasturlash yordamida dastur fragmentlarini qisqartirishga imkon beradi. Tabiiyki, bunday mexanizmlarni amalga oshiruvchi standart kutubxonalar ham mavjud. Bugungi kunda C++ dasturlash tilida eng samarali STL kutubxonasi hisoblanadi.

STL kutubxonasining ko‘plab tadbiqlari mavjud bo‘lib, ularning har biri aniq standart doirasida yaratilgan bo‘lsa-da, o‘z kengaytmalariga ega. Ammo bunday yondashuvning bir kamchiligi bor: dastur fragmentini har doim turli kompilyatorlar bilan bir xil tarzda ishlamaydi.

Shuning uchun, dasturchi qanchalik mohirlik bilan kutubxonani yaratsa va foydalansa, o‘ziga xos bajarilishini tushunsa ham, imkon qadar an‘anaviy usullardan foydalanishni tavsiya qilamiz.

C++ dasturlash tilining kutubxonalaridan eng mashhur to‘plamlarni ko‘rib chiqaylik. Ularning har biri muhim vazifalarni hal qilinishi mumkin doirasi uchun o‘z shablon parametrlariga ega.

To‘plamlarni dastur fragmentida ishlatish uchun quyidagi fragmentdan foydalaniladi.

```
#include <T>
```

Bunda T – to‘plamning nomi.

Odatda quyidagi to‘plamlar ko‘p ishlatiladi.

vector – elementlar to‘plami, o‘lchamini o‘zgartish kerak bo‘lgan massivda saqlanadigan elementlar to‘plami (odatda ortib boradigan); Dasturga ulanish uslubi:

```
#include <vector>
```

list – elementlar to‘plami, elementlarni ikki tomonlama bog‘langan ro‘yxat sifatida saqlaydigan to‘plam; Dasturga ulanish uslubi:

```
#include <list>
```

map – elementlar to‘plami, har bir elementi shakli <const kalit, T> juftlikda saqlanadigan to‘plam, bu oddiy bir juftlik <kalit, qiymati> juftligi (har bir kalit bitta qiymati mos keladi). Kalit –taqqoslash amali uchun qiymatini tavsiflovchi ma’lum bir qiymati. Juftlikda kalit tez qidirishni amalga oshirish imkonini beradi, kalit asosida tartiblashtirilgan shaklida saqlanadi. Tartiblashni amalga oshirish uchun oldindan tashkil qilish qonuniyatini aniqlab olish kerak. Dasturga ulanish uslubi:

```
#include <map>
```

set – elementlar to‘plami, faqat kalitlarning qiymati bo‘yicha tartiblangan to‘plamidir, ya‘ni taqqoslash amali qo‘llaniladigan, ammo takrorlanmaydigan qiymatlar — har bir kalit to‘plamda (inglizcha set-to‘plam degan ma‘noni beradi) faqat bir marta foydalaniladi; Dasturga ulanish uslubi:

```
#include <set>
```

multimap – map, juftlikda kalitlar unikal emas, takrorlanadigan to‘plam. Agar kalit bo‘yicha qidirsangiz, siz bitta qiymatni emas, balki bir xil kalit qiymatiga ega bo‘lgan elementlar to‘plamini olasiz. Dasturga ulanish uslubi:

```
#include <map>
```

multiset – set, juftlikda kalitlar unikal emas, takrorlanadigan to‘plam. Agar kalit bo‘yicha qidirsangiz, siz bitta qiymatni emas, balki bir xil kalit qiymatiga ega bo‘lgan elementlar to‘plamini olasiz. Dasturga ulanish uslubi:

```
#include<set>
```

Belgilar to‘plami bo‘lgan satr (qator)ni ham to‘plam sifatida qarash mumkin. Shuning uchun ixtiyoriy kutubxona satrlar bilan ishlash va ifodalash uchun o‘zining sinflarga ega. STLda satrlar ASCII va Unicode formatlarida ifodalanadi.

string — to‘plam, ASCII formatidagi bir baytli belgilar to‘plami;

wstring — to‘plam, Unicode formatidagi ikki baytli belgilar to‘plami;

Dasturga ulanish uslubi:

```
#include <string>
#include <wstring>
```

Shuningdek, belgilar to‘plami bo‘lgan, satrli oqimlarni ham to‘plam sifatida qarash mumkin. **strstream**-oddiy ma’lumotlar tiplarini STL-string ko‘rinishada saqlashni tashkil qilish uchun ishlatiladi. Dasturga ulanish uslubi:

```
#include <strstream>
```

Dasturlarni tahlil qilishni shu sinfdan boshlaymiz.

2.1-dastur. Satrli oqim to‘plam sifatida ishlatish

```
#include "stdafx.h"
#include <iostream>
#include <strstream>
#include <string>
using namespace std;

int main(){
    strstream xstr;
    for (int i = 0; i < 10; i++)
    {
        xstr << "Demo " << i << endl;
    }
    cout << xstr.str() << endl;
    string str;
    str.assign(xstr.str (), xstr.pcount());
```

```
    cout << str.c_str() << endl;
    system("pause");
    return 0;
}
```

2.1-dastur natijasi. Output

```
Demo 0
Demo 1
Demo 2
Demo 3
Demo 4
Demo 5
Demo 6
Demo 7
Demo 8
Demo 9
═══════════════════════════¤¤¤¤llllllllyu■yu■
Demo 0
Demo 1
Demo 2
Demo 3
Demo 4
Demo 5
Demo 6
Demo 7
Demo 8
Demo 9
```

Satrli oqim - oqim oxirida bir null terminator bilan tugaydigan buferdir. Shuning uchun uning bo‘sh qolgan elemetlari ixtiyoriy belgi bilan to‘ldiriladi. Satrli oqimning haqiqiy qismini olish uchun pcount() elementlarni sanagichidan foydalanish mumkin. Keyin esa, satrli oqimning "haqiqiy qismi" olinadi va chop qilinadi.

Iteratorlar ham dinamik strukturaga ega to‘plamdir. Iterator dinamik ma’lumotlar tuzilmalarini amalga oshirishda juda muhim tushunchadir. Tushunishimiz uchun, iteratorni ma’lum bir cheklovlar bilan bir ko‘rsatkich sifatida abstrakt ko‘rinishda aniqlash mumkin. Sirasini aytganda, iterator umumiy tushuncha bo‘lib va bir ko‘rsatkich

uchun obyekt to‘plami bo‘ladi, ammo ko‘rsatkich bu iterator emas. Iterator sinfini quyidagicha qurib olish mumkin:

2.2-dastur. Iteratorning sinfni qurish.

```
class Iterator{
    T* pointer;
    public:
        T* GetPointer (){
            return this -> pointer;
        }
        void SetPointer (T* pointer){
            this -> pointer = pointer;
        }
};
```

Iteratorning ba‘zi bir formallashtirilgan ta‘riflari:

Iteratorlar to‘plam elementlariga kirishni ta‘minlaydi. Har bir aniq STL sinfi uchun iteratorlar to‘plamda sinf ichida alohida aniqlanadi.

Iteratorlarning uch turi mavjud:

- `(forward) iterator` - to‘plamni kichik indeksdan kattaroq indeksga o‘tkazish uchun;
- `reverse iterator` - to‘plamni katta indeksdan kichikroq indeksli o‘tkazish uchun;
- `random access iterator` - to‘plamdan tasofidiy, har qanday yo‘nalishda tanlash uchun.

To‘plamning yarim elementlarini o‘chirishga doir misol keltiramiz.

2.3-dastur. To‘plamning yarim elementlarini o‘chirish.

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

void printInt (int number);

int main (){

    vector<int> myVec;
    vector<int>::iterator first, last;
    for (long i=0; i<10; i++){
```

```
        myVec.push_back(i);
    }
    first = myVec.begin();
    last = myVec.begin() + 5;
    if (last >= myVec.end()){
        return - 1;
    }
    myVec.erase(first, last);
    for_each (myVec.begin(), myVec.end(),
printInt);
    system("pause");
    return 0;
}
void printInt (int number)
{
cout << number << endl;
}
```

2.3-dastur natijasi. Output

```
5
6
7
8
9
```

2.3-dasturdagi ba‘zi funksiyalarni keyinroq tushuntirib o‘tamiz. Shuni ham tushunish kerakki, to‘plamning qandaydir elementi uchun iterator olishda to‘plamni ketma-ket o‘zgartirish iteratorni yaroqsiz holga keltirish mumkin.

Iteratorning iteratsiyasida oldinga va shunga o‘xshash orqaga o‘tish quyidagicha fragmenti asosida sodir bo‘ladi:

```
for (iterator element = begin(); element < end();
element++)
{ t = (*element); }
for (iterator element = end(); element < begin();
element--)
{ t = (*element); }
```

Tasodifiy tanlash iteratoridan foydalanganda, masalan, quyidagicha fragmentni yozish mumkin:

```
for (iterator element = begin(); element < end();
```

```
element+=2)
 { t = (*element); }
```

Barcha to‘plamlarda mavjud bo‘lgan asosiy usullar quyidagi 2.1-jadvalga keltiramiz.

2.1-jadval. To‘plam usullari.

№	nomi	vazifasi
1	`empty`	To‘plamni bo‘shligini tekshiradi
2	`size`	To‘plamning o‘lchamini qaytaradi
3	`begin`	To‘g‘ri iteratorning birinchi elementini ko‘rsatadi
4	`end`	To‘g‘ri iteratorning oxirgi elementini ko‘rsatadi. Elementi yo‘q to‘plamga oxirgisidan keyinga o‘tadi
5	`rbegin`	Teskari iteratorning birinchi elementini ko‘rsatadi
6	`rend`	Teskari iteratorning oxirgi elementini ko‘rsatadi.
7	`clear`	To‘plamni tozalash, barcha elementlarini o‘chiradi.
8	`erase`	Ajratilgan elementlarni to‘plamdan o‘chiradi.
9	`capacity`	To‘plamning sig‘imini qaytaradi, ya‘ni bu to‘plam uchun mumkin bo‘lgan elementlar soni (aslida to‘plam uchun qancha xotira ajratilganini qaytaradi);

To‘plamning sig‘imi (hajmi), boshida aytib o‘tilganidek, kerak bo‘lganda o‘zgaradi, ya‘ni, agar to‘plam uchun ajratilgan barcha xotiralar to‘lgan bo‘lsa, unda yangi element qo‘shilganda, to‘plamning sig‘imi oshiriladi va o‘sishdan oldin undagi barcha qiymatlar yangi xotira maydoniga ko‘chiriladi - bu dasturchilar uchun juda qimmat amal bo‘lib hisoblanadi. Dasturlashda to’plamning hajmi va quvvati ishonch hosil qilish muhim dasturlash elementi hisoblanadi. To‘plamning sig‘imi turlicha ekanligiga ishonch hosil qilish uchun quyidagi dasturni keltiramiz.

2.4-dastur. To‘plamning sig‘imini tekshirish.

```
#include "stdafx.h"
#include <iostream>
#include <vector>

using namespace std;
int main(){
vector<int> vec;
cout << "Real size of array in vector: " <<
vec.capacity() << endl;
for (int j = 0; j < 25; j++){
```

```
    vec.push_back (10);
}
cout << "Real size of array in vector: " <<
vec.capacity() << endl;
system("pause");
return 0;
}
```

2.3-dastur natijasi. Output

```
Real size of array in vector: 0
Real size of array in vector: 28
```

vector – eng ko‘p ishlatiladigan to‘plam vektor hisoblanadi. Bu to‘plamning operator[] operatoriga ega ekanligi juda qulay. Odatiy massiv kabi ishlatiladi. Xuddi shuningdek, bu operator map, deque, string va wstring to‘plamlariga ham mavjud.

Vektorning quvvati dinamik ravishda o‘zgarishini tushunish muhimdir. Odatda, multiplikativ yondashuv hajmini oshirish uchun ishlatiladi: zarur bo‘lsa, vektor uchun ajratilgan xotiraning marta soni ortadi, ya‘ni, yangi element qo‘shib to‘plam sig‘imining oshirishga sabab bo‘lsa, operatsion tizimi dasturi uchun yangi xotira maydoni ajratadi. Masalan, ikki barobar katta bo‘lish uchun, eski xotira maydoni barcha elementlari nusxasi yangi qiymat qilib qo‘shish.

STL kutubxona algoritmlari. STL kutubxonasini ishlab chiquvchilar shablonli ma’lumotlar tuzilmalari majmuasiga ega kutubxona yaratishda o‘zlariga ancha jiddiy vazifa qo‘yishgan. STL kutubxonasi to‘plamlari bilan ishlash imkonini beradigan, mashhur algoritmlarni optimal tatbiqlari va katta majmuini o‘z ichiga oladi. Barcha amalga oshirilgan funksiyalarni uch guruhga bo‘lish mumkin:

1. Barcha to‘plam elementlari tanlash va ularga ishlov berish usullari:

```
sount, count_if, find, find_if, adjacent_find,
for_each, mismatch, equal, search,copy,
copy_backward, swap, iter_swap,
swap_ranges, fill, fill_n, generate, generate_n,
replace,
replace_if, transform, remove, remove_if,
remove_copy,
remove_copy_if, unique, unique_copy, reverse,
```

```
reverse_copy, rotate, rotate_copy, random_shuffle,
partition,
stable_partition
```

Bu usularni vazifalarini ixtiyoriy manbadan, splusplus.com veb sahifasidan foydalanib bilish mumkin.

2. Saralash usullari:

```
sort, stable_sort, partial_sort, partial_sort_copy,
nth_element, binary_search, lower_bound,
upper_bound,
equal_range, merge, inplace_merge, includes,
set_union,
set_intersection, set_difference,
set_symmetric_difference, make_heap, push_heap,
pop_heap, ‘sort_heap, min, max, min_element,
max_element,
lexographical_compare, next_permutation,
prev_permutation
```

Bu usularni vazifalarini ixtiyoriy manbadan, splusplus.com veb sahifasidan foydalanib bilish mumkin.

3. To‘plam a’zolari ustida ma’lum arifmetik amallarni bajarish usullari:

```
Accumulate, inner_product, partial_sum,
adjacent_difference
```

Bu usularni vazifalarini ixtiyoriy manbadan, cplusplus.com veb sahifasidan foydalanib bilish mumkin. Ushbu usularni sanab o‘tishdan maqsad STL kutubxonasi tomonidan taqdim etilgan boy vositalar to‘plami bilan tanishtirishdir. Qo‘shimcha ma’lumot olish uchun, C++ dasturlash tiliga oid sinflarning tegishli hujjatlarida tanishib chiqish mumkin.

Pridekatlar. Ko‘pgina STL kutubxona algoritmlari uchun algoritm to‘plamning muayyan a’zosi bilan nima qilish kerakligini aniqlaydigan shartni o‘rnatishingiz lozim bo‘ladi. Predikat - bu funksiya, bir necha parametrlarni oladi va Boolean qiymatini qaytaradi (true/false). Standart predikatlar to‘plami ham mavjud.

Oqim xavfsizligi. Bu STL butunlay xavfsiz kutubxona emasligini tushunish muhim ahamiyatga ega. Lekin bu muammoni hal qilish juda oddiy: ikki oqimlar bir xil to‘plamdan foydalanayotgan bo‘lsa, Mutex seksiyasini amalga oshirish zarur.

STL cross-platform kutubxona hisoblanadi. Albatta, ushbu kutubxona kompilyatorning har qanday versiyasi uchun mavjudligiga mutloq kafolat yo‘q. Masalan, u kamdan-kam hollarda mobil qurilmalarda amalga oshiriladi, chunki amalga oshirilgan ma’lumotlar tuzilmalarining aksariyati xotirani tejamasdan, tezlik foydasini tanlaydi hamda xotira mobil platformalarda eng qimmatli texnik resursdir, kompyuterda esa u hozir juda ko‘p. Shuning uchun tez-tez o‘z STL kutubxonangizni lokalizatsiyasini yaratish kerak bo‘ladi, masalan, ilovasini mobil platformaga ko‘chirish uchun.

Konteyner sinflar. Konteyner sinflar muayyan tarzda tashkil qilingan ma’lumotlarni saqlash uchun mo‘ljallangan sinflar. Turli xil tipdagi ma’lumotlarni saqlash uchun bir xil turdagi konteynerdan foydalanishingiz mumkin. Bu xususiyat sinf shablonlari yordamida amalga oshiriladi, shuning uchun C++ kutubxonasining konteyner sinflarini, shuningdek algoritmlarni va iteratorlarni o‘z ichiga olgan qismi standart shablonlar kutubxonasi (STL) deb ataladi.

Ma’lumotlar konteynerlarda saqlanadi va ular bilan turli amallar konteyner usullari va moslanuvchan algoritmlar bilan aniqlanadi va bajariladi. Iteratorlar bu ikki elementni bir-biriga bog‘lagan holda ishlaydi. Ular tufayli har qanday algoritm har qanday konteyner bilan ishlashi mumkin.

Professonal dasturlashni kutubxona sinflarisiz foydalanishni tasavvur qilish mumkin emas, shuningdek alohida konteynerlarsiz ham. Ulardan foydalanish dasturlarning ishonchliligi, joriy qilish samaradorligi, moslashuvchanligi va ko‘p qirraliligini oshirish hamda dastur tuzish vaqtini kamaytirishga imkonini beradi. Kutubxonani yaratish ko‘p ish va mashaqqat talab qiladi, amao, dastur yaratish vaqtida o‘zini oqlaydi.

STL kutubxonasi dasturlarni yozishda ishlatiladigan asosiy ma’lumotlar tuzilmalarini amalga oshiruvchi konteynerlarni o‘z ichiga oladi: vektorlar, navbatlar, ro‘yxatlar, lug‘atlar va to‘plamlar. Konteynerlarni ikki turga bo‘lish mumkin: ketma-ket va assotsiativ konteynerlar .

Ketma-ket konteynerlar. Ular uzluksiz ketma-ketlikda o‘xshash miqdorlarning chekli sonini saqlashni ta‘minlaydi. Konteynerlar sifatida vektor (vector), ikki tomonlama navbat (deque), ro‘yxat (list) va bir aloqali ro‘yxati (forward_list), shuningdek konteyner variantlar asosida adapterlar, stek (stack), navbat (queue) va ustuvorlik bilan navbat

(priority_queue) sinflarini o‘z ichiga oladi. Massiv ham amallar bilan cheklangan holda konteynerning yana bir turidir.

Konteynerning har bir turi ma’lumotlar ustida o‘z amallar to‘plamini ta‘minlaydi. Siz tanlagan konteyner turi dasturdagi ma’lumotlar bilan nima qilishni xohlashingizga bog‘liq. Masalan, agar ketma-ketlik o‘rtasida ma’lumotlar tez - tez joylashtirish va o‘chirish kerak bo‘lsa, ro‘yxatlardan foydalanish kerak, ma’lumotlarni oxirida yoki boshida, birinchi navbatda joylatrish kerak bo‘lsa, ikki tomonlama navbatdan foydalanish maqsadga muvofiq.

Assotsiativ konteynerlar. Assotsiativ konteynerlar asosiy ma’lumotlarga kalitlar asosida tezkor murojaat qilishni ta‘minlaydi. Bu konteynerlar muvozanatli daraxtlarga asoslangan. Assotsiativ konteynerlarning besh turi mavjud: lug‘atlar (map), ko‘p lug‘atlar (multi) (multimap), to‘plamlar (set), multi to‘plamlar (multiset) va bitli to‘plam (bitset).

Dasturchi standart kutubxonada mavjud bo‘lgan sinflarga asoslanib o‘z konteyner sinflarini yaratishi mumkin. Bunga kirishishdan oldin muhim tushunchani bilishinggiz shart, ya‘ni STL kutubxonasining fundamental tushunchasi bu shablondir

Barcha konteyner sinflari standartlashtirilgan interfeysi bilan ta‘minlangan. Turli konteynerlar uchun bir xil amallarning ma‘nosi bir xil bo‘lishi tabiiydir. Asosiy amallar konteynerlarning barcha turlari uchun qo‘llaniladi. Standart esa faqat konteyner interfeysini belgilaydi, shuning uchun turli xil dasturlar samaradorlikda katta farq qilishi mumkin. Barcha konteynerlar o‘z xotirasini o‘zi boshqaradi, shuning uchun dasturchi bu haqida o‘ylashning hojati yo‘q.

Deyarli har qanday konteynerlarning quyidagi xususiyatlari bor:

2.2-jadval. Konteyner xususiyatlari.

№	Xususiyat nomi	mazmuni
1	`value_type`	Konteyner elementining tipi
2	`size_type`	Elementlar indeksining tipi
3	`iterator`	Iterator
4	`const_iterator`	O‘zgarmas iterator
5	`reverse_iterator`	Teskari iterator
6	`const_reverse_iterator`	O‘zgarmas teskari iterator
7	`reference`	Elementga havola
8	`const_reference`	O‘zgarmas elementga havola
9	`key_type`	Kalit tipi (assotsiativ konteynerlar uchun)

10	`key_compare`	Taqqoslash mezoni (assotsiativ konteynerlar uchun)

Iterator bir element uchun ko‘rsatkichga ekvivalentidir. Ular konteynerlarni to‘g‘ri yoki teskari yo‘nalishda ko‘rish uchun ishlatiladi. Iteratordan talab qilinadigan barcha amallar konteyner elementiga murojaat qilish va uning keyingi elementiga o‘tish amalini amalga oshirishdir. Konteynerlarning elementlarining qiymatlari o‘zgarmaganda o‘zgarmas iteratorlardan foydalaniladi.

Iteratorlar yordamida haqiqiy ma’lumotlar tiplari haqida o‘ylamasdan, konteyner elementlarga murojaat qilish uchun foydalanish mumkin. Buning uchun, har bir konteyner quyidagi 2.3-jadvalda keltirilgan bir necha usullardan foydalanishni tavsiya qilinadi. Har bir konteyner uchun bu tiplar va usullarni, ularni realizatsiyasini amalga oshirishga bog‘liq tarzda belgilanadi. Shuningdek, ixtiyoriy konteynerlarda ularning hajmi haqida ma’lumot olish uchun usullari mavjud:

`size()` – elementlar soni;

`max_size()` – konreynerning maksimal o‘lchami (1 miliard ta element uchun);

`empty()` – mantiqiy usuli, konteyner bo‘shligini tekshiradi;

Zaruriyat bo‘lganda dasturchi konteynerning maydonlari va usullari ketma-ket o‘zlashtirib boraveradi. Biz ham zaruriyat tug‘ilganda foydalanamiz.

2.3-jadval. Konteyner sinflarning umumiy usullari.

№	Usullar	Izoh
1	`iterator begin() const_iterator begin () const`	Birinchi elementni ko‘rsatadi
2	`iterator end() const_iterator end () const`	Oxirgisidan keyingi elementni ko‘rsatadi
3	`reverse_iterator rbegin() const_reverse_iterator rbegin () const`	Teskari ketma-ketlikda birinchi elementni ko‘rsatadi
4	`reverse_iterator rend()`	Teskari ketma-ketlikda oxirgisidan keyingi elementni ko‘rsatadi

	const_reverse_iterator rend () const	

Chiziqli konreynerlar (array, vector, deque, list, forward_list). Standart konteynerlarni ikkita katta guruhga chiziqli va assotsiativ konteynerlarga bo‘lishini bilamiz. O‘z navbatida chiziqli konteynerlarni bog‘langan ro‘yxat (forward_list va list) va tasodifiy kirish konteynerlariga (ekrin konteyner deb ham yuritiladi) (deque, vector va array) bo‘linadi.

Assotsiativ konteynerlar quyidagi variantlarning kombinatsiyasi bo‘lgan sakkizta konteyner bilan ifodalanadi (tegishli standart sinflar nomlarining qavslar ichida beriladi): to‘plam (*set) yoki lug‘at (*map), elementlarni takrorlashga imkon beruvchi (*multi*) yoki ruxsat bermaydigan, tartibga solingan solinmagan (unordered*).

Barcha konteynerlar iterator va const_iterator tiplari o‘z ichiga olish uchun o‘z navbatida o‘qish-yozish va faqat o‘qish iteratorlarini aniqlaydi. Konteyner tarkibida olgan iteratorlar oralig‘ini begin() va end() (o‘qish-yozish uchun iterator), shuningdek cbegin() va cend() (o‘qish uchun const_iterator) funksiyalari yordamida olish mumkin. Konteynerlar ikki tomonlama iteratorlar bilan ham teskari o‘tish uchun rbegin(), rend(), crbegin() va crend() funksiyalari mavjud.

Barcha konteynerlarni empty funksiyasi tomonidan bo‘sh uchun tekshirish mumkin. Qachonki cont.begin() == cont.end() bo‘lsa, o‘zgarmas konteyner bo‘sh hisoblanadi. end() konteynerning oxirgi elementidan keyingi shartli elementga havola qiluvchi iteratorni qaytaradi. Elementlar sonini size() funksiyasi yordamida olish mumkin (forward_list bundan mustasno). Konteyner tarkibidagi funksiyalarni tozalash uchun clear() funksiyasidan foylalaniladi (array bundan mustasno).

Iteratorda begin(), end(), cbegin(), cend(), rbegin(), rend(), crbegin() va crend() erkin funksiyalarining shablonlarini aniqlash mumkin. Bular bir xil nomli funksiyalarga havola qilinadi, shuningdek, bunday aniqlangan funksiyalar statik massiv va std::valarray uchun aniqlandi. for(:) takrorlanish operatori xuddi shunday shakliga tayanadi. Masalan, cr bir konteyner bo‘lsin (shuningdek, statik massiv ham bo‘lishi mumkin), shunda dasturda for() fragmenti quyidagicha ishlatiladi.

```
for (T x : cr) work(x);
```

Semantik jihatdan quyidagi fragmentga teng

```
{
```

```
  using std::begin;
  using std::end;
  const auto e = end(cr);
  for (auto p = begin(cr); p != e; ++p)
  { T x = *p; work(x); }
}
```

Bu yerda T tipi konteyner elementlarining tipiga mos kelishi shart emas va auto asosida konstruksiya yoki havolali tip ham bo‘lishi mumkin (eng ko‘p ishlatiladigan variantlar auto& va auto&&).

Chiziqli konteynerlarni (array dan boshqa) belgilangan qiymatlardan assign() funksiyasini chaqirib to‘ldirish mumkin (eski qiymalari o‘chiriladi) va resize() funksiyasi bilan o‘lchamlarini (hajmi kamayganda elementlar oxiridan o‘chiriladi va hajmi ortganda, yangi elementlar oxiridan qo‘shiladi) aniqlash mumkin.

Konteynerning birinchi elementiga to‘g‘ridan-to‘g‘ri murojaat (tartiblanmagan assotsiativ konteynerlardan tashqari) front() funksiya yordamida amalga oshiriladi. Iteratorlari ikki tomonlama bo‘lgan barcha konteynerlarning oxirgi elementga murojaat uchun back() funksiyasi, shuningdek, teskari yo‘naltirilgan iteratorlar reverse_iterator va const_reverse_iterator funksiyalaridan foydalaniladi. Tegishli intervallarni rbegin(), rend() i crbegin(), crend() funksiyalari yordamida olish mumkin.

Barcha konteynerlarni tenglik va tengsizlik uchun taqqoslash mumkin va ularning mazmunini swap() funksiyasi yordamida almashtirish mumkin. Chegaralanmagan assotsiativlardan tashqari barcha konteynerlarni <, <= , > va >= operatorlari leksikografik jihatdan taqqoslash mumkin.

Standart konteynerlarda dinamik xotirani boshqarish uchun Allocator ajratuvchi maxsus sinflardan foydalanadi. Allocator xotirani boshqarishning minimal birligini belgilaydigan va bir qator yordamchi ta‘riflarni taqdim etadigan element tipiga bog‘liqdir. Bu vazifa to‘rt asosiy funksiyalari yordamida amalga oshiriladi: elementlari berilgan qator uchun xotira ajratishda allocate, xotirani tozalash uchun deallocate, konstruktor qurish uchun construct, va destruktor uchun destroy. Allokator boshqa tipdagi elementlar uchun analog allokator olishda rebind metafunksiyasini amalga oshirishi kerak.

Alloc – elementlarning aniq tiplari uchun allokator berilgan bo‘lsin. U tipli elementlar uchun uning allokatorini olish varianti quyidagi dastur fragmentida keltirilgan.

```
using AllocForU = typename Alloc::template
rebind<U>::other;
```

C++da standart kutubxona (<memory> sarlavha fayl bilan) new/new[] va delete/delete[] operatorlarini qo‘llash orqali allocator<T> ni ta‘minlaydi. Bundan model va o‘zingizning allokatorlaringgizni yozishda foydalanish mumkin.

Bir aloqali ro‘yxat <forward_list>. forward_list<T, a = allocator<T>> bir tomonlama iterator orqali T tipdagi elementlarga kirishni ta‘minlaydi. Ro‘yxatning tugunlarini yaratish uchun A::rebind orqali yaratilgan allokatorday foydalaniladi. Bir aloqali ro‘yxatning o‘ziga xosligi shundaki, faqat belgilangan joriy tugundan keyin elementlarni qo‘shish va o‘chirish mumkin:

insert_after - element qo‘shish.

emplace_after – yangi element yaratish. Bunda belgilangan parametrlar uchun konstruktor chaqiriladi.

erase_after - element o‘chirish.

Birinchi elementdan oldin degan ko‘shimcha aniqlangan pozitsiya mavjud. Bu before_begin va cbefore_begin (const_iterator qaytaruvchilar uchun variant) funksiyalari yordamida bajariladi. Shuningdek, fl.push_front(item) funksiyasidan foydalanib, elementlarni oldindan joylashtirish mumkin (yuqoridagiga ekvivalent), masalan, quyidagicha dastur fragmenti:

```
fl.insert_after(fl.before_begin(), item)
```

Konstruktorga har qanday parametrlarni joylashtirish uchun fl.emplace_front(...) funksiyasi ishlatiladi. Bunga ekvivalent sifatida quyidagi dastur fragmentini yozish mumkin:

```
fl.emplace_after(fl.before_begin(), ...)
```

pop_front funksiyasi ro‘yxatdan birinchi elementni o‘chiradi.

C++ standart kutubxonasida ro‘yxat konteynerlar xususiyati uchun samarali bo‘lgan amallarni bajarishga faqat iteratorlarda foydalanishga ishonchsizligi yuqori darajali amallarni qo‘llab-quvvatlash hisoblanadi:

merge - ikkita tartiblangan ro‘yxatni bir-biriga birlashtiradi, elementlar ko‘chirilmaydi lekin o‘ngdan chap ro‘yxatga o‘tiladi

splice_after - berilgan ro‘yxatni ko‘rsatilgan elementdan keyin joylashtiradi.

remove – berilgan elementga teng bo‘lgan barcha elementlarni o‘chiradi.

remove – berilgan pridikat asosida barcha elementlarni o‘chiradi.

reverse – elementlarni teskati tartibiga murojaat qiladi.

unique - barcha ketma-ket dublikatlarni o‘chiradi.

sort - ro‘yxatni joyida tartiblaydi.

Umuman olganda, bu funksiyalar qandaydir standart algoritmlarga o‘xshaydi, lekin juda tez va qulay ishlaydi. Ularga murojaat qilish uchun umumiy ruxsat olish kerak, masalan, sort() funksiyasi uchun std::sort(from, to). Ammo, tasodifiy kirish iteratorlari kabi talab qilingan ro‘yxatlar uchun amal qilmaydi.

Ikki aloqali ro‘yxat (ikkilangan ro‘yxat) <list>. list<T, a = allocator<T>> ikki tomonlama iterator orqali T tipdagi elementlarga kirishni ta‘minlaydi. Bir aloqali ro‘yxatdan farqli o‘laroq, elementlarni belgilangan pozitsiyadan oldin joylashtirildi (insert, emplace, splice funksiyalari), oxiridan qo‘shish va o‘chirish (push_back, emplace_back, pop_back), va iterator ko‘rsatayotgan elementni o‘chirish (erase) funksiyalari bilan amalga oshiriladi. *_after funksiyasi mavjud emas.

Ikki tomonlama navbat <deque>. deque<T, A = allocator<T>> , maxsus kirish iteratori orqali elementlar uchun kirish imkonini beradi. Faqat ro‘yxati kabi, u samarali qo‘shish va har ikki tomonidan ma’lumotlar olib tashlash imkonini beradi. Indeksga kirish uchun ikkita funksiya ishlatiladi: [] operatori va at(indeks). Birinchisidan farqli o‘laroq, ikkinchisi indeksni tekshiradi va qiymati haqiqiy bo‘lmasa, out_of_range istisnoga murojaat qiladi.

Ikki tomonlama navbat konteyneri uchun ro‘yxatni bir xil tarzda har qanday holatda elementlarni kiritish va o‘chirish imkonini beradi. Lekin ikki tomonlama navbatda bu amallar ular konteyner hajmi, vaqti chiziqli bo‘lishini talab qilishi mumkin. Bundan tashqari, oldindan iteratorga saqlangan elementlarni tartibini kiritish va o‘chirishni buzishi mumkin shuning uchun xotirada saqlangan elementlarning ko‘rsatkichlarini yodda saqlash maqsadga muvofiq (Agar iteratorlar ro‘yxat kabi saqlanayotgan bo‘lsa, elementlarga bo‘lgan ko‘rsatkichlar o‘chirilmaydi).

Dinamik massiv (vektor) <vector>. Vektor<T, a = allocator <T>> elementlarga maxsus kirish iteratori orqali kirishni ta‘minlaydi. Deque dan farqli o‘laroq, u kiritish va boshida elementlarni o‘chirish uchun ruxsat bermaydi. Dinamik massiv xotiraning uzluksiz bo‘limida ketma-ket saqlangan elementlarning joylashishini kafolatlaydi, uning manzili data funksiyasi bilan qaytariladi. Elementlarni qo‘shish va oldindan ajratilgan saqlashni tugatishda elementlar ko‘chiriladigan joyga katta hajmdagi yangi dinamik qator ajratish mumkin. Eski saqlab o‘chiriladi,

va barcha saqlangan iteratorlar yo‘qoladi, o‘chirilgan obyektlar uchun ko‘rsatkich bo‘lib qoladi.

Dinamik massiv reserve funksiyasi yordamida yetarli hajmda saqlash uchun oldindan massivni tayyorlash imkonini beradi (u ko‘chib o‘tishda elementlarni olib kelishi mumkin). Saqlash hajmini capacity funksiyasi yordamida bilib olishingiz mumkin. shrink_to_fit funksiyasi haqiqiy ishlatiladigan hajmini ajratadi va elementlarni foydalanilmagan xotiradan olib tashlaydi.

Statik massiv < array>. array<T, N> iteratorning tasodifiy kirishi orqali elementlarga kirish imkonini beradi va statik array T[N] to‘plami hisoblanadi, kerakli manzili data funksiyasi tomonidan olinishi mumkin. T[N] massivdan farqli ravishda array<T, N> - qiymatlari ko‘chirilishi (qiymat bo‘yicha funksiyalarga o‘tishi) mumkin bo‘lgan va qiymatlari avtomatik ravishda ko’rsatkichlarga aylantirilmaydigan to‘la huquqli tipdir. array < T, N> dan tayanch sinf sifatida foydalanish mumkin (lekin ehtiyotkorlik bilan, faqat har qanday konteyner kabi, standart konteynerlar bu maqsad uchun mo‘ljallangan emas, chunki, xususan, virtual funksiyalarni o‘z ichiga olmaydi). deque va vector kabi bir xil tarzda indeksi tomonidan elementlarga murojaat kilish mumkin. Elementlar sonini o‘zgartirish mumkin emas, shuning uchun array elementlarni kiritish yoki o‘chirish uchun hech qanday funksiyalar aniqlanmaydi. fill funksiyasi massivni qiymat nusxalari bilan to‘ldiradi.

2.4-jadval. Ketma-ket konteynerlar uchun asosiy funksiyalar

Headers		**<array>**	**<vector>**	**<deque>**	**<forward_list>**	**<list>**
Members		**array**	**vector**	**deque**	**forward_list**	**list**
	constructor	*implicit*	vector	deque	forward_list	list
	destructor	*implicit*	~vector	~deque	~forward_list	~list
	operator=	*implicit*	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin before_begin	begin
	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin		rbegin
	rend	rend	rend	rend		rend
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin cbefore_begin	cbegin
	cend	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin		crbegin
	crend	crend	crend	crend		crend

capacity	size	size	size	size		size
	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty
	resize		resize	resize	resize	resize
	shrink_to_fit		shrink_to_fit	shrink_to_fit		
	capacity		capacity			
	reserve		reserve			
element access	front	front	front	front	front	front
	back	back	back	back		back
	operator[]	operator[]	operator[]	operator[]		
	at	at	at	at		
modifiers	assign		assign	assign	assign	assign
	emplace		emplace	emplace	emplace_after	emplace
	insert		insert	insert	insert_after	insert
	erase		erase	erase	erase_after	erase
	emplace_back		emplace_back	emplace_back		emplace_back
	push_back		push_back	push_back		push_back
	pop_back		pop_back	pop_back		pop_back
	emplace_front			emplace_front	emplace_front	emplace_front
	push_front			push_front	push_front	push_front
	pop_front			pop_front	pop_front	pop_front
	clear		clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap
list operations	splice				splice_after	splice
	remove				remove	remove
	remove_if				remove_if	remove_if
	unique				unique	unique
	merge				merge	merge
	sort				sort	sort
	reverse				reverse	reverse
observers	get_allocator		get_allocator	get_allocator	get_allocator	get_allocator
	data	data	data			

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. C++ dasturlash tilida eng samarali qaysi kutubxonasi hisoblanadi va nima uchun?
2. Kalitlarning qiymati bo‘yicha tartiblangan qanday to‘plamlarni bilasiz?
3. Bir va ikki baytli belgilar to‘plami nima deb nomlangan va ularning formatlari bo‘yicha nimalarni bilasiz?
4. Iteratorlar qanday to‘plam va nima uchun ?
5. Har bir aniq STL sinfi uchun iteratorlar to‘plamda sinfda qanday aniqlanadi va turlari nechata? Har bir turini tushuntirib bering?
6. STL kutubxonasi to‘plamlari bilan ishlash imkonini beradigan, mashhur algoritmlarni optimal tatbiqlari va katta majmuini o‘z ichiga oladi. Bu algoritmlar necha guruhga bo‘linadi va qaysilar?
7. Konteyner sinflar qanday sinf hisoblanadi?
8. Konteynerlarni nechta turga bo‘lish mumkin va qaysilar?
9. Har qanday konteynerlarning bo‘lishi shart bo‘lgan xususiyatlarni sanab bering?
10. Ixtiyoriy konteynerlarda ularning hajmi haqida ma’lumot olish uchun qanday usullari mavjud?
11. Iteratorda begin(), end(), cbegin(), cend(), rbegin(), rend(), crbegin() va crend() erkin funksiyalari nimalarini aniqlashi mumkin?
12. Chiziqli konteynerlarni belgilangan qiymatlardan qaysi funksiyasini chaqirib to‘ldirish mumkin?
13. Barcha konteynerlarni tenglik va tengsizlik uchun taqqoslash mumkin va ularning mazmunini qanday funksiya yordamida almashtirish mumkin?
14. Allocator nima uchun ishlatiladi?
15. Allocator xotirani boshqarishning minimal birligini belgilaydigan va bir qator yordamchi ta‘riflarni taqdim etadigan element tipiga bog‘liqdir. Bu vazifa nechta va qanday asosiy funksiyalari yordamida amalga oshiriladi?
16. Bir aloqali ro‘yxatning nimalarini yaratish uchun A::rebind orqali yaratilgan allokatorday foydalaniladi?
17. Qaysi funksiyasidan foydalanib, elementlarni oldindan joylashtirish mumkin.
18. Ikki aloqali ro‘yxat va ikki tomonlama navbatning farqlarini sanab bering?

19. Dinamik va statik massivlarning o‘xshash tomonlarini misollar yordamida tushuntirib bering?
20. Staiik massiv elementlarni kiritish yoki o‘chirish uchun qanday funksiyalar aniqlangan.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I

STL kutubxonasiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`#include <iostream>`
`#include <vector>`
 	
`using namespace std;`
 	
`int main(){`
`  vector v;`
`  int i;`
 	
`  cout <<  v.size() << endl;`
 	
`  for(i=0; i<10; i++) v.push_back(i);`
 	
`  cout <<  v.size() << endl;`
 	
`  for(i=0; i<10; i++) cout << v[i] << " ";`
`  cout << endl;`
	1. Dasturda STL kutubxonasining necha sinfi va uning funksiyalari ishlatilgan.
	_____
	_____
	_____
	_____
	
	2. Dasturdagi STL sinfni o‘zgartirib dasturni qaytadan tuzing. Qysi qismlar o‘zgarganini qayd qiling.
	_____
	_____
	_____
	_____
	
	3. Dasturda vektor elementlariga xaqiqiy tipli qimatlarni berish uchun dastur fragment aniq yozing.
	_____
	_____
	_____
	_____
	_____

```
    cout <<  v.front()
<< endl;
    cout <<  v.back()
<< endl;

    vector<int>::iterat
or p = v.begin();
    while (p !=
v.end()) {
    cout << *p << " " ;
    p++;
    }

    system("pause");
    return 0;
}
```

4. cout << v.front() << endl; fragmentga almatirish variantlarini aniqlang va dasturga yozib, tekshirib ko‘ring.

_____
_____
_____

5. `vector<int>::iterator p = v.begin();` dasturning fragmentida nima o‘z aksini topgan.

_____
_____
_____
_____
_____

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi.

_____

7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

IKKINCHI ASSISMENT TOPSHIRIG‘I

Konteyner sinflarga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`#include "stdafx.h"`
 	
`#include <iostream>`
 	
`int main()`
	1. Dasturdagi eng katta xatoni toping.
	_____
	_____
	_____
	_____

```
{
    std::vector<int>
myVector;
    for (int count=0;
count < 5; ++count)

myVector.push_back(count);


std::vector<int>::const_it
erator itV;
    itV =
myVector.begin();
    while (itV !=
myVector.end())
        {
        std::cout << *itV
<< " ";
        ++itV;
        }

    std::cout << '\n';

    std::list<int> myList;
    for (int count=0;
count < 5; ++count)

myList.push_back(count);


std::list<int>::const_iter
ator itL;
    itL = myList.begin();
    while (itL !=
myList.end())
    {
        std::cout << *itL
<< " ";
```

2. Dasturda aniqlangan myVector, myList, mySet konteynerlarning elementlari bir xil bo‘lishi uchun kerakli dastur fragmentlarini yozing.

_____
_____
_____
_____
_____
_____
_____

3. itV qanday konteyner va nima uchun.

_____
_____
_____
_____

4. Dasturda dasturchining yana bir oddiy ammo ko‘rol xatosini toping.

_____
_____
_____
_____

5. Dasturdagi barcha konteynerlarni yarim elementlarini chiqarish dastur fragmentlarini tuzing.

_____
_____
_____
_____
_____

```
        ++itL;
    }

    std::cout << '\n';

    std::set<int> mySet;
    mySet.insert(28);
    mySet.insert(01);
    mySet.insert(80);
    mySet.insert(9);
    mySet.insert(3);


std::set<int>::const_itera
tor itS;
    itS = mySet.begin();
    while (itS !=
mySet.end())
    {
        std::cout << *itS
<< " ";
        ++itS;
    }

    std::cout << '\n';
    system("pause");
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

UCHINCHI ASSISMENT TOPSHIRIG‘I	
🎓	Ketma-ket konteynerlardan foydalanishga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar
#include "stdafx.h" #include <iostream> #include <vector> #include <cstring> #include <string> using namespace std; struct student { char fio[80]; short age; student(const char* fio, short age) : age(age) { strncpy_s(this->fio, fio, sizeof(this->fio)); } }; inline ostream& operator <<(ostream& out, const student& obj) { return out << obj.fio << "\t: " << obj.age; } struct trimatrix : vector< vector<double> > { trimatrix(int n) { while (n > 0) { vector<double> row(n); for (int i = 0; i < n; i++) row[i] = n - i; push_back(row); n--; }	1. Dasturdagi yangi tiplarni aniqlang. _____ _____ _____ _____ _____ 2. Dasturdagi ketma-ket konteynerlar dastur fragmentini yozing. _____ 3. Dasturda keraksiz bo‘lgan dastur fragmentlarini o‘chirib tashlang _____ _____ _____ _____

```
 }
};
inline ostream& operator
<<(ostream& out, const
trimatrix& obj) {
 for (int i = 0; i <
obj.size(); i++) {
 for (int j = 0; j <
obj[i].size(); j++)
 cout << obj[i][j] << " ";
 cout << endl;
 }
return out;
}
int main(int argc, char
**argv) {
 setlocale(LC_ALL, "rus");

 vector<student> group;

group.push_back(student("Bo
ltavev B.B.", 18));

group.push_back(student("Am
inov A.A.", 21));

group.push_back(student("Ka
rimov A.Sh.", 19));

 for (int i = 0; i <
group.size(); i++)
 cout << group[i] << endl;

 cout << trimatrix(4);

  int k, n;
  vector<bool> a(n = rand()
+ 1, true);
```

4. vector konteynerini boshqa ketma-ket konteynerga almatirish uchun kerakli dastur fragmentlari yozing.

5. Dasturdagi ketma-ket konteynerlar uchun asosiy funksiyalarni yozing.

```
 a[0] = a[1] = false;
 for (k = 2; k < n; k++)
 for
(vector<bool>::iterator j =
a.begin() + k + k;
       j < a.end(); j += k)
    *j = false;
 const int line = 10;
 k = 0;
 for
(vector<bool>::iterator j =
a.begin(); j < a.end();
j++)
 if (*j) {
 cout << j - a.begin() <<
"\t";
 if (0 == ++k % line) cout
<< endl;
 }
 if (k % line != 0) cout <<
endl;

 system("pause");
 return 0;
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi.
_____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

1.3. Assosiativ va tartiblanmagan assosiativ konteynerlar

📖 Assosiativ konteynerlarning asosiy foydalanish ko‘nikmalari, assosiativ konteynerlar (set, map, multiset, multimap) va tartiblanmagan assosiativ konteynerlar (unordered_set, unordered_map, unordered_multiset, unordered_multimap) bilan yaqindan tanishib, ularning xususiyatlar, funksiyalari, sinflari, yaratish va foydalanish uslublari, amallari, talablari, vazifalari va usullari keltirilgan bo‘lib, dasturchining to‘plamlar bilan ishlashdagi tanlash imkoniyati bo‘yicha tajriba xulosalari, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. Shuningdek, assosiativ konteynerlarda qidirish ustunligi va

uni dasturda taqqoslash bo‘yicha miosllar keltirilgan. Bilimlarni mustahkamlash uchun 32 ta nazariy savol va amaliy ko‘nikma hamda malakalarni rivojlantrish uchun 4 ta assisment topshirig‘i va har bir assismentda 10(12) ta topshiriq, jami 44ta topshiriq berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

Kalit so‘zlar. to‘plam, shablon, map, set, multimap, multiset, unordered_set, unordered_map, unordered_multiset, unordered_multimap, satrli oqim, iterator, dinamik ma’lumotlar tuzilmasi, predikat, konteyner sinflar, allocator, equal_range, *operator==*, operator=, *tartiblangan assosiativ konteyner, tartiblanmagan assosiativ konteyner, unikal kalit, xesh funksiya*, xeshlash**,** konstruktor, modifikator, qidirish usullari, pair, try - catch funksiyasi.

Bilish shart bo‘lgan tushunchalar. Tip tushunchasi, sinf va sinf obyekti, to‘plam, statik va dinamik massiv, elementga murojaat va ko‘rsatkich, funksiya va ko‘rsatkich, konteyner, ketma-ket konteynerlar, xesh, tasodifiy qiymat, istisno holat, ma’lumotlarni kiritish va chiqarish, oqim, dasturlashga oid dastlabki tushunchalar va C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

Bilib olasiz. Assosiativ konteynerlarning asosiy foydalanish ko‘nikmalari, assosiativ konteynerlar (set, map, multiset, multimap), tartiblanmagan assosiativ konteynerlar (unordered_set, unordered_map, unordered_multiset, unordered_multimap) bilan yaqindan tanishib, ularning xususiyatlar, funksiyalari, sinflari, yaratish va foydalanish uslublari, amallari, talablari, vazifalari va usullarini, dasturchining assosiativ to‘plamlar bilan ishlashdagi tanlash imkoniyati bo‘yicha tajriba xulosalari, xeshlash, tartiblangan va tartiblanmagan konteynerlarning sinflari xususiyatlari, usullarini, dasturlashdagi o‘rni va allocator orqali yaratish va foydalanish usullarini o‘rganishingiz mumkin.

REJA

1. Assosiativ konteynerlar (*set, map, multiset, multimap*).
2. Tartiblanmagan assosiativ konteynerlar (*unordered_set, unordered_map, unordered_multiset, unordered_multimap*).

KIRISH

Konteynerlar va ketma-ket konteynerlar bo‘yicha nazariy va amaliy imkoniyatlarini bilamiz. Shuning uchun konteynerlarning ikkinchi turi bo‘lgan assosiativ konteynerlarni boshqa konteynerlar farqli bo‘lishi kerak. Assosiativ konteynerlari kalit asoisda

ma’lumotlarni to‘plamdan tez izlab topish uchun ishlatiladi. Tartiblangan konteynerlar muvozanatlashgan binar daraxt ustiga quriladi va qat‘iy tartibga tayanadi ("kichik" amali [<] operatori asosida hisoblangan). Ikkita element bir biridan kichik bo‘lmasa, ekvivalent hisoblanadi.

Buning kutubxonasi 4 ta asosiy *set* (to‘plam), *multiset* (multi to‘plam, ko‘p to‘plam), *map* (lug‘at) va *multimap* (mulilug‘at, ko‘p lug‘at) assosiativ konteynerlar bilan ishlashga qaratilgan. Bu konteynerlar Key kalitini o‘zlariga asosiy parametr sifatida oladi va Compare munosabati bo‘yicha tartiblaydi. Tartiblash *Key* parametr bilan amalga oshiriladi. Shuningdek, *map* va *multimap* erkin *T* tipida Key bilan assosiativlanadigandir (sherikdir). *Compare* obyektining tipi taqqoslanaligan konteyner obyekti (*comparison object)* deb aytiladi.

Shuning uchun bu konteynerlarning umumiy xususityati va amallaridan boshlaymiz. Barcha assotsiativ konteynerlar quyidagi amallarni qo‘llab-quvvatlaydi:

count – elementlar sonini qaytaradi, belgilangan kalit bo‘yicha elementlar sonini qaytaradi;

find –elementga ko‘rsatkichga mos bo‘lgan iteratorni qaytaradi, agar bunday bo‘lmasa end() funksiyasini vazifasini bajaradi.

equal_range - berilgan intervaldigi barcha elementlar uchun iteratorlar juftligini qaytaradi.

Katitlarning tengli xaqida *munosabatlarning ekvivalentligi* Kalitlar uchun shartli taqqoslash va (*not*) *operator== tahlil qilamiz.*

Key1 va key2 kalitlar o‘zaro teng hisoblanadi, agar taqqoslanadigan obyektlar uchun comp chin qiymat qaytarsa, ya‘ni:

comp(k1, k2) == *false* && comp(k2, k1) == *false*

Assosiativ konteynerlar har qanday kalit qiymat uchun bitta eng katta elementning qiymatini saqlasa, unikal (takrorlanmaydigan) kalitlarni (unique keys) qo‘llab quvvatlaydi. Aks hoda ular teng qiymatli kalitlarni (*equal keys*) ham qo‘llab quvvatlaydi. *set* va *map* konteynerlar unikal kalitlarni, *multiset* va *multimap* konteynerlar teng qiymatli kalitlar bilan ishlashga mo‘ljallangan.

set va *multiset* konteynerlar uchun aniqlanadigan tip va kalit tipi bir xil bo‘ladi.

map va *multimap* konteynerlar uchun *pair<const Key, T>* jufligi bo‘ladi.

Assosiativ konteynerlarning iteratorlari ikki tomonlama iteratorlar sinfiga kiradi. *insert* konteyner havolalari va iteratorlarning inkor qilmaydi. Elementlarni o‘chirishda *erase* faqat iterator va havolalarni inkor qiladi.

3.1-jadvalda assosiativ konteynerlarning talablari keltirilgan (konteynerlarga qo‘shimacha). Unda x bu assosiativ sinf, a – X ning qiymati, Agar X unikal kalitni qo’llab quvvatlasa, a_uniq – X ning qiymati, Agar X bir nechta kalitni qo’llab quvvatlasa, a_eq – X ning qiymati, i va j – iteratorning kirish talablarini qanoatlantiruvchi va value_type elementni ko‘rsatadi (beradi), [i, j) - interval, p – a uchun ruxsat berilgan iterator, q – a uchun o‘zgartiriladigan iterator, [q1, q2) - a da ruxsat berilgan interval, t - X::value_type ning qiymati, k - X::key_type ning qiymati.

3.1-jadvalda assosiativ konteynerlarning talablari.

Ifoda	Qaytarila-digan tipi	tasdiq/ oldingi yoki keyingi holatiga izox	murakkabligi
X::key_type	Key	.	Kompilyatsiya vaqti
X::key_compare	Compare	less<key_type> joriy holat	Kompilyatsiya vaqti
X::value_compare	binar predikat turi	xuddi, key_compare uchun set va multiset; munosabati tartiblangan juftlik, map va multimap uchun chaqirilgan birinchi elementdek (Key)	Kompilyatsiya vaqti
X(c) X a(c);	.	Bo‘sh konteyner yaratish; taqqoslash obyekti sifatida foydalanish uchun.	Doimiy
X() X a;	.	Bo‘sh konteyner yaratish; taqqoslash obyekti	Doimiy

		sifatida Compare() dan foydalanish uchun	
X(i,j,c) X a(i,j,c);	.	Bo‘sh konteyner yaratish [i, j) intervaldan qiymat qo‘shish; taqqoslash obyekti sifatida foydalanish uchun..	NlogN (N - i dan j gacha); chiziqli , agar [i, j) value_comp( ) bilan saralangan bo‘lsa
X(i,j) X a(i,j);	.	Bo‘sh konteyner yaratish [i, j) intervaldan qiymat qo‘shish; taqqoslash obyekti sifatida Compare() dan foydalanish uchun	NlogN (N - i dan j gacha); chiziqli , agar [i, j) value_comp( ) bilan saralangan bo‘lsa
a.key_comp()	X::key_compare	Taqqoslash obyektini qaytaradi	doimiy
a.value_comp()	X::value_compare	Taqqoslash obyekti uchun yaratilgan value_compare obyektni qaytaradi	doimiy
a_uniq.insert(t)	pair<iterator, bool>	Agar konteynerda t kalitli element bo‘lmasa, t ni qo‘shish, juftlikda komponentning bool qiymati qo‘shish bajarilganligini, iterator t kalitli elementni ko‘rsatadi	logarifmik
a_eq.insert(t)	iterator	t ni qo‘shadi va	logarifmik

		iteratorni qaytaradi, qo‘shishilgan elementni ko‘rsatadi.	
a.insert(p, t)	iterator	t ni qo‘shish, agar faqat va faqat unikal kalitli konteynerda t kalitga teng kalitli element bo‘lmasa; har doim nusxalari bilan konteynerlarga t ni qo‘shish. har doim t kalitli elementni ko‘rsatuvchi iteratorni qaytaradi. p iterator – qo‘shish qaerdan boshlanishini ko‘rsatuvchi izoh	umuman logarifmik, lekin domiyga intiladi, agar t to‘g‘ri p ni yoniga qo‘yilgan bo‘lsa.
a.insert(i, j)	Natija ishlatilmaydi	[i, j) intervaldan konteynerga elementlarni qo‘shadi;	Umuman, Nlog(size()+ N) (N – i dan j gacha); chiziqli, agar [i, j) value_comp( ) bilan kelishgan holda saralangan bo‘lsa
a.erase(k)	size_type	konteynerda k kalitga teng bo‘lgan barcha e lementlarni o‘chiradi. o‘chirilgan elementlarning sonini qaytaradi.	log(size()) + count(k)
a.erase(q)	Natijasi	q bilan ko‘rsatilgan	domiyga

	ishlatilmay-di	elementni o‘chiradi	intiladi
a.erase(ql, q2)	Natijasi ishlatilmay-di	[ql, q2) intervaldagi barcha elementlarni cho‘chiradi.	log(size())+ N, N - ql dan q2 gacha.
a.find(k)	iterator; const_iterator – a uchun o‘zgarmas	k kalitli elementni ko‘rsatuvchi iterator yoki agar bunday element topilmasa a.end() qaytaradi	logarifmik
a.count(k)	size_type	k kalitli elementlar sonini qaytaradi.	log(size()) + count(k)
a.lower_bound(k)	iterator; const_iterator a uchun o‘zgarmas	k kalitli elementdan kichik bo‘lmagan birinchi elementni ko‘rsatuvchi iteratorni qaytaradi	logarifmik
a.upper_bound(k)	iterator; const_iterator a uchun o‘zgarmas	k kalitli elementdan katta bo‘lgan birinchi elementni ko‘rsatuvchi iteratorni qaytaradi	logarifmik
a.equal_range(k)	pair<iterator, iterator>; pair<const_iterator, const_iterator> a uchun o‘zgarmas	make_pair(lower_boud (k), upper_bound (k)) ga ekvivalent	logarifmik

Assotsiativ konteyner iteratorlarining asosiy xususiyati shundaki, ular konteynerlar orqali o‘sish tartibidagi kalitlar tartibida iteratsiyalanadi, bu yerda o‘sish tartibi ularni yaratish uchun ishlatilgan taqqoslash bilan belgilanadi.

Ikkiia ixtiyoriy o‘zguruvchan I va j iteratorlar uchun I dan j gacha masofa musbat bo‘ladi, ya‘ni *value_comp (*j, *i) == false*. *Unikal kalitli* assotsiativ konteynerlar uchun kuchliroq *value_comp (*i, *j) == true* holat saqlanib turibdi.

Barcha tartiblangan konteynerlar ikki tomonlama iteratorlar yordamida elementlarga murojaat qilish imkonini beradi. Kalit asosida qidirishda berilgan kichik bo‘lmagan birinchi elementni lower_bound() funksiyasi va birinchi katta elementni upper_bound() funksiyasi yordamida amalga oshiriladi (3.1-jadvalga qarang). Agar mymap konteyner berilgan bo‘lsa, unda mymap.equal_range(key) funksiyasi bilan ekvivalnt bo‘ladi (3.1-dasturga qarang).

3.1-dastur. Tartiblangan konteynerlar funksiyalari.

```
// created by Mbbahodir
#include "stdafx.h"

#include <iostream>
#include <map>

using namespace std;
int main ()
{
  map<char,int> mymap;
  map<char,int>::iterator itlow,itup;

pair<map<char,int>::iterator,map<char,int>::iterator>
ret, make_ret;

  mymap ['a']=101;   mymap ['c']=303;   mymap
['f']=404;   mymap ['b']=202;

  cout << mymap.count('c') << endl;

  cout << mymap.find('c')->second << endl;

  ret = mymap.equal_range('b');

  char key = 'b';
  make_ret =
make_pair(mymap.lower_bound(key),mymap.upper_bound(ke
y));

  cout << "Kichik bo‘lmagam element: ";
  cout << ret.first->first << " => " << ret.first-
>second << '\t';
```

```
  cout << make_ret.first->first << " => " <<
make_ret.first->second << '\t';
  itlow = mymap.lower_bound ('b');
  cout << itlow->first << " => " << itlow->second <<
endl;

  cout << "katta bo‘lgan birinchi element: ";
  cout << ret.second->first << " => " << ret.second-
>second << '\t';
  cout << make_ret.second->first << " => " <<
make_ret.second->second << '\t';
  itup = mymap.upper_bound ('b');
  cout << itup->first << " => " << itup->second <<
endl;

  system("pause");
  return 0;
}
```

3.1-dastur. Output

```
1
303
Kichik bo‘lmagam element: b => 202      b => 202       b => 202
katta bo‘lgan birinchi element: c => 303       c => 303       c => 303
```

3.1-dastur tahlili. Dasturda 4 ta elementdan iborat map konteyner berilgan. ‘s‘ kalitli element soni count() funksiya bilan aniqlagan. Xuddi shu kalit bilan uning qiymati find() funksiya bilan topilgan. equal_range('b'), mymap.lower_bound('b'); mymap.upper_bound('b'); make_pair(mymap.lower_bound(key), mymap.upper_bound(key)); funksiyalarining ekvivalent ishlari ko‘rsatilgan.

Takrorlanmaydigan (unikal) tartiblangan elementlar to‘plami <set>. set<K, C = less<K>, A = allocator<K>> - Bu konteyner joyida qiymatlarni o‘zgartirishga yo‘l qo‘ymaydi. **<set>** - bu assosiativ konteyner bo‘lib, unikal kalitni qo‘llab quvvtlaydi (bir xil qiymatli kalitlarni qo‘llab quvvatlamaydi) va kalit yordamida tez qidirish imkonini beradi. set<K>::iterator va set<K>::const_iterator iteratorlar funksional ekvivalent hisoblanadi va o‘zgarganda const K& qiymat

qaytaradi. Set to‘plamda qiymatni o‘zgartirish uchun avval o‘chirib tashlab so‘ng yangi qiymatni yoki o‘zgartirilgan varianti qo‘shiladi.
<set> ning shabloni:

```
template <class Key, class Compare = less<Key>,
    template <class U> class Allocator = allocator>
```

<set> sinfning ochiq xususiyatlari, operatorlari va funksiyalari:

1. typedef operatorlari:

```
    typedef Key key_type;
    typedef Key value_type;
   typedef Allocator<Key>::pointer pointer;
    typedef Allocator<Key>::reference reference;
    typedef Allocator<Key>::const_reference
const_reference;
    typedef Compare key_compare;
    typedef Compare value_compare;
    typedef iterator;
    typedef iterator const_iterator;
    typedef size_type;
    typedef difference_type;
    typedef reverse_iterator;
    typedef const_reverse_iterator;
```

2. Xotirani ajratish va bo‘shatish operatorlari (allocation/deallocation):

```
set(const Compare& comp = Compare());
    template <class InputIterator>
    set(InputIterator first, InputIterator last,
        const Compare& comp = Compare());
    set(const set<Key, Compare, Allocator>& x);
    ~set();
    set<Key, Compare, Allocator>& operator=(const
set<Key, Compare,
        Allocator>& x);
    void swap(set<Key, Compare, Allocator>& x);
```

3. Ruxsat berish vositalarining operatorlari (accessors):

```
key_compare key_comp() const;
    value_compare value_comp() const;
    iterator begin() const;
    iterator end() const;
```

```
    reverse_iterator rbegin() const;
    reverse_iterator rend() const;
    bool empty() const;
    size_type size() const;
    size_type max_size() const;
```

4. Qo‘shish va o‘chirish operatorlari (insert/erase):

```
    pair<iterator, bool> insert(const value_type&
x);
    iterator insert(iterator position, const
value_type& x);
    template <class InputIterator>
    void insert(InputIterator first, InputIterator
last);
    void erase(iterator position);
    size_type erase(const key_type& x);
    void erase(iterator first, iterator last);
```

5. To‘plam amallari (set operations):

```
    iterator find(const key_type& x) const;
    size_type count(const key_type& x) const;
    iterator lower_bound(const key_type& x) const;
    iterator upper_bound(const key_type& x) const;
    pair<iterator, iterator> equal_range(const
key_type& x) const;
```

6. Taqqoslash operatorlari:

```
template <class Key, class Compare, class Allocator>
bool operator==(const set<Key, Compare, Allocator>&
x,
     const set<Key, Compare, Allocator>& y);

template <class Key, class Compare, class Allocator>
bool operator<(const set<Key, Compare, Allocator>& x,
    const set<Key, Compare, Allocator>& y);
```

<set> to‘plamda:

iterator (iterator) bu – const value_type ko‘rsatuvchi domiy ikki tomnlama iterator. Aniq tipni belgilash realizatsiya qilishga bog‘liq va Allocatorda aniqlanadi.

const_iterator - ham iterator (iterator) kabidir.

size_type - ishorasiz butun tip, Aniq tipni belgilash realizatsiya qilishga bog‘liq va Allocatorda aniqlanadi.

difference_type - ishorali butun tip, Aniq tipni belgilash realizatsiya qilishga bog‘liq va Allocatorda aniqlanadi.

Elementlarni qo‘shish insert() funksiyasi tomonidan amalga oshiriladi va bu funksiyaning bir nechta variantlari mavjud:

-juft iteratorlarni kitirishda intervaldan qiymat qo‘shish;

```
template <class InputIterator>
  void insert (InputIterator first,
InputIterator last);
```

-qo‘yish mumkin bo‘lgan joyga ko‘rsatish orqali bitta qiymat qo‘shish (agar yangi element bevosita oldin pozitsiyani egallasa, o‘rnatish o‘zgarmas vaqt mobaynida amalga oshiriladi);

```
pair<iterator,bool> insert (const value_type& val);
```

-berilgan qiymatni qo‘shish (asosiy usul);

```
iterator insert (iterator position, const
value_type& val);
```

Oxirgi variant insert funksiyasi bir juftli (iterator, Boolean) qaytaradi. Juftlikdagi birinchi element qo‘shish joyni yoki topilgan qiymatini, ikkinchi joylashtirilgan qiymatini qo‘shish bajarilganligi (true) yoki qo‘shish vaqtida to‘plamda borligini (flase) bildiradi.

Elementlarni o‘chirish erase funksiyasi orqali amalga oshiriladi. Bu funksiya qiymat yoki berilgan elementlar to‘plami yoki iterator yoki iteratorlar intervallari qabul qiladi.

```
(1)  size_type erase (const value_type& val);
(2)  void erase (iterator position);
(3)  void erase (iterator first, iterator last);
```

Birinchi parametr elementni o‘chiradi, qolgan ikka parametrlar elementni ko‘rsatadigan iterator qaytaradi. Birinchi variant o‘chirilgan elementlarning sonini beradi (set bo‘lsa, 1 yoki 0 qiymat beradi).

3.2-dastur. Set to‘plamga elementlarni qo‘shish.

```
// created by MBBahodir
#include "stdafx.h"
#include <iostream>
#include <set>
using namespace std;

void erase_subs(set<int> &s, const int &subs)
```

```
{
   set<int>::iterator p, pe, pd;
   p = s.begin(), pe = s.end();

  while (p != pe)
  {
      if (p == s.find(subs))
      p = s.erase(p);
    else
      ++p;
  }
}

int main ()
{
  set<int> myset;
  set<int>::iterator it;
  pair<set<int>::iterator,bool> ret;

  for (int i=1; i<=5; ++i) myset.insert(i*10);

  ret = myset.insert(20);

  if (ret.second == false) it=ret.first;

  myset.insert (it,25);
  myset.insert (it,24);
  myset.insert (it,26);

  int myints[] = {5,10,15,60,28};
  myset.insert(myints,myints+5);

  cout << "myset contains after insert:";
  for (it=myset.begin(); it!=myset.end(); ++it)
     cout << ' ' << *it;
   cout << '\n';

   // set to‘plam tltmentlarini o‘chirish
   it = myset.begin();
   ++it;
```

```
  myset.erase (it);

  myset.erase (40);

  it = myset.find (60);
  myset.erase (it, myset.end());

  cout << "myset contains after erase :";
  for (it=myset.begin(); it!=myset.end(); ++it)
     cout << ' ' << *it;
   cout << '\n';

  erase_subs(myset,24);

  cout << "myset contains after erase_subs :";
  for (it=myset.begin(); it!=myset.end(); ++it)
     cout << ' ' << *it;
   cout << '\n';
   system("pause");
  return 0;
}
```

3.2-dastur. Output

```
myset contains after insert: 5 10 15 20 24 25 26 28
30 40 50 60
myset contains after erase : 5 15 20 24 25 26 28 30
50
myset contains after erase_subs : 5 15 20 25 26 28
30 50
```

Tartiblangan multito‘plam <set> - <multiset>. <set> to‘plamdan farqli o‘laroq, insert funksiyasi qiymat qo‘shish uchun qo‘shiladigan qiymatni ko‘rsatuvchi iterator qaytaradi. <multiset> - assosiativ konteynet bo‘lib, teng qiymatli kalitlarni saqlaydi (mumkin qadar bir kalit qiymatli elmentlar to‘plamini saqlaydi) va kalit orqali tez qidirish imkonini beradi.

<multiset> ning shabloni (<set> ni shabloni bilan bir xil):

```
template <class Key, class Compare = less<Key>,
    template <class U> class Allocator = allocator>
```

<multiset> sinfning ochiq xususiyatlari, operatorlari va funksiyalari:

1. typedef operatorlari - <set> niki bir xil.
2. Xotirani ajratish va bo‘shatish operatorlari (allocation/deallocation):

```
multiset(const Compare& comp = Compare());
    template <class InputIterator>
    multiset(InputIterator first, InputIterator
last,
        const Compare& comp == Compare());
    multiset(const multiset<Key, Compare,
Allocator>& x);
    ~multiset();
    multiset<Key, Compare, Allocator>&
operator=(const multiset<Key,
            Compare, Allocator>& x);
    void swap(multiset<Key, Compare, Allocator>&
x);
```

3. Ruxsat berish vositalarining operatorlari (accessors) - <set> niki bilan bir xil.
4. Qo‘shish va o‘chirish operatorlari (insert/erase):

```
    iterator insert(const value_type& x);
    iterator insert(iterator position, const
value_type& x);
    template <class InputIterator>
    void insert(InputIterator first,
InputIterator last);
    void erase(iterator position);
    size_type erase(const key_type& x);
    void erase(iterator first, iterator last);
```

5. To‘plam amallari (set operations) - <set> niki bir xil.
6. Taqqoslash operatorlari:

```
template <class Key, class Compare, class
Allocator>
bool operator==(const multiset<Key, Compare,
Allocator>& x,
const multiset<Key, Compare, Allocator>& y);

template <class Key, class Compare, class
Allocator>
bool operator<(const multiset<Key, Compare,
Allocator>& x,
        const multiset<Key, Compare,
Allocator>& y);
```

<multiset> to‘plamda:

iterator (iterator) bu – const value_type ko‘rsatuvchi domiy ikki tomnlama iterator. Aniq tipni belgilash realizatsiya qilishga bog‘liq va Allocatorda aniqlanadi.

const_iterator - ham iterator (iterator) kabidir.

size_type - ishorasiz butun tip, aniq tipni belgilash realizatsiya qilishga bog‘liq va Allocatorda aniqlanadi.

difference_type - ishorali butun tip, aniq tipni belgilash realizatsiya qilishga bog‘liq va Allocatorda aniqlanadi.

<*set*> va <*multiset*> to‘plamlarning xususiyatlari, operatorlari va funksiyalari umuman olganda bir xil. Shuning uchun 2 ta to‘plamni birga o‘rganish maqsadga muvofiq.

set va *multiset* uchun foydalaniladigan iteratorlar o‘zgarmas ikki tomonlama iterator bo‘lganligi uchun *algorithm* kutubxonasida mazkur tipni qo‘llamaydigan funksiyalari uzatish mumkin emas. Shuning uchun joriy sinfda aniqlangan massiv elementlari bilan ishlaydigan usullari, funksiyalardan (umumlashgan algoritmmlari o‘rnida) foydalanish kerak.

Yuqorida ta‘kidlab o‘tilgandek, *multiset* to‘plamning *set* to‘plamdan farqi faqat kalit bir nechta bir xil qiymatli kalitlarni saqlashidadir. Shuning uchun ixtiyoriy sohada ishlatishda *multiset* sinfining ishlatilishi *set* sinfinikidan farq qilmaydi.

set va *multiset* sinflarining obyektlarining tipi kalit bilan yonmayon bitta shablonli parametr olishi mumikn. Bu shablon taqqoslash (*comp*) funksiyasidir. Agar shundan funksiya mavjud bo‘lsa, oshkormas `less<>` funksiyasi bilan beriladi (`< amali`).

set (*multiset)* sinflarining obyektlarini quyidagi konstruktolar bilan yaratish mumkin:

Bo‘sh to‘plam konstruktori - set<type, comp> ar; yoki set<type, comp> ar(Comp);

Nusxalash konstruktori - set<type, comp> ar(other);

Iteratorlar yordamida qo‘shish konstruktori - set<type, comp> ar(first, last); yoki set<type, comp> ar(first, last, Somp);

Ro‘yxat asosida initsializatsiya qilish konstruktori - set<type, comp> ar {init}; yoki set<type, comp> ar(init); yoki set<type, comp> ar(init, Comp);

Bu konstruktorlarda Comp – konteynerlarning kalitlarini taqqoslash funksyasi (ixtiyoriy). Agar dasturchi o‘zining allokator funksiyasini yaratsa, qo‘shish uchun taqqoslash funksiyasining yonida konstruktorga majburiy bo‘lmagan allocator() funksiyasi bor.

set (*multiset)* sinflarining obyektlarini o‘chirish uchun destruktor - ar.~set();

Hajmlar va o‘lcham bilan ishlash usullari: empty() – true qaytaradi, agar to‘plam bo‘sh bo‘lsa false, size() - to‘plamdagi elementlar sonini qaytaradi, max_size() – mumkin bo‘lgan elementlar sonini qaytaradi.

Modifikatorlari: clear() – konteynerni tozalaydi, insert() – element qo‘shish, erase() – elementni o‘chirish, swap() – o‘rin almashtirish, emplace() – joyida elementni yaratish.

Qidirish usullari: set to‘plam bilan ishlaganda qidirish muhim xisoblanadigan aspekt bo‘lib hisoblanadi. Qidirishni samarali tashkil qilish uchun bir nechta o‘zning usullariga ega: count(key), find(key), equal_range(key), lower_bound(key), upper_bound(key)

Taqqoslash va birlashtirish amallari: set to‘plam uchun barcha taqqoslash amallarini bajaradi. To’plamning ketma-ket konteynerlariga tegishli barcha amal o‘rinli.

[=] (operator=) operatorni elementlar to‘plamini olish uchun ishlatiladi. Masalan, other to‘plamning nusxasini yangi bir other_one to‘plamga nusxalash uchun ishlatiladi (other_one = other).

Semantik nusxalashni amalga oshirish usuli other to‘plamning nusxasini yangi bir other_one to‘plamga to‘liq nusxalash uchun ishlatiladi va other to‘plami o‘chirilmaydi, ammo uning holatini aniqlab bo‘lmaydi other_one = move(other).

<set> va <multiset> doir masalalr va dasturlarni keltiramiz.

1-masala. [10, 50] intervalda N ta ketma- ket sonlar tasofidiy berilgan. {10, 20, 30, 40, 50} berilan sonlardan joriy ketma-ketlikda nechtadan bor.

3.3(a)-dastur. 1-masalaning dasturi.

```
#include "stdafx.h"

#include <iostream>
#include <vector>
#include <set>
#include <random>
#include <ctime>
using namespace std;

int main() {
    default_random_engine rnd(time(0));
```

```
    uniform_int_distribution<unsigned> g(10, 50);
    vector<int> myVektor;
    int n, k = 0;
    cout << "n = "; cin >> n;
    for (int i = 0; i < n; i++) {
        int d = g(rnd);
        myVektor.push_back(d);
        cout << myVektor[i] << " ";
    }
    set<int> mySet;
    for (int i=1; i<=5; ++i) mySet.insert(i*10);
    cout << endl;
    for (int i = 0; i < n; i++)
        if (mySet.count(myVektor[i])) k++;
    cout << k << "  ta element topildi."  << endl;
    system("pause");
    return 0;
}
```

3.3(a)-dastur. Output

```
n = 255
47 50 18 29 44 47 31 31 25 19 40 32 43 29 38 22 10 39 47 10 43 46 26
47 18 50 30 24 34 47 31 13 35 41 29 26 25 26 19 46 18 17 23 32 18 47
37 14 13 13 32 28 14 42 41 29 21 19 23 38 12 31 26 48 34 44 20 21 34
13 43 42 30 26 44 10 34 22 15 26 23 26 30 22 44 41 44 12 36 33 29 37
20 17 10 28 33 26 19 12 50 31 49 33 50 41 32 42 38 35 27 23 18 25 28
31 12 36 18 42 36 44 24 37 10 34 49 26 29 11 39 33 31 15 11 42 37 40
49 34 41 20 50 39 19 29 30 24 38 47 14 41 37 21 49 21 42 14 37 45 12
40 17 46 17 10 45 43 37 19 25 39 11 49 33 17 11 12 10 22 19 26 12 47
34 16 21 33 25 44 26 40 28 38 35 15 31 26 10 32 40 42 45 32 17 20 41
30 36 13 23 24 44 19 23 29 21 38 30 26 20 23 33 31 39 20 22 50 34 18
24 21 30 21 42 13 22 41 19 35 40 14 34 37 16 14 35 17 44 40 13 37 28
45 24
34  ta element topildi.
```

Oldinroq juda tez-tez elementlarni kiritish va o‘chirish algoritmlarni foydalanish kerak emas deb aytgan. Vektor va to‘plam bilan ishlash samaradorligini taqqoslaylik. To‘plam va vektorning massivlarini

tasodifiy sonlar bilan to‘ldiradigan va belgilangan qiymatni qidiradigan dastur yarataylik.

3.3(b) – dastur. Vektor va to‘plam bilan ishlash samaradorligini taqqoslash.

```
// Created by MBBahodir

#include "stdafx.h"

#include <iostream>
#include <set>
#include <vector>
#include <algorithm>
#include <random>
#include <ctime>
#include <chrono>
using namespace std;

int main() {
    int n;
    default_random_engine rnd(time(0));
    uniform_int_distribution<unsigned> g(1, 10000);
    cout << "n = "; cin >> n;
    const int k = 100;
    int a = n;

    auto start = chrono::system_clock::now();
    vector<int> ar1;
    while (a--) ar1.push_back(g(rnd));
    auto result1 = find(ar1.begin(), ar1.end(), k);
    auto end = chrono::system_clock::now();
    auto elapsed = end - start;
    cout << "vector => "
         << elapsed.count()
         << endl;
    a = n;


    start = chrono::system_clock::now();
```

```
    set<int> ar2;
    while (a--) ar2.insert(g(rnd));
        auto result2 = ar2.find(k);
    end = chrono::system_clock::now();
    elapsed = end - start;
    cout << "set => " << elapsed.count() << endl;

    system("pause");
    return 0;
}
```

3.3(b)-dastur. Output

```
n = 10000
vector => 289989
set => 1160000
```

To‘plamning ishlashi vektordan sezilarli darajada past bo‘ladi. Shuni yodda tutingki, to‘plamni to‘ldirish amali saralash bilan amalga oshiriladi va bu albatta, qimmatbaho amal. Lekin, faqat qidirish uchun, har ikki konteynerlarni ishlash solishtiradigan bo‘lsa, vaziyat <set> foydasiga o‘zgaradi:

3.3(c)-dastur. Output

```
n = 10000
vector => 70042
set => 20044
```

Bundan qanday xulosa chiqarish mumkin? set yordamida (yoki multiset) takrorlanish jarayonlarini dasturlashda ketma ketlikdan elementlarni olish, yoki tez-tez o‘chirish va elementlar kiritish bu noto‘g‘ri yondashuv hisoblanadi. Set to‘plam obyektini yaratish uchun takrorlanish jarayonlari asosida shakllantirilgan ketma-ketlikdan foydalanish kerak yoki tayyor to‘plam uchun insert funksiyasidan foydalanish maqsadga muvofiq.

Ammo, dasturda elementlarni qidirish tez-tez ishlatilsa, set raqobatga chiqa olmaydi.

pair - yordamchi sinfi. Boshqa assotsiativ konteynerlar bilan ishlashni yanada ko‘rib chiqish uchun pair yordamchi shablon sinfi bilan tanishishimiz kerak. Bu sinf birlik sifatida ikkita obyektlarni saqlash imkoniyatini beradi. pair STD ning turli joylarida, xususan minmax() algoritmida, set equal_range() sinfining usulida, shuningdek juft (key,

value) bo‘lgan assosiativ konteynerlar elementlari bilan ishlash uchun ishlatiladi.

pair - bir juft obyekt yaratish uchun quyidagi konstruktor sintaktiki ishlatiladi:

```
pair<T1, T2> p1;
pair<T1, T2> p2(val1, val2);
pair<T1, T2> p3(p1);
```

Amalari:

p.first – havola bo‘yicha juftlikning birinchi elementiga murojaat.
p.second – havola bo‘yicha juftlikning ikkinchi elementiga murojaat.
p->first – ko‘rsatkich bo‘yicha juftlikning birinchi elementiga murojaat.
p->second – ko‘rsatkich bo‘yicha juftlikning ikkinchi elementiga murojaat.
p = other_p – qiymat qilib berish (C++ qoidasiga asosan oshkormas tip almatirish yordamida).
p.swap(other_p) yoki swap(p, other_p) –p va other_p lar uchun o‘rin almashtirish.
<, <=, >, >=, ==, != – taqqoslash amallari
make_pair(val1, val2) – juftlikni beradi.

3.4-dastur. Pair konstruktori va amallarini ishlatish.

```
// Created by MBBahodir
#include "stdafx.h"

#include <iostream>
using namespace std;

int main() {
    int a1 = 16, b1 = 11;
    int a2 = 18, b2 = 03;
    pair<int, int> par1(a1, b1);
    pair<int, int> *par2 = new pair<int, int>(a2,
b2);
    cout << par1.first << '.' << par1.second <<
endl;
    cout << par2->first << '.' << par2->second <<
endl;
    system("pause");
    return 0;
```

```
}
```

3.4-dastur. Output

```
16.11
18.3
```

Assosiativ konreynerlar uchun map va multimap sinflari. map assotsiativ konteyneri (xarita, lug‘at) o‘z-o‘zini muvozanatlovchi daraxtga (qizil-qora daraxt) asoslangan to‘plamlar bilan bir xil tarzda amalga oshiriladi. To‘plam va lug‘at o‘rtasidagi asosiy farq shundaki, map massivi kalitlardan tashkil topgan elementlar juftlarining tartibli assotsiativ massivi (konteyneri) va ularning mos qiymatlaridan iborat. Lug‘atda kalitlari unikal (takrorlanmaydigan) va multimap bir nechta nusxadagi kalitlari bo‘ladi. Lug‘atlarda kalit va qiymat turlari fundamental tip yoki abstrakt tip bo‘lishi mumkin. Agar kalit tipi abstrakt tip bo‘lsa, uning uchun saralash yo‘nalishini belgilovchi komparator funksiyasi (taqqoslash funksiyasi) aniqlanishi kerak. Kalit doimiy qiymatga ega va kalit qiymatini o‘zgartirish mumkin emas. Juftlikning ikkinchi element qiymatini (asosiy qiymatini) o‘zgartirish mumkin.

Map (yoki multimap) sinflari bilan ishlashni boshlash uchun dastur sarlavhasiga (ikkala sinf bilan birgalikda) quyidagi dastur fragmenti yozildi (bu oldin ham barcha assosiativ konteynerlar uchun aytilgan edi):

```
#include <map>
```

Konstruktorlar. Lug‘at obyektlari shablon parametrlarini: kalit tipi va kalitning qiymati tipi (key va T), taqqoslash (comp)funksiyasini o‘z ichiga oladi. Agar bunday funksiya bo‘lmasa, u oshkormas less <> funksiya (operation <) tomonidan o‘rnatiladi.

Quyidagi konstruktorlar yordamida map sinf obyektlarini yaratiladi:

Oddiy map konreyner konstruktorlari: map<Key, T, comp> ar;
yoki
map<Key, T, comp> ar(Comp);

Nusxalash uchun konstruktor: map<Key, T, comp> ar(other);

Iteratorlar yordamida qo‘shish uchun konstruktorlar: map<Key, T, comp> ar(first, last); yoki map<Key, T, comp> ar(first, last, Somp);

Ro‘yxat asosida initsializatsiya qilish konstruktorlari: map<Key, T, comp> ar {init}; yoki map<Key, T, comp> ar(init); yoki map<Key, T,

comp> ar(init, Comp); *Eslatma*. Ro‘yxatni initsializatsiya qilishda har bir juftlik alohida figurali qavs ichiga olinishi kerak!

```
map<string, int> ar {{"a1", 10}, {"www", 17},
{"j8", 100}};
```

Bu yerda Comp konteyner kalitlari solishtirish uchun funksiyasi (ixtiyoriy). Bundan tashqari, taqqoslash funksiyaning yonida konstruktor ixtiyoriy Allocator() vazifasini o‘z ichiga oladi (soddaligi uchun yozilmaydi), dasturchi uning allocator vazifasini ham belgilaydi.
Map sinfining destruktori: ar.~map();

Lug‘at usullari. Lug‘at usullari set to‘plam usullari bilan bir xil bo‘lganligi uchun ularni takrorlab keltirmaymiz. Bu usullar qanday ishlashini misol yordamida ko‘rsatamiz. Aytaylik, quyidagicha lug‘at yaratishimiz kerak: kalitga tasodifiy (string), qiymatga tasodifiy (integer) o‘rnatiladi. So‘ngra key1 va key2 orasidagi barcha key = > qiymat juftliklarining chiqishi talab qilinsin. Dastur oxirida key 3 kalit uchun maksimal qiymatni chiqarish talab qilinadi (dasturning o‘zidagi izohlarga qarang).

3.5-dastur. Lug‘at usullaridan foydalanish.

```
// Created by MBBahodir
#include "stdafx.h"

#include <iostream>
#include <random>
#include <ctime>
#include <string>
#include <map>
#include <vector>

using namespace std;

int main() {
    map<string, int> rat;
    vector<string> lit;
    lit.push_back("a1"); lit.push_back("a2");
lit.push_back("a3");
    lit.push_back("b1"); lit.push_back("b2");
lit.push_back("b3");
    lit.push_back("c1"); lit.push_back("c2");
lit.push_back("c3");
```

```
    default_random_engine rnd(time(0));
    uniform_int_distribution<int> d(-10, 20);
    uniform_int_distribution<int> c(0, 8);

//============================1=================//
    // lug‘atni to‘ldirish: tasofidiy kalit tasodifiy
olinadi     //

//============================================//
    int n;
    cout << "Lug‘at elementlari sonini kirit: ";
    cin >> n;
    while (n--) {
        string S = lit[c(rnd)];
        int D = d(rnd);
        pair<string, int> tpr = make_pair(S, D);
        rat.insert(tpr);
        // emplace uchun juftlik yaratish shartmas:
        //rat.emplace(S, D);
    }

//============================2=================//
    // 2 ta har xil amal bilan kalit va qiymat
qoshish            //

//============================================//
    rat.emplace("d1", 10);
    rat.insert(pair<const string, int>("d3", -15));
    rat.insert(pair<const string, int>("d3", 6));

//============================3=================//
    // birinchi va ikkinchi kalit orasidagi
elementlarni chiqarish//

//============================================//
    string el_l, el_r;
    cout << "kalitlarni kirit:\n";
    cout << "1-kalit: "; cin >> el_l;
```

```
    cout << "2-kalit: "; cin >> el_r;
    auto fst = rat.lower_bound(el_l);
    auto lst = rat.upper_bound(el_r);
    while (fst != lst) {
        cout << fst -> first
             << " => "
             << fst -> second
             << endl;
        fst++;
    }

//==============================4=================//
    // kalitga mos eng katta elementni aniqlash
//

//================================================//
    string tmp;
    cout << "Kalit kirit: ";
    cin >> tmp;
    auto first = rat.equal_range(tmp).first;
    auto last = rat.equal_range(tmp).second;
    if (first == last)
        cout << 0 << endl;
    else
        while (first != last) {
            auto r = first -> second;
            if (r >= 10)
                cout << r << " ";
            first++;
        }
        system("pause");
    return 0;
}
```

3.5-dastur. Birinchi sinov Output

```
Lug‘at elementlari sonini kirit: 10
a2 => 14
a3 => 13
```

```
b2 => -8
b3 => 3
c2 => -4
c3 => -7
d1 => 10
d3 => -15
kalitlarni kirit:
1-kalit: a1
2-kalit: c2
a2 => 14
a3 => 13
b2 => -8
b3 => 3
c2 => -4
Kalit kirit: c4
0
```

3.5-dastur. Ikkinchi sinov Output

```
Lug‘at elementlari sonini kirit: 9
a1 => 11
a2 => 10
b1 => 2
b2 => -10
c1 => 3
c2 => 9
d1 => 10
d3 => -15
kalitlarni kirit:
1-kalit: a1
2-kalit: c2
a1 => 11
a2 => 10
b1 => 2
b2 => -10
c1 => 3
c2 => 9
Kalit kirit: d3
```

Elementlarga murojaat. Lug‘atlarda elementlarga kirishning yana ikkita usuli mavjud (iteratorlar bilan ishlashdan tashqari). Birinchi usul at(key) usulini qo‘llashdir (bilsangiz kerak). Biroq, indeksni argument sifatida qabul qilmaydi, lekin kalitning qiymatini qabul qiladi. Bu usul kalitga ekvivalent bo‘lgan kalit bilan mos elemeni qiymatiga mos yozuvlar qaytaradi. Agar bu element mavjud bo‘lmasa, out_of_range istisno funksiyasining qiymati qaytariladi. Boshqacha aytganda, massivda bunday kalit bor-yo‘qligini tekshiradi. Ikkinchi usul overloaded[] funksiyadan foydalanishga asoslangan. Lekin quyidagi sabablarga ko‘ra ehtiyotkorlik bilan foydalanish zarur. ar(key) mavjud bo‘lmasa, standart konstruktor yordamida kalit bilan yangi element qator qo‘shiladi. Aks holda elementga ko‘rsatkich qaytariladi. Eslattb o‘tamizki, indekslash funksiyasini assotsiativ konteyner sinflarning ikki turga (map va unordered_map) foydalanish mumkin. Ushbu usullarni ishlab chiquvchilar tomonidan qo‘shilishining sabablari aniq: dastur juda qisqa va chiroyli bo‘ladi.

at() funksiyasi istisno qaytargani uchun try - catch funksiyasi bilan quyidagicha boshqarish mumkin:

```
map<string, int> rat;
try {
    rat.at("U");
}
// istesnoni ushlab olish
catch (std::out_of_range) {
    // uni qayta ishlash
    rat["U"] = 15;
}
```

Dastur fragmentida "U" kalitli juftlik topilmasa, uni yaratib, yangi qiymat beradi.

rat["U"] = 15; (hiyla, ruscha fishka) indekslash amali u faqat asosiy qiymatini olish emas, balki bu qiymatini o‘zgartirish mumkin degan ma‘noni anglatadi, bir qiymatini qaytaradi. Boshqacha qilib aytganda, ushbu dastur fragmentidagi amallarini bajarish mumkin:

```
rat["b1"]=1980;
cout << rat["b1"] << endl;
rat["b1"] +=1;
cout << rat["b1"] << endl;
--rat["b1"];
cout << rat["b1"] << endl;
```

map<K, T, C = less<K>, A = allocator<pair<const K, T>>> to‘plam pair<const K, T> tipidagi qiymatni saqlaydi. Bunda K kalit, tanlash imkonini beradi va T saqlanadigan qiymat. Shuning uchun iteratordan foydalangannimizda first kalit qiymatni va second saqlangan qiymatni qaytaradi, o‘zgartirish imkonini beradi. Yuqorida aytib o‘tilgandek operator[] asosiy amallardan hisoblanadi (vazifasini bilsangiz kerak). Agar map<K, T> m bo‘lsa va m[k] = t ma‘no jihatidan quyidagi fragmentga teng:

```
m.insert(make_pair(k, T())).first->second = t
```

Bu kabi elementlarni qo‘shish samarali hisoblanib, T obyektni yaratishga imkon bermaydi (yuqoridagi fragmentda aniq T() konstruktor chaqirilgan).

Agar operator[] noqulay va samarasiz deb hisoblasangiz, yana unga ikkita alternativ variantlar bor. Birinchisi find funksiyasidan foydalanib,kalit bo‘yicha (key, value) juftligiga mos iteratorni olish yoki kalit lug‘atda bo‘lmasa, end() funksiyadan foydalanish mumkin. Ikkinchisi at() funksiyasidan, ya‘ni kalit asosida qiymatni qaytaradi. Agar kalit bo‘lmasa, istisno holatga o‘tadi.

Map sinfining umumiy shabloni quyidagicha:

```
template <class Key, class T, class Compare = less<Key>,
        template <class U> class Allocator = allocator>
```

Map sinfining operatorlari, xususiyatlari, funksiyalari va usullarini qo‘rib chiqamiz.

typedef operatorlari :

```
typedef Key key_type;
    typedef pair<const Key, T> value_type;
    typedef Compare key_compare;
    class value_compare
    :public binary_function<value_type, value_type, bool> {
    friend class map;
    protected:
        Compare comp;
        value_compare(Compare c) : comp(c) {}
    public:
        bool operator()(const value_type& x, const value_type& y) {
```

```
            return comp(x.first, y.first);
        }
    };
    typedef iterator;
    typedef const_iterator;
    typedef Allocator<value_type>::pointer pointer;
    typedef        Allocator<value_type>::reference
reference;
    typedef   Allocator<value_type>::const_reference
const_reference;
    typedef size_type;
    typedef difference_type;
    typedef reverse_iterator;
    typedef const_reverse_iterator;
```

Xotirani ajratish va bo‘shatish (xolli qilish) operatorlari (allocation/deallocation):

```
map(const Compare& comp = Compare());
    template <class InputIterator>
    map(InputIterator first, InputIterator last,
    const Compare& comp = Compare());
    map(const map<Key, T, Compare, Allocator>& x);
    ~map();
    map<Key, T, Compare, Allocator>&
    operator=(const     map<Key,     T,     Compare,
Allocator>& x);
    void swap(map<Key, T, Compare, Allocator>& x);
```

Ruxsat etish vositalari (accessors):

```
key_compare key_comp() const;
    value_compare value_comp() const;
    iterator begin()
    const_iterator begin() const;
    iterator end();
    const_iterator end() const;
    reverse_iterator rbegin();
    const_reverse_iterator rbegin();
    reverse_iterator rend();
    const_reverse_iterator rend();
    bool empty() const;
```

```
    size_type size() const;
    size_type max_size() const;
    Allocator<T>::reference        operator[](const
key_type& x);
```

Qo‘shish va o‘chirish funksiyalar (insert/erase):

```
pair<iterator, bool> insert(const value_type& x);
    iterator   insert(iterator   position,   const
value_type& x);
    template <class InputIterator>
    void insert(InputIterator first, InputIterator
last);
    void erase(iterator position);
size_type erase(const key_type& x);
void erase(iterator first, iterator last);
```

Map sinf usulari:

```
iterator find(const key_type& x);
const_iterator find(const key_type& x) const;
size_type count(const key_type& x) const;
iterator lower_bound(const key_type& x);
const_iterator   lower_bound(const   key_type&   x)
const;
iterator upper_bound(const key_type& x);
const_iterator   upper_bound(const   key_type&   x)
const;
pair<iterator,      iterator>      equal_range(const
key_type& x);
pair<const_iterator,                 const_iterator>
equal_range(const key_type& x)const;
```

Mantiqiy operatorlari:

```
template <class Key, class T, class Compare, class
Allocator>
bool   operator==(const   map<Key,   T,   Compare,
Allocator>& x,
    const map<Key, T, Compare, Allocator>& y);
template <class Key, class T, class Compare, class
Allocator>
bool   operator<(const   mapr<Key,   T,   Compare,
Allocator>& x,
```

```
        const map<Key, T, Compare, Allocator>& y);
```

iterator- ikki tomonlama iterator bo‘lib value_type ishora beradi. Aniq tipi amalga oshirish bilan bog‘liq va Allocator asosida belgilanadi.

const_iterator- doimiy ikki tomonlama iterator bo‘lib const value_type ishora beradi. Aniq tipi amalga oshirish bilan bog‘liq va Allocator asosida belgilanadi. Bunda iterator va const_iterator uchun konstruktor bor, bu kafolatlanadi.

size_type – butun ishorasiz tip. Aniq tipi amalga oshirish bilan bog‘liq va Allocator asosida belgilanadi.

difference_type - butun ishorali tip. Aniq tipi amalga oshirish bilan bog‘liq va Allocator asosida belgilanadi.

Assotsiativ konteynerlar uchun standart usullar to‘plamidan tashqari, lug‘at *Allocator::reference operator[](const key_type&)* amalini ta‘minlaydi. Masalan, m lug‘atda k kalit bilan m[k] quyidagi fragment bilan ekvivalent:

```
(*((m.insert(make_pair(k, T()))).first)).second
```

Assotsiativ konteynerlar asosiy xususiyatlari

Headers		<set>		<map>	
Members		set	multiset	map	multimap
	constructor	set	multiset	map	multimap
	destructor	~set	~multiset	~map	~multimap
	assignment	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin
	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin	rbegin
	rend	rend	rend	rend	rend
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin
	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin	crbegin
	crend	crend	crend	crend	crend
capacity	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	

Tartiblangan unikal kalitga ega bo‘lmagan map – multimap sinfi hisoblanadi. Bu sinf `operator[]` standart operatorni qo‘llab quvvatlamaydi, va multiset juftligini eslatadi. Bunda qidirish faqat birinchi maydon, ya‘ni kalit maydon bilan amalga oshiriladi. multimap sinfini o‘rganib chiqish uchun, misol sifatida matnli faylga yozilgan lug‘atga murojaat amallarini ko‘rsatamiz. Faraz qilaylik lug‘at so‘zlar juftligidan tashkil topgan va birinchi so‘z kalit bo‘lsin va takrorlanishi mumkin.

“so‘z - tarjimasi” tipini aniqlash. Bunda so‘z juftligini olamiz va lug‘al element deb qaraymiz.

```
using Entry = pair<string, string>;
```

Lug‘at tipini aniqlash

```
using Dictionary = multimap<string, string>;
```

Standart kirish va chiqish operatorlari quyidagicha aniqlaymiz.

```
namespace std
{
 istream& operator>>(istream &is, Entry &en)
 {
  return is >> en.first >> en.second;
 }

 ostream& operator<<(ostream &os, const Entry &en)
 {
  return os << en.first << ' ' << en.second;
 }

 istream& operator>>(istream &is, Dictionary &d)
 {
  istream_iterator<Entry> begin(is), end;
  d = Dictionary(begin, end);
  return is;
 }

 ostream& operator<<(ostream &os, const Dictionary
&d)
 {
  ostream_iterator<Entry> out(os, "\n");
  copy(d.begin(), d.end(), out);
```

```
  return os;
 }
}
```

Lug‘atga kirish uchun amal. Belgilangan lug‘at uchun yangi lug‘at chiqaradi.

```
Dictionary reverse(const Dictionary &d)
{
 Dictionary result;
 for (const auto &el : d)
  result.emplace(el.second, el.first);
 return result;
}
```

Lug‘at elementiga murojaat va o‘qish.

```
void dictionary_reverse(istream &from, ostream &to)
{
 Dictionary read;
 from >> read;
 to << reverse(read);
}
```

Std sinfida aniqlangan istream_iterator va ostream_iterator tiplaridan foydalanganlik uchun [<<] va [>>] operatorlari shu fazoga moslab yaratilgan. Bu foydalanuvchiga kirish va chiqish operatorlari va standart tiplardan foydalanish ruhiyatiga ta‘sir qilmaydi.

Multimar sinfning konstruktori map sinfiniki bilan bir xil.

Multimar sinfning map sinfinikidan farf qiladigan operatorlari:

```
friend class multimap;
    protected:
        Compare comp;
        value_compare(Compare c) : comp(c) {}
    public:
        bool operator()(const value_type& x, const
value_type& y) {
            return comp(x.first, y.first);
        }
    };
```

Mantiqiy operatorlari:

```
template <class Key, class T, class Compare, class
```

```
Allocator>
bool operator==(const multimap<Key, T, Compare,
Allocator>& x,
        const multimap<Key, T, Compare, Allocator>&
y);

template <class Key, class T, class Compare, class
Allocator>
bool operator<(const multimap<Key, T, Compare,
Allocator>& x,
        const multimap<Key, T, Compare, Allocator>&
y);
```

Tartiblanmagan assosiativ konteynerlar <unordered_set> va <unordered_map>. Tartiblanmagan konteynerlar <unordered_set>, <unordered_map> xesh – jadval asosida qurilgan (odatda bu xesh jadvallar ekvivalent elementlar ro‘yxati - bloklar) va xesh funksiyalarga (joriy holatida standart tiplar uchun <functional> sinfidan std::hash<T> bilan aniqlanadi) va tenglik amaliga (joriy holatda [==] operatori) tayanadi.

Ikkita element uchun xeshlari teng bo‘lsa va tenglik amali chin (true) qiymat qaytarsa, ular ekvivalent hisoblanadi. Tartiblanmagan konteynerlari bir tomolama iteratorlar bilan elementlariga murojaat qilish kerak.

Tartiblanmagan standart assosiativ konteynerlarga quyidagi shablon sinflar kiradi*:*

```
unordered_set<K, H = hash<K>, E = equal_to<K>, A =
allocator<K>>,
unordered_map<K, T, H = hash<K>, E = equal_to<K>, A =
allocator<pair<const K, T>>>,
unordered_multiset,
unordered_multimap.
```

Bularning funksionalligi tartiblangan assosiativ konteynerlarga o‘xshaydi.

Tartiblanmagan konteynerlarda xesh jadvalni shakllantirishning o‘ziga xosligi lower_bound va upper_bound funksiyalariga egamasligidir. Ammo, berilgan kalitga mos bir intervaldagi elementlarni olish uchun equal_range funksiyasi ishlatiladi.

O‘rtacha tartiblanmagan konteynerlarda elemetnlarni qo‘shish, qidirish va o‘chirish o‘rtacha doimiy vaqt talab va tartiblanganlarga nisbatan sezilarli darajada tezroq bo‘lishi mumkin. Biroq, eng yomon holatda ham vaqtga chiziqli erishish mumkin. Samaradorligi hesh funksiyasini amalga oshirish sifatiga bog‘liq, lekin butunlay bunday vaqt ajratishni oldini olish mumkin emas. Shuning uchun, interaktiv dasturlarda (masalan, o‘yinlar), tartibga solinmagan konteynerlar davriy kechikishlar tufayli tartiblanganlardan ko‘ra yomonroq bo‘lishi mumkin.

Katta sondagi elementlar qo‘shish uchun, kerakli vaqtda bir qator kiritishda "rehash"ga oshirish mumkin (ko‘p vaqt oladigan ajratilgan saqlash va qayta tarqatish elementlar hajmini oshirish) rehash funksiyasini chaqirib to‘g‘ri hisoblanadi (vektor uchun reserve funksiyasiga o‘xshaydi).

Ayrim vazifalarda saralash uchun vaqt ko‘p ajratilsa avtomatik saralashni muhim kamchilik deb hisoblash mumkin. Bu holda, tartiblangan to‘plam va lug‘atlar sinflarini tartiblanmagan assotsiativ konteynerlar sinflar (unordered_set, unordered_multiset, unordered_map va unordered_multimap) tomonidan o‘zgartirilishi mumkin. Tartiblanmagan konteynerlarda qidirish, qo‘shish va o‘chirish amallari o‘rtacha doimiy vaqt murakkabligiga ega va u O (1) ga tengdir.

Tartiblanmagan konteynerlar xesh jadvallar sifatida amalga oshiriladi. Xesh-jadvalda amalni bajarish xesh-funksiyada (xeshlash) kalitdan boshlanadi. Olingan xesh qiymati (shuningdek, xesh yoki xesh kodi) massivda indeks rolini bajaradi. Xesh jadvali oddiy lug‘atga o‘xshaydi, unda tom ma‘noda xesh qiymati deb hisoblash mumkin.

Tartiblanmagan konteynerlarning xesh jadvallarida yacheykalari bor va ular cheksiz ko‘p elementlarni o‘z ichiga olishi mumkin. Bu yacheykani segment deb ataymiz. Aslida, segment (xesh bilan bog‘liq) ulangan ro‘yxatdir.

To‘plam hajmiga nisbatan saqlanadigan elementlar soni mos kelsa, (xesh funksiyasi uchun mumkin bo‘lgan elementlar soni) xesh jadvalni to‘ldirish (load factor) koeffitsienti deb yuritiladi va o‘rtacha amal ijro vaqt belgilaydigan muhim parametr hisoblanadi.

Unordered_set va unordered_multiset sinflari bilan ishlashni boshlash uchun unordered_set sarlavhasini (ikkala sinf bilan birgalikda) quyidagicha qo‘shish kerak:

```
#include <unordered_set>
```

unordered_map va unordered_multimap sinflari ham xuddi yuqoridagidek qo‘shiladi:

```
#include <unordered_map>
```

Tartiblanmagan konteynerlarning tiplari quyidagi shablon parametrlarini o‘z ichiga oladi:

- const Key – kalit tipi (unordered_set, unordered_multiset, unordered_map, unordered_multimap).
- T – qiymatning tipi (unordered_map, unordered_multimap).
- alloc –allokator funksiya (majburiy emas, erkin).
- size_type bucket_count – initsializatsiya qilishda foydalanish uchun minimal yacheykalar soni (agar ko‘rsatilmagan bo‘lsa, joriy holat bo‘yicha olinadi).
- hash – xesh-funksiya (erkin).
- equal – taqqoslash funksiyasi, kalitlarni solishtirish uchun ishlatiladi (erkin).

Quyidagi konstruktorlar asosida unordered_set va unordered_map sniflarining obyektini quradi:

```
unordered_set<Key> ar;
unordered_set<Key, hash, equal> ar;
unordered_map<Key, T> ar;
unordered_map<Key, T, hash, equal> ar;
```

Agar hash xesh funksiya yozilmagan bo‘lsa, joriy holat bo‘yicha hash<key_type> funksiyasi ishlatiladi.

Agar equal taqqoslash funksiyasi yozilmagan bo‘lsa, joriy holat bo‘yicha equal_to<> (operator==) funksiyasi ishlatiladi.

Soddaligi uchun Allocator () va bucket_count yozilmaydi.

Nusxalash konstruktrlari:

```
unordered_set<Key> ar(other);
unordered_map<Key, T> ar(other);
```

Iteratorlar asosida qo‘shish:

```
unordered_set<Key> ar(first, last);
unordered_set<Key, hash, equal> ar(first, last);
unordered_map<Key, T> ar(first, last);
unordered_map<Key, T, hash, equal> ar(first, last);
```

Ro‘yxat bo‘yicha **initsializatsiya qilish:**

```
unordered_set<Key> ar {init};
unordered_set<Key> ar(init);
unordered_set<Key, hash, equal> ar(init);
unordered_map<Key, T> ar {init};
unordered_map<Key, T> ar(init);
unordered_map<Key, T, hash, equal> ar(init);
```

unordered_set va unordered_map sinf obyektlarini o‘chirish uchun destruktorlar:

```
ar.~unordered_set();
ar.~unordered_map();
```

Tartiblanmagan konteynerlarda ishlatilmaydigan usullar:

- [>], [<,] [>=], [<=] taqqoslash amallari ([==]va [!=]amallarni qo‘llab quvvatlaydi)
- lower_bound va upper_bound
- Ikki tomonlama iteratorlar: rbegin, crbegin, rend, crend

Tartiblanmagan konteynerlarning usullari:

Xeshlash uslubi (Bu usullar tartibsiz konteynerlarning ishlashini boshqarish imkonini beradi):

- load_factor() – to‘ldirish koeffitsentini qaytaradi (yuqoriga qarang)
- max_load_factor() – Agar argumentsiz foydalanilsa, maksimal to‘ldirish qiymati uchun float tipini qaytaradi, agar argument sifatida float tipi foydalanilsa, maksimal to‘ldirish qiymati uchun o‘rnatiladi . Agar terminator omili ushbu chegaradan oshsa, konteyner avtomatik ravishda segmentlar sonini oshirishi mumkin.
- rehash(size_type count) – konteyner uchun qayta xeshlash o‘tkazadi, ya‘ni yacheykalarni sonini (backet_count) backet_count >= count va backet_count > size/max_load_factor bo‘lishi uchun.
- reserve(size_type count) – songa yacheykalar sonni o‘rnatadi, konteynerni qayta xeshlash o‘kazmaslik va maksimal to‘ldirish koeffitsienti uchun, oxirgi yacheykaning count hisoblanadi.

Maksimal samaradorlikka erishish uchun har doim maksimal yacheykalar sonini aniq belgilash kerak. Tavsiya etilgan qiymatlar 0.7 dan 0.8 gacha oraliqda bo‘ladi. Element qidirish uchun standart maksimal yacheykalar soni o‘rnatiladi (1.0 ga teng) va dasturchilar tomonidan 0.7 o‘rnatiladi.

Xesh funksiyasining o‘zi konteyner faoliyatini yaxshilashi mumkin, ammo bu mavzu doirasidan chiqib ketishga olib keladi.

Interfeys segmentlari. Segment elementlari bilan bevosita ishlash uchun (bog‘langan ro‘yxat sifatida) quyidagi usullardan foydalaniladi:

- begin(), end(), cbegin(), cend() – iteratorlar. E‘tibor bering, bular segmentlar uchun, massiv uchun emas! Konteynerlar uchun shularning anologlari yaratilgan.
- size_type bucket_count() – massivdagi segmentlar sonini qaytaradi.

• size_type max_bucket_count() – muuumkin bo‘lgan maksimal elementlar soni (tizimga va joriy qilishga bog‘liq).

• size_type bucket_size(size_type n) –n indeksli segmentdagi elementlar soni.

• size_type bucket(Key) – kalit asosida segment sonini qaytaradi.

Bu usullar asosida xesh jadvalni vizual ko‘rinishini tasavvur qilish mumkin.

3.6-dastur. Tartiblanmagan konteynerlardan foydalanish.

```
#include "stdafx.h"

#include <iostream>
#include <iomanip>
#include <unordered_set>
#include <random>
#include <ctime>
#include <chrono>
using namespace std;

int main() {
    default_random_engine rnd(time(0));
    uniform_int_distribution<unsigned> g(100, 999);
    unordered_multiset<int> unm;
    size_t n = 60;
    unm.reserve(n);
    while (n--) unm.emplace(g(rnd));
    cout << "Konteynerdagi segmentlar soni: "
         << unm.bucket_count()
         << "\n"
         << "Maksimal to‘ldirishlar ko'ffitsenti: "
         << unm.max_load_factor()
         << endl;
    auto start = chrono::system_clock::now();
    auto res = unm.find(200);
    auto end = chrono::system_clock::now();
    auto elapsed = end - start;
    cout << "Natija => "
```

```
          << elapsed.count()
          << endl;
    // O‘zimiz tomondam Maksimal to‘ldirishlar
ko'ffitsenti o‘rnatamiz
    unm.max_load_factor(0.7);
    unm.rehash(200);
    cout << "Konteynerdagi segmentlar soni: "
          << unm.bucket_count()
          << "\n"
          << "Maksimal to‘ldirishlar ko'ffitsenti: "
          << unm.max_load_factor()
          << endl;
    start = chrono::system_clock::now();
    res = unm.find(200);
    end = chrono::system_clock::now();
    elapsed = end - start;
    cout << "Natija => "
          << elapsed.count()
          << endl;
    cout << " Hash tuzilmasi " << endl;
    for (size_t i = 0; i < unm.bucket_count(); i++)
{
        cout << "Segment N:" << setw(3) << i << "
=> ";
        // Iteratorga lokal segmentlarni olish
        auto first = unm.cbegin(i);
        auto last = unm.cend(i);
        while (first != last) {
            cout << *first++ << " ";
        }
        cout << endl;
    }
    system("pause");
    return 0;
}
```

3.6-dastur. Output

```
Konteynerdagi segmentlar soni: 32
Maksimal to‘ldirishlar ko'ffitsenti: 1
```

```
Natija => 30023
Konteynerdagi segmentlar soni: 256
Maksimal to‘ldirishlar ko'ffitsenti: 0.7
Natija => 10004
Segment N:  0 =>        Segment N:  1 =>
Segment N:  2 =>        Segment N:  3 =>
Segment N:  4 =>
Segment N:  5 =>        Segment N:  6 =>
Segment N:  7 =>        Segment N:  8 =>
Segment N:  9 => 646
Segment N: 10 =>        Segment N: 11 =>
Segment N: 12 =>        Segment N: 13 =>
Segment N: 14 =>
Segment N: 15 =>        Segment N: 16 =>
Segment N: 17 =>        Segment N: 18 =>
Segment N: 19 =>
Segment N: 20 =>        Segment N: 21 =>
Segment N: 22 =>        Segment N: 23 =>
Segment N: 24 => 940
Segment N: 25 =>        Segment N: 26 =>
Segment N: 27 =>        Segment N: 28 =>
Segment N: 29 =>
Segment N: 30 => 717    Segment N: 31 =>
Segment N: 32 =>        Segment N: 33 =>
Segment N: 34 =>
Segment N: 35 =>        Segment N: 36 =>
Segment N: 37 =>        Segment N: 38 =>
Segment N: 39 =>
Segment N: 40 =>        Segment N: 41 =>
Segment N: 42 =>        Segment N: 43 =>
Segment N: 44 =>
Segment N: 45 => 841 738
Segment N: 46 =>        Segment N: 47 =>
Segment N: 48 =>        Segment N: 49 =>
Segment N: 50 =>        Segment N: 51 =>
Segment N: 52 => 768    Segment N: 53 =>
Segment N: 54 =>        Segment N: 55 =>
Segment N: 56 =>        Segment N: 57 =>
```

```
Segment N: 58 =>         Segment N: 59 =>
Segment N: 60 => 631     Segment N: 61 =>
Segment N: 62 => 316     Segment N: 63 =>
Segment N: 64 =>         Segment N: 65 =>
Segment N: 66 =>         Segment N: 67 =>
Segment N: 68 =>         Segment N: 69 =>
Segment N: 70 =>         Segment N: 71 =>
Segment N: 72 =>         Segment N: 73 =>
Segment N: 74 =>         Segment N: 75 =>
Segment N: 76 =>         Segment N: 77 =>
Segment N: 78 =>         Segment N: 79 =>
Segment N: 80 =>         Segment N: 81 =>
Segment N: 82 =>         Segment N: 83 =>
Segment N: 84 =>         Segment N: 85 =>
Segment N: 86 =>         Segment N: 87 =>
Segment N: 88 =>         Segment N: 89 =>
Segment N: 90 =>         Segment N: 91 =>
Segment N: 92 => 498     Segment N: 93 =>
Segment N: 94 => 187 476
Segment N: 95 =>         Segment N: 96 =>
Segment N: 97 =>         Segment N: 98 => 280
Segment N: 99 =>         Segment N:100 => 225
Segment N:101 =>         Segment N:102 =>
Segment N:103 =>         Segment N:104 =>
Segment N:105 => 973     Segment N:106 =>
Segment N:107 =>         Segment N:108 =>
Segment N:109 =>         Segment N:110 =>
Segment N:111 =>         Segment N:112 =>
Segment N:113 =>         Segment N:114 =>
Segment N:115 =>         Segment N:116 => 960
Segment N:117 =>         Segment N:118 =>
Segment N:119 =>         Segment N:120 =>
Segment N:121 => 630     Segment N:122 =>
Segment N:123 =>         Segment N:124 => 567
Segment N:125 =>         Segment N:126 =>
Segment N:127 =>         Segment N:128 =>
Segment N:129 =>         Segment N:130 =>
Segment N:131 =>         Segment N:132 =>
```

```
Segment N:133 => 351    Segment N:134 => 227
Segment N:135 =>        Segment N:136 =>
Segment N:137 =>        Segment N:138 =>
Segment N:139 =>        Segment N:140 =>
Segment N:141 =>        Segment N:142 =>
Segment N:143 =>        Segment N:144 =>
Segment N:145 =>        Segment N:146 => 761
Segment N:147 =>        Segment N:148 => 145
Segment N:149 =>        Segment N:150 =>
Segment N:151 =>        Segment N:152 =>
Segment N:153 =>        Segment N:154 =>
Segment N:155 =>        Segment N:156 =>
Segment N:157 =>        Segment N:158 =>
Segment N:159 =>        Segment N:160 =>
Segment N:161 =>        Segment N:162 =>
Segment N:163 =>        Segment N:164 =>
Segment N:165 =>        Segment N:166 =>
Segment N:167 => 242    Segment N:168 =>
Segment N:169 =>        Segment N:170 =>
Segment N:171 =>        Segment N:172 =>
Segment N:173 =>        Segment N:174 =>
Segment N:175 =>        Segment N:176 =>
Segment N:177 =>        Segment N:178 =>
Segment N:179 =>        Segment N:180 => 458
Segment N:181 =>        Segment N:182 =>
Segment N:183 =>        Segment N:184 =>
Segment N:185 =>        Segment N:186 =>
Segment N:187 =>        Segment N:188 =>
Segment N:189 =>        Segment N:190 =>
Segment N:191 =>        Segment N:192 => 852
Segment N:193 =>        Segment N:194 =>
Segment N:195 =>        Segment N:196 =>
Segment N:197 => 287    Segment N:198 =>
Segment N:199 =>        Segment N:200 =>
Segment N:201 =>        Segment N:202 =>
Segment N:203 =>        Segment N:204 =>
Segment N:205 =>        Segment N:206 =>
Segment N:207 =>        Segment N:208 =>
```

```
Segment N:209 => 483    Segment N:210 => 697
Segment N:211 =>        Segment N:212 =>
Segment N:213 =>        Segment N:214 =>
Segment N:215 => 967    Segment N:216 =>
Segment N:217 =>        Segment N:218 =>
Segment N:219 =>        Segment N:220 =>
Segment N:221 =>        Segment N:222 =>
Segment N:223 =>        Segment N:224 =>
Segment N:225 =>        Segment N:226 =>
Segment N:227 =>        Segment N:228 =>
Segment N:229 => 666    Segment N:230 => 484
Segment N:231 =>        Segment N:232 => 603
Segment N:233 =>        Segment N:234 =>
Segment N:235 => 254    Segment N:236 =>
Segment N:237 =>        Segment N:238 =>
Segment N:239 =>        Segment N:240 =>
Segment N:241 =>        Segment N:242 =>
Segment N:243 =>        Segment N:244 =>
Segment N:245 =>        Segment N:246 =>
Segment N:247 =>        Segment N:248 =>
Segment N:249 =>        Segment N:250 =>
Segment N:251 =>        Segment N:252 =>
Segment N:253 =>        Segment N:254 =>
Segment N:255 =>
```

Tartiblanmagan konteynerlarning sinflari murakkab tuzilishga ega, ammo standart vositalar bilan ham tartiblangan konteynerlardan kam bo‘lmagan mukammal ishlashni ko‘rsatadi. Tartiblangan konteynerlar yoki tartiblanmagan konteynerlar orasidagi tanlov vazifaga bog‘liq bo‘ladi. Assotsiativ konteynerlarning asosiy usullari keng tarqalganligi sababli, dasturning bir bajarilishidan boshqasiga o‘tish oson va testlarni o‘tkazganingizdan so‘ng tanlovingizni bir yoki bir nechta turli konteynerda amalga oshirib, tanlov qilishingiz mumkn.

Tartiblanmagan konteynerlarning sinflari xususiyatlari.

Headers		**<unordered_set>**		**<unordered_map>**	
Members		**unordered_ set**	**unordered_ multiset**	**unordered_ map**	**unordered_ multimap**
	constructor	unordered_ set	unordered_ multiset	unordered_ map	unordered_ multimap
	destructor	~unordered_	~unordered_	~unordered_	~unordered_

		set	multiset	map	multimap
	assignment	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin
	end	end	end	end	end
	rbegin				
	rend				
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin
	cend	cend	cend	cend	cend
	crbegin				
	crend				
capacity	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size
	empty				

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Assosiativ konteynerlarga qaysi sinflar kiradi?
2. C++ barcha assotsiativ konteynerlar qanday amallarni qo‘llab-quvvatlaydi?
3. Key1 va key2 kalitlar o‘zaro teng hisoblanadi, agar taqqoslanadigan obyektlar uchun qaysi funksiya chin qiymat qaytarsa?
4. Qanday konteynerlar uchun aniqlanadigan tip va kalit tipi bir xil bo‘ladi?
5. Assosiativ konteynerlarning iteratorlari necha tomonlama iteratorlar sinfiga kiradi?
6. Kalit asosida qidirishda berilgan kichik bo‘lmagan birinchi elementni qanday funksiya va birinchi katta elementni qanday funksiya yordamida amalga oshiriladi?
7. Set to‘plamda qiymatni o‘zgartirish uchun avval nima funksiyani bajarib, so‘ng yangi qiymatni yoki o‘zgartirilgan varianti qo‘shiladi?
8. Xotirani ajratish va bo‘shatish operatorlari set sinfi uchun sanab bering?
9. Ishorali butun tip, aniq tipni belgilash realizatsiya qilishga bog‘liq va Allocatorda aniqlanadi. Gap qaysi funksiya haqida.
10. To‘plam - assosiativ konteynet bo‘lib, teng qiymatli kalitlarni saqlaydi (mumkin qadar bir kalit qiymatli elmentlar to‘plamini saqlaydi) va kalit orqali tez qidirish imkonini beradi. Gap qaysi to‘plam haqida bormoqda.

11. *set* va *multiset* sinflarining obyektlarining tipi kalit bilan yonma-yon bitta shablonli parametr olishi mumikn. Bu shablon qaysi funksiya?
12. [=] (operator=) operatorni nima uchun ishlatiladi?
13. To‘plam va vektorning massivlarini tasodifiy sonlar bilan to‘ldiradigan va belgilangan qiymatni qidiradigan dastur tuzing.
14. Pair qanday sinf va nima uchun kerak?
15. p.first – havola bo‘yicha juftlikning nechanchi elementiga murojaat?
16. Lug‘atda kalitlari unikal (takrorlanmaydigan) va qaysi sinfda bir nechta nusxadagi kalitlari bo‘ladi.
17. Oddiy map konreyner konstruktorlari: map<Key, T, comp> ar; va map<Key, T, comp> ar(Comp); ni ishlash tartibini tushuntirib bering.
18. Ro‘yxatni initsializatsiya qilishda har bir juftlik alohida qanday qavs ichiga olinishi kerak.
19. Lug‘atlarda elementlarga kirishning yana ikkita usuli mavjud (iteratorlar bilan ishlashdan tashqari). Birinchi va ikkinchi usularni tushuntirib bering.
20. Indekslash funksiyasini assotsiativ konteyner sinflarning qaysi sinflariga foydalanish mumkin.
21. at() funksiyasi istisno qaytargani uchun try - catch funksiyasi bilan boshqarish mumkinmi?
22. operator[] noqulay va samarasiz deb hisoblasangiz, yana unga ikkita alternativ variantlar bor. Bu variantlarni ayting.
23. Qaysi to‘plamda qidirish faqat birinchi maydon, ya‘ni kalit maydon bilan amalga oshiriladi.
24. Qanday konteynerlar xesh – jadval asosida qurilgan va xesh funksiyalarga va tenglik amaliga tayanadi.
25. Ayrim vazifalarda saralash uchun vaqt ko‘p ajratilsa avtomatik saralashni muhim kamchilik deb hisoblash mumkin. Bunda dasturchi qanday yo‘l tutadi.
26. Tartiblanmagan konteynerlarning xesh jadvallarida nima bor va ular cheksiz ko‘p elementlarni o‘z ichiga olishi mumkin.
27. Tartiblanmagan konteynerlarning tiplari uchun shablon parametrlarini sanab bering.
28. Tartiblanmagan konteynerlarda agar equal taqqoslash funksiyasi yozilmagan bo‘lsa, joriy holat bo‘yicha qaysi funksiya ishlatiladi.

29. Xeshlash uslubi nima uchun kerak.
30. rehash(size_type count) – konteyner uchun qayta xeshlash o‘tkazadi, ya‘ni yacheykalarni sonini qaysi shartda bo‘lishi uchun.
31. Segment elementlari bilan bevosita ishlash uchun qanday usullardan foydalaniladi.
32. Tartiblangan konteynerlar yoki tartiblanmagan konteynerlar orasidagi tanlov nimaga bog‘liq bo‘ladi.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I

	Assosiativ konteynerlarga (set, multiset) oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`// Created by MBBahodir`
`#include "stdafx.h"`
`#include <iostream>`
`#include <set>`
`using namespace std;`
 	
`void myPrint(set<int> &myset){`
 	
`  for (set<int>::iterator it=myset.begin(); it!=myset.end(); ++it)`
`  cout << ' ' << *it;`
`  cout << '\n';`
`}`
`void myPrintmulti(multiset<int> &myset){`
	1. Dasturda nechta global funksiya yaratilgan.
	_____
	_____
	_____
	_____
	
	2. Dasturda nechta set to‘plam mavjud.
	_____
	_____
	_____
	_____
	
	3. Dasturda nechta multiset to‘plam mavjud.
	_____
	_____
	_____
	_____
	_____
	_____

```
    for
(multiset<int>::iterator
it=myset.begin();
it!=myset.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';
}
void mySizeSets(set<int>
&myset){
    cout << int
(myset.size()) << '\n';
}

int main ()
{
  set<int> myset;
  set<int>::iterator it;

  int myints[]={
12,82,37,64,15,15 };

   set<int> first
(myints,myints+6);
   multiset<int> mfirst
(myints,myints+6);
   set<int> second;
   second = first;

   myPrint(first);
   myPrintmulti(mfirst);

   mySizeSets(second);
   mySizeSets(first);

  for (int i=1; i<=5;
i++) myset.insert(i*10);

  myPrint(myset);
```

4. Set va multiset to‘plamlarning farqi dasturning qaysi fragmentida ko‘rsatilgan .

_____
_____
_____

5. Nima uchun oxirgi myPrint(myset); funksiya ishlamaydi.

_____
_____
_____

6. reverse_iterator ni o‘rniga iterator ni o‘rnatsa, nima bo‘ladi. Xato bo‘ladi. Xatoni tuzatish uchun nima qilish kerak.

_____
_____
_____

7. myset.swap(second); ni o‘rniga second.swap(first); yozilsa nima hodisa sodir bo‘ladi.

_____
_____
_____

8. Set va multiset ning operator= dasturning qaysi qismlarida foydalanilgan va nima uchun.

_____
_____
_____

```
  it=myset.find(20);
  myset.erase (it);
  myset.erase
(myset.find(40));

  myset.swap(second);

  for (
set<int>::reverse_iterat
or rit=myset.rbegin();
rit != myset.rend();
++rit)
     std::cout << ' ' <<
*rit;
  cout << '\n';

  while (!myset.empty())
  {
     cout << ' ' <<
*myset.begin();

myset.erase(myset.begin(
));
  }

  myPrint(myset);

  system("pause");
  return 0;
}
```

9. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
10. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

IKKINCHI ASSISMENT TOPSHIRIG‘I	
☛	Assosiativ konteynerlarga (map, multimap) oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish

	orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`// Created by MBBahodir`
`#include "stdafx.h"`
 	
`#include <iostream>`
`#include <map>`
`#include <string>`
 	
`using namespace std;`
`void myPrint(map<char,int> &mymap){`
`  for (map<char,int>::iterator it=mymap.begin(); it!=mymap.end(); ++it)`
`  cout << it->first << " => " << it->second << endl;`
`  cout << endl;`
`}`
`void myPrintmulti(multimap<string,int> &myMultimap){`
`  for (auto& x : myMultimap)`
`  cout << " [" << x.first << ':' << x.second << ']';`
`  cout << endl;`
	1. Dasturda nechta global funksiya yaratilgan.
	_____
	_____
	_____
	_____
	_____
	
	2. Dasturda nechta map va multimap to‘plam mavjud.
	_____
	_____
	_____
	_____
	_____
	_____
	
	3. map va multimap to‘plamlarning farqi dasturning qaysi fragmentida ko‘rsatilgan .
	_____
	_____
	_____
	
	4. Qachon oxirgi myPrint(mymap); funksiya ishlamaydi.
	_____
	_____
	_____

```
}
int main ()
{
  map<char,int> mymap,
mymap_one;

map<char,int>::iterator
it;

pair<map<char,int>::ite
rator,map<char,int>::it
erator> ret;

  mymap['a']=10;
mymap_one['x'] = 100;
  mymap['b']=20;
mymap_one['y'] = 200;
  mymap['c']=30;
mymap_one['z'] = 300;


multimap<std::string,in
t> mymultimap;


mymultimap.emplace("Ali
",1.50);

mymultimap.emplace("Val
i",2.71);

mymultimap.emplace("Sol
i",1.40);

mymultimap.emplace("Sod
iq",3.14);

  it = mymap.find('b');
```

5. mymultimap ni o‘rniga map ni o‘rnatsa, nima bo‘ladi. Xato bo‘ladi. Xatoni tuzatish uchun nima qilish kerak.

_____
_____
_____

6. `mymap.swap(mymap_one)` ni o‘rniga `mymap_one.swap(mymap)` yozilsa nima hodisa sodir bo‘ladi.

_____
_____
_____

7. map va multimap ning operator= dasturning qaysi qismlarida foydalanilgan va nima uchun.

_____
_____
_____
_____
_____

8. Dasturdagi izohga olingan dastur fragmenti olib tashlansa, dasturda qanday o‘zgarishlar bo‘ladi va natijadachi.

_____
_____
_____
_____
_____

```
  if (it !=
mymap.end())
mymap.erase (it);

  myPrint(mymap);
  myPrint(mymap_one);

/* while
(!mymap.empty())
  {
    std::cout <<
mymap.begin()->first <<
" => " <<
mymap.begin()->second
<< '\n';

mymap.erase(mymap.begin
());
  }
*/


mymap.swap(mymap_one);

  myPrint(mymap_one);
  myPrint(mymap);


  ret =
mymap.equal_range('b');

  cout << ret.first-
>first << " => " <<
ret.first->second <<
endl;
  cout << ret.second-
>first << " => " <<
ret.second->second <<
```

endl;
 	
myPrintmulti(mymultimap
);
 	
  system("pause");
  return 0;
}	

9. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____ 10. Shu dasturning analogini yaratish sizga mustaqil vazifadir.	

UCHINCHI ASSISMENT TOPSHIRIG‘I	
🎓	Tartiblanmagan assosiativ konteynerlarga (unordered_set, unordered_multiset) oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.
dastur	topshiriqlar
// Created by MBBahodir #include "stdafx.h" #include <iostream> #include <string> #include <unordered_set> using namespace std; template<class T>	1. Dasturda nechta global funksiya yaratilgan. _____ _____ _____ _____ _____

```
T cmerge (T a, T b) {
  T t(a);
t.insert(b.begin(),b.end());
return t;
}
void
myPrint(unordered_set<string>
&myset){
    for (const string& x:
myset) cout << " " << x;
   cout <<  endl;
}
void
myPrintmultiset(unordered_mult
iset<string> &mymultiset){
    for (const std::string& x:
mymultiset) std::cout << " "
<< x;
  std::cout << std::endl;
}
int main ()
{
  unordered_set<string> myset,
myset_str;
  unordered_set<int>
myset_one;
  unordered_multiset< string>
first, second, third;

  myset.emplace ("Laseti");
myset_one.insert(1);
  myset.emplace ("Tico");
myset_one.insert(15);
  myset.emplace ("Damas");
myset_one.insert(45);

  myPrint(myset);
```

2. Dasturda nechta unordered_set to‘plam mavjud.

_____
_____
_____
_____
_____
_____
_____

3. Dasturda nechta unordered_multiset to‘plam mavjud.

_____
_____
_____
_____
_____
_____
_____

4. Dasturda myPrint(myset_one) funksiyasi yaaqirilganda natija qanday bo‘ladi va nima sababdan.

_____
_____
_____

5. To‘plamdan berilgan qiymatni izlash dastur fragmentini aniqlang va myset_one uchun ham izlash funksiyasini yarating.

_____
_____
_____
_____
_____
_____

```
  for (const int& x:
myset_one) cout << " " << x;
   cout <<  endl;

  string input;
  cout << "Mashina nomini
kirit: ";
  getline (cin,input);


unordered_set<::string>::const
_iterator got =
myset.find(input);

  if ( got == myset.end() )
    cout << "to‘plamda yo‘q";
  else
    cout << *got << " nomli
mashina bor";
  cout << endl;

   myset.rehash(12);

  myset.insert("mashina");
  myset.insert("uy");
  myset.insert("daraxt");
  myset.insert("eshik");
  myset.insert("choynak");

  cout << "joriy bucket_count:
" << myset.bucket_count() <<
endl;

  myset_str.swap(myset);

  cout << "first =>";
  myPrint(myset);
```

_____
_____
_____
_____
_____
_____

6. myset.rehash(12) nima vazifani bajaradi.

_____
_____
_____

7. `myset_str.swap(myset);` ni o‘rniga `myset.swap(myset_str);` yozilsa nima hodisa sodir bo‘ladi.

_____
_____
_____

8. operator= dasturning qaysi qismlarida foydalanilgan va nima uchun.

_____
_____
_____
_____
_____
_____

9. myPrintmultiset(second); funksiyasi ekranga nima natijani chiqaradi.

_____
_____
_____
_____
_____
_____

```
  cout << "second =>";
  myPrint(myset_str);

  first.insert("www.uz");
first.insert("tuit.uz");
first.insert("edu.uz");
  second.insert("www.uz");
second.insert("uzedu.uz");
second.insert("edu.uz");
  third = cmerge (first,
second);
  first = third;

  std::cout << "first =>";
  myPrintmultiset(first);

  second.clear();
  second.insert("bed");
  second.insert("bed");
  second.insert("wardrobe");
  second.insert("nightstand");

  cout << "second =>";
  myPrintmultiset(second);

 string mystring = "red";

 second.insert (mystring);
 second.insert
(mystring+"dish");
  cout << "second =>";
  myPrintmultiset(second);
     system("pause");
  return 0;
}
```

10. third = cmerge (first, second); funksiyasi nima vazifani bajaradi va natijasi qanday.

_____
_____
_____
_____
_____
_____

11. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi.

_____

12. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

TO‘RTINCHI ASSISMENT TOPSHIRIG‘I	
🎓	Tartiblanmagan assosiativ konteynerlarga (unotdered_map, unordered_multimap) oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`// Created by MBBahodir`
`#include "stdafx.h"`
`#include <iostream>`
`#include <string>`
`#include <unordered_map>`
 	
`using namespace std;`
 	
`template<class T>`
`T cmerge (T a, T b) {`
`  T t(a);`
`t.insert(b.begin(),b.end()); return t;`
`}`
 	
`void myPrint(unordered_map<int,string> &mymap){`
`  for (auto& x: mymap) cout << "[" << x.first << "," << x.second << "]" ;`
`  cout <<  endl;`
`}`
`void myPrintmultimap(unordered_multimap<int,string> &mymultimap){`
	1.Dasturda nechta global funksiya yaratilgan.
	_____
	_____
	_____
	_____
	
	2. Dasturda nechta unordered_map to‘plam mavjud.
	_____
	_____
	_____
	_____
	_____
	_____
	
	3. Dasturda nechta unordered_multimap to‘plam mavjud.
	_____
	_____
	_____
	_____
	_____
	_____
	
	4. Dasturda myPrint(mymap_one) funksiyasi chaqirilganda natija qanday bo‘ladi va nima sababdan.
	_____
	_____
	_____

```
    for (auto& x:
mymultimap) cout << "["
<< x.first << "," <<
x.second <<"]" ;
   cout <<  endl;
}
int main ()
{


unordered_map<int,strin
g> mymap, mymap_str;

unordered_map<int,int>
mymap_one;

unordered_multimap<int,
string> first, second,
third;

  mymap.emplace
(80686,"Laseti");
mymap_one.insert(make_p
air<int,int>(1,2));
  mymap.emplace
(10010,"Tico");
mymap_one.insert(make_p
air<int,int>(15,16));
  mymap.emplace
(70707,"Damas");
mymap_one.insert(make_p
air<int,int>(45,46));

  myPrint(mymap);

  for (auto x:
```

5. To‘plamdan berilgan qiymatni izlash dastur fragmentini aniqlang va mymap_one uchun ham izlash funksiyasini yarating.

_____
_____
_____
_____
_____
_____
_____
_____
_____

6. mymap.rehash(20) nima vazifani bajaradi.

_____
_____
_____

7. `mymap_str.swap(mymap);` ni o‘rniga `mymap.swap(mymap_str);` yozilsa nima hodisa sodir bo‘ladi.

_____
_____
_____

8. operator= dasturning qaysi qismlarida foydalanilgan va nima uchun.

_____
_____
_____
_____
_____

```
mymap_one) cout << "["
<< x.first << "," <<
x.second <<"]" ;
   cout <<  endl;

  int input;
  cout << "Mashina
raqamini kirit: ";
  cin >> input;
```

9. myPrintmultimap(second); funksiyasi ekranga nima natijani chiqaradi.

_____
_____
_____
_____
_____
_____
_____
_____

```
unordered_map<int,strin
g>::const_iterator got
= mymap.find(input);

  if ( got ==
mymap.end() )
    cout << "to‘plamda
yo‘q";
  else
      cout << "[" <<
got->first << "," <<
got->second <<"]" << "
raqamli mashina bor";
  cout << endl;

  mymap.rehash(20);

  mymap[1] = "mashina";
  mymap[10] = "uy";
  mymap[15] = "daraxt";
  mymap[20] = "eshik";
  mymap[25] =
"choynak";

  cout << "joriy
bucket_count: " <<
mymap.bucket_count() <<
```

10. third = cmerge (first, second); funksiyasi nima vazifani bajaradi va natijasi qanday.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

```
endl;


mymap_str.swap(mymap);

  cout << "first =>";
  myPrint(mymap);

  cout << "second =>";
  myPrint(mymap_str);

  pair<int,string>
mypair (1,"www.uz");
  pair<int,string>
mypair_one
(2,"tuit.uz");

  first.insert
(mypair);

second.insert(mypair_on
e);
  first.insert
(pair<int,string>(50,"G
OOG"));

second.insert(pair<int,
string>(501,"GGOD"));
  first.insert
(second.begin(),
second.end());

  third = cmerge
(first, second);
  first = third;

  cout << "first =>";
```

```
myPrintmultimap(first);

  second.clear();

second.insert(pair<int,
string>(80,"BUXORO"));

second.insert(pair<int,
string>(01,"TOSHKENT"))
;

second.insert(pair<int,
string>(70,"QARSHI"));

second.insert(make_pair
(75,"TERMIZ"));

  cout << "second =>";

myPrintmultimap(second)
;

  system("pause");
  return 0;
}
```

11. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi.
    _____
12. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

1.4.Konteynerlarning adapterlari.

📖 Adapterdarning muhim tushunchalari asosida stek, navbat, ustivor navbat, ikki tomonlama navbat sinf obyekttlari va ularning xususiyatlari, funksiyalari, dasturlashda o‘zlarini tutishlari, konteynerlar bilan ishlashga mo‘ljallangan asosiy algoritmlar va ularning dasturlashdagi ahamiyati, funktor funksiyalarni yaratish va foydalanish uslublari, amallari, talablari, vazifalari va usullari keltirilgan bo‘lib, dasturchining adapterlar, algoritmlar va funktor funksiyalar bilan ishlashdagi tanlash imkoniyati bo‘yicha tajriba xulosalari, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni

mustahkamlash uchun 35 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 5 ta assisment topshirig‘i va har assismentda 10(9,7) ta topshiriq, jami 46ta topshiriq berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

Kalit so‘zlar. Konteyner, adapter, shablon, sinf, moslashtirish, stack (stek), queue (navbat), priority_queue (ustuvor bilan navbat), deque (ikki tomonlama navbat), ma’lumotlar tuzilmasi, push_back, pop_back, pop_front, LIFO, FILO, LIFO, for_each(), find(), funksional obyektlar, Operator(), Standart ML va funktor, Haskell va funktor.

☑Bilish shart bo‘lgan tushunchalar. Tip tushunchasi, sinf va sinf obyekti, to‘plam, statik va dinamik massiv, elementga murojaat, funksiya va ko‘rsatkich, konteyner, ketma-ket konteynerlar, xesh, tasodifiy, istisno holat, ma’lumotlarni kiritish va chiqarishi, oqim, dasturlashga oid dastlabki tushunchalar va C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

Bilib olasiz. stack (stek), queue (navbat), priority_queue (ustuvor bilan navbat), deque (ikki tomonlama navbat) sinf obyekttlari va ularning xususiyatlari , funksiyalari, dasturlashda o‘zlarini tutishlari (LIFO, FILO, LIFO), konteynerlar bilan ishlashga mo‘ljallangan asosiy algoritmlar (for_each(), find()) va ularning dasturlashdagi ahamiyati, funktor funksiyalarni yaratish va foydalanish uslublari, amallari, talablari, vazifalari va usullarini o‘rganishingiz mumkin.

REJA

1. Stack, queue, priority_queue, deque.
2. Konteynerlar bilan ishlash algoritmlari.
3. Funktorlarning qo‘llanilishi.

KIRISh

Adapterlar standarti shablon kutubxonasining alohida toifasidir (vakilidir). Adapterlar yangi tushunchalar yoki ilovalar emas, balki mavjud kutubxona tushunchalarining o‘ziga xos, tez-tez ishlatiladigan maqsadlar uchun moslashtirishlari. Bu moslashtirish (adaptatsiyalash) ko‘pincha adapterning so‘rovlarnnig asosiy tushunchasiga asoslanib, funksiyalarini cheklash orqali amalga oshiriladi. Kutubxona konteynerlarning adapterlari, iteratorlar va funksiyalarni o‘z ichiga oladi.

Konteynerlarning adapterlari. Konteynerlarning adapterlari asosida adapterlarni hosil qilishning eng oson yo‘li misol orqali tushuntirishdir. Bular stack (stek), queue (navbat), priority_queue

(ustuvor bilan navbat) misol bo‘ladi. Bu misollardan tushunarli bo‘ldiki, adapterlar:

- juda keng va tez-tez ishlatiladigan ma’lumotlar tuzilmalari;
- har qanday realizatsiyani alohida bajarilishi kerak emas. Ularning funksiyalarini ta‘minlash uchun STL standart konteyneri har qanday moslashtirishda tayanch sifatida push_back, pop_back, yoki pop_front amallardan (adapter turiga qarab) foydalanish mumkin;
- adapter uchun tayanch konteynerning to‘plamidan qo‘shimcha amallar chiqarib tashlanishi kerak (vahimalar yaratmaslik uchun, masalan, indekslash amali uchun, konteyner vektor bo‘lsa, stek moslashtirish ishlatiladi);

Konteynerlarning adapterlari uchun sintaktik ta‘riflar misollar orqali qarab chiqish yaxshi natija beradi, shuning uchun misollar bilan tushntirib o‘tamiz. Birinchi navbatda konteynerlarning adapterlari uchun *<stack>, <queue> kabi sinflarni dasturga qo‘shish lozim.*

Stek (stack, ma’lumotlar yig‘indisi): stekda uning elementlariga faqat bir uchidan murojaat qilish mumkinligi bilan xarakterlanadi va stekning yuqori qismi deb ataladi. Bu LIFO (Last In — First Out , birinchi kirgan oxiri chiqadi) tamoyili bo‘yicha faoliyat ko‘rsatuvchi ma’lumotlar to‘plamidir. Stekka oid deyarli barcha funksiyalarini ko‘rsatadigan dastur keltirilgan (4.1-dastur).

4.1-dastur. Stekka oid funksiyalardan foydalanish.

```
// Created by MBBahodir
#include "stdafx.h"

#include <iostream>
#include <vector>
#include <stack>
#include <string>
#include <array>
using namespace std;

int main() {

 stack<int> st, st_one, st_two;
 stack<string,vector<string>> vst;

 vst.emplace("str1");
 vst.emplace("str2");
```

```
 vst.push("eostr");

  for (int i=0; i<5; ++i) st.push(i);

  for (int i = 0; i < 15; i++)
st_one.emplace(i*10);

  st_two = st;
  st.swap(st_one);

  cout << st.size() << " " << st_one.size() << " "
<< st_two.size() << " " << vst.size() << endl;

  while (!vst.empty()) {
    cout << vst.top() << " : stack dagi oʻrni " <<
vst.size() << endl;
    vst.pop();
    }

 int size = st.size();
 while (size--) {
     cout << st.top() << " : stack dagi oʻrni " <<
st.size() << endl;
     st.pop();
 }

 while (st.size() != 0){
     cout << st_two.top() << " : stack dagi oʻrni "
<< st_two.size() << endl;
     st_two.pop();
 }

 while (!st.empty()) {
     cout << st.top() << " : stack dagi oʻrni " <<
st.size() << endl;
       st.pop();
 }
```

```
 cout << st.size() << " " << st_one.size() << " "
<< st_two.size() << " " << vst.size() << endl;

 cout << " stack st " << (st.empty() ? "" : "not ")
<< " empty" << endl;
 cout << " stack vst" << (vst.empty() ? "" : "not
") << " empty" << endl;
 cout << " stack st_one " << (st_one.empty() ? "" :
"not ") << " empty" << endl;
 system("pause");
 return 0;
}
```

4.1-dastur. Output.

```
15 5 5 3
eostr : stack dagi o‘rni 3
str2 : stack dagi o‘rni 2
str1 : stack dagi o‘rni 1
140 : stack dagi o‘rni 15
130 : stack dagi o‘rni 14
120 : stack dagi o‘rni 13
110 : stack dagi o‘rni 12
100 : stack dagi o‘rni 11
90 : stack dagi o‘rni 10
80 : stack dagi o‘rni 9
70 : stack dagi o‘rni 8
60 : stack dagi o‘rni 7
50 : stack dagi o‘rni 6
40 : stack dagi o‘rni 5
30 : stack dagi o‘rni 4
20 : stack dagi o‘rni 3
10 : stack dagi o‘rni 2
0 : stack dagi o‘rni 1
0 5 5 0
 stack st empty
 stack vst empty
 stack st_one not empty
```

4.1 - dasturda keltirilgan stek adapterining funksiyalari bo‘yicha dasturni tahlil qilish orqali ba‘zi xulosalar keltiramiz:

- Stack<string> elementlari satr bo‘lgan o‘zgaruvchini e‘lon qiladi. String obyektlar o‘zlari STL konteynerlar ekanligiga e‘tibor bering. Shunday qilib, stek konteynerlarning har qanday elementlarini o‘z ichiga olgan konteynerlarni olishi mumkin (bu boshqa STL konteynerlari uchun xosdir).

- stack<string> e‘lon qilishni misollarning ko‘pchiligida stekka tavsif sifatida ko‘rish mumkin. Ko‘pchilik dasturchi buni boshqa bo‘lishi bilmasligi va tassafur qilmasligi mumkin. Lekin bir xil ta‘rif mavjud stack<string, vector<string>> satrlari bir steki, bunda tayanch sinf vector<string> tanlab olingan. Masalan, *vector*, *list* va *deque* yoki hatto dasturchining konteyner sinfi tayanch sinflar sifatida foydalanish. Ba‘zan so‘rashadi, nima uchun stack< vektor<string>> yozish mumkin emas (birinchi stringni o‘chirgan holda)? Chunki stack konstruktorini amalga oshirishda bu yo‘l belgilangan va bu juda mumkin. Ammo, butunlay boshqa turdagi tavsif bo‘ladi: string vektorlari to‘plami stack< vektor<string>> umuman boshqa stek bo‘ladi (konteynerlarning tarkibiy joylashtirish haqidagi eslatma).

- Juda ko‘p holarda stekni vektor ko‘rinishida initsializatsiya qilinadi, ammo bu faqat C++11 variantida mavjud. 4.1-dasturda emplace() funksiyasidan stekka element qo‘shish uchun foydalanilgan.

- Keyingi fragmentlarda, deyarli barcha amallarni (usullarini) ko‘rish mumkin: stack:push() – stekka bitta element qo‘shish, top() – stekning eng yuqorida turgan elementiga ko‘rsatkich olish, pop() – yuqori elementini chiqarib tashlash, size() – stekning joriy hajmi, empty() – stekning bo‘shligini tekshirish, swap() – ikki stekning almashtirish, operatop= stekni qiymat qilib boshqa bir tekka berish (o‘zlashtirish).

- Dasturdan ko‘rish mumkinki, stek adapteri tayanch konteynerga xos usullarni yo‘qotdi (at(), operator[] va hokazo.), lekin push(), pop() funksiyalarni qayta aniqlash orqali orttirilgan hisoblanadi.

Stekni tushinib olib, o‘xshash funksiyalarni navbatga joriy qilish qiyin masala emas. Stekdan farqli o‘laroq, navbat FIFO (First In — First Out birinchi kirgan birinchi chiqadi) tamoyili bo‘yicha faoliyat yuritadigan ma’lumotlar to‘plamidir. (Bu usulni bir uchiga oqib kirish, keyin boshqa uchidan oqib chiqadigan quvurga o‘xshatish mumkin).

4.1- dasturda til semantikasi talabi bo‘yicha tahrirlarni qilib navbat uchun deyarli o‘zgarishsiz qoldiramiz va 4.2 dasturni olamiz.

4.2-dastur. Navbat sinfining funksiyalari.

```
// Created by MBBahodir
#include "stdafx.h"

#include <iostream>
#include <list>
#include <queue>
#include <string>
#include <array>
using namespace std;

int main() {

    queue<int> st, st_one, st_two;
    queue<string,list<string>> vst;

 vst.emplace("str1");
 vst.emplace("str2");
 vst.push("eostr");

  for (int i=0; i<5; ++i) st.push(i);

  for (int i = 0; i < 15; i++)
st_one.emplace(i*10);

  st_two = st;
  st.swap(st_one);

  cout << st.size() << " " << st_one.size() << " "
<< st_two.size() << " " << vst.size() << endl;

  while (!vst.empty()) {
      cout << vst.front() << " : stack dagi o‘rni "
<< vst.size() << endl;
    vst.pop();
    }
```

```
 int size = st.size();
 while (size--) {
     cout << st.front() << " : stack dagi o‘rni "
<< st.size() << endl;
     st.pop();
 }

 while (st.size() != 0){
     cout << st_two.front() << " : stack dagi o‘rni
" << st_two.size() << endl;
     st_two.pop();
 }

 while (!st.empty()) {
     cout << st.front() << " : stack dagi o‘rni "
<< st.size() << endl;
       st.pop();
 }

 cout << st.size() << " " << st_one.size() << " "
<< st_two.size() << " " << vst.size() << endl;

 cout << " stack st " << (st.empty() ? "" : "not ")
<< " empty" << endl;
 cout << " stack vst" << (vst.empty() ? "" : "not
") << " empty" << endl;
 cout << " stack st_one " << (st_one.empty() ? "" :
"not ") << " empty" << endl;
 system("pause");
 return 0;
}
```

4.2-dastur.Output

```
15 5 5 3
str1 : stack dagi o‘rni 3
str2 : stack dagi o‘rni 2
eostr : stack dagi o‘rni 1
```

```
0 : stack dagi o‘rni 15
10 : stack dagi o‘rni 14
20 : stack dagi o‘rni 13
30 : stack dagi o‘rni 12
40 : stack dagi o‘rni 11
50 : stack dagi o‘rni 10
60 : stack dagi o‘rni 9
70 : stack dagi o‘rni 8
80 : stack dagi o‘rni 7
90 : stack dagi o‘rni 6
100 : stack dagi o‘rni 5
110 : stack dagi o‘rni 4
120 : stack dagi o‘rni 3
130 : stack dagi o‘rni 2
140 : stack dagi o‘rni 1
0 5 5 0
 stack st empty
 stack vst empty
 stack st_one not empty
```

4.1-dasturda o‘zgartirish talab qilinganlarini o‘zgartirib, quyidagi xulosalarga olib keladi:

- navbatda pop_front() usuli vektor konteyner ustiga qurib bo‘lmaydi, , lekin uni ro‘yxat konteyner uchun qurish mumkin.

- Deque sinfi uchun *front(), push_back(), pop_front*()), usullari har qanday konteynerlarda ishlatiladi (faqat pop_front() usuli borlarida).

- shuning uchun navbat uchun yasaladigan obyektda, ya‘ni deque sinfida top() usuli o‘rniga front() usuli ishlatiladi.

Ma’lum bir tartibda qayta ishlanishi kerak ma’lumotlarni uchun umumiy ma’lumotlar tuzilishiga asoslangan to‘plami bu – stek va navbatdir (stack, deque). Masalan, biror funksiya o‘z navbatida uchinchi funksiyani chaqiradigan boshqa funksiyani chaqirsa, uchinchi funksiya birinchi funksiyani emas, ikkinchi funksiyaga qaytishi muhimdir.

Bu ma’lumotlarni qayta ishlash tartibini amalga oshirish uchun dasturchi o‘zining navbat funksiyasi tashkil qilish lozim. Stekka qo‘shilgan oxirgi qiymat birinchi bajariladi va aksincha, stekka qo‘shilgan birinchi qiymat oxirgi bajariladi. Shunday qilib, ma’lumotlar tuzilmasining o‘zi chaqirishlarning to‘g‘ri belgilangan tartibda berilishini ta‘minlaydi.

Konseptual, ma’lumotlar tuzilishi - stek juda oddiy: u ma’lumotlarni kiritish yoki chiqarish uchun ma’lum bir tartibni belgilaydi Har safar bir element qo‘shiladi, bu stekka yuqori element bo‘lib tugaydi. Stekdan olib tashlanadigan yagona element stekning yuqori qismida joylashgan elementdir. Shunday qilib, stekka “birinchi kirish, oxirgi chiqish — FILO” yoki “oxiri kirish, birinchi chiqish — LIFO” deb ham aytiladi. Demak, Stekka qo‘shilgan birinchi element undan oxirgi chiqariladi.

Xo‘sh, bu nima? Nima uchun stek kerak? Yuqorida aytib o‘tganimizdek, funksiyalarni (qiymatlarni) chaqirishni tashkil qilish uchun qulay yo‘lidir. Aslida, stek chaqirish (call stack) tez-tez ishlaydigan yoki boshqa funksiyalarning qaytish qiymatini kutayotgan funksiyalar ro‘yxatiga murojaat qilish uchun ishlatiladigan atamadir.

Shuningdek, stek informatika faning fundametntal tilining bir qismidir. Agar birinchi kelib, oxirish ketish tamoyili asosida fikrlasangiz, unda stek terminini ishlatish lozim.

Bundan tashqari, bunday (stek kabi) navbatlar ko‘p jarayonlarda ishtirok etadi, nazariy informatikadan tortib, bunday push-down funksiyalari va juda ham ko‘p bag usullarida.

Stek assosiativ usullarga ega:

Push – stekka element qo‘shish..

Pop – stekdan element o‘chirish

Top – elementni ko‘rish.

LIFO - stek xarakati.

FILO - stek xarakati (ekvivalent LIFO ga).

Stack (stek) ma’lumotlar tuzilmasini amalga kutubxona yaratish masalani qaraymiz. Umumlashgan dasturlash usuliga mos keladigan umumiy stack, ya‘ni Stack sinfi uchun shablon yaratish degani. Agar shablonlar bilan tanish bo‘lmasangiz, funksiya shablonlariga oid nazariy ma’lumotlarga qarang, shablonlar bilan ishlash mexanizmi batafsil bilish lozim. Agar shablonlar bilan tanish, lekin sinf shablonlari bilan ishlashni unutgan bo‘lsangiz, sinf shablonlari haqida nazariy materiallarni o‘qib o‘rganing.

Bu stack shablonlari bilan amalga oshirildigan kutubxonani deyarli har qanday ma’lumotlar tipi uchun foydalanish mumkin. Bundan tashqari, stack hajmi dastur ijrosi davomida jadal belgilanadi. Stackga qo‘shimcha funksiya ham qo‘shildi peek(), bu stackning yuqori qismidan n- elementini qaytaradi.

4.3-dastur. Stack shablon sinfi

```
#ifndef STACK_H
#define STACK_H
// Created by MBBahodir
#include <cassert>
#include <iostream>

#include <iomanip>

template <typename T>
class Stack{
private:
    T *stackPtr;                        // stackka
ko‘satkich
    const int size;                     // stackdagi
maksimal elementlar soni
    int top;                            // stackning
joriy elementi
public:
    Stack(int = 10);                    // stackning
joriy holatda elementlar soni 10 ta
    Stack(const Stack<T> &);            // nusxalsh
konstruktori
    ~Stack();                           // destruktor

    inline void push(const T & );       // elementni
stackning yuqori qismiga joylashtirish
    inline T pop();                     // stack
ichidan eng yuqoridagi elementni o‘chirish
    inline void printStack();           // stackni
ekranga chiqarish
    inline const T &Peek(int ) const; // stackning
yuqorisidagidam keyingi n-element
    inline int getStackSize() const;  // stack
o‘lchamini olish
    inline T *getPtr() const;           // stackka
ko‘rsatkich olish
    inline int getTop() const;          // joriy
elementni olish
```

```
};

// Stack sinfi shablon funksiyalarini realizatsiyz
qilish

// konstruktor Steka
template <typename T>
Stack<T>::Stack(int maxSize): size(maxSize) //
o‘zgarmaslarni initsalizatsiyalash
{
    stackPtr = new T[size]; // xotira ajratish
    top = 0; // joriy elementno 0 bilan toldirish;
}

// nusxalsh konstruktori
template <typename T>
Stack<T>::Stack(const Stack<T> & otherStack) :
    size(otherStack.getStackSize()) //
o‘zgarmaslarni initsalizatsiyalash
{
    stackPtr = new T[size]; // yangi xotira
ajratish
    top = otherStack.getTop();

    for(int ix = 0; ix < top; ix++)
        stackPtr[ix] = otherStack.getPtr()[ix];
}

// destruktor
template <typename T>
Stack<T>::~Stack()
{
    delete [] stackPtr; // ochirish
}

// element qo‘shish
template <typename T>
inline void Stack<T>::push(const T &value)
```

```
{
    // o‘lchamni o‘zgartirish
    assert(top < size); // joriy element soni
o‘lchamdan kichik bo‘lishi kerak

    stackPtr[top++] = value; // elementni
joylashtirish
}

// elementni o‘chirish
template <typename T>
inline T Stack<T>::pop()
{
    // o‘lchamni tekshirish
    assert(top > 0); // joriy element 0 katta
bo‘lishi kerak

    stackPtr[--top]; //  elementni o‘chirish
return 0;
}

// yuqori elementdan n-elementni qaytaradi
template <class T>
inline const T &Stack<T>::Peek(int nom) const
{
  //
  assert(nom <= top);

  return stackPtr[top - nom]; // n-elementni
qaytarish
}

// ekranga chiqarish
template <typename T>
inline void Stack<T>::printStack()
{
    for (int ix = top - 1; ix >= 0; ix--)
        cout << "|" << setw(4) << stackPtr[ix] <<
```

```
endl;
}

// o‘lchamni qaytarish
template <typename T>
inline int Stack<T>::getStackSize() const
{
    return size;
}

// ko‘rsatkich qaytarish (nusxalash konstruktori
uchun)
template <typename T>
inline T *Stack<T>::getPtr() const
{
    return stackPtr;
}

// o‘lchamni qaytarish
template <typename T>
inline int Stack<T>::getTop() const
{
    return top;
}

#endif // STACK_H
```

Stack shablon sinfi *.h (header) faylda yaratildi. Shablon sinf interfeysi va amalga oshirish ham bir xil faylda bo‘lishi kerak, bo‘lmasa, shunga o‘xshash mazmun bilan xato berishi mumkin.

```
error undefined reference to "method of a class template»
```

Ya‘ni, xatolik, shablon sinf usullari havolalarda aniqsizlik.

Shablon sinf interfeysining fragmentlarida 9 -dan 28gacha yangilandi. Sinfning barcha usullari izohlarni o‘z ichiga oladi va mening fikrimcha, ularning funkiyalarini alohida ta‘riflash mantiqiy to‘g‘ri emas. Stack shablon sinfning barcha usullari inline funksiyalari sifatida e‘lon qilinadi. Bu sinf ishini tezlashtirish maqsadida amalga oshiriladi. Sinfning ajralmas funksiyalari tashqi funsiyalarga qaraganda tezroq ishlaydi.

Shablon interfeysidan so‘ng, stack sinf usullari amalga oshiriladi, kutubxona fragmentlarida 32-117gacha yangilandi. Agar stack sinf,

shablonlar va sinflar usullarini amalga oshirish haqida bilsangiz shablon sinfni yaratishda murakkab hech narsa yo‘q. Sinfda ikkita konstrkutor bor, birinchisi fragmentning 32-33 qatorlarida - bu standart konstruktor hisoblanadi. Fragmentning 41-50 qatordagi nusxa konstruktor hisoblanadi. Bir obyektni boshqasiga nusxalash uchun kerak. Peek usuli (funksiyasi), fragmentning 80-88 fatorlarida stack elementlarni ko‘rish imkonini beradi. Faqat element raqamini kiritishingiz kerak, hisoblash stack yuqorisidagi elementdan boshlanadi va keraklisini qaytaradi. Boshqa funksiyalari xizmatchi funksiyalar hisoblanadi, ular tashqi sinfdna foydalanish uchun mo‘ljallangan, ya‘ni prinstack() funksiyasidan tashqari, bu funksiya stack elementlarini ekranga chiqaradi.

Endi stack shablon sinf uchun dastur tuzaylik va har doimgidek, stack sinf shablonni sinov bo‘ladi hamda asosiy funksiyalaridan foydalanishni ko‘ramiz.

4.4-dastur. Shablon sinfdan foydalanish.

```
// Created by MBBahodir
include "stdafx.h"

#include <iostream>

using namespace std;

#include "stack.h"

int main()
{
    Stack<char> stackSymbol(5);
    int ct = 0;
    char ch;

    while (ct++ < 5){
        cin >> ch;
        stackSymbol.push(ch); // element qo‘shish
    }

    cout << endl;

    stackSymbol.printStack(); // ekranga chiqarish
```

```
    cout << "\n\n Element o‘chirish\n";
    stackSymbol.pop();

    stackSymbol.printStack(); // ekranga chiqarish
    Stack<char> newStack(stackSymbol);
    cout << "\n\nNusxalash konstruktori!\n";
    newStack.printStack();
    cout << "2 tartibdagi element: "<<
newStack.Peek(2) << endl;
    system("pause");
    return 0;
}
```

4.4-dastur. Output.

```
b
a
h
o
q

|   q
|   o
|   h
|   a
|   b


 Element o‘chirish
|   o
|   h
|   a
|   b


Nusxalash konstruktori!
|   o
|   h
```

```
|   a
|   b
2 tartibdagi element: h
```

4.4-dasturda Bir stack obyektini yaratilgan, stack hajmi 5ga teng va stack ko‘pi o‘z ichiga olishi mumkin bo‘lgan elementlar soni 5 ga teng. stackni while takrorlanish operatori bilan to‘ldiriladi. Ekranda stack elementlarini chiqariladi, keyin stackdan bir elementi o‘chiriladi va yana ekranda stack elementlarini chiqariladi. Natija asosida u o‘zgardi, aniq bir element o‘chirildi.

Shuningdek, dasturda nusxa konstruktor ishlatiladi. Peek() funksiyasi stackning ikkinchi elementini qaytardi.

Stack shablon sinf yaxshi yaratildi va to‘g‘ri ishlamoqda. uni, masalan, int ma’lumotlar turida sinab ko‘rcangiz bo‘ladi. Ishonchim komilki, hamma narsa to‘g‘ri ishlaydi.

Navbat (queue). Navbat ma’lumotlar tuzilmasi bo‘lib (yuqorida aytib o‘tilganidek) bu LILO tamoyilini qo‘llab quvvatlaydi (last in-last out: oxirgi kirgan oxirgi chiqadi). C++ allaqachon tayyor STL konteyner-navbat (queue) mavjud.

Navbatda, birinchi elment kiritilgan bo‘lsa, u ham birinchi chiqariladi. 4 ta elementni qo‘shsangiz, birinchi qo‘shilgan element birinchi, ikkinchi qo‘shilgan element ikkinchi chiqadi.

Navbat qanday ishlashini tushunish uchun do‘kon navbatini tasavvur qilishingiz mumkin. Siz esa uning o‘rtasida turibsiz, shunda kassaning oldida turibsiz, avval oldingizdagi barcha odamlarga xizmat qilinishi. Lekin oxirgi navbatda o‘tgan kishi uchun, o‘zi tashqari barcha odamlarga xizmat qilish kerak emas.

1	9	8	0	1	2	8

Bu yuqorida keltirilgan ro‘yxatga qarang. Bulardan birortasini chiqarish uchun shu ketma-ketlik asosida chiqarib, keraklisiga to‘xtaladi. Masalan, 0 sonnini chiqarish, uchun avvl 3 ta son chiqariladi so‘ng kerakli son.

Stack shablon sinfida bir peek() (bu indeks bo‘yicha elementni qaytaradi), bu navbat shablonida muayyan elementni ko‘rish mumkin emas.

Agar barcha navbat elementlarga kirish kerak bo‘lsa, lekin, navbatda buni amalga oshirish mumkin. Keyinroq bu qanday amalga oshirishni ko‘rsatib o‘tamiz.

C++da navbat yaratish. Agar C++ da navbat shablonini ishlatmoqchi bo‘lsangiz, avval <queue>kutubxonasini ulashingiz kerak.

Keyin navbatni e‘lon qilish uchun quyidagi sintaktikni ishlatishingiz kerak:

```
queue <type> <name>;
```

Birinchi queue so‘zini yozishimiz kerak.

Type ga navbatni to‘ldirish uchun kerakli tipni ko‘rsatishimiz lozim.

Navbat obyektining nomini ko‘rsatishamiz kerak.

Masalan,

```
queue <int> q;
```

Navbat usullari va funksiyalari:

Navbatda funksiya bu bir xil funksiyadir, lekin u faqat STL konteynerlari bilan ishlaydi. Bu funksiyalarga push(), pop(), front(), back(), empty() funksiyalar kiradi.

push() – navbatga yangi element qo‘shish.

pop() – navbatdan birinchi elementni o‘chirish, ikkinchi element birinchi element o‘rniga suriladi.

front() – navbatdagi birinchi elemeniga murojaat

back() - navbatdagi birinchi elemeniga murojaat

empty() – navbat bo‘sh yoki bo‘shmasligini tekshiradi. Agar bo‘sh bo‘lsa true, aks holda false qiymat qaytaradi.

Bu keltirilgan funksiyalarga doir 4.5-dasturini keltiramiz.

4.5-dastur. Navbat funksiyalaridan foydalanish.

```
// Created by MBBahodir
#include "stdafx.h"
#include <iostream>
#include <queue>
using namespace std;

int main() {
  queue <int> q;

  for (int h = 0; h < 7; h++) {
      q.push(rand()%100);
  }

  cout << "Navbatdigi birinchi element " <<
q.front() << endl;
```

```
  cout << "Navbatdigi oxirgi element " << q.back()
<< endl;
  q.pop();

  cout << "Navbatdigi birinchi element
(o‘chirishdan so‘ng): " << q.front() << endl;

  cout << "Navbat bo‘sh " << (q.empty()?"":"emas")
<< endl;

  system("pause");
  return 0;
}
```

4.5-dastur.Output.

```
Navbatdigi birinchi element 41
Navbatdigi oxirgi element 78
Navbatdigi birinchi element (o‘chirishdan so‘ng):
67
Navbat bo‘sh emas
```

Massiv yordamida navbatlarni yaratish. Yuqorida aytganimizdek, navbatni massiv orqali amalga oshirish mumkin. Odatda, agar dasturchi bunday navbatni yaratsa, massiv navbat deb ataladi. Shuningdek, buni massiv deb ham aytish mumkin, ammo massiv so‘zi C++ da bor. Shuning uchun, navbat shablon deb aytish ham mumkin va bir xil tarzda uni chaqiriladi.

Buning uchun 2 ta qo‘shimcha o‘zgaruvchi yaratish kerak. Birinchisi start bo‘lsin va navbatning birinchi elemenitini bildirsin. Ikkinchisi ends bo‘lsin va navbatning oxirgi elemenitini bildirsin.

queue[ends] - ifodasini ishlatib navbatning oxirgi elementini qiymatini qaytaradi.

queue[start] - ifodasini ishlatib navbatning oxirgi elementini qiymatini qaytaradi.

Navbatdagi birinchi elementni o‘chirish uchun start ni birga kamaytirish kerak xolos, navbatni bo‘shligini tekshirish uchun start=ends mantiqiy shartni tekshirishning o‘zi kifoyadir. Agar mantiqiy shart true qaytarsa navbat bo‘sh, aks holda bo‘sh emas.

4.6-dastur. Massiv yordamida navbatlardan foydalanish.

```
// Created by MBBahodir
#include "stdafx.h"
#include <iostream>
#include <queue>
using namespace std;

int main() {
  //queue <int> q;
  int q[3];
  int start = 0, ends = 0;
  for (int h = 0; h < 3; h++) {
      int a; cin >> a;
        q[start++] = a;
  }

  cout << "Navbatdagi birinchi element " << q[start
- 1] << endl;

  start--;  // bir elementni o‘chirish

  cout << " Navbatdagi birinchi element
(o‘chirilgandan so‘ng) " << q[start - 1] << endl;

  cout << " Navbatdagi oxirgi element " << q[ends]
<< endl;

  cout << "Navbat bo‘sh " << ((start == ends)?"":"
emas ") << endl;

  system("pause");
  return 0;
}
```

4.6-dastur.Output
9 7 5 Navbatdagi birinchi element 5

```
 Navbatdagi birinchi element (o‘chirilgandan so‘ng)
7
 Navbatdagi oxirgi element 9
Navbat bo‘sh  emas
```

Ustuvor navbat (Priority_queue). Ustuvor navbat (Priority_queue) bilan navbat (queue) - oddiy navbatdek, lekin ustuvor navbatga yangi element qo‘shilasa, shunda navbat kamayish tartibida saralanadi. Shu tariqa ustuvor navbatda eng katta element birinchi o‘ringa chiqadi. Ustuvor navbat shablonidan quyida sintaktik orqali foydalanish kerak:

```
priority_queue <type> <name>;
```

Birinchi priority_queue so‘zini yozishimiz kerak.

Type ga navbatni to‘ldirish uchun kerakli tipni ko‘rsatishimiz lozim.

Navbat obyektining nomini ko‘rsatishamiz kerak.

Ustivor navbatda yangi element qo‘shish uchun push() funksiyasi, birinchi o‘rindagi elementiga murojaat qilish uchun top() funksiyasi ishlatiladi. Front() – funksiyasi ishlatilmaydi. Shuningdek, back() – oxirgi elementga murojaat funksiyasi ham ishlatilmaydi.

4.7-dastur. Ustuvor navbatni (Priority_queue) yaratish va usullaridan foydalanish.

```
// Created by MBBahodir
#include "stdafx.h"
#include <iostream>
#include <queue>
using namespace std;

int main() {
    priority_queue <int> priority_q,
priority_q_one;

  for (int h = 0; h < 7; h++) {
      priority_q.push(rand()%100);
  }

  priority_q_one  = priority_q;

  cout << "priority_q navbatdigi birinchi element "
<< priority_q.top() << endl;
```

```
   cout << " priority_q_one navbatdigi birinchi
element " << priority_q_one.top() << endl;
  priority_q.pop();

  cout << "priority_q navbatdigi birinchi element
(o‘chirishdan so‘ng): " << priority_q.top() <<
endl;

  cout << "priority_q navbat bo‘sh " <<
(priority_q.empty()?"":"emas") << endl;

  while (!priority_q_one.empty())
  {
      cout << "priority_q_one navbatdigi element :
" << priority_q_one.top() << endl;
      priority_q_one.pop();
  }
  cout << " priority_q_one navbat bo‘sh " <<
(priority_q_one.empty()?"":"emas") << endl;

  system("pause");
  return 0;
}
```

4.7-dastur. Output

```
priority_q navbatdigi birinchi element 78
 priority_q_one avbatdigi birinchi element 78
priority_q navbatdigi birinchi element
(o‘chirishdan so‘ng): 69
priority_q navbat bo‘sh emas
priority_q_one navbatdigi element : 78
priority_q_one navbatdigi element : 69
priority_q_one navbatdigi element : 67
priority_q_one navbatdigi element : 41
priority_q_one navbatdigi element : 34
priority_q_one navbatdigi element : 24
priority_q_one navbatdigi element : 0
 priority_q_one navbat bo‘sh
```

Navbatlarni yaratishni ikki usulini ko’rib chiqdik. Queue sinf shabloni va massiv asosida. Aslida ko‘proq queue sinf shablonidan foydalangan maqul. Chunki bu massivga nisbattan tez ishlaydi.

Ikki tomonlama navbat (deque, double-ended queue). Deque – shunday ma’lumotlar tuzilmasiga aytiladiki, unda ma’lumotlarni qo‘shish va o‘chirish oldindan va orqadan amalga oshiriladi. Deque ham xuddi queue dek xotirada saqlanadi. Dek (Deque) bu shunday dinamik massivki, u ikki tomondan kengayishi va boshqa navbatlar va to‘plamlarga qaraganda tezroq.

Bundan tashqari, u har qanday holatda o‘zi elementlarni boshida yoki oxirida kiritish uchun ruxsat beradi, lekin o‘zboshimchalik holatda elementlarni tez kiritish imkoniyati saqlab qolinmasligi mumkin. Vektor o‘xshash amallari kam bo‘lishi mumkin.

Dek konteyner bilan tanishar ekanmiz u nimasi bilan yaxshi, boshqa konteynerlardan qanday farq qiladi degan savollar paydo bo‘lishi tabiiy albatta. Dekning umumiy tavsifidan boshlaymiz. STL da bu dek o‘zi nima? Tushuntirish uchun, uzoqqa bormaymiz, jismoniy tarbiya darsini olamiz. Ko‘pchiligimiz "bo‘y bo‘yicha saflanish" nimaligini bilamiz. Shunday qilib, haqiqiy Dek shunday qurilish mexanizmiga ega. Unda shunday bir qator hosil bo‘ladiki va bu qator uning oxirida va uning boshidan to‘ldiriladi, zarur bo‘lsa, bunda ketma-ket elementlarni o‘rnini almashtirishi ham mumkin. Qancha tezroq turish kerak bo‘lsa, shuncha yaxshi va stek, navbat ma’lumotlar tuzilmasiga qaraganda? Albatta, yo boshida yoki qatorning oxirida va qatorning o‘rtasida taxminan sizning bo‘y balandligiga qarash va keyin solishtirish va o‘rinlarni o‘zgartirish kerak. Biroq, bir qator o‘rtasida joylashtirilgan bo‘lsa, butun elementlar suriladi. Bu hayotdan dekka qiyosiy misol edi.

Shakllantirilayotgan to‘plamning boshida yoki oxirida elementlarni qo‘shish kerak bo‘lganda, ba‘zi elementlarni o‘rtaga qo‘yish kerak bo‘ladi dekni ishlatish mantiqan to‘g‘ri keladi (mos ravishda shu tipdagi dek tanlash kerak). Dekda ketma-ketlik o‘rtasida qo‘shish va o‘chirish qo‘llab-quvvatlash bo‘lsa-da, bu amallar chiziqli vaqtida amalga oshiriladi. Dasturda bu kabi amallar ko‘p bajarish kerak bo‘lsa, ro‘yxatlar eng yaxshi tanlov bo‘ladi.

Aslida, Dek-bu boshida yoki oxirida elementlarning tezroq qo‘shilishi imkoni bo‘lgan vektordir. Shu bilan birga, Dekning boshqa amallari vektorlarning tegishli amallari bilan bir xil ishlaydigan vaqtga (amallar ko‘paysa sekinroq) ega.

Dek (navbat queue) bilan va vektor (vector) o‘rtasidagi farqlar:

1. Deklarning boshiga va oxiriga ham elementlarni qo‘shish tezkor amaldir, ammo vektorlar uchun faqat oxiriga qo‘shish tezkor amal hisoblanadi. Bu qo‘shish doimiy vaqt talab etadi.
2. Dekning funksiyalar va elementlarni qo‘shishga nisbattan vektordagi shunga o‘xshash funksiyalar odatda, bir oz sekin.
3. Dekning iteratorlari maxsus tipdagi aqlli ko‘rsatkichlarga ega bo‘lishi kerak, chunki doimiy ko‘rsatkichlar ular turli qismlari o‘rtasida biridan ikkinchisiga o‘tish kerak.
4. Xotira bloklari hajmiga chegaralari bor tizimlarida ikki tomonlama navbat dek xotirani ortiqg‘i bilan ajratadi, chunki vektor olishi mumkin bo‘lgan ma’lumotlarga nisbattan ko‘p. Shunday qilib, max_size () funksiyasining natijasi vektorlarga nisbatan ikki tomonlama navbatlarga katta bo‘lishi mumkin
5. Ikki tomonlama navbatlar imkoniyatlarni boshqarishni ta‘minlamaydi va xotira qayta ajratilganda unga joy aniqlanmaydi. Xususan, boshida yoki oxirida elementlarni qo‘shish boshqa har qanday qo‘shish yoki o‘chirish kabi ikki tomonlama navbatda elementlar bilan bog‘liq barcha ko‘rsatkichlarni, murojaatlarni iteratorlar bekor qilmaydi. Biroq, xotiraning taqmislanishi vektorlarga qaraganda yaxshiroq ishlashi mumkin, chunki ular odatda ichki tuzilishiga ko‘ra, ikki tomonlama navbatlar xotirani qayta joylashtirishda barcha elementlarining nusxalash shart emas.
6. Deklarda ishlatilmaydigan xotira bloklari band qilinmaydi va real o‘lchamdan so‘ng xotiraning bo‘sh joylari ozod qilinadi. Bu esa Dekning xotira hajmi kamaytirishga imkon beradi (ammo, qachon va qanday qilib bu sodir bo‘lshi uni amalga oshirishga bog‘liq).

Dekka ta‘luqli bo‘lgan vektorlarning afzalligi.

1. To‘plamning o‘rtasida elementlarni qo‘shish va o‘chirish sekinroq amal hisoblanadi. Chunki bir o‘zgarish uchun barcha elementlarning o‘rinlarini almashtirish kerak bo‘ladi.
2. Vektorlar va deklarning iteratorlari erkin ruxsat turiga mansub.
3. Dek sinfi vektor sinfining funksiyalari bilan bir xil funksiyalarga ega.

Dekning kamchiliklariga

1. Tez-tez qo‘shish yoki o‘chirish dekning boshida yoki oxirida sodir bo‘lishi.
2. Konteyner elementlariga havola qila olmaysiz.

3. Konteyner ishlatilmay qolganda xotirani xoli qilish amalga oshirilishiga C++ kafolat bermaydi (amalga oshirish tamoyiliga qarab, xoli bo‘lshi va bo‘lmasligi ham mumkin)

Inobatga olishini kerak bo‘lgan holatlar:

1. Dekning at() boo‘qa biror bir funksiyasi iterator va indeksni tekshira olmaydi.

2. Qo‘shish va o‘chirish qo‘shimcha xotira talab qilish mumkin. Shuning uchun qo‘shish va o‘chirishga doir amallar bajarilganda dekning boshqa elementlari uchun ko‘rsatkichlar, havolalar va iteratorlarni aniq ko‘rsata olmaydi. Bunga istisno bo‘lib, Dekning boshiga yoki oxiriga element qo‘shish olinadi. Boshiga yoki oxiriga qo‘shilganda ko‘rsatkich va havolalar aniq bo‘ladi, ammo iterator yo‘q.

3. Dek va vektorning interfeyslari deyarli bir xil. Farqi shundaki, ba‘zi bir funksiyalash yaratilgan bo‘lishi mumkin. Masalan, dek sinfi capacity, reserve kabi funksiyalarga ega emas. Buning sababi dek vektorlarga o‘xshab ishlashini yaxshilash kerak emas.

4. Deklar bilan ishlaganda, ayniqsa ma’lumotlarni qo‘shishda ko‘rsatkich va iteratorlarga bohliq dastur fragmenini yozishdan ehtiyot bo‘lish va mustaqil yozish kerakmas.

Umuman olganda, shaxsiy deklar bu vektorlardir, ular bilan ishlash ham vektorlar kabi amalga oshiriladi. Ammo qachon dekni, qachon vektorni tanlashni bilish kerak.

Ma’lumotlar tuzilmasining boshiga va oxiriga qo‘shish uchun optimizatsiyalangan (yaxshilangan, ixcham qilingan) vektor – bu dekdir. Bunda qo‘shish aniq o‘zgarmas vaqtda bajariladi va nusxalash konstruktorini bir marta chaqiradi.

Agar element dekning o‘rtasiga qo‘shilsa, eng yomoni bu amal vaqt talab qiladi, chunki qo‘shish nuqtadan boshigacha va oxirigacha bo‘lgan masofalar orasida kichik aniqlash lozim.

insert, push_front i push_back funksiyalari dekdagi hamma iteratorlarni bekor qilishi mumkin. Shuningdek dekning o‘rtasiga ma’lumot qo‘shilganda havolalarni xam bekor qiladi.

Berilgan navbatdagi elementlarni guruhlash masalasini qaraylik. Guruhlashni amalga oshirish uchun guruhlash shartini tanlaymiz va shu asosida navbat ichida guruhlaymiz, shartga tegishli bo‘lganlar navbat boshidan, bo‘lmaganlar oxiridan joylatiriladi.

4.8-dastur. Navbvtdagi elementlarni guruhlash.

```
// Created by MBBahodir
#include "stdafx.h"
```

```
#include <iostream>
#include <deque>
#include <algorithm>
using namespace std;

  bool mypred(const int x){
     return x <= 51;   // guruhlash uchun shart
  }

int main(){

    int Arr[]={1,78,89,23,51,49,100,18,50};
    deque<int> d(&Arr[0],&Arr[9]);

    cout << "Joriy deque: " << endl;
    for (deque<int>::iterator
it=d.begin();it!=d.end();it++) std::cout<<*it<<" | ";
    cout << endl;

    stable_partition(d.begin(),d.end(),mypred);

    cout << "Natija: " << endl;
    for (deque<int>::iterator
it=d.begin();it!=d.end();it++) std::cout<<*it<<" | ";
    cout << endl;
    system("pause");
    return 0;
 }
```

4.8-dastur.Output

```
Joriy deque:
1 | 78 | 89 | 23 | 51 | 49 | 100 | 18 | 50 |
Natija:
1 | 23 | 51 | 49 | 18 | 50 | 78 | 89 | 100 |
```

4.7-dasturdagi fragment asosiy guruhlash sharti hisoblanadi.

```
bool mypred(const int x){
     return x <= 51;   // guruhlash uchun shart
```

```
  }
```

Bunda agar x <= 51 bo‘lsa true, aks holda flase qaytaradi. Bunga ekvivalent sifatida quyidagicha fragment ham yozish mumkin.

```
bool mypred(const int x){
     if (x <= 51) return true;   // guruhlash uchun
shart
     return false;
  }
```

Dek uchun keltirilgan materiallar dastlabki o‘rganuvchilar uchun yetarli deb hisoblaymiz. Masalaning shartiga qarab dek yoki vektorni tanlash dasturchining mahoratiga bog‘liq.

Dekning funksiyalariga quyidagilar kiradi:

4.1-jadval. Dekning funksiyalari

push_front	Boshidan yangi element qo‘shish
push_back	Oxiridan yangi element qo‘shish
pop_front	Birinchi elementni olish
pop_back	Oxirgi elementni olish
front	Birinchi element qiymatini ko‘rish
back	Oxirgi element qiymatini ko‘rish
size	Elementlar sonini ko‘rish
clear	Barcha elementlarni o‘chirish

Konteynerlar bilan ishlash algoritmlari. Barcha konteynerlar umumiy yondashuvi tufayli, amalda qiziqtirgan asosiy algoritmlar konteynerlarning har qanday turi uchun qo‘llanilishi mumkin bo‘lgan umumiy shaklda amalga oshirilishi mumkinligi tufayli STL konteynerlar uzoq amaliy foydalanish uchun bir arzigulik ixtiro bo‘ldi.

Algoritmlar kutubxonaning eng katta va ommabop qismidir. Juda ko‘p algoritmlar mavjudki, ularning hammasini batafsil bayon qilish uchun katta kitob kerak. Quyida ularni guruhlarga ajratib, ularning nomlari bilan ataymiz (ularning hammasi emas) va ulardan faqat ayrimlari misol tariqasida keltiramiz.

for_each()algoritmi. for_each() eng ko‘p ishlatiladigan algoritmdir. Konteyner elementlar guruhi (ehtimol, barcha elementlari) uchun ketma-ket harakatni amalga oshiradi. U har qanday STL konteynerda foydalanish mumkin. Quyidagi 4.9-dasturda for_each() algoritm massiv va vektor uchun ishlatilgan.

4.9-dastur. for_each() algoritmidan foydalanish.

```
// Created by MBBahodir
```

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

inline ostream& operator <<( ostream& out, const
vector< unsigned > & obj ) {
   cout << "< ";
   for( auto& p: obj )
      cout << p << " ";
   return out << ">";
}

void pow2( unsigned& i ) { i *= i; }

int main( void ) {
   const int examples = 4;
   for( int i = 0; i < examples; i++ ) {
      unsigned ai[] = { 1, 2, 3, 4 , 5, 6, 7, 8, 9
},
           ni = sizeof( ai ) / sizeof( ai[ 0 ] );
      vector< unsigned > vi( ai, ai + ni );
      cout << vi;
      switch( i ) {
         case 0:
            for_each( vi.begin(), vi.end(), pow2 );
            cout << " => " << vi << endl;
            break;
         case 1:
            for_each( ai, ai + ni, pow2 );
            cout << " => " << vector< unsigned >(
ai, ai + ni ) << endl;
            break;
         case 2:
            for( auto& i : ai ) pow2( i );
            cout << " => " << vector< unsigned >(
```

```
ai, ai + ni ) << endl;
            break;
         case 3:
            for_each( vi.begin() + 2, vi.end() - 2,
pow2 );
            cout << " => " << vi << endl;
            break;
      }
   }
   system("pause");
   return 0;
}
```

4.9-dastur.Output.

```
< 1 2 3 4 5 6 7 8 9 > =>< 1 4 9 16 25 36 49 64 81 >
< 1 2 3 4 5 6 7 8 9 > =>< 1 4 9 16 25 36 49 64 81 >
< 1 2 3 4 5 6 7 8 9 > =>< 1 4 9 16 25 36 49 64 81 >
< 1 2 3 4 5 6 7 8 9 > =>< 1 2 9 16 25 36 49 8 9 >
```

Dasturning birinchi global (auto &x : ...) takrorlanish fragmenti ishlashini ko‘rsatilgan (C++11 standarti tomonidan joriy qilingan). Bu kabi fragmentni massivlar va konteynerlarga ham qo‘llanilishi mumkin (bu vektorni oqimga chiqarish operatori variant funksiyasida ko‘rsatilgan). Bu takrorlanish fragmenti aslida STL kutubxona yoki algoritmlarni qismi emas, lekin u for_each() algoritm bilan bir xil ta‘sir ko‘rsatadi, ya‘ni to‘plam barcha elementlar uchun izchil qo‘llash mumkin.

Bu yuqoridagi dastur barcha algoritmlarni berilgan interval asosida tashkiliy asosiy mantiq qismini ko‘rsatadi (shart emas, barcha konteynerlar). Boshi va oxiri bilan cheklangan iteratorga (ko‘pincha, birinchi berilgan 2 ta parametrlar) funksiya, funktor va predikat (bu funksiyalarga - qaysidir elementlari bo‘yicha saralash imkoni bo‘lsa va mantiqiy qiymat qaytaradigan funksiyalar kiradi) navbat bilan qo‘llaniladi.

Find(). Navbatdagi algoritm – bu find(). Bu nomidan ma’lumki, to‘plamdagi elementlarni qidirish funksyasidir. E‘tibor bersak, juda ko‘p konteynerlar bunday find() – funksiyasiga ega va u obyekt uchun **obj.find(…) sietaksisida ishlatiladi. Shu vaqtda algoritm quyidagicha funksiyani find(obj:iteator, …)** ishga tushuradi. Aslida, bu bitta algoritm emas, balki ularning butun bir katta guruhi mavjud,

ular to‘plam elementlarini ba‘zi atribut, shart bilan tanlashlari asosida birlashtirilishi mumkin. Bu guruh predikatlariga **find()**, **find_if()**, **find_if_not()**, **find_first_of()**, **find_end()**, **adjacent_find()** funksiyalari kiradi. Shuningdek bu guruhda kengaytirilgan algoritmlar ham tegishlidir va bularga **count()**, **count_if()**, **search()**, **binary_search()**, **min()**, **max()**, **minmax_element()**, **min_element()**, **max_element()**, **equal() kabi bir qator algoritmlar kiradi.**

To‘plamga ishlov berish (aralashtirish) bir shartli guruh algoritmlar mavjud va ular joylarda elementlarni qayta tartibga soluvchi va qiymatlarni o‘zgartiruvchi algoritmlardir: **fill()**, **replace_copy()**, **reverse()**, **rotate()**, **rotate_copy()**, **shuffle()**, **random_shuffle()**, **transform()**, **replace()**, **replace_if() va boshqa funksiyalar.**

Ikki to‘plamlar ustida bajariladigan algoritimlar guruhi ham mavjud. Bular asosan nusxalash, tarkibini ko‘chirib o‘tish (xar tipdagi konteynerlar uchun bo‘lishi mumkin. Masalan, **vector<>** ni **set<> ga**) funksiyalarini bajaradi. Bularga **copy()**, **copy_if()**, **move()**, **swap_ranges()**, **remove_copy()**, **remove_copy_if()**, **merge()**, **set_intersection()**, **set_difference() va boshqa algoritmlar.**

Nihoyat, juda maxsus guruh algoritmlari, to‘plam ichida elementlar turli saralash bilan bog‘liq funksiyalar **sort()**, **stable_sort()**, **is_sorted()**, **is_sorted_until()va boshqa algoritmlar.** Ushbu qiziqarli guruhni keyinroq, alohida batafsil ko‘rib chiqish kerak.

Kutubxonada vaqt o‘tishi bilan ortib borayotgan algoritmlarning ko‘pligi va ularning aksariyati adabiyotda hech qanday ta‘riflanmasligiga qaramasdan, tabiiy savol tug‘iladi: bu xilma-xillikni qanday tushunish kerak? Bu qiyinchiliklar qanday yengish ‘ mumkin:

- Barcha STL obyektlar (konteynerlar, algoritmlar) shablon sintaktiki bo‘yicha tasvirlangan. Shuning uchun, ularning tavsiflari, ularning header fayllarida dastur fragmentlari sifatida tuzilgan bo‘lishi kerak.
- Kerakli algoritimlar uchun header fayllarni standart </usr/include/c++> katalogiga o‘ting va stl_algo* kabi sarlavha fayllarini toping - uda barcha algoritm funksiyasi prototiplarini (o‘xshashlarini) topasiz. Bundan tashqari, har bir prototipi algoritm maqsadini tushuntirib va parametrlarini tushuntirishning batafsil izoh oldindan yozib qo‘yilgan.
- Bir necha asosiy STL algoritmlarni foydalanishga doir misollar ko‘rib chiqing – kutubxonalar tizimida ularning ko‘pchiligi bor. O‘xshashlik bilan boshqa barcha algoritmlarni yaratish mumkin.

Shablon sinf kutubxonalar shablon jihatidan **template** termin bilan belgilangan, chunki o‘sha, kompilyatsiya sintaktiki xato bo‘yicha xabarlarni chiqaradi. Birinchisi yozma xabarlar fragmentlari bo‘yicha va ikkinchi xatolarni izlash uchun. Bu shablon kabi kuchli mexanizmi tomonidan qayyta ishlanidi (tekshiriladi) va dasturchi bu uchun tayyor bo‘lishi kerak.

Yuqorida aytib o‘tilganidek, misollarni o‘rganish juda ko‘p savollarga javob beradi, shuning uchun dastur keltiramiz. Endi o‘rgangan algoritmlarni diqqat bilan eslang (har birini sharhlab, kutubxona sinflaridan tashqarida bo‘lgan juda ko‘p STL algoritmlari mavjud, ammo ularning barchasi bir-biriga o‘xshab ishlaydi):

4.10-dastur. Standart algoritmlardan foydalanish.

```
#include "stdafx.h"
#include <iostream>
#include <cstring>
#include <vector>
#include <map>
#include <set>
#include <algorithm>
using namespace std;

inline ostream& operator <<( ostream& out, const
vector<char>& obj ) {
   for( auto p: obj ) cout << p;
   return out;
}

int main( void ) {
   char s[] = "Muhammad al-Xorazmiy nomidagi
Toshkent Axborot Texnologiyalari Universiteti"; //
nazvanie knigi
   vector<string> vs;
   vs.push_back("Muminov");
   vs.push_back("Bahodir");
   // copy & find :
   vector<char> v1( strlen( s ) );
   copy( s, s + v1.size(), v1.begin() );
   int nb = 0;
```

```
   for( auto is = find( v1.begin(), v1.end(), ' '
); is != v1.end();
             is = find( ++is, v1.end(), ' ' ) )
nb++;
   cout << "bo‘sh joylar soni: " << nb << " (" <<
nb + 1
        << " so‘z)" << endl;
   // min & max :
   auto mm = minmax_element( v1.begin(), v1.end()
);
   cout << "belgilar uzunligi: '" << *mm.first <<
"' ... '"
        << *mm.second << "'" << endl;
   // fill & reverse & rotate & shuffle :
   vector<char> suv( vs[ 0 ].size() );
   copy( vs[ 0 ].begin(), vs[ 0 ].end(),
suv.begin() );
   cout << suv << endl;
   random_shuffle( suv.begin(), suv.end() );
   cout << suv << endl;
   reverse( suv.begin(), suv.end() );
   cout << suv << endl;
   rotate( suv.begin(), suv.begin() + suv.size() /
2, suv.end() );
   cout << suv << endl;
   // set_intersection & set_difference
   set< char > sus, pns;
   for( char s: vector<char>( vs[ 0 ].begin(), vs[
0 ].end() ) )
      sus.insert( s );
   for( char s: vector<char>( vs[ 1 ].begin(), vs[
1 ].end() ) )
      pns.insert( s );
   vector<char> outi( 100 ), outd( 100 );
   auto ret = set_intersection( sus.begin(),
sus.end(), pns.begin(),
                                pns.end(),
outi.begin() );
```

```
   cout << "umumiy belgi " << ( ret - outi.begin()
) << " : "
        << outi << endl;
   ret = set_difference( sus.begin(), sus.end(),
pns.begin(),
                         pns.end(), outd.begin() );
   cout << "unikal belgi " << ( ret - outd.begin()
) << " : "
        << outd << endl;
   system("pause");
   return 0;
}
```

4.10-dastur. Output

```
      bo‘sh joylar soni: 6 (7 so‘z)
      belgilar uzunligi: ' ' ... 'z'
      Muminov
      nuvmMoi
      ioMmvun
      mvunioM
      umumiy belgi 2 : io
unikal belgi 5 : Mmnuv
```

Dasturda char uchun konteynerlardan foydalanilgan. Deyarli barcha berilgan konteynerlar uchun turli xil algoritmlar amalga oshirilgan. (ixcham, ammo zerikarli fragmentlar).

Funktorlarning qo‘llanilishi. Funktor so‘zi C++ da “funksional obyektlar” ning qisqartmasidan olingan. C++ da funksional obyekt - operator() bilan aniqlangan sinfning nusxasi hisoblanadi. Agar C++ sinfi operator() aniqlasangiz, u funksiya kabi ishladi, ammo holatni saqlay olmaydi. Masalan, 4.11-dasturga qarang.

4.11-dastur. Operator()ni qayta ishlatish.

```
#include "stdafx.h"

#include <iostream>
#include <string>

using namespace std;
```

```
class SimpleFunctor {
    string name_;
public:
    SimpleFunctor(const char *name) : name_(name)
{}
    void operator()() {  cout << "Salom, " << name_
<< " ishlar zo‘rmi!" << endl; }
};

int main() {
    SimpleFunctor sf("Akbar");
    sf();
    system("pause");
    return 0;
}
```

4.11-dastur. output

```
Salom, Akbar ishlar zo‘rmi!
```

Dasturda sf obyekt bo‘lishidan qa‘tiy nazar sf() funksiyasini chaqiramiz. Chunki SimpleFunctop sinfida operator() aniqlangan.

Ko‘p hollarda STL kutubxonasining taqqoslash va xatolik funksiyasining algoritmlarida funktorlar C++ da predikatlar sifatida ishlatiladi. Faraz qilamiz, sonlar ketma ketligi berilgan. Bu ketma ketlikdagi barcha juft sonlar va toq sonlar yig‘indisini hisoblash kerak bo‘lsin. Bu masalani yechish uchun funktor va for_each algoritmidan foydalanish mumkin.

4.12-dastur. Funktor va for_each algoritmidan foydalanish.

```
#include "stdafx.h"
#include <algorithm>
#include <iostream>
#include <list>
using namespace std;
class EvenOddFunctor {
    int even_;
    int odd_;
public:
    EvenOddFunctor() : even_(0), odd_(0) {}
    void operator()(int x) {
```

```
        if (x%2 == 0) even_ += x;
        else odd_ += x;
    }
    int even_sum() const { return even_; }
    int odd_sum() const { return odd_; }
};

int main() {
    EvenOddFunctor evenodd;

    int my_list[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
};
    evenodd =  for_each(my_list,

my_list+sizeof(my_list)/sizeof(my_list[0]),
                  evenodd);

     cout << " Juftlar yig‘indisi: " <<
evenodd.even_sum() << endl;
     cout << " Toqtlar yig‘indisi: " <<
evenodd.odd_sum() <<  endl;
     system("pause");
     return 0;
}
```

4.12-dastur. output
Juftlar yig‘indisi: 30 Toqtlar yig‘indisi: 25

Dasturda for_each da EvenOddFunctor sinf obyekti berilgan. for_each my_list bo‘yicha ketma ket o‘tganda funktor funksiyani chaqirmoqda. Har takrorlanishda evenodd funktor nusxasini qaytaradi. Evenodd funktor esa juft yoki toq elmentlarni yig‘indisini saqlaydi.

Standart MLda funktorlar. ML dagi funksiya interfeyslarning umumiy tadbiqlari mavjud va OYD jihatidan shakllantirish qiyin masallardan hisoblanadi. ML standart jihatidan funktorlar ML modul tizimining bir qismidir va ular tuzilmalarni yaratishga imkon beradi. Masalan, plagin tizimini yozish masalasini olaylik. Aytaylik barcha

plaginlar soddaroq bo‘lishi uchun, faqat perform funksiyasini o‘z ichiga oladigan zarur interfeysini amalga oshirish kerak bo‘lsin. MLda avval plaginlar uchun maxsus tasdiqlovchi (imzolovchini)ni yaratish zarur.

```
Signature Plugin =
sig
  val perform: unit -> unit
end;
```

Plaginlar uchun tasdiqlovchi interfeysni aniqladik, ikkita plaginni, aytaylik, Loudplugin va Silentpluginni tuzilmalar orqali amalga oshirishimiz mumkin.

LoudPlugin ni amalga oshirish:

```
structure LoudPlugin :> Plugin =
struct
  fun perform() = print"KATTA HARFLAR!\n"
end;
```

Silentplugin ni amalga oshirish:

```
structure SilentPlugin :> Plugin =
struct
  fun perform () = print "kichik harlar\n"
end;
```

Endi funktorlarga yaqinlashamiz. ML dagi funktorlar tuzilmalarni argument sifatida qabul qilishadi, shuning uchun **Plugin** argument sifatida berilgan.

```
functor Performer(P : Plugin) =
struct
  fun job () = P.perform ()
end;
```

Bu funktor Plugin ni P argument sifatida qabul qiladi va P plaginni perform funksiyasini chaqiradigan job funksiyasi uchun ishlatadi.

Performer funktorni ishlatamiz (Funktor tuzilma qaytarishini bilamiz).

```
structure LoudPerformer = Performer(LoudPlugin);
structure SilentPerformer = Performer(SilentPlugin);

LoudPerformer.job ();
SilentPerformer.job ();
```

Standard ML uchun bk oddiy misol edi. Ammo chuqurroq o‘rganish uchun S tilini ham bilish talab qilinadi.

Haskellda funktorlar. Haskelldagi funktorlar haqiqiy funktorlar bo‘lishi bilan farqlanadi. Haskelldagi funktorlar kategoriya nazariyasidan matematik funktorlarga juda o‘xshaydi (Kategoriya nazariyasi-matematikaning obyektlarning ichki tuzilishiga bog‘liq bo‘lmagan matematik obyektlar o‘rtasidagi munosabatlarning xususiyatlarini o‘rganadigan bo‘limi). Kategoriya nazariyasida funktor F kategoriyalar o‘rtasida shunday taqsimlanishi kerakki, kategoriya strukturasi saqlanib qolsin yoki boshqacha qilib aytganda, u ikki kategoriya orasidagi gomomorfizm bo‘lsin.

Bu ta‘rif Haskellda oddiy tipdagi sinf sifatida amalga oshiriladi.

```
class Functor f where
 fmap :: (a -> b) -> f a -> f b
```

ML ga qarab, masalan, Haskell tipi plagin sinfiga o‘xshaydi, joriy Sinfning nusxasi bo‘lish uchun qaysi amallarni qanday qilib yaratishni aniqlaydi. Bu holatda, birgina Functor tiplar bo‘yicha aniqlanmagan, balki f konstruktor asosida aniqlangan. Bundan shuni bilish mumkinki Functor – fmap funksiyasini yaratib beradi. fmap funksiyasi a tipini qabul qiladi va b tipini qaytaradi. fa (f konstruktori asosida qurilgan tip, a ga qo‘llaniladi) qabul qilinadigan tip va fb qaytariladigan tip.

Uni tushunish uchun, ba‘zi konreynerda har bir element uchun amal qiladigan funksiyasi sifatida fmap funksiyasini o‘ylab ko‘ring. Funktorlarning eng oddiy misoli muntazam ro‘yxatlar va map funksiyasi bo‘lib, u ro‘yxatdagi har bir element uchun funksiyani qo‘llaydi.

```
Prelude> map (+1) [1,2,3,4,5]
[2,3,4,5,6]
```

Bu oddiy misolda fmap funksiyasi faqat map va f tipidagi konstruktor [] - list tipidagi konstruktordir. Shuning uchun, ro‘yxatlar uchun, masalan, Functor sifatida belgilangan.

```
instance Functor [] where
 fmap = map
```

Bu map o‘rniga fmap yordamida fragmentni shakllantirib olsak, albatta, to‘g‘ri bo‘ldi.

```
Prelude> fmap (+1) [1,2,3,4,5]
[2,3,4,5,6]
```

Lekin funktor ta‘rifi tuzilishini asrab-avaylash haqida hech narsa demaydi, unutmang! Shuning uchun har qanday normal funktorlar matematik funksiyalar ta‘rifiga kiruvchi funktorlar qonunlarini qanoatlantirishi kerak. Fmapning ikki qoidalari bor:

```
fmap id = id
fmap (g. h) = fmap g. fmap h
```

Birinchi qoidada konteynerda har bir element uchun bir xil funksiyani xaritalash hech qanday ta‘siri yo‘q deb aytilgan. Ikkinchi qoidaga ko‘ra, konteynerdagi har bir element ustida ikkita funksiyaning tarkibi birinchi funksiyani ko‘rsatish bilan, so‘ngra ikkinchisini ko‘rsatish bilan bir xil bo‘ladi.

Ularni yaqqol ko‘rsatish uchun funktorlarga yana bir misol daraxtlar ustida bajariladigan amallar hisoblanadi. Bir konteyner sifatida daraxt, shajara (ierarxiya, ichma ich joylashgan obyektlar) berilgan bo‘lsin. Daraxt tuzilishini saqlab qolish uchun, daraxt xususityalariga fmap funksiyasini qo‘llaymiz. Buning uchun birinchi daraxtni belgilab boshlaymiz.

```
data Tree a = Node (Tree a) (Tree a)
        | Leaf a
         deriving Show
```

Bu fragmentda daraxt turi ikki daraxtning (chap va o‘ng tugunlaruga) tuguni (node) yoki barg (leaf) ekanligi aytiladi. **`deriving Show`** ifodasi show funksiyasi orqali ko‘rish imkonini beradi. Endi Tree daraxti ustida funktorni aniqlay olamiz.

```
instance Functor Tree where
  fmap g (Leaf v) = Leaf (g v)
  fmap g (Node l r) = Node (fmap g l) (fmap g r)
```

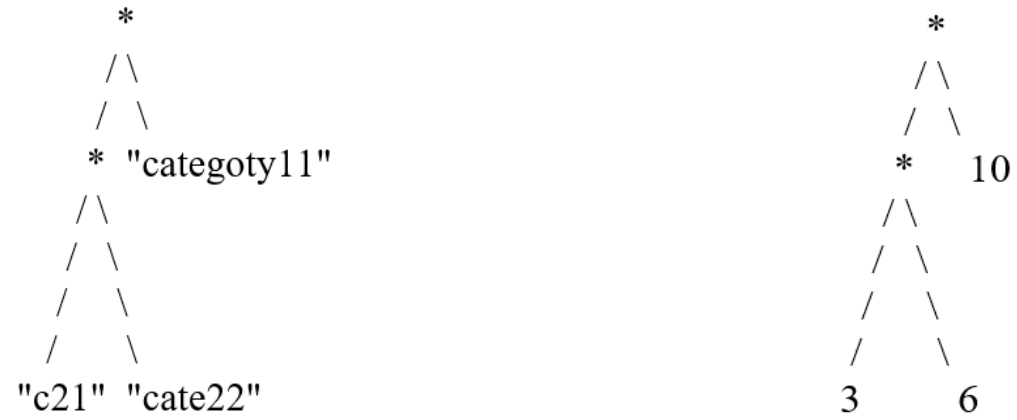
Bu fragmentda fmap tipidagi g funksiyasi argumuent sifatida Leaf tipli v qiymatni qabul qiladi va Leaf tipidagi gni vga qiymat qilib beradi. Ikkinchi qoidada fmap tipiga oid g funksiyasi Node tipidagi l va r qiymatlarni qabul qiladi va natijada Node funksiyasi fmap tipidagi gl va gr argumentlarni qiymat qilib beradi.

Fmap funksiyasini tree obyektlari bilan ishlash uchun tayyorlaymiz. Tree sinfni satrli (string) ro‘yxat bilan quramiz va har bir ro‘yxat uzunligini length()funksiyasi bilan aniqlaymiz.

```
Prelude> let tree = (Node (Node (Leaf “c21”) (Leaf “cate22”) (Leaf
“categoty11”))
```

```
Prelude> fmap length tree
Node (Node (Leaf 3) (Leaf 6)) (Leaf 10)
```

Quyidagicha tree quriladi:



(a) (b)

4.1-rasm. (a) tree ko‘rinishi, (b) length() bo‘yicha ko‘rinishi.

Aslida, Haskell da funktorlar fundamental funktor sinfi hisoblanadi va applicative functors va strelkalar barchasi bular ham funktorlaridir. Haskell funksiyalar murakkab hisoblanadi, agar bu funktorlarni haqiqiy xayot qonuniyatlari bag‘lab o‘rganmoqchi bo‘lsangi, alohida fan va manbalarni o‘rganish lozim.

Prolog funktorlari. Bu hammasidan oson funktorlardir. Ikkita misol qaraymiz. Birinchisi atom - bu tuzilmaning birinchisidir. Ifodani qaraymiz:

```
?- likes(olma, nok)
```

Birinchi atom funktordir – likes

Ikkinchisi- functor deb nomlangan ajralmas predmet. Bu argumentlari va funktor tuzilmasini qaytaradi. masalan,

```
?- functor(likes(olma, nok), Functor, Arity).
Functor = likes
Arity = 2
```

Bu Prolog funktorlarga misol edi.

Funktorlarni yaratish dasturchilarga xos emas albatta. Chunki bugungi kunda tayyor foydalanish uchun ko‘plab funktorlar yaratilgan. Ularni asosan instrument yaratuvchilar, ya‘ni dasturchilar uchun dastur tuzuvchilar yaratadi ap mantiqiy dasturlash funktorlarning o‘rni muhim ahamiyat kasb etadi.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Adapterlar nima va qaysi sinflar kiradi?
2. Konteynerlarning adapterlari qaysi sinflar kiradi.
3. Juda keng va tez-tez ishlatiladigan ma’lumotlar tuzilmalari bu – nima?
4. Konteyner adapterlarining funksiyalarini ta‘minlash uchun STL standart konteyneri har qanday moslashtirishda tayanch sifatida qaysi amallardan (adapter turiga qarab) foydalanish mumkin.
5. Birinchi navbatda konteynerlarning adapterlari uchun qanday sinflarni dasturga qo‘shish lozim.
6. Stek nima va u nimasi bilan xarakterlanadi.
7. Birinchi kirgan oxiri chiqadi tamoyili bo‘yicha faoliyat ko‘rsatuvchi ma’lumotlar to‘plami bu nima.
8. Navbat nima va u nimasi bilan harakterlanadi.
9. Birinchi kirgan Birinchi chiqadi tamoyili bo‘yicha faoliyat ko‘rsatuvchi ma’lumotlar to‘plami bu nima.
10. Navbatning qaysi usuli vektor konteyner ustiga qurib bo‘lmaydi, lekin uni ro‘yxat konteyner uchun qurish mumkin.
11. Deque sinfi uchun *front()*, *push_back()*, *pop_front*()), usullari har qanday konteynerlarda qachon ishlatiladi?
12. Deque sinfida qaysi usuli o‘rniga front() usuli ishlatiladi?
13. Ma’lum bir tartibda qayta ishlanishi kerak ma’lumotlarni uchun umumiy ma’lumotlar tuzilishiga asoslangan to‘plami bu – qanday to‘plamlar?
14. Ma’lumotlar tuzilishi - stek juda oddiy: u qanday tartibni belgilaydi?
15. Stekning assosiativ usullarini sanab bering?
16. Navbatda, birinchi elment kiritilgan bo‘lsa, u qachon chiqariladi?
17. Agar C++ da navbat shablonini ishlatmoqchi bo‘lsangiz, avval qanday kutubxonasini ulashingiz kerak.
18. empty() – navbat uchun nima vazifani bajaradi.
19. Massiv yordamida navbatlarni yaratish mumkinmi?, mumkin bo‘lsa izohlab bering.
20. Ustuvor navbat (Priority_queue) bilan navbat (queue) obyektlarining farqi nimada.

21. Ustivor navbatda yangi element qo‘shish uchun push() funksiyasi, birinchi o‘rindagi elementiga murojaat qilish uchun esa qaysi funksiyasi ishlatiladi.
22. Ikki tomonlama navbat va odiiy navbatning farqi nimada?
23. Dek va vektorning o‘xshash tomonlari va farqlari.
24. Ikki tomonlama navbatning afzalliklarini sanab bering.
25. Nima uchun ikki tomonlama navbat dasturchiga kerak. 3 ta asos keltiring.
26. Dekning maummolarini bilasizmi? Dek navbat qachon dasturchiga pand berishi mumkin.
27. Konteyner elementlar guruhi (ehtimol, barcha elementlari) uchun ketma ket harakatni amalga oshiradigan algoritmning aniqlang.
28. To‘plamdagi elementlarni qidirish funksyasi find()ga ta‘rif bering.
29. To‘plamga ishlov berish (aralashtirish)uchun bir shartli guruh algoritmlar mavjud va ular joylarda elementlarni qayta tartibga soluvchi va qiymatlarni o‘zgartiruvchi algoritmlardir. Ularni sanab bering.
30. Ikki to‘plamlar ustida bajariladigan algoritimlar guruhi qaysi algoritmlar misol bo‘la oladi.
31. Funktor so‘zi C++ da qaysi so‘zning qisqartmasidan olingan.
32. C++ da funksional obyekt – qaysi operator bilan aniqlangan sinfning nusxasi hisoblanadi.
33. Ko‘p hollarda STL kutubxonasining taqqoslash va xatolik funksiyasining algoritmlarida funktorlar C++ da nimalar sifatida ishlatiladi.
34. ML standart jihatidan funktorlar ML modul tizimining bir qismidir va ular nimalarini yaratishga imkon beradi.
35. Haskelldagi funktorlar kategoriya nazariyasidan qaysi funktorlarga juda o‘xshaydi.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I

Stack – stek konteynerlar adarteriga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur

```
// Created by MBBahodir
#include "stdafx.h"

#include <iostream>
#include <vector>
#include <stack>
#include <string>
#include <array>
#include <string>
using namespace std;

int main() {

    stack<int> st, st_one, st_two;
    stack<string,vector<string>> vst;

    vst.emplace("str1");
    vst.emplace("str2");
    vst.push("eostr");
    for (int i = 0; i < 255; i++)
    {
        vst.push("belgi - " + to_string(int(i)) + to_string(i));
    }
```

topshiriqlar

1. Dasturda nechta global funksiya yaratilgan.

_____
_____
_____
_____

2. Dasturda nechta stack to‘plam mavjud.

_____
_____
_____
_____

3. Dasturda vst.push("belgi - " + to_string(int(i)) + to_string(i)); qanday vazifani bajaradi.

_____
_____
_____
_____
_____
_____
_____

4. Ikki xil tipdagi stek to‘plam elementlarini almashtirish dastur fragmentini yozing.

_____
_____
_____

```
    for (int i=0; i<50;
++i) st.push(i*15);

    for (int i = 0; i <
105; i++)
st_one.emplace(i*10);

    st_two = st;
    st.swap(st_one);

    cout << st.size() << "
" << st_one.size() << " "
<< st_two.size() << " " <<
vst.size() << endl;

    while (!vst.empty()) {
        cout << vst.top()
<< " : stack dagi o‘rni "
<< vst.size() << endl;
        vst.pop();
    }

    int size = st.size();
    while (size--) {
        cout << st.top()
<< " : stack dagi o‘rni "
<< st.size() << endl;
        st.pop();
    }

    while (st.size() !=
0){
        cout <<
st_two.top() << " : stack
dagi o‘rni " <<
st_two.size() << endl;
        st_two.pop();
```

5. (!vst.empty()) qachon chin va qachon yolg‘on qiymatlarni qaytaradi.

_____
_____
_____

6. Stekdan elementlarni ishlash funksiyasini tuzing.

_____
_____
_____

7.
```
while (st.size() !=
  0){
      cout <<
st_two.top() << " : stack
dagi o‘rni " <<
st_two.size() << endl;
      st_two.pop();
   }
```

Dastur fragmentini stek elementlarini saqlab qolgan xolda ishlatish funksiyasini tuzing.

_____
_____
_____

8. Stek va vektor bilan farqlari aniqlovchi dastur tuzing.

_____
_____
_____
_____
_____
_____
_____
_____

```
    }

    while (!st.empty()) {
        cout << st.top()
<< " : stack dagi o‘rni "
<< st.size() << endl;
        st.pop();
    }

    cout << st.size() << "
" << st_one.size() << " "
<< st_two.size() << " " <<
vst.size() << endl;

    cout << " stack st "
<< (st.empty() ? "" : "not
") << " empty" << endl;
    cout << " stack vst"
<< (vst.empty() ? "" :
"not ") << " empty" <<
endl;
    cout << " stack st_one
" << (st_one.empty() ? ""
: "not ") << " empty" <<
endl;
    system("pause");
    return 0;
}
```

9. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
10. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

IKKINCHI ASSISMENT TOPSHIRIG‘I	
🎓	queue – navbat konteyner adapteriga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan

	oshriladi.

dastur	topshiriqlar

// Created by MBBahodir
#include "stdafx.h"
 	
#include <iostream>
#include <list>
#include <queue>
#include <string>
#include <array>
using namespace std;
 	
int main() {
 	
queue<int> st, st_one, st_two;
queue<string,list<string>> vst;
 	
vst.emplace("str1");
vst.emplace("str2");
vst.push("eostr");
 	
for (int i=0; i<5; ++i) st.push(i);
 	
for (int i = 0; i < 15; i++) st_one.emplace(i*10);
 	
st_two = st;
st.swap(st_one);
 	
cout << st.size() << " " << st_one.size() << " " << st_two.size() << " " << vst.size() <<
	1. Dasturda nechta global funksiya yaratilgan.
	_____
	_____
	_____
	_____
	_____
	
	2. Dasturda nechta queue sinfining obyektining to‘plam mavjud.
	_____
	_____
	_____
	_____
	_____
	_____
	
	3. Dasturdagi ortiqcha kutubxonalarni o‘chirib tashlang va sabablarini ayting.
	_____
	_____
	_____
	
	4. Navbatda qachon oxirgi element qoladi. Dastur fragmenti asosida ko‘rsatib bering.
	_____
	_____
	_____
	
	5. (st.size() != 0) shartni matiqan takrorlanishga shart qilib qo‘yish to‘g‘rimi va nim uchun. Anologini yozing.
	_____
	_____
	_____

```
endl;

  while (!vst.empty())
{
      cout <<
vst.front() << " :
stack dagi o‘rni " <<
vst.size() << endl;
    vst.pop();
    }

 int size = st.size();
 while (size--) {
     cout << st.front()
<< " : stack dagi o‘rni
" << st.size() << endl;
     st.pop();
 }

 while (st.size() !=
0){
     cout <<
st_two.front() << " :
stack dagi o‘rni " <<
st_two.size() << endl;
     st_two.pop();
 }

/* while (!st.empty())
{
     cout << st.front()
<< " : stack dagi o‘rni
" << st.size() << endl;
       st.pop();
 } */

 cout << st.size() << "
" << st_one.size() << "
```

6. `st.swap(st_one);` ni o‘rniga `st_one.swap(st);` yozilsa nima hodisa sodir bo‘ladi.

_____
_____
_____

7. Navbatning dasturda ishlatimagan funksiyasini aniqlang.

_____
_____
_____
_____
_____

8. Dasturdagi izohga olingan dastur fragmenti olib tashlansa, dasturda qanday o‘zgarishlar bo‘ladi va natijasichi.

_____
_____
_____
_____
_____

<pre>" << st_two.size() << "
" << vst.size() <<
endl;
 	
cout << " stack st "
<< (st.empty() ? "" :
"not ") << " empty" <<
endl;
cout << " stack vst"
<< (vst.empty() ? "" :
"not ") << " empty" <<
endl;
cout << " stack st_one
" << (st_one.empty() ?
"" : "not ") << "
empty" << endl;
system("pause");
return 0;</pre>	

9. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____ 10. Shu dasturning analogini yaratish sizga mustaqil vazifadir.	

UCHINCHI ASSISMENT TOPSHIRIG‘I	
🎓	Ustuvor navbat (Priority_queue) va ikki tomonlama navbat (deque) oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.
dastur	topshiriqlar

<pre>// Created by MBBahodir
#include "stdafx.h"
#include <iostream>
#include <queue></pre>
	1. Dasturda nechta tipdagi navbat obyektlari yaratilgan.
	_____
	_____
	_____
	_____

```
using namespace std;
   bool mypred(const
int x){
     return x <= 51;
// guruhlash uchun
shart
  }

int main() {
    priority_queue
<int> priority_q,
priority_q_one;

  for (int h = 0; h <
7; h++) {

priority_q.push(rand()%
100);
  }

  priority_q_one  =
priority_q;

  cout << "priority_q
navbatdigi birinchi
element " <<
priority_q.top() <<
endl;
   cout << "
priority_q_one
navbatdigi birinchi
element " <<
priority_q_one.top() <<
endl;
  priority_q.pop();

  cout << "priority_q
```

2. Dasturda priority_q.push(rand()%100); ni o‘rniga boshqa operatorni yozing.

_____
_____
_____
_____
_____
_____

3. Dasturda nechta priority_queue to‘plam mavjud.

_____
_____
_____
_____
_____
_____

4. Dasturda `deque<int> d(&Arr[0],&Arr[9]);` funksiyasi qanday vazifani bajaradi.

_____
_____
_____

5. To‘plamdan berilgan qiymatni izlash dastur fragmentini yozing.

_____
_____
_____
_____
_____
_____
_____

6. Ustivor navbat shartkiritib ikkita yangi navbat yaratuvchi dastur tuzing.

_____
_____
_____

```
navbatdigi birinchi
element (o‘chirishdan
so‘ng): " <<
priority_q.top() <<
endl;

  cout << "priority_q
navbat bo‘sh " <<
(priority_q.empty()?"":
"emas") << endl;

  while
(!priority_q_one.empty(
))
  {
      cout <<
"priority_q_one
navbatdigi element : "
<< priority_q_one.top()
<< endl;

priority_q_one.pop();
  }
  cout << "
priority_q_one navbat
bo‘sh " <<
(priority_q_one.empty()
?"":"emas") << endl;

  int
Arr[]={1,78,89,23,51,49
,100,18,50};
    deque<int>
d(&Arr[0],&Arr[9]);

    cout << "Joriy
deque: " << endl;
    for
```

7. Dasturda it qanday tip, nima uchun aynan dek navbatga foydalanilgan.

_____
_____
_____

8. operator= dasturning qaysi qismlarida foydalanilgan va nima uchun.

_____
_____
_____

```
(deque<int>::iterator
it=d.begin();it!=d.end(
);it++)
std::cout<<*it<<" | ";
    cout << endl;


stable_partition(d.begi
n(),d.end(),mypred);

    cout << "Natija: "
<< endl;
    for
(deque<int>::iterator
it=d.begin();it!=d.end(
);it++)
std::cout<<*it<<" | ";
    cout << endl;

  system("pause");
  return 0;
}
```

9. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
10. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

TO‘RTINCHI ASSISMENT TOPSHIRIG‘I	
🎓	Konteynerlar bilan ishlash algoritmlariga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`// Created by MBBahodir`
`#include "stdafx.h"`
`#include <vector>`
`#include <algorithm>`
 	
`using namespace std;`
 	
`inline ostream&`
`operator <<( ostream&`
`out, const vector<`
`unsigned > & obj ) {`
`  cout << "< ";`
`  for( auto& p: obj )`
`  cout << p << " ";`
`  return out << ">";`
`}`
 	
`void pow2( unsigned& i`
`) { i *= i; }`
 	
`int main( void ) {`
`  const int examples =`
`4;`
`  for( int i = 0; i <`
`examples; i++ ) {`
`  unsigned ai[] = {`
`1, 2, 3, 4 , 5, 6, 7,`
	1. Dasturda nechta global funksiya yaratilgan.
	_____
	_____
	_____
	_____
	
	2. Dasturda xatolarni toping.
	_____
	_____
	_____
	_____
	_____
	_____
	
	3. for(auto& p: obj) fragmentni tushuntirib bering va anoligini yozing.
	_____
	_____
	_____
	_____
	_____
	_____
	
	4. Dasturning qaysi fragmentlarini find() funksiyasi bilan almashtirish mumkin.
	_____
	_____
	_____

```
8, 9 },
           ni = sizeof(
ai ) / sizeof( ai[ 0 ]
);
      vector< unsigned
> vi( ai, ai + ni );
      cout << vi;
      switch( i ) {
         case 0:
            for_each(
vi.begin(), vi.end(),
pow2 );
            cout << "
=> " << vi << endl;
            break;
         case 1:
            for_each(
ai, ai + ni, pow2 );
            cout << "
=> " << vector<
unsigned >( ai, ai + ni
) << endl;
            break;
         case 2:
            for( auto&
i : ai ) pow2( i );
            cout << "
=> " << vector<
unsigned >( ai, ai + ni
) << endl;
            break;
         case 3:
            for_each(
vi.begin() + 2,
vi.end() - 2, pow2 );
            cout << "
=> " << vi << endl;
            break;
```

5. To‘plamdan berilgan qiymatni izlash dastur fragmentini aniqlang va o‘z izlash funksiyasingizni yarating.

_____
_____
_____
_____
_____
_____
_____
_____
_____

```
      }
   }
   system("pause");
   return 0;
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

BESHINCHI ASSISMENT TOPSHIRIG‘I

Funktorlarning ko‘llanilishiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur

```
// ConsoleApplication1.cpp
: main project file.

#include "stdafx.h"
#include <iostream>
#include <string>
using namespace std;
class myFunctorClass
{
private:
    int _x;
    char _ch;
public:
    myFunctorClass (int x)
: _x( x ) {}
    myFunctorClass (char
ch): _ch(ch){}
    int operator() (int y)
{ return _x + y; }
    int operator() ()
{return _x*_x; }
```

topshiriqlar

1. Dasturda nechta konstruktor funksiya yaratilgan.

_____
_____
_____
_____

2. Dasturda xatolarni toping.

_____
_____
_____
_____
_____
_____

3. operator() fragmentni tushuntirib bering va anoligini yozing.

_____
_____
_____
_____

```
    void operator()(char
ch, int a,  int b) {
        switch (ch)
        {
        case '+' : cout <<
a + b; break;
        case '-' : cout <<
a - b; break;
        case '*' : cout <<
a * b; break;
        case '/' : cout <<
((b>0) ? to_string(a/b) :
"maxraj 0"); break;
        case '%' : cout <<
((b>0) ? to_string(a%b) :
"maxraj 0"); break;
        case '^': {
            int ret = a;
            while( b-- > 1
) ret *= a;
            cout << ret;
                  } break;
        default: cout <<
"amal to‘g‘rimas !!!";
            break;
        }
    }

    int operator()( int a,
int b) {
        switch (_ch)
        {
        case '+' : return a
+ b; break;
        case '-' : return a
- b; break;
        case '*' : return a
* b; break;
```

_____
_____

4. Massivning juft va toq elementlari yig‘indisini df sonini hisoblovchi funktor yozing.

_____
_____
_____

5. To‘plamdan berilgan qiymatni izlash funktor funksiyasingizni yarating.

_____
_____
_____
_____
_____
_____
_____
_____
_____

6. Taqqoslash uchun funktor funksiya yarating.

_____
_____
_____
_____
_____
_____
_____

```
        case '/' : return
((b>0) ? a/b : 0); break;
        case '%' : return
((b>0) ? a%b : 0); break;
        case '^': {
            int ret = a;
            while( b-- > 1
) ret *= a;
            return ret;
                  } break;
        default: {cout <<
"amal to‘g‘rimas !!!";
return 0; }
                 break;
        }
    }
};

int main()
{
    myFunctorClass add( 5
);
    cout << add( 6 );
    cout << endl;
    myFunctorClass F(5);
    F('+',3,4);
    cout << endl << F() <<
endl;

    cout <<
myFunctorClass('^')(2,10);
    system("pause");
    return 0;
}
```

7. Funktor funksiyalarning qaysi uslubi sizga yoqdi. Dasturni to‘liq shu uslubga almashtiring.

_____
_____
_____
_____
_____

8. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi.

_____

9. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

2-BOB. DASTURLASH TILINING TAKOMILLASHTIRILGAN IMKONIYATLARI

2.1. Standart algoritmlar va iteratorlar.

📖 C++ tilining standart algoritmlar kutubxonasi, iteratorlar, xotirani taqsimlash talablari, dinamik xotira ajratish va foydalanishga asosida qidirish, saralash, taqqoslash funksiyalari, dasturlashda o‘zlarini tutishlari, iteratorlar, o‘zgarmas iteratorlar, teskari iteratorlar bilan ishlashga mo‘ljallangan asosiy funksiyalar va ularning dasturlashdagi ahamiyati, xotirani taqsimlovchi sinflarni yaratish va talablari, turli tipli massivlar uchun dinamik xotirani ajratish va foydalanish operatorlar uslublari, amallari, talablari, vazifalari va usullari keltirilgan bo‘lib, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni mustahkamlash uchun 30 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 3 ta assisment topshirig‘i va har assismentda 10(9,7) ta topshiriq, jami 26ta topshiriq berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

🖋 ***Kalit so‘zlar.*** Algoritm, kutubxona, iterator, o‘zgarmas iterator, teskari iterator, qidirish, saralash, o‘rin almashtirish, shablon, sinf, find(), sort(), swap(), begin(), end(), rbegin(), rend(), crbegin(), crend(), xotira, pul, xotira bloki, new, delete, for_each(), funksional obyektlar.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Tip tushunchasi, sinf va sinf obyekti, merosxo‘r sinf, to‘plam, statik va dinamik massiv, elementga murojaat, funksiya va ko‘rsatkich, konteyner, xesh, istisno holat, ma’lumotlarni kiritish va chiqarish, dasturlashga oid dastlabki tushunchalar va C++ tilini qo‘llab-quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Inspektorlar, mutatorlar, koordinatorlar, find() algoritmi, find_if() algoritmi, count() va count_if() algoritmlari, sort() algoritmi, <functional> kutubxonasi, for_each() algortmi, algoritmlarning bajarilish tartibi, iterator amallari, iteratorlarning qo‘shima amallari, xotirani taqsimlovchilar, new() operatori, delete() operatori, xotirani ishlashini tezlashtirish usullari, pul obyektlari, konteyner va uni xotirada qo‘llanilishi, standart bo‘yicha taqsimlovchilar usullarini o‘rganishingiz mumkin.

REJA

1. Standart algoritmlar.
2. Iteratorlar va ularning qo‘llanilishi.
3. Xotirani taqsimlovchilar va ularga qo‘yilgan talablar.

4. Standart bo‘yicha taqsimlovchi.

KIRISh

Bugunki kunda dasturlash juda soddalashib bormoqda, ba‘zi bir mutaxassislar dasturlash uchun matematika kerakmas deb aytsa, ba‘zilari matematika dasturlashning asosi deb aytadi. Shuning negizida nima yotibdi. Albatta algoritmlar, isteratorlar va xotirani taqsimlovchilar va ularning talablari yotadi. Algoritmlar tayyor bo‘lsa, faqat bu algoritmlarini joy joyiga qo‘yish bu juda osondir. Ammo algoritmlarni yaratish va uni dasturlash joriy qilish o‘ta mushkul va mashaqqatli masaladir. Shuning uchun bugungi kunda juda ko‘plab algoritmlarni saqlovchi standart va no standart kutubxonalar mavjud.

Standart algoritmlar. Yangi dasturlashni boshlanuvchilar odatda saralash, qidirish yoki qator elementlarini sanash kabi nisbatan oddiy vazifalarni bajarish uchun maxsus takrorlanishga asoslangan usullarni yozish uchun juda ko‘p vaqt sarflashadi. Bu usullar ularda xato qilish qanchalik osonligi jihatidan ham, umumiy ishonchlilik va mavjudlik nuqtai nazaridan ham muammoli bo‘lishi mumkin, chunki bu usullar tushunish qiyin bo‘lishi mumkin.

Qidiruv, sanash va saralash dasturlashda juda keng tarqalgan amallar bo‘lganligi sababli, C++ standart kutubxonasi dastlab bu vazifalarni bir necha qator kodda bajaradigan katta funksiyalar to‘plamini o‘z ichiga olgan. Bundan tashqari, bu xususiyatlar oldindan sinovdan o‘tgan, samarali va turli xil konteynerlarni qo‘llab-quvvatlaydi. Bu xususiyatlarni ham paralel qo‘llab – quvvatlash tezroq uni bajarish uchun bir xil masala uchun bir necha MP (markaziy protsessor) oqimlarini ajratish qobiliyati ega.

Algoritm kutubxonasi tomonidan taqdim etiladigan funksiyalar odatda uch toifadan biriga kiradi:

Inspektorlar (nazoratchilar) - konteynerdagi ma’lumotlarni (masalan, qidirish yoki elementlarni soni hisoblash amallari) ko‘rish uchun ishlatiladi.

Mutatorlar - konteynerdagi ma’lumotlarni o‘zgartirish uchun ishlatiladi (masalan, saralash yoki elementlarni qayta tartibga solish amallari).

Fasilitatorlar (koordinatorlar) - elementlarining qiymatlari asosida natija hosil qilish uchun ishlatiladi (masalan, qiymatlarni ko‘paytiruvchi obyektlar yoki elementlarning qaysi tartib juftliklarida tartiblanishi kerakligini aniqlovchi obyektlar).

Bu algoritmlar algoritm kutubxonasida (<algorithm >sarlavha faylida) joylashgan. Oldingi biladigan va eng ko‘p tarqalgan algoritmlardan ba‘zilarini imkoniyatlarini chuquroq ko‘rib chiqamiz va bu algoritmlarning barchasi iteratorlardan foydalanadi.

find() algoritmi – berilgan qiymat bo‘yicha elementlarni qidirish algoritmi. Berilgan qiymatni birinchi topgunicha ishlaydi. Argument sifatida 3 ta parametrni oladi.

ketma-ketlikda boshlang‘ich element uchun iterator;

ketma-ketlikda oxirgi element uchun iterator;

qidirishning qiymati.

Qidirish natijada qidirilayotgan qiymatli elementga (agar u topilsa) yoki iteratorning oxiriga (agar bunday element topilmasa) ishora qiluvchi iterator qaytadi.

5.1-dastur. find () funksiyasidan foydalanish.

```
// Created by MBBahodir
#include "stdafx.h"

#include <algorithm>
#include <array>
#include <iostream>

using namespace std;
int main(){
    array<int,10> arr = {1, 3, 5, 7, 9, 11, 13, 15,
17, 19 };

    cout << "Qidirish va o‘zgartirish uchun son
kirit: ";
    int search;
    int replace;
    cin >> search >> replace;

//  array<int,10>::iterator found;

    auto found = find(arr.begin(), arr.end(),
search);

    if (found == arr.end())
    {
```

```
        cout << search << " topilmadi. " << endl;
    }
    else
    {
        *found = replace;
    }

    for (int i : arr)
    {
        cout << i << ' ';
    }

    cout << endl;
    system("pause");
    return 0;
}
```

5.1-dastur.Output
Birinchi sinov
Qidirish va o‘zgartirish uchun son kirit: 19 21 1 3 5 7 9 11 13 15 17 21
Ikkinchi sinov
Qidirish va o‘zgartirish uchun son kirit: 10 25 10 topilmadi. 1 3 5 7 9 11 13 15 17 19

find_if() algoritmi - shart orqali elementni izlash. Juda ko‘p hollarda konteynerda mavjud elementlar orasida ma’lum bir qiymat saqlaydigan element bormi degan masala duch kelamiz. Masalan, konteynerda satrli elementlar berilgan bo‘lsa, biror bir satr qismi bormikan degan masala bo‘lsin. Mana shunday masalalarni yechishda find_if() algoritmi eng mos va qulay hisoblanadi. Bu algoritm find() algoritmi bilan analog sifatida ishlaydi ammo, qidirish uchun qiymat o‘rniga funksiya ko’rsatkichlariga moslangan element qiymatida o‘xshashlik bor bo‘lgan obyekt chaqiriladi. find_if() funksiyasi buning uchun har bir elementni chaqiradi va to shunday elementni topilmaguncha yoki konteynerda elementlar tugaguncha.

Ma’lum bir berilgan satr bo‘yicha, berilgan konteynerda shu satrni o‘zida qism satr qilib saqlovchi element bor yoki yo‘qligini aniqlovchi dasturni tuzish kerak bo‘lsin.

5.2-dastur. find_if() funksiyasidan foydalanish.

```
// Created by MBBahodir
#include "stdafx.h"
#include <algorithm>
#include <array>
#include <iostream>
#include <string>

using namespace std;

bool containsNut(string str)
{
    return (str.find("oro") != string::npos);
}

int main()
{
    array<string, 4> arr = { "Toshkent", "Buxoro",
"Samarqand", "Navoiy" };

    auto found = find_if(arr.begin(), arr.end(),
containsNut);

    if (found == arr.end())
    {
        cout << " Topilmadi" << endl;
    }
    else
    {
        cout << " Topildi: " << *found << endl;
    }
    system("pause");
    return 0;
}
```

Agar odatdagi standart tarzdagidek muammoni hal qilshganimizda, kamida ikki takrorlanish kerak edi (birinchi takrorlanish elementlarni saralash uchun va ikkinchisi satr qismini solishtirish uchun). Standart C++ kutubxonasining vazifalari bir nechta fragment satrlarida ham shunday qilish imkonini beradi.

count() va count_if() algoritmlari – konteynerdan berilgan elementlarini sanash vazifasini bajaradi. Berilgan mezon (shart) bo‘yicha barcha elementlarni tekshirib, sanaydi. Berilgan konteynerda nechta element berilgan satr qismi borligini sanash dasturini keltiramiz.

5.3-dastur. count() va count_if() algoritmlaridan foydalanish.

```
// Created by MBBahodir
#include "stdafx.h"
#include <algorithm>
#include <array>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

bool containsNut(string str)
{
    return (str.find("daryo") != string::npos);
}
bool counter(string str){
    return true;
}

int main()
{
    array<string, 6> arr = { "Sirdaryo",
"Amudaryo", "Qashadaryo", "Suxondaryo",
"Zarafshon", "Qashadaryo" };

    auto countWithIf = count_if(arr.begin(),
arr.end(), containsNut);
    int countofArr = count(arr.begin(), arr.end(),
"Qashadaryo");
    cout <<  "daryo satr qismi borlarining soni: "
```

```
<< countWithIf << endl;
    cout <<  " elementlar soni: " << countofArr <<
endl;

    system("pause");
    return 0;
}
```

5.3-dastur. Output

```
daryo satr qismi borlarining soni: 5
 elementlar soni: 2
```

sort() algoritmi – foydalanuvchining saralash funksiyasini qurish uchun ishlatiladi. Oldinlari sort() funksiyasi bilan massivlarni saralagan bo‘lishingiz mumkin, ammo sort() funksiya bu bilan chegaralanib qolmaydi. Bu funksiya yordamchi bir funksiya oladi va bu argument asosida foydalanuvchi o‘zining saralash uslubini amalga oshirishi mumkin. Funksiya esa taqqoslash uchun parametr ikkita parametr oladi. Agar birinchi element ikkinchisidan oldin bo‘lishi kerak bo‘lsa, ikkinchi parametr true qiymat qaytarishi kerak. Odatda sort() funksiyasi o‘sish tartibida saralaydi. Berilgan massivni kamayish tartibida saralash dasturini tuzaylik.

5.4-dastur. sort() funksiyasidan foydalanish.

```
#include "stdafx.h"
#include <algorithm>
#include <array>
#include <iostream>

using namespace std;
bool greater(int a, int b)
{
  return (a > b);
}
bool d_greater(int a, int b)
{
  return (b > a);
}
```

```
int main()
{
   array<int,5> arr = { 28, 18, 16, 25, 25 },
arr_one;

   arr_one = arr;

   sort(arr.begin(), arr.end(), greater);
   sort(arr_one.begin(), arr_one.end(),d_greater );
   for (int i : arr) {
    cout << i << ' ';
   }
   cout << endl;
   for (int i : arr_one ) {
       cout << i << ' ';
   }
  cout << endl;

  system("pause");
  return 0;
}
```

5.4-dastur. Output
28 25 25 18 16 16 18 25 25 28

Dasturdan ko‘rinadiki, o‘zimiz saralash algoritmini boshidan yozishimiz shart emas ekan, buning uchun bir qator fragment yetarli bo‘ldi. Ammo C++ tili, juda ko‘p saralashning kamayish tartibda foydalanganligi uchun greater{} tipli funksiyasi bo‘lishi mumkin (<functional> kutubxonasida bor). Biz o‘zimizning greater funksiyasidan foydalanish uchun sort(arr.begin(), arr.end(), greater); dastur fragmentini yozdik, ammo kutubxonanikini chaqirish uchun sort(arr.begin(), arr.end(), std::greater{}); kabi fragment yozishimiz kerak. E‘tibor bering, greater{} funksiyasi figurali qavsga muxtoj, shuning uchun uni funksiya deb atalmaydi, ma’lumot tipi deb yuritiladi va undan foydalanish uchun nusxasini yaratishimiz kerak. Figurali qavslar anonim obyektlarni yaratish uchun xizmat qiladi.

for_each() algortmi – konteynerning barcha elementlarini o‘qish. Bu for_each() funksiyasi kiruvchi ma’lumot sifatida to‘plamni qabul qiladi

va foydalanuvchining funksiyasini joriy to‘plamdagi har bir element uchun qabul qiladi. Buning qulaylik tomoni shundaki, agar bir amalni to‘plamning barcha elementlari bo‘yicha bajarish kerak bo‘lsa, for_each() funksiyasi as qotadi. Berilgan massivning barcha elementlarini chiqarish uchun for_each() funksiyasidan foydalanishni dasturini keltiraylik.

5.5-dastur. for_each() funksiyasidan foydalanish.

```
#include "stdafx.h"

#include <algorithm>
#include <array>
#include <iostream>

using namespace std;

void doubleNumber(int &i){
  i = i;
}

int main(){
    array<int,4> arr = { 1, 2, 3, 4 };

   for_each(arr.begin(), arr.end(), doubleNumber);

  for (int i : arr){
     cout << i << ' ';
  }

   cout << endl;
   system("pause");
  return 0;
}
```

Yosh dasturchilar uchun bu takrorlanish jarayonini oddiy for operatoriga aniq parametrlarni qo‘yib ishlatish maquldir. for_each() funksiyasining yutuqlari shandan iboratki, takrorlanish tanasidan qayta foydalanish va paralellashtirish, katta proyektlarda katta ma’lumotlar, turli tiplari har xil ma’lumotlar tuzilmasi bilan ishlgandan eng yaxshi isntrument bo‘lib xizmat qiladi.

Algoritmlarning bajarilish tartibi . Algoritm kutubxonasida eng ko‘p algoritmlar ma’lum bir ijro tartibini kafolatlamaydi. Bunday algoritmlar uchun har qanday belgilangan funksiyalar bajarilish tartibini o‘z zimmasiga olmasligiga ishonch hosil qilish kerak, chunki bu funksiyalarni chaqirish tartibi ishlatiladigan kompilyatorga qarab farq qilishi mumkin. Quyidagi algoritmlar aniq bajariladi:

```
for_each()
  copy()
  copy_backward()
  move()
  move_backward()
```

Agar parametr sifatida biror funksiya berilmagan bo‘lsa, aniqlanmagan holat o‘tadi. Buning uchun har bir algoritmga arr.begin() va arr.end() aniq berilishi kerak. Ammo C++20 standartida faqat massiv berilsa o‘zi aniqlanadi.

Algoritm kutubxonasining algoritmlari ro‘yxati:

1	**O‘zgartirilmaydigan ketma-ketlik amallar:**	
1.1	all_of	to‘plamdagi barcha elementlar uchun holat shartini tekshirish
1.2	any_of	to‘plamdagi har qanday element uchun holat shartini bajarishini tekshirish
1.3	none_of	to‘plamdagi hech bir element uchun holat shartga javob bermasligini tekshirish
1.4	for_each	funksiyani to‘plamga qo‘llash
1.5	find	to‘plamda qiymatini topish
1.6	find_if	to‘plamda shartni bajaruvchi elementni topish
1.7	find_if_not	to‘plamda shartni bajarmaydigan elementni topish
1.8	find_end	to‘plamdagi oxirgi elementni topish
1.9	find_first_of	to‘plamdagi elementlardan birinchisini topish
1.10	adjacent_find	to‘plamda teng qo‘shni elementlarni topish
1.11	count	to‘plamdagi elementlar sonini hisoblash
1.12	count_if	shartga javob beradigan to‘plamdagi elementlar sonini qaytaradi
1.13	mismatch	ikki farqli intervallarni birinchi o‘rnini qaytaradi
1.14	equal	to‘plamdagi Ikki elementlar tengligini tekshirish

1.15	is_permutation	to‘plamda almashtirish bor yoki yo‘qligini tekshirish
1.16	search	ketmk-ketlikdan qidirish
1.17	search_n	intervaldan qidirish
2	**Ketma-ketlik amallarini o‘zgartirish:**	
2.1	copy	to‘plam elementlardan nusxa olish
2.2.	copy_n	elementlarni ko‘paytirish
2.3	copy_if	shart asosida diapazon elementlarini nusxalash
2.4	copy_backward	elementlar to‘plamidan orqaga nusxa ko‘chirish
2.5	move	elementlar qatorini ko‘chirish
2.6	move_backward	elementlar to‘plamni orqaga ko‘chirish
2.7	swap	ikki obyektning qiymatlarini almashtirish
2.8	swap_ranges	ikki intervalning qiymatlarini almashtirish
2.9	iter_swap	ikki iteratorlarning ko‘rsatikich obyektlari qiymatlarini almashish
2.10	transform	to‘plam obyektlarini aylantirish
2.11	replace	to‘plamning qiymat almashtirish
2.12	replace_if	Qiymatlarni shart asosida to‘plamda almashtirish
2.13	replace_copy	almashtirish qiymatlari to‘plamni nusxalash
2.14	replace_copy_if	almashtirish qiymatlari to‘plamni shart asosida nusxalash
2.15	fill	to‘plamni qiymat bilan to‘ldirish
2.16	fill_n	ketma-ketligini qiymat bilan to‘ldirish
2.17	generate	funksiya bilan to‘plam uchun qiymatlarni yaratish
2.18	generate_n	funksiya bilan ketma-ketlik uchun qiymatlarni hosil qilish
2.19	remove	to‘plam qiymatini o‘chirish
2.20	remove_if	to‘plam ma’lumotlar shart asosida o‘chirish
2.21	remove_copy	qiymatni o‘chirish diapazonini nusxalash
2.22	remove_copy_if	qiymatni o‘chirish diapazonini shart asosida nusxalash
2.23	unique	to‘plamdagi takrorlangan elementlarni

		o‘chirish
2.24	unique_copy	to‘plamdagi takrorlangan elementlarni nusxalash
2.25	reverse	teskari to‘plam
2.26	reverse_copy	teskari to‘plamni nusxalash
2.27	rotate	to‘plamda elementlarning chapga aylantirish
2.28	rotate_copy	to‘plamda elementlarning chapga aylantirishni nusxalash
2.29	random_shuffle	to‘plamdagi elementlarning tasodifiy qo‘yish
2.30	shuffle	to‘plamdagi elementlarning generator yordamida tasodifiy qo‘yish
3	**Bo‘laklar (qismlar) bilan amallari**	
3.1	is_partitioned	to‘plamning bo‘linganligini tekshirish
3.2	partition	ikkiga bo‘lish oralig‘i
3.3	stable_partition	qonuniyat asosida Ikkiga bo‘lish oralig‘i
3.4	partition_copy	bir to‘plamni ikkiga bo‘lish
3.5	partition_point	bo‘lish nuqtasini olish
4	**Saralash amallari:**	
4.1	sort	to‘plam ma’lumotlar saralash
4.2	stable_sort	ekvivalentlar tartibini saqlab qolgan holda elementlarni saralash
4.3	partial_sort	to‘plamda elementlarni qisman saralash
4.4	partial_sort_copy	to‘plamda elementlarni qisman saralash va nusxalash
4.5	is_sorted	to‘plamning tartiblanganligini tekshirish
4.6	is_sorted_until	to‘plamdagi ajratilmagan elementni topish
4.7	nth_element	to‘plamdagi elementni saralash
5	**Binar qidiruv (bo‘lingan / tartiblangan intervallar bilan ishlash):**	
5.1	lower_bound	iteratorni quyi chegarasini qaytarish
5.2	upper_bound	iteratorni yuqori chegarasini qaytarish
5.3	equal_range	teng elementlardan to‘plam osti olish
5.4	binary_search	qiymatning tartiblangan ketma-ketlikda mavjudligini tekshirish
6	**Birlashtirish (tartiblangan intervallar bilan ishlash):**	
6.1	merge	tartiblangan to‘plamlarni

		birlashtirish
6.2	inplace_merge	ketma-ket tartiblangan intervallarni birlashtirish
6.3	includes	tartiblangan to‘plamlarni boshqa tartiblangan to‘plamni o‘z ichiga olishi yoki olmasligini tekshirish
6.4	set_union	ikkita tartiblangan to‘plamni birlashtirish
6.5	set_intersection	ikki tartiblangan intervallarning kesishishi
6.6	set_difference	ikki tartiblangan intervallar orasidagi farq
6.7	set_symmetric_difference	ikki tartiblangan intervallar orasidagi simmetrik farq
7	**Elementlar yig‘indisi amallari:**	
7.1	push_heap	elementlar yig‘indisini qatoriga surish
7.2	pop_heap	elementlar yig‘indisidan elementi o‘chirish
7.3	make_heap	elementlar yig‘indisini yasash
7.4	sort_heap	elementlar yig‘indisini tartiblash
7.5	is_heap	elementlar yig‘indisi ni tekshirish
7.6	is_heap_until	elementlar yig‘indisidan birinchisini topish
8	**Kichik va katta amallari:**	
8.1	min	eng kichik elementni qaytaradi
8.2	max	eng katta elementni qaytaradi
8.3	minmax	eng kichik va eng katta elementni qaytaradi
8.4	min_ element	to‘plamdagi eng kichik elementni qaytaradi
8.5	max_element	to‘plamdagi eng katta elementni qaytaradi
8.6	minmax_element	to‘plamdagi eng kichik va eng katta elementni qaytaradi
9	**Boshqa yordamchi amallar:**	
9.1	lexicographical_compare	leksikografik jihatdan qiyoslash
9.2	next_permutation	to‘plam aylantirish uchun keyingi ideks
9.3	prev_permutation	to‘plam aylantirish uchun oldingi ideks

Bu yerda 85 ta algoritmlar keltirilgan, ammo bularning hammasi shablon funksiya bo‘lganligi uchun C++ dasturlash versiyalarida farqi

bo‘lishi mumkin. Shuningdek, eslab ko‘ring, bu algoritmlarning ko‘plaridan oldin mavzularimizda foydalanganmiz.

Iteratorlar va ularning qo‘llanilishi. Iteratorlar konteynerlarning elementlariga murojaat qilish uchun foydalaniladi. Iteratorlar bilan elementlar bilan ishlash juda qulay hisoblanadi. Iterator iterator tipi bilan yoziladi. Har qanday konteyner uchun iteratorlarning tiplari farq qiladi. Masalan, list<int> tipidagi konteyner uchun **list<int>::iterator tipi,** vector<int> tipidagi konteyner uchun esa **vector<int>::iterator** tipi ishlatiladi. C++ tiladi konteynerlardan iteratorlarni ajratib olish uchun **begin()** va **end()** funksiyalaridan foydalaniladi. **begin()** funksiyasi konteynerdagi birinchi element ko‘rsatuvchi iteratorni qaytaradi (agar konteynerda elementlar bo‘lsa). **end()** funksiyasi konteynerdagi oxirgi elementdan keyingi pozitsiyani ko‘rsatuvchi iteratorni (ya‘ni konteyner oxirini) qaytaradi. Agar konteyner bo‘sh bo‘lsa, **begin()** va **end()** funksiyalari bir xil qiymat qaytaradi. Agar **begin()** va **end()** funksiyalari o‘zaro teng bo‘lmasa, ularning orasidan kamida bitta element bor. Bu funksiyalar aniq tipli konteyner uchun iterator qaytaradi. Iterator yaratishga misol:

```
std::vector<int> v = { 1,2,3,4 };
std::vector<int>::iterator iter = v.begin();
```

Bu misolda tipiga mansub bo‘lgan vektor tipidpgi konteyner - vektor yaratilgan. Konteyner doimiy qiymatlar bilan to‘ldirilgan. **begin()** funksiyasi (usuli) bilan vektor elementini olish uchun iterator keltirilgan. Bu iterator vektor konteynerning birinchi elementini ko‘rsatadi.

Iterator amallari:

-***iter** – iterator ko‘rsatadigann elementni olish;

-**++iter** - keyingi elementga murojjat qilish uchun iteratorni harakatlantirish

-**--iter** - oldingi elementga murojjat qilish uchun iteratorni harakatlantirish. forward_list konteyner iteratorlari dekrement amalini qo‘llab quvvatlamaydi.

-**iter1 == iter2** - ikki iterator teng, agar ular bir xil iteratorni aniqlagan bo‘lsa.

-**iter1 != iter2** ikki iterator teng emas, agar ular bir xil iteratorni aniqlagan bo‘lsa.

5.6-dastur. Iterator amallaridan foydalanish.

```
#include "stdafx.h"
#include <iostream>
#include <vector>
```

```
  using namespace std;
int main()
{
    vector<int> myvector;
    for (int i = 0; i < 15; i++)
    {
        myvector.push_back(rand() % 100);
    }
     //  vector<int>::iterator iter =
myvector.begin();
    auto iter = myvector.begin();
    while(iter!=myvector.end())
    {
        cout << *iter << " | ";
        ++iter;
    }
    cout << endl;
    system("pause");
    return 0;
}
```

Konteynerlar bilan ishlaganda, konteynerdagi elementlarni qo‘shish yoki o‘chirish ushbu konteyner uchun barcha joriy iteratorlarni, shuningdek, uning elementlariga havola va ko’rsatkichlarni bekor qilishiga olib kelishi mumkin. Iteratorlar nafaqat elementlarni olish, balki ularni o‘zgartirish imkonini beradi.

5.7-dastur. Iterator yordamida konteynerni elementlari qiymatini o‘zgartirish.

```
#include "stdafx.h"

#include <iostream>
#include <vector>
using namespace std;
int main()
{
     vector<int> myvector;
    for (int i = 0; i < 15; i++)
    {
        myvector.push_back(rand() % 10);
```

```
    }
     vector<int>::iterator iter = myvector.begin();
    //auto iter = myvektor.begin();
    while(iter!=myvector.end())
    {
        *iter = (*iter) * (*iter);
        ++iter;
    }

    for(iter = myvector.begin();
iter!=myvector.end(); ++iter)
    {
        cout << *iter << " | ";
    }
    cout << endl;
    system("pause");
    return 0;
}
```

5.7-dastur. Output
1 | 49 | 16 | 0 | 81 | 16 | 64 | 64 | 4 | 16 | 25 | 25 | 1 | 49 | 1 |

Dasturda while takrorlanish operatori konteynerning elementlarini iteratorga olib, o‘zini o‘ziga ko‘paytirib, yana shu iteratorga yozadi. Shuning uchun ekranga konteynerdagi sonlarning kvadratlari chiqadi.

O‘zgarmas iteratorlar. Agar konteynerda o‘zgarmas qiymatli elementlar bo‘lsa, bu holda konteyner elementlariga murojaat qilish uchun o‘zgarmas iteratorlardan foydalanishsh kerak. Buning uchun const_iteratop tipi ishlatiladi. Bu iteratorlar faqat elementlarni sanash imkonini beradi. O‘zgartirish mumkin emas.

```
vector<int>::const_iterator iter ;

     for(iter = myvector.begin();
iter!=myvector.end(); ++iter)
    {
        cout << *iter << " | ";
        //*iter = (*iter) * (*iter);
    }
```

O‘zgarmas iteratorning qiymatlarini olish uchun cbegin() va cend() funksiyalari ham ishlatiladi. Agar iterator o‘zgarmas deb olinmagan bo‘lsa ham, bu funksiyalar uni o‘zgarmas qilib beradi. Shungdek, o‘zgartirish mumkin emas bo‘ladi.

5.8-dastur. O‘zgarmas iteratordan foydalanish.

```
// ConsoleApplication1.cpp : main project file.

#include "stdafx.h"
#include <iostream>
#include <vector>
using namespace std;

int main(){
    int myints[] = {16,2,77,29,28};
    const vector<int> myvector (myints, myints +
sizeof(myints) / sizeof(myints[0]) );
    vector<int> myvector_one;
    for (int i = 0; i < 15; i++)
        myvector_one.push_back(rand() % 10);

    vector<int>::const_iterator it;
    for (it = myvector.begin(); it !=
myvector.end(); ++it)
        cout << ' ' << *it;
    cout << endl;
    for (it = myvector_one.cbegin(); it !=
myvector_one.cend(); ++it)
        cout << ' ' << *it;
    cout << endl;
    system("pause");
    return 0;
}
```

5.8-dastur.Output

```
    16 2 77 29 28
 1 7 4 0 9 4 8 8 2 4 5 5 1 7 1
```

Teskari iteratorlar. Bu iteratorlar konteynerning elementlarini teskari olishni amalga oshiradi. Buning uchun reverse_iterator tipi ishlatiladi. Bu tipdagi iteratorlarga konteynerning elementlarini olish uchun rbegin() i rend() funksiyalaridan foydalaniladi.

```
#include "stdafx.h"
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    int myints[] = { 1, 2, 3, 4, 5 };
    vector<int> v (myints, myints+
sizeof(myints)/sizeof(myints[0]));
    for (vector<int>::reverse_iterator iter =
v.rbegin(); iter != v.rend(); ++iter)
            std::cout << *iter << " | ";

    cout << endl;
    system("pause");
    return 0;
}
```

Agar konteynerni o‘zgartirishdan himoyaya qilish va teskarisi kerak bo‘lsa, const_reverse_iterator tipidagi iteratordan va uning crbegin() va crend() funksiyalaridan foydalanish maqsadga muvofiqdir. Yuqoridagi dasturga quyidagi fragmentni yozish yetarli bo‘ladi.

```
for (vector<int>::const_reverse_iterator iter = v.crbegin(); iter !=
v.crend(); ++iter)
    std::cout << *iter << " | ";
```

Shuningdek iteratorlari qo‘shima amallarga ham ega (list va forward_list konteynerlaridan tashqari):

- **iter + n** - n ta pozitsiya keyingi elementni ko‘rsatuvchi iteratorni qaytaradi.
- **iter – n** - n ta pozitsiya oldingi elementni ko‘rsatuvchi iteratorni qaytaradi.
- **iter += n** - iteratorni n - chi pozitsiyaga o‘tkazadi
- **iter -= n** - iteratorni n - chi pozitsiyaga o‘tkazadi
- **iter1 - iter2** - iter1 va iter2 lar orasidagi pozitsiyalar sonini qaytaradi

- [>], [>=], [<], [<=] - taqqoslash amallari. Agar iterator oxiriga yaqin elementni ko‘rsatsa boshqasidan katta.

5.9-dastur. Iteratorning qo‘shimcha amallaridan foydalanish.

```
#include "stdafx.h"
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    int myints[] = { 1, 2, 3, 4, 5 };
    vector<int> v (myints, myints+
sizeof(myints)/sizeof(myints[0]));
    for (auto i = v.begin(); i < v.end(); i++)
    {
        cout << *i << " | ";
    }
    cout << endl;
    auto it1 = v.begin();
    auto it2 = it1 + 2;
    cout << "it1 + 2 => " << *it2 << endl;
    auto it3 = v.end() - 3;
    cout << "it1 - 3 => " << *it3 << endl;
    it2 += 2 ;
    cout << "it2 + 2 => " << *it2 << endl;
    it3 -= 2;
    cout << "it3 - 2 => " << *it3 << endl;
    cout << "it2 - it1 => " << it2 - it1 << endl;
    cout << "(it1 < it2) => " << (it1 < it2) <<
endl;
    cout << "(it1 >= it2) => " << !(it1 < it2) <<
endl;
    cout << "(it1 != it2) => " << !(it1 == it2) <<
endl;
    cout << "(it1 == it2) => " << (it1 == it2) <<
endl;
    cout << "(it1 != it2) => " << (it1 != it2) <<
endl;
    cout << endl;
```

```
    system("pause");
    return 0;
}
```

5.9-dastur.Output

```
1 | 2 | 3 | 4 | 5 |
it1 + 2 => 3
it1 - 3 => 3
it2 + 2 => 5
it3 - 2 => 1
it2 - it1 => 4
(it1 < it2) => 1
(it1 >= it2) => 0
(it1 != it2) => 1
(it1 == it2) => 0
(it1 != it2) => 1
```

Taqqoslash amallari uchun ko‘chirib o‘tish iteratorlari va bazaviy iteratorlar uchun ba‘zi C++ dasturlashda farqlari bor.

5.1-jadval. Taqqoslash amallari yozish uslublari.

Ko‘chirib o‘tish iteratorlari uchun.	Ekvivalent bazaviy iteratorlari uchun.
move_ita == move_itb	basic_ita == basic_itb
move_ita != move_itb	!(basic_ita == basic_itb)
move_ita < move_itb	basic_ita < basic_itb
move_ita <= move_itb	!(basic_itb < basic_ita)
move_ita > move_itb	basic_itb < basic_ita
move_ita >= move_itb	!(basic_ita < basic_itb)

Xotirani taqsimlovchilar va ularga qo‘yilgan talablar. Dinamik xotira bilan ishlashda maxsus struklarni, fragmentlardan foydalanish bo‘lmasa, ko‘pigina algoritmlardan foydalanish samarasiz bo‘lshi mumkin. Bunga misol sifatida ikkita holatni ko‘rib chiqamiz. new va delete operatorlarining qayta aniqlanib yuklanishida sintaktik konstruktorlar kichikroq bo‘ladi va dastur lokalizatsiya qilish oddiy bo‘ladi. Shuningdek protsessordagi amallar tizimida ham qayta aniqlash sodir bo‘ladi.

Avvalo, xotirani ishlashini tezlashtirishda aqlli allocator qanchalik foydasi borligini aniqlash lozim. Buning uchun oddiy test misollarini

(C++ va C#) tillarida ko‘rib chiqaylik (bu xotira bilan ishlash uchun yaxshi menejer hisoblanadi va obyektlarni avlodlarga ajratadi, turli o‘lchamdagi obyektlar uchun turli pullardan (joylar) foydalanadi).

C++ dagi dastur

```
class Node {
public:
   Node* next;
};
// ...
for (int i = 0; i < 10000000; i++) {
   Node* v = new Node();
}
```

C# dagi dastur

```
class Node
{
  public Node next;
}
// ...
for (int l = 0; l < 10000000; l++)
{
  var v = new Node();
}
```

Dasturlar sharsimon vakum usulini qo‘llab ham vaqt bo‘yicha solishtirilsa 10 barovar (62 ms va 650 ms) farq bo‘lmoqda. Shuningdek C# ishini tugatdi, ammo C++ da esa yaxshigina xotira ajratilgan, buni hali o‘chirish kerak.

Pul obyektlari. Pul obyektlari uchun aniq yechim - OS dan katta xotira blokini olish va uni teng blokli o‘lchamlarga bo‘lishdir sizeof(node). Xotirani ajratishda blokni puldan olinadi va uni bo‘shatishda uni pulga qaytariladi. Pulni tashkil qilishning eng oson usuli - bitta aloqa stekni (stack) ishlatishdir.

Maqsad dasturda minimal qayta aniqlashlar bo‘lgani uchun, BlockAlloc sinfni merosxo‘ri sifatida Node sinfini tanlashdir.

```
class Node : public BlockAlloc<Node>
```

Avvalo, OS yoki C-runtime dan katta bloklar uchun bir pul olish kerak. Buni malloc va free funksiyalari bilan tashkil qilish mumkin, lekin katta samaradorligi uchun (ortiqcha abstraksiya bosqichini

o‘tkazish uchun), Virtualloc/Virtualfreye funksiyalaridan foydalanish maqsadga muvofiq. Bu vazifalar 4 karrali bo‘lgan bloklarga xotira ajratadi va 64 karrali bo‘lgan bloklar jarayon manzili uchun oraliq saqlab turadi. Bir vaqtning o‘zida commit va reserve imkoniyatlari ko‘rsatilgan holda yana bir darajada ko‘tarilib, manzil oraliqni rezervlash va bitta chaqirish bilan xotira bloklarini ajratamiz.

PagePool sinfi dastur fragmenti.

```
inline size_t align(size_t x, size_t a) { return
((x-1) | (a-1)) + 1; }
//#define align(x, a) ((((x)-1) | ((a)-1)) + 1)

template<size_t PageSize = 65536>
class PagePool
{
public:
        void* GetPage() {
                void* page = VirtualAlloc(NULL,
PageSize, MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);
                pages.push_back(page);
                return page;
        }

        ~PagePool() {
                for (vector<void*>::iterator i =
pages.begin(); i != pages.end(); ++i) {
                        VirtualFree(*i, 0,
MEM_RELEASE);
                }
        }
private:
        vector<void*> pages;
};
```

Berilgan o‘lchamga mos, pul bloklarni hosil qilish uchun BlockPool sinfini yaratamiz.

BlockPool sinfining dastur fragmentlari:

```
template<class T, size_t PageSize = 65536, size_t
Alignment = 8 /* sizeof(void*) */>
class BlockPool : PagePool<PageSize>
{
```

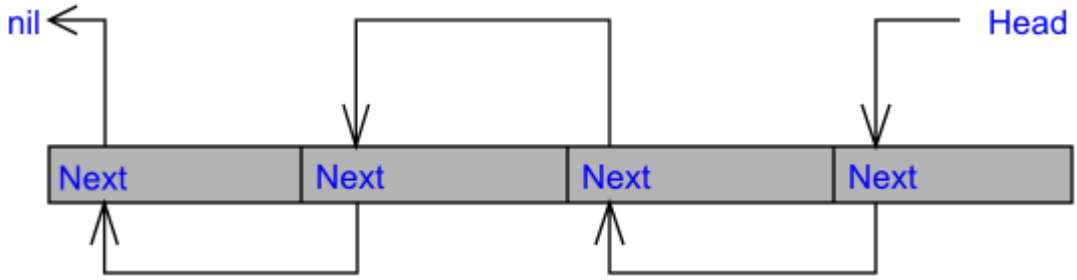
```
public:
        BlockPool() : head(NULL) {
                BlockSize = align(sizeof(T),
Alignment);
                count = PageSize / BlockSize;
        }

        void* AllocBlock() {
                // todo: lock(this)
                if (!head) FormatnewPage();
                void* tmp = head;
                head = *(void**)head;
                return tmp;
        }

        void FreeBlock(void* tmp) {
                // todo: lock(this)
                *(void**)tmp = head;
                head = tmp;
        }
private:
        void* head;
        size_t BlockSize;
        size_t count;

        void FormatnewPage() {
                void* tmp = GetPage();
                head = tmp;
                for(size_t i = 0; i < count-1; i++)
{
                        void* next = (char*)tmp +
BlockSize;
                        *(void**)tmp = next;
                        tmp = next;
                }
                *(void**)tmp = NULL;
        }
};
```

Potoklar orasidagi sinxronzatsiyani tashkil qilish uchun berilgan manzil ***todo: lock(this)*** izohga olib qo‘yilgan. Buning uchun EnterCriticalSection yoki boost::mutex dan foydalanish ham mumkin.

Agar biror narsa yozilgan bo‘lsa, bloklar majmuini formatlashda abstrakt FreeBlock pulga blok qo‘shish ishlatilmaydi. O‘ylab ko‘ring.

```
for (size_t i = 0; i < PageSize; i += BlockSize)
FreeBlock((char*)tmp+i);
```

FIFO tamoyilidagi bloklar majmui teskari bo‘lib qoladi.



5.1-rasm. FIFO tamoyilidagi bloklar majmui.

Puldan ketma-ket so‘ralgan bir necha blok manzillar kamayib ketadi. Shuningdek protsessor orqaga qaytishni yoqtirmaydi, chunki uning Prefetch amalii buziladi (**UPD**: zamonaviy protsessorlar uchun tegishli emas). Agar takrorlanish bilan almashtirilsachi?

```
for (size_t i = PageSize-(BlockSize-(PageSize%BlockSize)); i != 0; i -= BlockSize) FreeBlock
```

Takrorlanishda almashtirishlar manzillar bo‘yicha orqaga bajariladi. Tayyorgalik ko‘rilganidan so‘ng, sinf obyektni yozish mumkin:

```
 template<class T>
class BlockAlloc {
public:
   static void* operator new(size_t s) {
      if (s != sizeof(T)) {
         return ::operator new(s);
      }
      return pool.AllocBlock();
   }
   static void operator delete(void* m, size_t s) {
      if (s != sizeof(T)) {
         ::operator delete(m);
      } else if (m != NULL) {
         pool.FreeBlock(m);
      }
```

```
    }
    static void* operator new(size_t, void* m) {
       return m;
    }
    static void operator delete(void*, void*) {
    }

private:
    static BlockPool<T> pool;
};
template<class T> BlockPool<T> BlockAlloc<T>::pool;
```

Sinfda ular ishlaganda nima uchun if (s != sizeof(T)) kerak. T bazaviy sinfdan merosxo‘r sinf yaratilgan va o‘chirilgan bo‘lsa? degan savol bo‘lishi mumkin.

Merosxo‘rlar odatda oddiy new/delete operatorlari bilan ishlatiladi, shuningdek BlockAlloc ni ham joylatirish mumkin. Bu yo‘l bilan oson va xavfsiz dasturda biror narsa buzib qo‘rqmasdan, pullar foydalanishga qaratilgan sinflar aniqlash mumkin. Bir nechta merosxo‘rlar ham bu bilan yaxshi ishlaydi.

BlockAlloc dan merosxo‘r sifatida Node ni olamiz va yana sinov o‘tkazamiz.

Sinov natijasiga qarasak, 5 barovar tez va 120 msda bajarilmoqda. Ammo, C# allokator hali ham yaxshi. Mumkin u yerda oddiy bo‘lmagan bir aloqali ro‘yxatdan foydalanilgandir. Agar, new operatoridan so‘ng zudlik bilan delete operatori ishlatilsa, xotira kamroq ishlatiladi va xeshda ma’lumotlar ham kamayadi va natijani 62 msda olish mumkin. Qiziqarli tomoni shundaki, . NET CLR da GC ni kutmasdan pulga mos bo‘sh qolgan lokal o‘zgaruvchini tez qaytaradi.

Konteyner va uni qo‘llanilishi. Xotirada ko‘p hollarda merosxo‘r obyektlarini saqlovchi sinflar vujudga keladi. Ularning xotiradan o‘chirilishi bazaviy sinf bilan bog‘liqdir. Masalan, sinf bo‘lsin va tugun ichidagi matndan olingan merosxo‘rlari Node va Attribute hamda (char*) qatorlardan to‘ldirilgan. Yoki papkani qayta o‘qish va yana o‘zgartirishda, bir marta yuklanadi fayl menejeri fayllar va kataloglar ro‘yxati.

Yuqorida ko‘rsatilgandek, new operatoriga nisbattan delete operatori foydalanish qiyinroq va muammoliroq bo‘ladi. Bazaviy sinf obyekti bilan bog‘liq katta blokda merosxo‘r sinf obyektlari uchun xotira ajratish kerak. Bazaviy sinf obyektini yo‘q qilish paytida,

destruktor odatdagidek, merosxo‘r sinf uchun chaqiriladi, lekin xotirani qaytarishingiz shart emas — bu katta blokda ozod qilinadi.

PointerBumpAllocator sinf yaratamiz. Bu sinf katta blokdan turli o‘lchamdagi qismlarni kesib tashlash va eskisi tugagach, yangi katta blokni tanlashga imkon beradi.

PointerBumpAllocator sinfining dastur fragmenti.

```
template<size_t PageSize = 65536, size_t Alignment
= 8 /* sizeof(void*) */>
class PointerBumpAllocator
{
public:
        PointerBumpAllocator() : free(0) { }

        void* AllocBlock(size_t block) {
                // todo: lock(this)
                block = align(block, Alignment);
                if (block > free) {
                        free = align(block,
PageSize);
                        head = GetPage(free);
                }
                void* tmp = head;
                head = (char*)head + block;
                free -= block;
                return tmp;
        }

        ~PointerBumpAllocator() {
                for (vector<void*>::iterator i =
pages.begin(); i != pages.end(); ++i) {
                        VirtualFree(*i, 0,
MEM_RELEASE);
                }
        }

private:
        void* GetPage(size_t size) {
                void* page = VirtualAlloc(NULL,
```

```
size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
                pages.push_back(page);
                return page;
        }

        vector<void*> pages;
        void* head;
        size_t free;
};
typedef PointerBumpAllocator<> DefaultAllocator;
```

Nihoyat, new va delete operatorlari bilan childObject ni tuzamiz va berilgan allocator bilan murojaat qilamiz:

```
template<class T, class A = DefaultAllocator>
struct ChildObject
{
    static void* operator new(size_t s, A&
allocator) {
        return allocator.AllocBlock(s);
    }
    static void* operator new(size_t s, A*
allocator) {
        return allocator->AllocBlock(s);
    }

    static void operator delete(void*, size_t) { }
    static void operator delete(void*, A*) { }
    static void operator delete(void*, A&) { }
private:
    static void* operator new(size_t s);
};
```

Child sinfiga o‘zgaruvchilarni qo‘shish uchun barcha e‘lonlarni new operatori orqali amalga oshirish kerak bo‘ladi. new operatori quyidagicha bo‘ladi.

```
new (... parametrlar... ) ChildObject (...konstruktor parametrlari... )
```

Qulaylik uchun A& va A* uchun new operatorlarni qo‘llaymiz.

```
node = new(allocator) XmlNode(nodename);
```

Agar allokator ajratuvchi sifatida qo‘shilsa, ikkinchisidan foydalanish qulayroq:

```
node = new(this) XmlNode(nodename);
```

Bundan tushinarli bo‘ladiki, ortiqcha belgilardan qochishda amallarni bo‘lish uchun ko‘rsatkich va havolalar konvertatsiya bo‘ladi.

Obyekt yaratishda qaysi new operatoridan foydalanganligiga qaramasdan, delete operatori yordamida o‘chirish amalga oshirilmaydi, kompilyatorning o‘zi standart delete operatoridan foydalanadi. Quyidagicha sintaktik asosida:

```
delete node;
```

Agar ChildObject sinfi obyekti yoki uning merosxo‘ri obyektidan foydalanayotgan bo‘lsangiz, new istisno vaqtida operatoriga mos delete orperatori chaqiriladi. Shuning uchung bu obyektdan foydalanganda birinchi size_t parametrni void*ga o‘zgartirish lozim.

new operatorini private bo‘limiga joylashtirish, uni allokatorsiz ishlashaga ruxsat bermaydi.

Standart bo‘yicha taqsimlovchilar.

1.Tuzilmali o‘zgaruvchisi uchun dinamik xotira ajratish. Tuzilmali o‘zgaruvchisi uchun xotira ajratish va bo‘shatish. Quyidagi tavsifga ega bo‘lgan data tuzilmasi berilgan bo‘lsin:

```
struct Date
{
    int day;
    int month;
    int year;
};
```

struct Date tipidagi o‘zgaruvchi uchun xotiradan joy ajratish va foydalanish uchun quyidagicha dastur fragmentini yozish kerak:

5.10-dastur. xotiradan joy ajratish va foydalanish.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

// tuzilma, kun, oy, yil;
struct Date
{
    int day;
    int month;
```

```
    int year;
};

int main(){
    // new operatori bilan xotiradan joy ajratish
    struct Date * pd; // struct Date ga ko‘rsatkich

    try{
        pd = new struct Date; // xotiraga joy
ajratish
    }
    catch (bad_alloc ba)    {
        cout << "Xorita ajratilmadi" << endl;
        return -1;
    }

    // Agar xotira ajratilgan bo‘lsa, unga
16.14.2020 qimat qilib beramiz.
    pd->day = 16;
    pd->month = 04;
    pd->year = 2020;

    cout << pd->day << ":" << pd->month << ":" <<
pd->year << endl;

    //  pd o‘zgaruvchisini xotiradan bo‘shatish
    delete pd;
    system("pause");
    return 0;
}
```

2. Sinf obyekti uchun dinamik xotira ajratish.

CDayWeek sinf obyektining ko‘rsatkichiga xotira ajratish va foydalanishni keltiramiz.

5.11-dastur. xotira ajratish va foydalanish.

```
// ConsoleApplication1.cpp : main project file.

#include "stdafx.h"
#include <iostream>
```

```
using namespace std;

// sinf, hafta kunlari
class CDayWeek{
    int d;
    public:
    // konstruktor
    CDayWeek() { d = 1; }

    // funksiyalari
    int Get(void) { return d; }
    void Set(int nd) {
        if ((1<=nd)&&(nd<=7))
            d = nd;
    }
};

int main(){
    //  new bilan xotiradan joy ajratish
    CDayWeek *p;

    try {
        //  p ko‘rsatkich uchun xotiradan joy
ajratish
        p = new CDayWeek(); // CDayWeek sinf
konstruktorini chaqirish
    }
    catch (bad_alloc ba) {
        cout << "Xotira ajratilmadi" << endl;
        cout << ba.what() << endl;
        return -1;
    }

    //  qimat berish
    p->Set(3);
    cout << p->Get() << endl;
    //  p ko‘rsatkichdan xotirani bo‘shatish
    delete p;
```

```
    system("pause");
    return 0;
}
```

3. new operatori yordamida massivlarga xotiradan joy ajratish va foydalanish. new operatori massiv uchun xotira ajratish uchun foydalanish mumkin. Massivga xotira ajratishda new operatorning umumiy shakli:

```
ptrArray = new type[size];
```

Bunda, *ptrArray* – xotira ajratilishi kerak bo‘lgan massiv nomi, type - elementlarning tiplari, size - ular bazaviy va ixtiyoriy bo‘lishi mumkin, massiv o‘lchami (elementlar soni).

delete operatori yordamida massiv obyektidan xotirani bo‘shatish uchun quyidagi fragment yoziladi:

```
delete[] ptrArray;
```

4. Bazaviy tipli ko‘rsatkich massiviga dinamik xotira ajratish va foydalanish. Quyida float tipidagi ko‘rsatkich massivi uchun xotira ajratadi. So‘ngra qator elementlari ixtiyoriy qiymatlar bilan to‘ldiriladi. Shundan so‘ng ajratilgan xotira delete [] operatori tomonidan bo‘shatiladi.

```
float * ptrArray;
ptrArray = new float[10];
int d;
for (int i = 0; i < 10; i++)
    ptrArray[i] = i * 2 + 1;

d = ptrArray[3];
delete[] ptrArray;
```

5. Tizimli o‘zgaruvchilar massivi uchun xotira ajratish va undan foydalanish. 3 ta talaba tuzilmalar massivi uchun xotira ajratish va bo‘shatish ko‘rsatilgan. shuningdek, tuzilmalar massivining element maydonlariga murojaat qilish ham ko‘rsatilgan.

5.12-dastur. Xotira ajratish va foydalanish.

```
#include "stdafx.h"
#include <iostream>
using namespace std;


struct TStudent
```

```
{
    string name;
    string numBook;
    int course;
    float rank;
};

int main(){
    int n_students = 3;
    TStudent * pS;

    try{

        pS = new struct TStudent[n_students];
    }
    catch (bad_alloc ba){
        cout << "Xotira ajratilmadi " << endl;
        cout << ba.what() << endl;
        return -1;
    }

    pS->name = "Aliyev Akbar";
    pS->numBook = "965874123";
    pS->rank = 3.93;
    pS->course = 1;

    (pS + 1)->name = "Bahromov Batir";
    (pS + 1)->numBook = "20930032";
    (pS + 1)->rank = 4.98;
    pS[1].course = 3;

    pS[2].name = "Qobilov Usmon";
    pS[2].numBook = "l2l28983";
    pS[2].rank = 4.32;
    pS[2].course = 2;

    cout << pS->name.c_str() << endl;
    cout << pS[1].rank << endl;
```

```
    cout << (pS + 2)->numBook.c_str() << endl;

    delete[] pS;
    system("pause");
    return 0;
}
```

6. Massiv obyektlari uchun xotira ajratish va undan foydalanish. new operatoridan foydalangan holda massiv obyektlari uchun xotira ajratish ko‘rsatiladi va delete operatori bilan xotiradan massiv obyektlarini bo‘shatish ko‘rsatiladi.

5.12-dastur. Xotira ajratish va foydalanish.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

class CMonth{
    private:
    int month;

    public:
    CMonth() { month = 1; }

    CMonth(int nmonth){
        if ((1 <= nmonth) && (nmonth <= 12))
            month = nmonth;
        else
            month = 1;
    }

    int Get(void) { return month; }
    void Set(int nmonth){
        if ((1 <= nmonth) && (nmonth <= 12))
            month = nmonth;
    }
};

int main(){
    int nMonths = 12;
```

```
    CMonth * pM;

    try {
        pM = new CMonth[12];
    }
    catch (bad_alloc ba)
    {
        cout << "Xotira ajratimadi: " << endl;
        cout << ba.what() << endl;
        return -1;
    }

    for (int i = 1; i <= 12; i++)
        pM[i - 1].Set(i);

    cout << pM[3].Get() << endl;
    cout << pM[5].Get() << endl;

    delete[] pM;
    system("pause");
    return 0;
}
```

Yuqoridagi dastur fragmentida massiv obektlar ichki o‘zgaruvchi 1 qiymatiga initsializatsiya qilinadi, chunki bu qiymat Cmonth() parametrlarisiz konstruktorda o‘rnatiladi. Bu konstruktor massiv uchun initsializator vazifasini bajaradi. Biroq, sinf boshqa konstruktorni amalga oshiradi, ya‘ni konstruktor bilan 1 qiymatli parametr yoki parametrli konstruktorni. C++ sintaktikiga ko‘ra, massiv obektlarini parametrli konstruktor bilan ishga tushirib bo‘lmaydi. Shuning uchun, Cmonth sinf parametrlarsiz konstruktor amalga oshirish kerak. Cmonth() parametrsiz konstruktor sinf dastur fragmentidan olib tashlansa, massiv obektlar uchun xotira ajratish mumkin bo‘lmaydi. Bitta obyektlar uchun xotira ajratish mumkin, massiv uchun mumkin emas.

Agar ma’lum bir sinf massiv obyektlari uchun xotira ajratish kerak bo‘lsa, bu sinf parametrlarsiz konstruktorga ega bo‘lishi va bajarilishi kerak.

7. Massiv o‘lchamini dinamik kattalashtirish (kichiklashtirish) uchun xotira ajratish va foydalanish. Tuzilmalar uchun xotirani qayta ajratish, tuzilmalarni ishga tushirish. masalan, Dayweek tuzilishi tip

uchun xotira qayta ajratish jarayoni ko‘rsatiladi. Xotirani dinamik ravishda ajratish va qayta ajratish bu usulning xotirani statik ajratishga nisbatan asosiy afzalligi hisoblanadi. Kerak bo‘lganda va kerak bo‘lganda dasturda xotira ajratishingiz mumkin. Dayweek tuzilishi parametrlarsiz konstruktor yordamida amalga oshiriladi (yuqoridagidek).

5.13-dastur. Xotira ajratish va foydalanish.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

struct DayWeek{
    int day;
    DayWeek() { day = 1; }
};

int main(){
    DayWeek * p;

    try {
        p = new DayWeek[5];
    }
    catch (bad_alloc ba){
        cout << "Xotira ajratilmadi" << endl;
        cout << ba.what() << endl;
        return -1;
    }
    for (int i = 0; i < 5; i++)
        p[i].day = i + 1;
    int d;
    d = (p + 1)->day;

    DayWeek * p2;

    try    {
        p2 = new DayWeek[7];
    }
    catch (bad_alloc ba){
        cout << "Xotira ajratilmadi" << endl;
```

```
        cout << ba.what() << endl;
        return -1;
    }
    for (int i = 0; i < 5; i++)
        p2[i] = p[i];
    delete[] p;

    p = p2;

    for (int i = 5; i < 7; i++)
        p[i].day = i + 1;

    d = p[5].day;
    d = (p + 6)->day;

    delete[] p;

    return 0;
}
```

main() funkssiyasida xotira avval 5 ta tuzidmadan iborat massiv uchun ajratiladi. Keyin bu xotira 7 tuzilmali massiv uchun qayta ajratiladi (reallocated). Buning uchun qo‘shimcha p2 ko‘rsatikich ishlatiladi. Qayta taqsimlashda avval p2 (7 element) uchun xotira ajratiladi. So‘ngra ma’lumotlar p dan p2 ga ko‘chiriladi. Bu p ko‘rsatkich uchun ajratilgan edi, xotira bo‘shatiladi (5 ta elementdan). Keyingi qadamda p qiymat p2 ga o‘rnatiladi. Shunday qilib, har ikki ko‘rsatkich bir xil xotira maydoni ko‘rsatadi. Massiv bilan ishlagandan so‘ng, p massiv uchun xotira ajratiladi.

Bu yuqorida ko’rsatib o‘tilgan dasturlash dinamik xotirani ajratish va foydalanishning standart shakllari bo‘lib hisoblanadi.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Standart algoritmlar kutubxonasi nima uchun yaratilgan.
2. Algoritm kutubxonasi tomonidan taqdim etiladigan funksiyalar odatda nechta toifaga bo‘linadi va qaysilar?
3. Mutatorlar nima va qanday vazifani amalga oshiradi.
4. Berilgan qiymat bo‘yicha elementlarni qidirish algoritmi qaysi va nechta parametr qabul qiladi?

5. Konteynerda mavjud elementlar orasida ma’lum bir qiymat saqlaydigan element qidirish uchun qaysi funksiya ishlatiladi?

6. Konteynerdan berilgan elementlarini sanash vazifasini bajaruvchi algoritmlarni sanab bering?

7. Funksiya taqqoslash uchun parametr sifatida ikkita parametr oladi. Bu qaysi funksiya.

8. Ikki obyektning qiymatlarini almashtirish algoritmni ayting.

9. Tartiblangan to‘plamlarni boshqa tartiblangan to‘plamni o‘z ichiga olishi yoki olmasligini tekshirish algoritmini ayting.

10. Iteratorlar nima uchun foydalaniladi.

11. **begin()** va **end()** funksiyalaridan nima maqsadda foydalaniladi.

12. Agar **begin()** va **end()** funksiyalari o‘zaro teng bo‘lmasa, ularning orasidan kamida nechta element bor.

13. Keyingi elementga murojjat qilish uchun iteratorni harakatlantirish amalni ayting.

14. Agar konteynerda o‘zgarmas qiymatli elementlar bo‘lsa, bu holda konteyner elementlariga murojaat qilish uchun nimadan foydalanishsh kerak.

15. cbegin() va cend()funksiyalaridan nima saqsadda foydalaniladi.

16. Agar konteynerda o‘zgaruvchi qiymatli elementlar bo‘lsa, bu holda konteyner elementlariga murojaat qilish o‘zgarmas iteratorlardan foydalanishsh mumkinmi.

17. Iteratorlari qo‘shima amallar qaysi konteynerlar uchun ishlatilmaydi.

18. Qaysi amal iter1 va iter2 lar orasidagi pozitsiyalar sonini qaytaradi.

19. (move_ita != move_itb) va !(move_ita == move_itb) munosabatlar bir xilmi, bir xil bo‘lsa nima uchun, farqi bo‘lsa, nima uchun.

20. Xotirani taqsimlovchilar va ularga qo‘yilgan talablar nima uchun kerak.

21. Merosxo‘rlar odatda oddiy new/delete operatorlari bilan ishlatiladi, shuningdek BlockAlloc ni ham nima uchun joylatirish kerak.

22. Agar, new operatoridan so‘ng zudlik bilan delete operatori ishlatilsa, xotira kamroq ishlatiladi va qaerdagi ma’lumotlar ham kamayadi.

23. Qaysi operatordan new operatori yoki delete operatori foydalanish qiyinroq va muammoliroq, sababini tushintirib bering.

24. new operatori qaysi bo‘limiga joylashtirilsa, uni allokatorsiz ishlashaga ruxsat bermaydi.

25. Tuzilmali o‘zgaruvchisi uchun xotira ajratish va bo‘shatish uchun qanday amallar bajarish kerak.

26. Sinf obyekti uchun dinamik xotira ajratish uchun qanday amallar bajarish kerak.

27. new operatori yordamida massivlarga xotiradan joy ajratish va foydalanish uchun qanday amallar bajarish kerak.

28. Bazaviy tipli ko‘rsatkich massiviga dinamik xotira ajratish va foydalanish uchun qanday amallar bajarish kerak.

29. Ma’lum bir sinf massiv obyektlari uchun xotira ajratish kerak bo‘lsa, bu sinf qanday parametrli konstruktorga ega bo‘lishi kerak.

30. Massiv o‘lchamini dinamik kattalashtirish (kichiklashtirish) uchun xotira ajratish va foydalanish uchun qanday amallar bajarish kerak.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I	
	Standart algoritmlarga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`// Created by MBBahodir`
`#include "stdafx.h"`
 	
`#include <algorithm>`
	1. Dasturda nechta global funksiya yaratilgan.
	_____
	_____
	_____
	_____

```
#include <array>
#include <iostream>
#include <utility>
#include <vector>
#include <string>

using namespace std;
bool myfunction (int
i, int j) { return
i<j; }
int main()
{
    array<int,10> arr
= {1, 3, 5, 7, 9, 11,
13, 15, 17, 19 };
    int container[] =
{5,10,15,20,25,30,35,
40,45,50};
    int continent[] =
{40,30,20,10};
    string myStr[] =
{"air","water","fire"
,"earth"};
    vector<string>
foo (myStr, myStr +
sizeof(myStr)/sizeof(
myStr[0]));
    vector<string>
bar (4);
    int myints[] =
{10,20,30,30,20,10,10
,20};
    vector<int>
v(myints,myints+8);

    cout << "Qidirish
```

2. Massivda berilgan massiv elementlari borligini tekshiruvchi fragmentni o‘zgartiring.

_____
_____
_____
_____

3. Dasturning qaysi fragmentini izohlab qo‘ysa, massivda berilgan massiv elementlari bor deb chiqadi.

_____
_____
_____
_____
_____
_____

4. Massivda berilgan massiv elementlari bor degan fragment qachon ishlamaydi.

_____
_____
_____

5. Move amali necha marta bajarilgan.

_____
_____
_____

6. Birinchi qaysi kutubxonaga ta‘luqli.

_____
_____
_____

7. Ikkinchi qaysi kutubxonaga ta‘luqli.

_____
_____

```
va o‘zgartirish uchun
son kirit: ";
    int search;
    int replace;
    cin >> search >>
replace;
    auto found =
find(arr.begin(),
arr.end(), search);
    if (found ==
arr.end())  {
        cout <<
search << "
topilmadi. " << endl;
    } else {
        *found =
replace;
    }
    for (int i :
arr){
        cout << i <<
' ';
        container[i]
= i;
    }
    cout << endl;
     sort
(container,container+
10);
     sort
(continent,continent+
4);
     if
(includes(container,c
ontainer+10,continent
,continent+4,
myfunction) )
     cout << "ichma
```

8. Foo vektorning elementlari ekranga chiqarish fragmentini kiriting

_____
_____
_____
_____
_____
_____
_____

```
ich massiv !\n";
     else
     cout << "ichma
ich massiv emas !\n";
     cout << endl;

     move (
foo.begin(),
foo.begin()+4,
bar.begin() );

    foo = move (bar);

    cout << endl;
    system("pause");
    return 0;
}
```

9. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
10. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

IKKINCHI ASSISMENT TOPSHIRIG‘I

Iteratorlar va ularning qo‘llanilishiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi

dastur	topshiriqlar

`#include "stdafx.h"`
`#include <iostream>`
`#include <vector>`
`  using namespace std;`
`int main()`
`{`
`    vector<int>`
`myvector;`
`    for (int i = 0; i <`
	1. Dasturda auto iter = myvector.begin(); uchun iterator yozing.
	_____
	_____
	_____
	_____
	_____
	_____
	_____

```
15; i++)
    {

    myvector.push_back(
rand() % 100);
    }
    int myints[] =
{16,2,77,29,28};
    const vector<int>
myvector_one (myints,
myints + sizeof(myints)
/ sizeof(myints[0]) );

    auto iter =
myvector.begin();
    while(iter!=myvecto
r.end())
    {
        cout << *iter
<< " | ";
        ++iter;
    }
/*
vector<int> v (myints,
myints+
sizeof(myints)/sizeof(m
yints[0]));
    for (auto i =
v.begin(); i < v.end();
i++)
    {
        cout << *i << "
| ";
    }
*/

    cout << endl;
    system("pause");
```

2.Dasturda vector<int> myvector; uchun o‘zgarmas iterator yarating.

_____
_____
_____
_____
_____
_____

3. Dasturdagi ortiqcha kutubxonalarni o‘chirib tashlang va sabablarini ayting.

_____
_____
_____

4. Ikkita vektorni o‘zaro solishtiring.

_____
_____
_____

5. Vektorlarni teskari chiqarish dastur fragmentlarini yozing.

_____
_____
_____

6. Vector saralang va vektordagi 2,5,7 elementlarni pozitsiyasini aniqlang.

_____
_____
_____

7. Dasturdagi izohga olingan dastur fragmenti olib tashlansa, dasturda qanday o‘zgarishlar bo‘ladi va natijasichi.

_____
_____
_____

```
    return 0;
}
```

_____
_____

8. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
9. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

UCHINCHI ASSISMENT TOPSHIRIG‘I

Xotirani ajratish va foydlanishga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.
☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur

```
// Created by MBBahodir
#include <iostream>
using namespace std;
struct TStudent
{
    string name;
    string numBook;
    int course;
    float rank;
};
int main(){
    int n_students = 3;
    TStudent * pS;
    try{
        pS = new struct
TStudent[n_students];
    }
    catch (bad_alloc
ba){
        cout << "Xotira
```

topshiriqlar

1. Dasturda nechta yangi tipdagi obyektlari yaratilgan.

_____
_____
_____
_____

2. Dasturdagi barcha talabalar haqidagi ma’lumotni ekranga chiqaring.

_____
_____
_____
_____
_____
_____

```
ajratilmadi " << endl;
        cout <<
ba.what() << endl;
        return -1;
    }
    pS->name = "Aliyev
Akbar";
    pS->numBook =
"965874123";
    pS->rank = 3.93;
    pS->course = 1;
    (pS + 1)->name =
"Bahromov Batir";
    (pS + 1)->numBook =
"20930032";
    (pS + 1)->rank =
4.98;
    pS[1].course = 3;
    pS[2].name =
"Qobilov Usmon";
    pS[2].numBook =
"12128983";
    pS[2].rank = 4.32;
    pS[2].course = 2;
    cout << pS-
>name.c_str() << endl;
    cout << pS[1].rank
<< endl;
    cout << (pS + 2)-
>numBook.c_str() <<
endl;
    delete[] pS;
    system("pause");
    return 0;
}
```

3. Massiv o‘lchamini dinamik kattalashtirish uchun fragment yozing, masalan 5 talaba uchun.

_____
_____
_____
_____
_____
_____

4. Yangi tip uchun get() va set() funksiyalarini yarating.

_____
_____
_____

5. Massivdan berilgan qiymatni izlash dastur fragmentini yozing.

_____
_____
_____
_____
_____
_____
_____

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____

7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

2.2. Sonli sinflar bilan ishlash

📖 Sonli sinf va ularning funksiyalari asosida complex, valarray, slice, gsilce, indirect_array, mask_array sinflari va funksiyalari, interval sonli sinf va funksiyalari, sonli sinf elementlarini yaratish, foydalanishga asosida taqqoslash funksiyalari, dasturlashda o‘zlarini tutishlari, uslublari, amallari, talablari, vazifalari va usullari keltirilgan bo‘lib, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni mustahkamlash uchun 35 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 6 ta assisment topshirig‘i va har assismentda 7(6) ta topshiriq, jami 49ta topshiriq berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

🖋 ***Kalit so‘zlar.*** kutubxona, sinf, shablon, matematik amallari, kompleks son, to‘plam, to‘plam osti, to‘lam ustida amalar, saralash, interval son.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Tip tushunchasi, sinf va sinf obyekti, merosxo‘r sinf, to‘plam, massiv, elementga murojaat, funksiya va ko‘rsatkich, ma’lumotlarni kirish va chiqishi, dasturlashga oid dastlabki tushunchalar va C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👌 ***Bilib olasiz***. Sonli sinf va ularning funksiyalari, kompleks sonlar bilan ishlash usullari va funksiyalari, valarray soni obyekti bilan ishlash, muo‘rosxo‘r sinf asosida to‘plamlarni ustida amallarni bajarish, surish, shartlar asosida yangi to‘plamni hosil qilish, qism to‘plamlarni yaratish, interval son va uning funksiyalarni o‘rganishingiz mumkin.

REJA

1. Complex sonli sinf va funksiyalari
2. Vallaray sonli sinf va funksiyalari
3. Slice va gslice sonli sinflari va funksiyalari
4. Qo‘shimcha sonli sinflar

KIRISH

Hayot turmush tarzida foydalaniladigan sonlar to‘plamlardan boshqa sonlar to‘plami va ular ustida bajarilajigan amallar mavjud. Masalan, maktabdan bilamizki, kompleks sonlar, oraliq sonlar, qolaversa, zamonaviy ilmiy sohalarda turli fazolarda turlicha sonlar to‘plami ishlatiladi. Interval sonlari, noravshan to‘plamlar shular kabi. Bu sonlar to‘plami hayotdagi qaysidir tizimni modellashtirish va uni jarayonlariga oid masalalarni yechishga xizmat qiladi. Bu masalarni ayniqsa kompyuterda yechish kerak. Shuning uchun ba‘zi bir ko‘p

foydalaniladigan sonlar to‘plami uchun sinflar yaratilgan. Bu sonlarda ularning amallari, funksiyalari xususiyatlari keltirilgan. Bunday sonlarga complex, vallaray, slice, gslice kabi sinflar va muallif tomonidan 2013 yilda yaratilgan interval sinf kutubxonalarini misol qilib olish mumkin.

Complex (kompleks) sonli sinf va funksiyalari. C++ standarti kutubxonasidagi kompleks sonlar sinfi obyekt modelidan foydalanishning yaxshi namunasidir. Arifmetik amallar qayta aniqlash tufayli bu sinf obyektlari o‘rnatilgan ma’lumot tiplaridan biriga tegishli kabi ishlatiladi. Bundan tashqari, an‘anaviy arifmetik o‘zgaruvchilar va kompleks sonlar bir vaqtning o‘zida ixtiyoriy amallarda ishtirok etishi mumkin. (Eslatib o‘tamiz, bu yerda kompleks sonlar matematikasiga oid umumiy savollar bilan shug‘ullanmayapmiz. Matematika bo‘yicha kopleks sonlar nazariyasini o‘qish kerak).

Bu sinfni ishlatish uchun complex sarlavha fayl qo‘shilishi kerak. Kichik dastur fragmentiga qarang:

```
#include <complex>
// ...
complex <double> a;
complex <double> b;
// ...
complex< double > c = a * b + a / b;
```

Kompleks va arifmatik amallar o‘zaro hamkorlikda bajarailadi.

```
complex< double > a;
complex< double > complex_obj = a + 3.14159;
```

Kompleks tiplarni arifmetik tiplar bilan ham yaratish va ular uchun operator= ni ishlatish mumkin.

```
complex< double > complex_obj;
double dval = 3.14159;
complex_obj = dval;
// ....
int ival = 3;
complex_obj = ival;
```

Ammo teskarisi, ya‘ni kopmleks tipni arifmetik tipga tenglashtirish mumkin emas.

```
complex< double > complex_obj;
double dval = complex_obj;
```

Bunda arifmetik tipni akslantirish xato degan xabar beradi.

Kompleksning 2 qismi bo‘ladi haqiqiy va mavhum. Haqiqiy (real) yoki mavhum – qismini oddiy arifmetik tipga qiymat qilib berish

mumkin. Kompleks sonlar sinfi o‘z navbatida real va mavhum qismlarni qaytaruvchi ikkita funksiyaga ega. Biz sinf a’zolaridan foydalanish uchun sintaktiki yordamida ularni olishimiz mumkin:

```
double re = complex_obj.real();
double im = complex_obj.imag();
```

Yoki bu sintaktikga ekvivalent sintaktik:

```
double re = real(complex_obj);
double im = imag(complex_obj);
```

Kompleks son sinfi to‘rt matematik amallar - operatorlari qo‘llab-quvvatlaydi. Bularga [+=], [-=], [*=] va [/=] operatorlari kiradi. Masalan,

```
complex< double > complex_obj;
complex< double > second_complex_obj;
complex_obj += second_complex_obj;
```

Kompleks sonlarni kiritish/chiqarish ham qo‘llab-quvvatlanadi. Chiqish operatori vergullar bilan ajratilgan real va mavhum qismlarni qavslar ichida chop etadi. Masalan, chiqish operatorlarini bajarish natijasi:

```
        complex< double > complex0( 3.14159, -2.171
);
    complex< double > complex1( complex0.real() );

    cout << complex0 << " " << complex1 << endl;
```

output
(3.14159,-2.171) (3.14159,0)

Kirishtish operatorlari quyidagi formatlar bilan ishlaydi: Kompleks soning real qismini kiritish oddiy kiritiladi, masalan, 525.25; mavhum qismi qavslar ichida kiritiladi, masalan, (123.5), to‘liq kiritish uchun qavs ichida real va mavhum qismlari vergul bilan ajratiladi, Masalan, (12.5, 2.5). 6.1-dasturga qarang.

6.1-dastur. Kopleks sonlarni kiritish va chiqarish.

```
#include "stdafx.h"
#include <iostream>
#include <complex>

using namespace std;

int main(){
```

```
    complex< double > a, b, c;
    cout << "Complex sinfi oid turli t=farmatli 3
ta son kiritibg:" << endl;
    cin >> a >> b >> c;
    cout << "Complex sinfiga oid sonlar:" << endl;
    cout << a << b << c << endl;
    system("pause");
}
```

6.1-dastur. Output

```
    Complex sinfiga oid turli formatli 3 ta son
    kiritibg:
    3.1415
    (-0.25)
    (2.71, -9.8)
    Complex sinfiga oid sonlar:
    (3.1415,0)(-0.25,0)(2.71,-9.8)
```

Ushbu funksiyalardan tashqari, kopmleks sonlar sinfi quyidagi funksiyalariga ega:

```
sqrt(), abs(), polar(), sin(), cos(), tan(), exp(),
log(), log10() va pow().
```

Yozish vaqtida mavjud C++ standart kutubxonasi o‘ng operandli (masalan, qo‘shishi +=) operatorlar kompleks soni bo‘lmasa murakkab tayinlash amallarini qo‘llab-quvvatlamaydi. Masalan, bu yozuvga ruxsat berilmaydi:

```
complex_obj += 1;
```

C++ standarti ko‘ra, bu operatorlar bo‘lishi kerak, ishlab chiqaruvchilar ko‘pincha standartlarni yozib tugatishga yetib kelolmaydilar.

Bunday amallarni amalga oshirish uchun o‘z operatorimizni aniqlashimiz mumkin. Bu yerda qo‘shish operatorini amalga oshiruvchi funksiyaning varianti keltirilgan.

```
inline complex<double>& operator+=( complex<double>
&cval, double dval ){
   return cval += complex<double>( dval );
}
```

Bu misol yordamida, turi boshqa murakkab tayinlash uchun uch operatorlari funksiyalarini yozish muammo emas. Quyidagi dastur

uchun murakkab tayinlash uchun uch operatorlari funksiyalarini qo‘shing va tekshirish uchun uni ishga tushirish.

```
int main() {
   complex< double > cval ( 4.0, 1.0 );
   cout << cval << endl;
   cval += 1;
   cout << cval << endl;
   cval -= 1;
   cout << cval << endl;
   cval *= 2;
   cout << cval << endl;
   cout /= 2.;
   cout << cval << endl;
   system("pause");
}
```

C++ standartida kompleks son uchun inkrement va dekrement amallarining bajarilishi ko‘rsatilmagan. Biroq, ularning semantikasi juda aniq, agar cval + = 1 ni yoza olsak, bu cvalning haqiqiy qismini 1 bilan oshirishni anglatadi, unda inkrement amali juda oson ko‘rinadi. Murakkab < double> tipi uchun bu amallarni amalga oshirish va quyidagi dasturni ishga tushirish:

```
int main() {
 complex< double > cval( 4.0, 1.0 );
 cout << cval << endl;
 ++cva1;
 cout << cval << endl;
}
```

Bu yuqoridagilardan tashqari kompleks sonlari bilan ishlash sinfining ko‘plab, matematik funksiyalar mavjud. Ammo C++ standartiga qarab ayrim amallardan, funksiyalardan foydalanib bo‘lmaydi. Shuning uchun foydalanuvchining o‘z mustaqil kutubxona yaratib ishlashi maqul.

Valarray son sinfi va uning funksiyalari. Yaqin orada C++ da massiv elementlari bilan ishlash va samarali saqlash uchun Valarray konteyneri paydo bo‘ldi. Bu hamma C++ standartlarida bo‘lmasligi ham mumkin.

- Bu sinf massiv element indekslari bo‘yicha birlashtrish operatorlari va elementlararo matematik amallarni bajaradi.

-Vektor bilan solishtirganda Valarray sinf matematik amallarni bajarishda samaraliroq hisoblanadi.

Valarray sinfning ochiq funksiyalari:

apply() – bu massiv elementlari bo‘yicha bir vaqtda o‘zgarishlarni bajaradi va yangi massiv qaytaradi.

sum() – massivdagi elementlar yig‘indisini qaytaradi.

6.2-dastur. apply () va sum () funksiyalaridan foydalanish.

```
#include "stdafx.h"
#include<iostream>
#include<valarray>
using namespace std;
int main(){
    int myints[] = { 5, 25, 55, 85, 115 };
    valarray<int> varr (myints,
sizeof(myints)/sizeof(myints[0]));
    valarray<int> varr_one ;

    varr_one = varr.apply([](int x){return x=x+5;});

    cout << "Massivning yangi qiymatllar: ";
    for (int &x: varr_one) cout << '\t' << x;
    cout << endl;

    cout << "Oldingi massiv sum() => ";
    cout << varr.sum() << endl;
    cout << "Yangi massiv sum() => ";
    cout << varr_one.sum() << endl;
    system("pause");
    return 0;
}
```

6.2-dastur.output
Massivning yangi qiymatllar: 10 30 60 90 120 Oldingi massiv sum() => 285 Yangi massiv sum() => 310

Min() va max() fnuksiyalari mos ravishda massiv eng kichik va eng katta elementlari qimatlarini qaytaradi.

```
cout << "min() => ";
```

```
    cout << varr.min() << endl;
    cout << "max() => ";
    cout << varr.max() << endl;
```

shift() – berilgan son qiymat bo‘yicha indeks asosida massivni surib yangi massiv qaytaradi. Agar son musbat bo‘lsa chapga suriladi, manfiy bo‘lsa, o‘nga suriladi.

cshift() - berilgan son qiymat bo‘yicha indeks asosida massivni aylana shaklida surib yangi massiv qaytaradi. Agar son musbat bo‘lsa chapga suriladi, manfiy bo‘lsa, o‘nga suriladi.

6.3-dastur. shift() va cshift()dan foydalanish.

```
#include "stdafx.h"
#include<iostream>
#include<valarray>
using namespace std;
int main(){
    int myints[] = { 5, 25, 55, 85, 115 };
    valarray<int> varr (myints,
sizeof(myints)/sizeof(myints[0]));
    valarray<int> varr_one, varr_two;
    cout << "Massivning eski qiymatllar: ";
    for (int &x: varr_one) cout << '\t' << x;
    cout << endl;

    varr_one = varr.shift(-2);
    cout << "Massivning shift(-2) qiymatllar: ";
    for (int &x: varr_one) cout << '\t' << x;
    cout << endl;

    varr_one = varr.shift(2);
    cout << "Massivning shift(2) qiymatllar: ";
    for (int &x: varr_one) cout << '\t' << x;
    cout << endl;

    varr_one = varr.cshift(2);
    cout << "Massivning cshift(2) qiymatllar: ";
    for (int &x: varr_one) cout << '\t' << x;
    cout << endl;
```

```
    varr_one = varr.cshift(-2);
    cout << "Massivning cshift(-2) qiymatllar: ";
    for (int &x: varr_one) cout << '\t' << x;
    cout << endl;

    system("pause");
    return 0;
}
```

6.3-dastur.output

```
Massivning eski qiymatllar:
Massivning shift(-2) qiymatllar:    0    0    5   25   55
Massivning shift(2) qiymatllar:     55   85   115   0    0
Massivning cshift(2) qiymatllar:    55   85   115   5    25
Massivning cshift(-2) qiymatllar:   85   115   5    25   55
```

swap () - bir massivni ikkinchsi bir massivga almashtiradi.

```
varr.swap(varr1);
```

Yuqoridagilardan tashqari quyidagi funksiyalari mavjud:

operator=	Massiv qiymatini o‘zlashtirish
operator[]	Massiv indeksi orqali elementiga murojaat
resize	Massiv elemmentlar soni (hajmi) ni qayta aniqlash
size	Massiv elemmentlar soni (hajmi)

Valarray ning standart amallari

```
valarray operator+() const;
valarray operator-() const;
valarray operator~() const;
valarray<bool> operator!() const;
```

Valarray tiplari ustida amallar:

```
valarray& operator*= (const valarray& rhs);
valarray& operator/= (const valarray& rhs);
valarray& operator%= (const valarray& rhs);
valarray& operator+= (const valarray& rhs);
valarray& operator-= (const valarray& rhs);
valarray& operator^= (const valarray& rhs);
valarray& operator&= (const valarray& rhs);
valarray& operator|= (const valarray& rhs);
valarray& operator<<= (const valarray& rhs);
valarray& operator>>= (const valarray& rhs);
```

Valarray tipi va odatiy oddiy tiplar ustida amallar:

```
valarray& operator*= (const T& val);
valarray& operator/= (const T& val);
valarray& operator%= (const T& val);
valarray& operator+= (const T& val);
valarray& operator-= (const T& val);
valarray& operator^= (const T& val);
valarray& operator&= (const T& val);
valarray& operator|= (const T& val);
valarray& operator<<= (const T& val);
valarray& operator>>= (const T& val);
```

Yuqorida keltirilgan operatorilar Valarray sinfining funksiyalari hisoblandi.

Valarray tiplari bilan bajariladigan amallarni qayta aniqlash mumkin. Bunda birinchi amal Valarray tipi = Valarray tipi * Valarray tipi, ikkinchi amal Valarray tipi = oddiy tip * Valarray tipi, uchinchi tip Valarray tipi = Valarray tipi* oddiy tip ketma ketligi asosida aniqlangan.

[*] - amalini qayta aniqlash:

```
template <class T> valarray<T> operator* (const valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator* (const T& val, const valarray<T>& rhs);
template <class T> valarray<T> operator* (const valarray<T>& lhs, const T& val);
```

[*] - amalini qayta aniqlash:

```
template <class T> valarray<T> operator* (const valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator* (const T& val, const valarray<T>& rhs);
template <class T> valarray<T> operator* (const valarray<T>& lhs, const T& val);
```

[/] - amalini qayta aniqlash:

```
template <class T> valarray<T> operator/ (const valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator/ (const T& val, const valarray<T>& rhs);
template <class T> valarray<T> operator/ (const valarray<T>& lhs, const T& val);
```

[%] - amalini qayta aniqlash:

```
template <class T> valarray<T> operator% (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator% (const T&
val, const valarray<T>& rhs);
template <class T> valarray<T> operator% (const
valarray<T>& lhs, const T& val);
```

[+] - amalini qayta aniqlash:

```
template <class T> valarray<T> operator+ (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator+ (const T&
val, const valarray<T>& rhs);
template <class T> valarray<T> operator+ (const
valarray<T>& lhs, const T& val);
```

[-] - amalini qayta aniqlash:

```
template <class T> valarray<T> operator- (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator- (const T&
val, const valarray<T>& rhs);
template <class T> valarray<T> operator- (const
valarray<T>& lhs, const T& val);
```

[^] - amalini qayta aniqlash:

```
template <class T> valarray<T> operator^ (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator^ (const T&
val, const valarray<T>& rhs);
template <class T> valarray<T> operator^ (const
valarray<T>& lhs, const T& val);
```

[&] – mantiqiy amalini qayta aniqlash:

```
template <class T> valarray<T> operator& (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator& (const T&
val, const valarray<T>& rhs);
template <class T> valarray<T> operator& (const
valarray<T>& lhs, const T& val);
```

[|] – mantiqiy amalini qayta aniqlash:

```
template <class T> valarray<T> operator| (const
valarray<T>& lhs, const valarray<T>& rhs);
```

```
template <class T> valarray<T> operator| (const T&
val, const valarray<T>& rhs);
template <class T> valarray<T> operator| (const
valarray<T>& lhs, const T& val);
```

[<<] – chapga surish amalini qayta aniqlash:

```
template <class T> valarray<T> operator<< (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator<< (const T&
val, const valarray<T>& rhs);
template <class T> valarray<T> operator<< (const
valarray<T>& lhs, const T& val);
```

[>>] – o‘ngga surish amalini qayta aniqlash:

```
template <class T> valarray<T> operator>> (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<T> operator>> (const T&
val, const valarray<T>& rhs);
template <class T> valarray<T> operator>> (const
valarray<T>& lhs, const T& val);
```

[&&] – tezkor mantiqiy amalini qayta aniqlash:

```
template <class T> valarray<bool> operator&& (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<bool> operator&& (const
T& val, const valarray<T>& rhs);
template <class T> valarray<bool> operator&& (const
valarray<T>& lhs, const T& val);
```

[||] – tezkor mantiqiy amalini qayta aniqlash:

```
template <class T> valarray<bool> operator|| (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<bool> operator|| (const
T& val, const valarray<T>& rhs);
template <class T> valarray<bool> operator|| (const
valarray<T>& lhs, const T& val);
```

[==] – mulohaza amalini qayta aniqlash:

```
template <class T> valarray<bool> operator== (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<bool> operator== (const
T& val, const valarray<T>& rhs);
template <class T> valarray<bool> operator== (const
```

```
valarray<T>& lhs, const T& val);
```

[!=] – mulohaza amalini qayta aniqlash:

```
template <class T> valarray<bool> operator!= (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<bool> operator!= (const
T& val, const valarray<T>& rhs);
template <class T> valarray<bool> operator!= (const
valarray<T>& lhs, const T& val);
```

[<] – mulohaza amalini qayta aniqlash:

```
template <class T> valarray<bool> operator< (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<bool> operator< (const
T& val, const valarray<T>& rhs);
template <class T> valarray<bool> operator< (const
valarray<T>& lhs, const T& val);
```

[>] – mulohaza amalini qayta aniqlash:

```
template <class T> valarray<bool> operator> (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<bool> operator> (const
T& val, const valarray<T>& rhs);
template <class T> valarray<bool> operator> (const
valarray<T>& lhs, const T& val);
```

[<=] – mulohaza amalini qayta aniqlash:

```
template <class T> valarray<bool> operator<= (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<bool> operator<= (const
T& val, const valarray<T>& rhs);
template <class T> valarray<bool> operator<= (const
valarray<T>& lhs, const T& val);
```

[>=] – mulohaza amalini qayta aniqlash:

```
template <class T> valarray<bool> operator>= (const
valarray<T>& lhs, const valarray<T>& rhs);
template <class T> valarray<bool> operator>= (const
T& val, const valarray<T>& rhs);
template <class T> valarray<bool> operator>= (const
valarray<T>& lhs, const T& val);
```

Slice sinfi. Bu - xizmatchi sinf bo‘lib, valarray sinfning merosxo‘ridir va bir o‘lchovli to‘plam ostilarni yaratishda ishlatiladi. Agar valarray massivni ikki o‘lchovli matritsa sifatida qarasak, undan bir o‘lchovli vektorni hosil qilish mumkin.

Bu sinf slice_array tipdagi obyektni tavsiflovchi parametrlarni saqlaydi. Agar to‘plam osti bilvosita yaratilgan valarray sinf tipida bo‘lsa, sinf obyekti valarray obyekti uchun argument sifatida ko‘rsatiladi. To‘plam ostida saqlanadigan qiymatlar valarray tipini oladi va quyidagi xususiyatlarni o‘z ichiga oladi:

valarraydagi boshlang‘ich indeks.

massivning umumiy uzunligi va undagi elementlar soni;

valarray elementlarning ketma-ket indekslari orasidagi qadam yoki masofani aniqlash.

Agar qism to‘plam yordamida aniqlangan massiv doimiy valarrayning kichik bo‘lagi bo‘lsa, bu massiv yangi valarraydir. Doimiy valarray tipidagi to‘plam bir qism to‘plam yordamida aniqlangan bo‘lsa, dastlabki valarray uchun mos havolalar semantikasidir. O‘zgarmas bo‘lgan valarray lar uchun baholash mexanizmi vaqt va xotira sarflaydi.

Qism to‘plam asosida aniqlangan bo‘lsa va farqlansa, barcha elementlariga murojaat bo‘lsa, valarray tipidagi massivlar uchun amallar bajarilishi kafolatlanadi.

Slice sinfi uchun Slice konstruktori ishlatiladi. Bir-biridan bir xil masofada bo‘lgan va belgilangan element bilan boshlanadigan, bir necha elementlardan iborat valarray bir to‘plam ostisini yaratadi.

Slice sinfining funksiyalari quyidagilar:

size() – valarray to‘plam ostining elementlari sonini aniqlaydi.

start() – valarray to‘plam ostining boshlang‘ich indeksini qaytaradi

stride() - valarray to‘plam ostining elementlari orasidagi masofani topadi.

6.3-dastur. Slice sinfidan foydalanish.

```
#include "stdafx.h"

#include <valarray>
#include <iostream>
using namespace std;
int main( ){
   int i;
   size_t sizeVA, sizeVAR;
   size_t startVAR, strideVAR;
```

```
   valarray<int> va(28), vaResult;

   for ( i = 0 ; i < 28 ; i += 1 )
       va [ i ] =  (rand()%100)+1;

   cout << "valarray elementlari: \n ( ";
      for ( i = 0 ; i < 28 ; i++ )
         cout << va[i] << '\t';
   cout << ")." << endl;

   sizeVA = va.size();
   cout << "valarray elementlar soni: "
        << sizeVA << "." << endl << endl;
   // slice konstruktori
   slice vaSlice( 0, 6, 4);
   vaResult = va[vaSlice];

   cout << "valarray to‘plam ostisi vaResult = "
        << "va[slice( 0, 6, 4)] =\n ( ";
   for ( i = 0 ; i < vaSlice.size() ; i++ )
         cout << vaResult [ i ] << " ";
   cout << ")." << endl;

   sizeVAR = vaSlice.size( );
   cout << "to‘plam ostining elementlar soni: "
        << sizeVAR << "." << endl;

   startVAR = vaSlice.start( );
   cout << "to‘plam ostining birinchi element
indeksi: "
        << startVAR << "." << endl;


   strideVAR = vaSlice.stride( );
   cout << "to‘plam ostining elementlari orasidagi
masofa: "
        << strideVAR << "." << endl;
```

```
   system("pause");
}
```

6.3-dastur. output

```
valarray elementlari:
 ( 42   68      35      1       70      25      79
59      63      65      6       46      82      28
62
92      96      43      28      37      92      5
3       54      93      83      22      17      ).
valarray elementlar soni: 28.

valarray to‘plam ostisi vaResult = va[slice( 0, 6,
4)] =
 ( 42 70 63 82 96 92 ).
to‘plam ostining elementlar soni: 6.
to‘plam ostining birinchi element indeksi: 0.
to‘plam ostining elementlari orasidagi masofa: 4.
```

Dasturga sinfga berilgan ta‘rif asosida uning konstruktori 3 ta parametr qabul qilar ekan. Birinchi parametr belgilangan elementlar indeksi, ikkinchisi elementlar soni, elementlar indeksi orasidagi masofa.

Gslice sinfi. Xizmatchi sinf bo‘lib, valarray ning ko‘p o‘lchamli to‘plam ostilari bilan ishlashga mo‘ljallangan. Agar valarray massivda ko‘p o‘lchamli to‘plam sifatida qaralsa, Gslice sinfi undan ko‘p o‘lchamli vektorni oladi.

Sinf gslice_array tip bilan tavsiflanuvchi parametrlarni saqlaydi. agar _valarray**<Type> tipidagi obyekt uchun argument sifatida class gslice obyekti berilgan bo‘lsa, Valarray to‘plam ostisi bilvosita qurilgan hisoblanadi. ** to‘plam osti elementlari bo‘lib, Valarray bazaviy sinfdan olingan va 3 ta parametrni qabul qiladi:

- Boshlang‘ich indeks;
- Vektor uzunligi valarray<size_t> sinf;
- Vektor qadami (elementlari orasidagi mosofa) valarray<size_t> sinf;

Vektorlarning uzinligi bir xil bo‘lishi kerak.

Agar to‘plam gslice bilan aniqlangan va valarray ning to‘plam ostisi bo‘lsa, gslice yangi valarray hisoblanadi. O‘zgarmas bo‘lgan Valarray uchun baholash mexanizmi vaqt va xotirani tejaydi.

boshlang‘ich va oxirgi to‘plamlar gslice bilan aniqlangan bo‘lsa, farqli, barcha indekslariga murojaat bo‘lsa, Valarray to‘plam ustidagi barcha amallar kafolatlanadi.

gslice sinfi uchun gslice konstruktori ishlatiladi. Belgilangan element bilan boshlanadigan, bir necha elementlardan iborat valarray to‘plam ostisini yaratadi.

gslice sinfining funksiyalari quyidagilar:

size() – valarray to‘plam ostining umumiy massiv elementlari sonini aniqlaydi.

start() – valarray to‘plam ostining umumiy boshlang‘ich indeksini qaytaradi

stride() - valarray to‘plam ostining umumiy elementlari orasidagi masofani topadi.

Gslice konstruktori:

```
gslice(
  size_t _StartIndex,
  const valarray<size_t>& _LenArray,
  const valarray<size_t>& _IncArray);
```

Parametrlari:

_StartIndex – to‘plam ositdagi birinchi elemeni indeksi

_LenArray – har bir to‘plam osti uchun massiv elementlari soni

_IncArray - har bir to‘plam osti uchun massiv elementlari orasidagi masofa (qadam);

Standart konstruktor boshlang‘ich indeksi nol uzunlikdagi vektorlar va vektorlari uchun uzunlik va qadam nol qiymatni saqlaydi. Ikkinchi konstruktor boshlang‘ich indeksi uchun _startindex saqlaydi, qator uzunligi uchun _lenarray va qadam uchun _incarray qiymatlarni saqlaydi.

gslice – bir nechta qismdan tashkil topgan valarray uchun to‘plam osti valarrayni aniqlaydi va hammasi belgilangan bir element indeksidan boshlanadi. gslice i slice ning farqlari birgina bir nechta qism valarray lar bilan ishlashdadir.

6.4-dastur. gslice sinfidan foydalanish.

```
#include "stdafx.h"
#include <valarray>
#include <iostream>
```

```
using namespace std;
int main(){

   int i;

   valarray<int> va ( 20 ), vaResult;
   for ( i = 0 ; i < 20 ; i+=1 )
      va [ i ] =  i+1;

   cout << "valarray massiv elementlari: " << endl
<< "(";
   for ( i = 0 ; i < 20 ; i++ )
      cout << '\t' << va [ i ];
   cout << " )" << endl;

   valarray<size_t> Len (2), Stride(2);
   Len [0] = 6;
   Len [1] = 4;
   Stride [0] = 2;
   Stride [1] = 4;

   gslice vaGSlice (0, Len, Stride );
   vaResult = va[vaGSlice];

   cout << " vaGSlice asosida vaResult
elementlari:" << endl
        << "va[vaGSlice] = (";

   for ( i = 0 ; i < 10 ; i++ )
      cout << '\t' << vaResult [ i ];
   cout << ")" << endl;

   const valarray <size_t> sizeGS = vaGSlice.size (
);

      cout << "vaResult elementlar soni:"
        << " vaGSlice.size ( ) = ( ";
      for ( i = 0 ; i < 2 ; i++ )
```

```
         cout << sizeGS[ i ] << '\t';
   cout << ")." << endl;

   size_t vaGSstart = vaGSlice.start ( );

   cout << "vaResult uchun birinchi element
indeksi: "
        << vaGSstart << "." << endl;

   const valarray <size_t> strideGS =
vaGSlice.stride ( );

   cout << "vaResult ychun qadamlar:"
        << " vaGSlice.stride ( ) = ( ";
      for ( i = 0 ; i < 2 ; i++ )
         cout << strideGS[ i ] << '\t';
   cout << ")." << endl;
   system("pause");
}
```

6.4-dastur. Output

```
valarray massiv elementlari:
(       1       2       3       4       5       6
7       8       9       10      11      12      13
14
15      16      17      18      19      20 )
 vaGSlice asosida vaResult elementlari:
va[vaGSlice] = (        1       5       9       13
3       7       11      15      5       9)
vaResult elementlar soni: vaGSlice.size ( ) = ( 6
4       ).
vaResult uchun birinchi element indeksi: 0.
vaResult ychun qadamlar: vaGSlice.stride ( ) = ( 2
4       ).
```

Dastur tahlili bilan mutaqil shug‘ullaning.

Slice_array sinfi. Ichki yordamchi shablon sinf hisoblanadi. Valarray qismlari bilan aniqlangan to‘plam osti massivlar uchun amallarni qo‘llab quvvatlaydi. Sinf sintaktiki quyidagicha:

```
template <class Type>
class slice_array : public slice {
public:
    typedef Type value_type;
    void operator=(const valarray<Type>& x) const;
    void operator=(const Type& x) const;
    void operator*=(const valarray<Type>& x) const;
    void operator/=(const valarray<Type>& x) const;
    void operator%=(const valarray<Type>& x) const;
    void operator+=(const valarray<Type>& x) const;
    void operator-=(const valarray<Type>& x) const;
    void operator^=(const valarray<Type>& x) const;
    void operator&=(const valarray<Type>& x) const;
    void operator|=(const valarray<Type>& x) const;
    void operator<<=(const valarray<Type>& x)
const;
    void operator>>=(const valarray<Type>& x)
const;
}
```

Bu sinf obyekti - valarray<Type> obyektidagi elementlar ketma-ketligi bilan yoziladigan slice sinf obyekti bilan birgalikda valarray <Type> sinfiga havola saqlaydigan obyektlarni tasniflaydi.

Shaboln sinf muayyan valarray amallari tomonidan bilvosita yaratilgan va dasturda bevosita foydalanish mumkin emas. Ichki yordamchi shablon sinfi quyidagi sintaktik bilan ishlatiladi:

```
slice_array< Type> valarray< Type:: operator[] ( slice);
```

Slice_array < Type > obyekti faqat [SL] formatli va SL slice valarray ifodasini yozish orqali yaratiladi. slice_array sinfining funksiyalari valarray< Type > uchun belgilangan mos funksiyalari bilan bir xil ishlatiladi, faqat tanlangan elementlar ketma-ketligi ta‘sir ko‘rsatadi. Slice_array sinf tomonidan nazorat natijasida konstruktorga uch parametrlar belgilanadi: massivning birinchi element indeksi, elementlar soni va elementlar orasidagi masofa.

gslice_array sinfi. Ichki yordamchi shablon sinf hisoblanadi. Valarray qismlari bilan aniqlangan to‘plam osti massivlar uchun amallarni qo‘llab quvvatlaydi. Sinf sintaktiki quyidagicha:

```
template <class Type>
class gslice_array : public gsplice {
public:
```

```
    typedef Type value_type;
    void operator=(const valarray<Type>& x) const;
    void operator=(const Type& x) const;
    void operator*=(const valarray<Type>& x) const;
    void operator/=(const valarray<Type>& x) const;
    void operator%=(const valarray<Type>& x) const;
    void operator+=(const valarray<Type>& x) const;
    void operator-=(const valarray<Type>& x) const;
    void operator^=(const valarray<Type>& x) const;
    void operator&=(const valarray<Type>& x) const;
    void operator|=(const valarray<Type>& x) const;
    void operator<<=(const valarray<Type>& x)
const;
    void operator>>=(const valarray<Type>& x)
const;
}
```

Bu sinf obyekti - valarray<Type> obyektidagi massiv elementlar ketma ketligi bilan yoziladigan gslice sinf obyekti bilan birgalikda valarray <Type> sinfiga havola saqlaydigan obyektlarni tasniflaydi.

Shaboln sinf muayyan valarray amallari tomonidan bilvosita yaratilgan va dasturda bevosita foydalanish mumkin emas. Ichki yordamchi shablon sinfi quyidagi sintaksis bilan ishlatiladi:

```
gslice_array< Type> valarray< Type>:: operator[] ( constgslice& )
```

gslice_array < Type > obyekti faqat [GL] formatli va GL gslice valarray ifodasini yozish orqali yaratiladi. slice_array sinfining funksiyalari valarray<Type> uchun belgilangan mos funksiyalari bilan bir xil ishlatiladi, faqat tanlangan elementlar ketma-ketligi ta‘sir ko‘rsatadi. Slice_array sinf tomonidan nazorat natijasida konstruktorga uch parametrlar belgilanadi: massivning birinchi element indeksi, elementlar soni va elementlar orasidagi masofa. Dastur fragmentiga qarang.

```
const size_t lv[] = {2, 3};
const size_t dv[] = {7, 2};
const valarray<size_t> len(lv, 2), str(dv, 2);
```

indirect_array sinfi. Ichki yordamchi shablon sinf hisoblanadi. Bazaviy valarray sinf elementlari asosidagi to‘plam osti bo‘lib, belgilangan to‘plam osti o‘rtasida amallarni ta‘minlash orqali valarray

obyektlarining to‘plam osti obyektlar bo‘lgan obyektlarni qo‘llab-quvvatlaydigan yordamchi sinf.

Bu sinf obyekti - valarray<Type> obyektidagi elementlar ketma ketligi bilan yoziladigan valarray<size_t> sinf obyekti xa bilan birgalikda valarray <Type> sinfiga havola saqlaydigan obyektlarni tasniflaydi.

Indirect_array<type> obyekti faqat va[xa] shaklining ifodasini yozish orqali yaratiladi. Tanlangan elementlar ketma ketligi asosida Indirect_array sinf funksiyalari valarray uchun belgilangan tegishli funksiyalar bir xil ishlaydi. Ketma-ketlik *xa* dan iborat bo‘ladi. *va* ichidagi *xa[i]* indeksiga akslatirilganda I elementlar o‘lchami bo‘ladi.

6.5-dastur. indirect_array sinfidan foydalanish.

```
#include "stdafx.h"
#include <valarray>
#include <iostream>

int main( )
{
   using namespace std;
   int i;

   valarray<int> va ( 10 );
   for ( i = 0 ; i < 10 ; i += 2 )
      va [ i ] =  i;
   for ( i = 1 ; i < 10 ; i += 2 )
      va [ i ] =  -1;

   cout << "Valarray maasiv elementlari:  ( ";
      for ( i = 0 ; i < 10 ; i++ )
         cout << va [ i ] << '\t';
   cout << ")." << endl;

    valarray<size_t> indx(5);
    int j = 0;
    for (int i = 1; i < 10; i+=2)
    {
        indx [j] = i;
        //cout << indx[j] << '\t';
        j++;
```

```
    }
    cout << endl;
   va[indx] = 10;

   cout << "O‘zgartirilgan valarray elementlari:  (
";
      for (i = 0 ; i < 10 ; i++ )
         cout << va [ i ] << '\t';
   cout << ")." << endl;
   system("pause");
}
```

6.5-dastur.output

```
Valarray maasiv elementlari:  ( 0        -1       2
-1      4       -1      6       -1      8       -1
).

O‘zgartirilgan valarray elementlari:  ( 0        10
2       10      4       10      6       10      8
10 ).
```

mask_array sinfi. Ichki yordamchi shablon sinf hisoblanadi. Bazaviy valarray sinf elementlari asosidagi to‘plam osti bo‘lib, belgilangan to‘plam osti uchun mantiqiy amallarni ta‘minlash orqali valarray obyektlarining to‘plam osti obyektlar bo‘lgan obyektlarni qo‘llab-quvvatlaydigan yordamchi sinf.

Bu sinf obyekti - valarray<Type> obyektidagi elementlar ketma ketligi bilan yoziladigan valarray <bool> sinf obyekti ba bilan birgalikda valarray <Type> sinfiga havola saqlaydigan obyektlarni tasniflaydi.

Ketma-ketlik ba.size elementlardan ko‘pmas. J element faqat ba[J] ga chin (true) o‘rnatilgandagina faol bo‘ladi. Ketma-ketlikda elementni soni ba elementlarga teng bo‘ladi. Agar i - ba da eng kichik haqiqiy chin (true) elementning indeksi bo‘lsa, u holda tanlangan ketma-ketlikda a[i] ning qiymati nolga teng.

6.6-dastur. mask_array sinfi sinfidan foydalanish.

```
#include "stdafx.h"
#include <valarray>
#include <iostream>
using namespace std;
```

```
int main( ){
   int i;
   valarray<int> va ( 10 );
   for ( i = 0 ; i < 10 ; i++ )
      va [ i ] =  i*3;

   cout << "valarray elementlari:  ( ";
      for ( i = 0 ; i < 10 ; i++ )
         cout << va [ i ] << '\t';
   cout << ")." << endl;

   va [(va > 20 || va < 10) ] = 10;
   cout << "o‘zgartirikgan valarray elementlari:  (
";
      for ( i = 0 ; i < 10 ; i++ )
         cout << va [ i ] << '\t';
   cout << ")." << endl;
   system("pause");
}
```

6.6-dastur.Output

```
valarray elementlari:
 ( 0    3       6       9       12      15      18
21      24      27      ).
o‘zgartirikgan valarray elementlari:
 ( 10   10      10      10      12      15      18
10      10      10      ).
```

In sinfi[2]. Bu sinf interval matematikaning amallarini bajarishga yo‘naltirilgan bo‘lib, C++ ning eski standarti asosida keltiriladi va taqribiy hisoblashlar uchun yaratilgan. Sinf quyidagicha aniqlangan:

```
class In {
    private:
    public:
       double _a, a_;
       double m,n;
```

[2] Muallif MO‘MINOV Bahodir Boltaevich tomonidan 2013 yillarda tadqiqot ishlari uchun yaratilgan. Barcha xoxlovchilar uchun foydalanish mumkin.

```
   In() {_a=a_=0;}
   In(double a, double b) {
        if (a>b)
        { a=a+b;
          b=a-b;
          a=a-b;
        }
        _a=a, a_=b;
        }

 //kiritish va chqarish...
 friend istream& operator>>(istream & S, In & d)
throw()
    {
     S>>d._a>>d.a_;
    return S;
    }
 friend ostream& operator << (ostream & S, In &d)
throw()
   { S<<"["<<d._a<<","<<d.a_<<"]\t";
    return S;}

 //  In a = 9;
 inline In & In::operator= (const double & a)
{
   _a = a_ =a;
   return *this;
}
// - ishorasi
 friend In operator -(In &a) throw()
 {
    return In(-a.a_,-a._a);
    }
```

Sinfda 2 ta konstruktor bo‘lib, birinchisi interval sonni 0 ga tenglashtiradi, ikkinchisi mos parametrlarga asoslanib interval soni yaratiladi.

In kutubxonasi funksiyalari:

class In – interval sinf.

intervalning _a, - quyi chegarasi a_ - yuqori chegarasi;
In() {_a=a_=0;} Intervalning qiymatini 0 ga tenglashtirish;
In(double a, double b) Intervalga qiymat berish. Agar _a> a_ bo‘lsa, qiymatlar o‘rni almashadi.

1) [>>] - interval sonni o‘qish :: 1 2
2) [<<] - interval sonni chqarish, masalan: [a,b]
3) [=] - interval songa oddiy sonni tenglashtirish, masalan: In a; a=1;
4) [-]- interval sonni ishorasini almashtirish, masalan: In a; b=-a;
5) [+] - interval qo‘shish, masalan: [a]+[b]; [a]+b; a+[b];
6) [-] - interval ayirish, masalan: [a]-[b]; [a]-b; a-[b];
7) [*] - interval ko‘paytirish, masalan: [a]*[b]; [a]*b; a*[b];
8) [/] - interval bo‘lish, amallari [a]/[b]; [a]/b; a/[b];
9) [+=]- operator interval qo‘shish [a]+=[b] [a]+=b
10) [-=]- operator interval ayirish [a]-=[b] [a]-=b
11) [*=]- operator interval ko‘paytirish [a]*=[b] [a]*=b
12) [/=]- operator interval bo‘lish [a]/=[b] [a]/=b
13) [|] - operator interval yoki
14) [&]- operator interval va
15) [|=] -operator interval yoki
16) [&=]- operator interval va
17) [==]- operator interval tenglik
18) [!=]- operator interval tengmas
19) [<=]- operator interval kichik yoki teng
20) [>=]- operator interval katta yoki teng
21) [<]- operator interval kichik
22) [>]- operator interval katta
23) [!]- operator interval 0 intervalga tegishligini aniqlaydi.
24) GetInf funksiyasi - intervalning kichik qiymati olish
25) GetSup funksiyasi - intervalning katta qiymati olish
26) SetInf funksiyasi -intervalning kichik qiymatini o‘zgartirish
27) SetSup funksiyasi -intervalning katta qiymatini o‘zgartirish
28) IsEmpty funksiyasi - intervalning xatoligi, masalan: [2,1] – xato;
29) abs funksiyasi -interval moduli
30) Mid funksiyasi -intervalning markazi
31) Rad funksiyasi -intervalning radusi
32) Wid funksiyasi -intervalning uzunligi 2*Rad
33) MaxIn funksiyasi -intervalning kattasi modulda
34) MinIn funksiyasi -intervalning kichigi modulda

35) Sing funksiyasi -intervalning ishorasi -1,0,1 ko‘rinishda

Indefine kutubxonasi

1) tab - "\t" dir.
2) newLine - "\n" dir.
3) ZERO - 0.0000000 qiymat
4) real - double tipidir

ERROR kutubxonasi

1) DIV_BY_ZERO_ERROR
2) ERROR_INTERVAL_EMPTY_INTERVAL
3) ERROR_INTERVAL_SING_INTERVAL
4) ERROR_INTERVAL

INTMATH kutubxonasi

1) min funksiyasi – ikki haqiqiy sonning kichigi
2) max funksiyasi – ikki haqiqiy sonning kattasi
3) abs funksiyasi – haqiqiy sonning moduli

INMATH interval matematika kutubxonasida standart qiymatlar.

1) #define E 2.7182818284590452354
2) #define LOG2E 1.4426950408889634074
3) #define LOG10E 0.43429448190325182765
4) #define LN2 0.69314718055994530942
5) #define LN10 2.30258509299404568402
6) #define PI 3.14159265358979323846
7) #define PI_2 1.57079632679489661923
8) #define PI_4 0.78539816339744830962
9) #define PI1_ 0.31830988618379067154
10) #define PI2_ 0.63661977236758134308
11) #define SQRTPI2 1.12837916709551257390
12) #define SQRT2 1.41421356237309504880
13) #define SQRT1_2 0.70710678118654752440
14) sin – sin(x);
15) cos – cos(x);
16) tan – tan(x);
17) sinh – sinh(x);
18) cosh – cos(x);
19) tanh – tanh(x);
20) asin – asin(x);
21) acos – acos(x);
22) atan – atan(x);

23) exp – exp(x);
24) log – ln(a);
25) log10 – log(x);
26) pow(In &a, int n) funksiyasi – a interval sonning n darajasi
27) pow(In &a, double n)
28) pow(const In &b, In &a)
29) pow(const double b, In &a)
30) pow(In &b, In &a)
31) sqrt (In &a) funksiyasi intarval sondan kvadrat ildiz olish
32) sqrt(In &a, double n) funksiyasi intarval sondan n-chi darajali ildiz olish
33) ceil funksiyasi interval sonni yaxlitlash
34) floor funksiyasi interval sonning katta yaxliylash
35) round funksiyasi interval sonni kichikka yaxlitlash

INDEFMURAKKAB hosila olish uchun kutubxona
class def_murakkab murakkab hosilalar sinfi…

1) div_Def_uv funksiyasi - u/v ni hisoblash;
2) mult_Def_uv funksiyasi - u*v ni hisoblash;
3) add_Def_uv funksiyasi - u+v ni hisoblash;
4) sub_Def_uv funksiyasi - u-v ni hisoblash;
5) pow_Def_u funksiyasi - f(x)ning d darajasining hosilasi;
6) f_sqrt funksiyasi - ildiz osti f(x)ning hosilasi

class def_fx x ning n darajasinig hosilasi sinfi…

1) fx(In x) funksiyasi - x ning 1 darajasini qaytaradi;
2) fx(In x, int n) funksiyasi - x ning n darajasini qaytaradi;
3) dfx(In x, int n) funksiyasi - x ning n darajasini hosilasi;
4) dfx(In x, int n, int m) funksiyasi - x ning n darajasini m tartibli hosilasi;

class def_func – matematik funksiyalardan hosila;

1) d_exp(In x,int m) funksiyasi – exp(x) ning m tartibli hosilasi;
2) d_exp(In x, In k, int m) funksiyasi – exp(kx) ning m tartibli hosilasi;
3) const_a(In b) funksiyasi – o‘zgarmas interval sonning hosilasi;
4) f_ax(In a, In x,int m) funksiyasi- a^x ning m tartibli hosilasi;
5) f_ax(In a, In x,int m, int k) funksiyasi-a^kx ning m tartibli hosilasi;
6) f_sin(In x,int m) funksiyasi-sin ning m tartibli hosilasi;
7) f_cos(In x,int m) funksiyasi- cos ning m tartibli hosilasi;

8) f_sqrt(float x, int m) funksiyasi-sqrt ning m tartibli hosilasi;
9) f_ln(In x, int m) funksiyasi- ln(x) ning m tartibli hosilasi;
10) f_log(float a, In x, int m) funksiyasi-log(a,x) ning m tartibli hosilasi;
11) f_tan(In x) funksiyasi-tg(x) ning 1 tartibli hosilasi;
12) f_ctan(In x) funksiyasi-ctg(x) ning 1 tartibli hosilasi;
13) f_acos(In x) funksiyasi-arccos(x) ning 1 tartibli hosilasi;
14) f_asin(In x) funksiyasi-arcsin(x) ning 1 tartibli hosilasi;
15) f_atan(In x) funksiyasi-arctan(x) ning 1 tartibli hosilasi;
16) f_actan(In x) funksiyasi-arcctan(x) ning 1 tartibli hosilasi;
17) f_cosh(In x) funksiyasi-giperbolik cos(x) ning 1 tartibli hosilasi;
18) f_sinh(In x) funksiyasi-giperbolik sin(x) ning 1 tartibli hosilasi;
19) f_tanh(In x) funksiyasi-giperbolik tan(x) ning 1 tartibli hosilasi;
20) f_ctanh(In x) funksiyasi-giperbolik ctan(x) ning 1 tartibli hosilasi;

class def_arg_func argumentida f(x) bo‘lgan funksiyalar hosilasi sinf;
1) d_exp(func f, func df, In x) funksiyasi-exp(f(x)) ning hosilasi;
2) f_ax(In x, double a, func f, func df) funksiyasi-a^f(x) ning hosilasi;
3) f_ax(In x, In a, func f, func df) funksiyasi-a^f(x) ning hosilasi;
4) f_sin(In x, func f, func df) funksiyasi-sin(f(x)) ning hosilasi;
5) f_cos(In x, func f, func df) funksiyasi-cos(f(x)) ning hosilasi;
6) f_sqrt(In x, func f, func df) funksiyasi-sqrt(f(x)) ning hosilasi;
7) f_ln(In x, func f, func df) funksiyasi-ln(f(x)) ning hosilasi;
8) f_log(float a, In x, func f, func df) funksiyasi-log(a,(f(x)) ning hosilasi;
9) f_log(In a, In x, func f, func df) funksiyasi-log(a,(f(x)) ning hosilasi;
10) f_tan(In x, func f, func df) funksiyasi-tan(f(x)) ning hosilasi;
11) f_ctan(In x, func f, func df) funksiyasi-ctan(f(x)) ning hosilasi;
12) f_acos(In x,func f, func df) funksiyasi-acos(f(x)) ning hosilasi;
13) f_asin(In x,func f, func df) funksiyasi-asin(f(x)) ning hosilasi;
14) f_atan(In x, func f, func df) funksiyasi-atan(f(x)) ning hosilasi;
15) f_actan(In x,func f, func df) funksiyasi- actan(f(x)) ning hosilasi;
16) f_cosh(In x,func f, func df) funksiyasi-cosh(f(x)) ning hosilasi;
17) f_sinh(In x, func f, func df) funksiyasi-sinh(f(x)) ning hosilasi;
18) f_tanh(In x, func f, func df) funksiyasi-tanh(f(x)) ning hosilasi;
19) f_ctanh(In x, func f, func df) funksiyasi-ctanh(f(x)) ning hosilasi;

20) pow_arg_func(func f, func df, func dx, In x, int d) funksiyasi- (f(x)^d) ning hosilasi;
21) f_msqrt(func f, func df, In x, float m) funksiyasi- m_sqrt(f(x)) (1/m) ning hosilasi;

INGAUSSK_H – Integralni Gauss usulida hisoblash uchun koeffisentlar uchun kutubxona

1) P – ko‘p hadni hisoblash;
2) dP – ko‘p hadni hosilasini hisoblash
3) A – Koeffisentlarni hisoblash;
4) x0 – x uchun boshlang‘ish yaqinlashish;
5) xeps – x ni eps aniqlikda olish uchun;
6) convert_x – olingan x larni a,b oraliqqa convertlash;
7) convert_A - olingan A larni a,b oraliqqa convertlash;

INTEGRALIN_H interval integrallash uchun kutubxonaintegral_MTT1 – markaziy to‘g‘ri to‘rtburchak usuli;

1) integral_Trap funksiyasi– trapetsiya usuli;
2) integral_Simson funksiyasi – Simson usuli;
3) integral_gauss1 funksiyasi – Gauss usuli;
4) integral_gauss2 funksiyasi - Gauss usuli ikki karrali

INGAUSS_H CHATSni gauss usuli bilan yechish uchun kutubxona N = 3 uchun

1) PrintArray – massivni chiqarish;
2) PrintVector – vektorni chiqarish;
3) ScanArray – massivni kiritish;
4) ScanVectorB – vektorni kiritish;
5) SwapColumns – ustunlarni almashtirish;
6) SwapLines – qatorlarni almashtirish;
7) Destroy – quyi uchburchakni aylantirish;
8) Solution – yechimni topish
9) F – yechimning aniqligini aniqlash;
10) GAUSS – birlashgan Gauss usulini qo‘llash;

6.7-dastur. Chiziqli algebraik tenglamalar sistemasini yechish dasturi.

```
#include <iostream.h>
#include <conio.h>
#include <e:\mylibs\interval.h>

 int main(){
int n;
```

```
cout<<"n ni kirit:>>";
cin>>n;
In a[n][n],b[n];
//________A ni kirit:____________
for(int i=0; i<n; i++)
 for(int j=0; j<n; j++)
     a[i][j]=KiritIn();
//________b ni kirit:____________
for(int j=0; j<n; j++)
     b[j]=KiritIn();
//________to‘g‘ri yo‘l____________
In r[n][n];
for(int j=0; j<n-1; j++)
 for(int i=j+1; i<n; i++){
    r[i][j] =DivIn(a[i][j],a[j][j]);
       for(int k=j; k<n; k++)

a[i][k]=SubIn(a[i][k],MultIn(r[i][j],a[j][k]));
        b[i]=SubIn(b[i],MultIn(r[i][j],b[j])); }
//________a[i][i]:_________________
 for(int j=0; j<n; j++){
    b[j] =DivIn(b[j],a[j][j]);
   for(int i=0; i<n; i++)
     r[i][j] =DivIn(a[i][j],a[i][i]); }

//__________a[][[]________convert_
 for(int j=0; j<n; j++)
   for(int i=0; i<n; i++)
     a[i][j] = r[i][j];
//________ out _____________
  for(int i=0; i<n; i++) {
   for(int j=0; j<n; j++)
        { cout<<"[";chopIn(a[i][j]); cout<<"]\t"; }
          cout<<"= [";  chopIn(b[i]); cout<<"]\t";
         cout<<endl;}

  In x[n],s;
   for(int i=n-1; i>-1; i-- )
```

```
    {  s. _a=0; s. a_=0;
       for( int j=n-1; j>i; j--)
          s=AddIn(s,MultIn(a[i][j],x[j]));
           x[i]=DivIn(SubIn(b[i],s),a[i][i]);
      }

    for(int i=0; i<n; i++){
       cout<<"[";chopIn(x[i]); cout<<"]"<<endl;}

  getch();
return 0;
}
```

6.7-dastur. Ikki karrali integralni gauss usuli bilan taqribiy hisoblash dasturi.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
double P(double x, float n){
  if (n==0) return 1;
  if (n==1) return x;
  return (((2*n-1)/n)*x*P(x,n-1))-(((n-1)/n)*P(x,n-
2));
  }

double dP(double x, float n){
  return ((n/(1-x*x))*P(x,n-1))-(x*P(x,n));
  }

double A(double x, float n){
  return 2./((dP(x,n)*dP(x,n))*(1-x*x));
  }

double x0(int i, double n){
    return -cos(M_PI*((4*(i+1)-1)/(4*n+2)));  }

double xeps(double x, double n, double eps){
  double temp=x-P(x,n)/dP(x,n);
```

```
  while (fabs(temp-x)>eps) {
     x=temp;
     temp=x-P(x,n)/dP(x,n);
     }
 return x;
  }

double convert_x(double xx, float a, float b){
     return (0.5*(b+a))+(0.5*(b-a)*xx);
       }
double convert_A(double AA, float a, float b){
     return ((b-a)/2)*AA;
       }

int main(){
  float n,eps;
 freopen("InputData.in","r",stdin);
   cin>>n; cin>>eps;
  float a,b; cin>>a>>b;
  float x[(int)n];
 freopen("OutputData[a,b].out","w",stdout);
 cout<<"*** --- ---- n="<<n<<" ---- --- ***"<<endl;
 cout<<"*** -- a="<<a<<" b="<<b<<" -- ***"<<endl;
 cout<<"*** -- -- --- --- --- -- -- ***"<<endl;
    for(int i=0; i<n; i++) {
      x[i]=x0(i,n); cout<<i+1;
      x[i]=xeps(x[i],n,eps);
      printf(" :%.15f",convert_x(x[i],a,b));;
      printf(" :%.15f\n",convert_A(A(x[i],n),a,b));
   }
  //getch();
return 0;
}
```

6.7-dastur. Output

```
---------- 6 ----------
eps=5
1.40656 3.17244
```

```
2.2895:2.2895
2.22385
6
0.882941
***____HISOBLANDI_____***
*** --  n=5 uchun -- ***
1 :-0.90963196754455566000 :0.19411758457043865000
2 :-0.54064083099365234000 :0.47043172906368108000
3 :-0.00000000000000006123 :0.56888888888888889000
4 :0.54064083099365234000 :0.47043172906368108000
5 :0.90963196754455566000 :0.19411758457043865000
_____

I=2.21087712506044950000
```

Ushbu in sinfi odatda ilmiy amaliy matematika va matematika-fizika tenglamalarini taqribiy kompyuterda hisoblash ishlarini amalga oshirish uchun ishlatiladi. Bu kutubxona aynan A254-S loyihasi doirasida ishalb chiqilgan. Loyihani asosiy vazifasi fizik jarayoinlarni matematik modellashtirish va ularni sonli yechimlarni olish uchun dasturiy ta‘minotlarni ishlab chiqishga bag‘ishlangan. Kutubxonada 10 dan ortiq merosxo‘r sinf, 200 dan ortiq funksiyalar, matematik fuksiyalarning modifikatsiyalangan funksiyalari bor.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Nima uchun sonli sinflar yaratiladi?
2. Qanday sonli sinflarni bilasiz?
3. Sonli sinflarda amallarni bajarish uchun funksiyalarni qaysi usul yordamida aniqlanadi?
4. Kompleks va arifmatik amalar o‘zaro hamkorlikda ishlaydimi?
5. Kopmleks tipni arifmetik tipka tenglashtirish mumkinmi?
6. Haqiqiy (real) yoki mavhum qismini oddiy arifmetik tipga qiy mat qilib berish mumkinmi?
7. Kopmleks sonning real qismi qaysi funksiya bilan olinadi?
8. Kopmleks sonning mavhum qismi qaysi funksiya bilan olinadi?
9. double re = complex_obj.real(); va double re = real(complex_obj); sintaktiklarning farqi bormi?
10. Kompleks soning mavhum qismini qanday kiritiladi?

11. Kompleks sonni to‘liq kiritish uchun qavs ichida real va mavhum qismlari qanday belgi bilan ajratiladi?

12. Kompleks sinfi qanday operandlarni qo‘llab quvvtlamaydi?

13. Valarray konteyneri nima uchun kerak?

14. Massiv element indekslari bo‘yicha birlashtrish operatorlari va elementlararo matematik amallarni hisoblash sinfi ayting?

15. **apply()** – bu massiv elementlari bo‘yicha bir vaqtda o‘zgarishlarni bajaradi va nimani qaytaradi?

16. **cshift()** - berilgan son qiymat bo‘yicha indeks asosida massivni qanday shaklida surib yangi massiv qaytaradi?

17. Valarray tiplari bilan bajariladigan amallarni qayta aniqlash mumkinmi?

18. Qaysi sinf valarray sinfning merosxo‘ridir va bir o‘lchovli to‘plam ostilarni yaratishda ishlatiladi?

19. slice sinf qanday tipdagi obyektni tavsiflovchi parametrlarni saqlaydi.

20. slice sinfining konstruktorida nechta parametr bor va qaysilar, sanab bering?

21. Agar qism to‘plam yordamida aniqlangan massiv doimiy valarrayning kichik bo‘lagi bo‘lsa, bu massiv yangi nima?

22. **stride() -** valarray to‘plam ostining elementlari orasidagi nimani topadi.

23. Qaysi sinf valarray ning ko‘p o‘lchamli to‘plam ostilari bilan ishlashga mo‘ljallangan.

24. Sinf gslice - Valarray bazaviy sinfdan olingan va nechta parametrni qabul qiladi?

25. Agar to‘plam gslice bilan aniqlangan va valarray ning to‘plam ostisi bo‘lsa, gslice yangi nima hisoblanadi.

26. gslice – bir nechta qismdan tashkil topgan valarray uchun to‘plam osti valarrayni aniqlaydi va hammasi belgilangan bir element indeksidan boshlanadimi yoki xar xil element indeksidan boshlanadimi?

27. gslice i slice ning farqlari nimadan iborat.

28. Qaysi sinf Valarray qismlari bilan aniqlangan to‘plam osti massivlar uchun amallarni qo‘llab quvvatlaydi.

29. Shaboln sinf muayyan valarray amallari tomonidan bilvosita yaratilgan va dasturda bevosita foydalanish mumkinmi yoki mumkin emasmi?

30. Bazaviy valarray sinf elementlari asosidagi to‘plam osti bo‘lib, belgilangan to‘plam osti o‘rtasida amallarni ta‘minlash orqali valarray

obyektlarining to‘plam osti obyektlar bo‘lgan obyektlarni qo‘llab-quvvatlaydigan yordamchi sinfni ayting?

31. Tanlangan elementlar ketma ketligi asosida Indirect_array sinf funksiyalari valarray uchun belgilangan tegishli funksiyalar qanday ishlaydi.

32. Bazaviy valarray sinf elementlari asosidagi to‘plam osti bo‘lib, belgilangan to‘plam osti uchun mantiqiy amallarni ta‘minlash orqali valarray obyektlarining to‘plam osti obyektlar bo‘lgan obyektlarni qo‘llab-quvvatlaydigan yordamchi sinf ayting?

33. Interval matematikaning amallarini bajarishga yo‘naltirilgan sinf nomini ayting?

34. In sinfda nechta konchtruktor bor va ular qanday ishlaydi?

35. In(double a, double b) intervalga qiymat berish funksiyasida agar _a> a_ bo‘lsa, nima vazifa bajariladi.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I

Kompleks sonlar sinfiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`// Created by MBBahodir`
`#include "stdafx.h"`
`#include <complex>`
 	
`using namespace std;`
 	
`int main(){`
`  complex< double > a, b, c;`
`  cout << "Complex sinfi oid turli`
	1. Dasturda xatosini toping.
	_____
	_____
	_____
	
	2. Dasturda kompleks sonlardan iborat massivni yarating.
	_____
	_____
	_____
	_____
	_____
	_____
	_____

```
t=farmatli 3 ta son
kiritibg:" << endl;
    cin >> a >> b >>
c;
    cout << "Complex
sinfiga oid sonlar:"
<< endl;
    cout << a << b <<
c << endl;
    system("pause");
}
```

3. Kompleks sonlarni yig‘indisini hisoblovchi funksiya yarating.

_____
_____
_____
_____

4. Kompleks sonlar ustida matematik amallarni bajaruvchi funksiyalarni yarating

_____
_____
_____
_____
_____
_____

5. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi.

_____

6. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

IKKINCHI ASSISMENT TOPSHIRIG‘I

Valarray son sinfi va uning funksiyalariga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.
☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`#include "stdafx.h"`
`#include<iostream>`
`#include<valarray>`
`using namespace std;`
`int main(){`
`  int myints[] = { 5, 25, 55, 85, 115 };`
`  valarray<int> varr (myints, sizeof(myints)/sizeof(m`
	1. Dastur xatolarini toping.
	_____
	_____
	_____
	_____
	_____
	_____
	_____
	_____
	_____

```
yints[0]));
    valarray<int>
varr_one ;

    varr_one =
varr.apply([](int
x){return x=x+5;});

    cout << "Massivning
yangi qiymatllar: ";
    for (int &x:
varr_one) cout << '\t'
<< x;
    cout << endl;

    cout << "Oldingi
massiv sum() => ";
    cout << varr.sum()
<< endl;
    cout << "Yangi
massiv sum() => ";
    cout <<
varr_one.sum() << endl;
cout << "min() => ";
    cout << varr.min()
<< endl;
    cout << "max() =>
";
    cout << varr.max()
<< endl;
varr_one =
varr.shift(2);
    cout << "Massivning
shift(2) qiymatllar: ";
    for (int &x:
varr_one) cout << '\t'
<< x;
    cout << endl;
```

2.Dasturda `varr_one = varr.apply([](int x){return x=x+5;});` nima vazifani amalga oshiriradi.

_____
_____
_____
_____
_____
_____

3. Dasturda berilgan valarrayning eng kichik va eng katta elementlari farqini topuvchi dastur fragmentini yozing.

_____
_____
_____
_____
_____

4. Ikkita massivni o‘zaro solishtiring, teng bo‘lsa 0, katta bo‘lsa 1, kichik bo‘lsa -1 qilib valarray massiv yarating.

_____
_____
_____

5. Massivlarni teskari chiqarish dastur fragmentlarini yozing.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

```
    varr_one =
varr.cshift(-2);
    cout << "Massivning
cshift(-2) qiymatllar:
";
    for (int &x:
varr_one) cout << '\t'
<< x;
    cout << endl;
    system("pause");
    return 0;
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

UCHINCHI ASSISMENT TOPSHIRIG‘I

🎓 Slice sinfiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur

```
// Created by MBBahodir
#include "stdafx.h"

#include <valarray>
#include <iostream>
using namespace std;
int main( ){
   int i;
   size_t sizeVA,
sizeVAR;
   size_t startVAR,
strideVAR;
   valarray<int>
va(28), vaResult;
```

topshiriqlar

1. Dasturda nechta yangi tipdagi obyektlari yaratilgan.

_____
_____
_____
_____

2. Dasturdagi barcha va elementlarini ekranga chiqaring.

_____
_____
_____
_____
_____
_____
_____

```
   for ( i = 0 ; i < 28
; i += 1 )
       va [ i ] =
(rand()%100)+1;

   sizeVA = va.size();
   cout << "valarray
elementlar soni: "
        << sizeVA <<
"." << endl << endl;
   // slice
konstruktori
   slice vaSlice( 0, 6,
4);
   vaResult =
va[vaSlice];
   cout << "valarray
to‘plam ostisi vaResult
= "
        << "va[slice(
0, 6, 4)] =\n ( ";
   for ( i = 0 ; i <
vaSlice.size() ; i++ )
         cout <<
vaResult [ i ] << " ";
   cout << ")." <<
endl;
   sizeVAR =
vaSlice.size( );
   cout << "to‘plam
ostining elementlar
soni: "
        << sizeVAR <<
"." << endl;
   startVAR =
vaSlice.start( );
   cout << "to‘plam
ostining birinchi
```

3. Massivdan 2 indeksdan 4 ta element N qadam bilan yangi massiv chiqarish mumkin. N ning qiymatlarini aniqlang.

_____
_____
_____
_____
_____
_____

4. vaSlice.size( ); ning tipi qanday?

_____
_____
_____

5. Massivdan berilgan qiymatni izlash dastur fragmentini yozing.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

```
element indeksi: "
        << startVAR <<
"." << endl;
   strideVAR =
vaSlice.stride( );
   cout << "to‘plam
ostining elementlari
orasidagi masofa: "
        << strideVAR <<
"." << endl;
   system("pause");
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

TO‘RTINCHI ASSISMENT TOPSHIRIG‘I

gslice sinfiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur

```
// Created by MBBahodir
#include "stdafx.h"
#include <valarray>
#include <iostream>
using namespace std;
int main(){
   int i;
   valarray<int> va (
20 ), vaResult;
   for ( i = 0 ; i < 20
; i+=1 )
      va [ i ] =  i+1;
```

topshiriqlar

1. Dasturda nechta yangi tipdagi obyektlari yaratilgan.

_____
_____
_____
_____

2. Dasturdagi barcha va elementlarini ekranga chiqaring.

_____
_____
_____
_____
_____
_____

```
   valarray<size_t> Len
(2), Stride(2);
   Len [0] = 6;
   Len [1] = 4;
   Stride [0] = 2;
   Stride [1] = 4;
   gslice vaGSlice (0,
Len, Stride );
   vaResult =
va[vaGSlice];
   cout << " vaGSlice
asosida vaResult
elementlari:" << endl
        <<
"va[vaGSlice] = (";
   for ( i = 0 ; i < 10
; i++ )
      cout << '\t' <<
vaResult [ i ];
   cout << ")" << endl;
   const valarray
<size_t> sizeGS =
vaGSlice.size ( );
      cout << "vaResult
elementlar soni:"
        << "
vaGSlice.size ( ) = (
";
      for ( i = 0 ; i <
2 ; i++ )
         cout <<
sizeGS[ i ] << '\t';
   cout << ")." <<
endl;
   size_t vaGSstart =
vaGSlice.start ( );
   cout << "vaResult
uchun birinchi element
```

3. Massivdan 2 indeksdan (4,6) ta element (N,M) qadam bilan yangi massiv chiqarish mumkin. (N,M) ning qiymatlarini aniqlang.

_____
_____
_____
_____
_____
_____

4. vaSlice.size( ); ning tipi qanday va natijasichi ?

_____
_____
_____
_____

5. Massivdan berilgan qiymatni izlash dastur fragmentini yozing.

_____
_____
_____
_____
_____
_____
_____

```
indeksi: "
        << vaGSstart <<
"." << endl;
   const valarray
<size_t> strideGS =
vaGSlice.stride ( );
   cout << "vaResult
ychun qadamlar:"
        << "
vaGSlice.stride ( ) = (
";
      for ( i = 0 ; i <
2 ; i++ )
         cout <<
strideGS[ i ] << '\t';
   cout << ")." <<
endl;
   system("pause");
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

BESHINCHI ASSISMENT TOPSHIRIG‘I

indirect_array sinfiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`// Created by MBBahodir`
`#include "stdafx.h"`
`#include <valarray>`
`#include <iostream>`
 	
`int main( )`
	1. Dasturda nechta yangi tipdagi obyektlari yaratilgan.
	_____
	_____
	_____
	_____

```
{
   using namespace std;
   int i;
   valarray<int> va ( 10 );
   for ( i = 0 ; i < 10 ; i += 2 )
      va [ i ] =  i;
   for ( i = 1 ; i < 10 ; i += 2 )
      va [ i ] =  -1;
    valarray<size_t> indx(5);
    int j = 0;
    for (int i = 1; i < 10; i+=2)
    {indx [j] = i;
        //cout << indx[j] << '\t';
        j++;
    }
    cout << endl;
   va[indx] = 10;
   cout << "O‘zgartirilgan valarray elementlari: ( ";
      for (i = 0 ; i < 10 ; i++ )
         cout << va [ i ] << '\t';
   cout << ")." << endl;
   system("pause");
}
```

2. Dasturdagi barcha va elementlarini ekranga chiqaring.

_____
_____
_____
_____
_____
_____

3. Massivdan 2, 4 indeksli elementlarni qiymatini 100 o‘zgartirish dasturini tuzing.

_____
_____
_____
_____
_____
_____

4. va[indx] = 10; ning tipi qanday?

_____
_____
_____

5. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____

6. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

OLTINCHI ASSISMENT TOPSHIRIG‘I

mask_array sinfiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

`// Created by MBBahodir`
`#include "stdafx.h"`
`#include <valarray>`
`#include <iostream>`
`using namespace std;`
`int main( ){`
`  int i;`
`  valarray<int> va ( 10 );`
`  for ( i = 0 ; i < 10 ; i++ )`
`  va [ i ] =  i*3;`
`va [(va > 20 || va < 10) ] = 10;`
`  cout << "o‘zgartirikgan valarray elementlari: ( ";`
`  for ( i = 0 ; i < 10 ; i++ )`
`  cout << va [ i ] << '\t';`
`  cout << ")." << endl;`
`  system("pause");`
`}`
	1. Dasturda nechta yangi tipdagi obyektlari yaratilgan.
	_____
	2. Dasturdagi barcha va elementlarini ekranga chiqaring.
	_____
	3. Massivdan 10 katta elementlari aniqlovchi dasturini tuzing.
	_____
	4. va [(va > 20 || va < 10)] = 10; ning tipi qanday va nima va nima vazifani bajaradi?
	_____

5. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____

6. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

2.3. Sintaktik tahlil

📖 Sintaktik tahlil qiluvchi konstruktorlar, matnli-erkin grammatika, kompilyatorlar yaratuvchilarning muammolari, ifodalarning sintaktik tahlili, leksik va sintaktik tahlil bosqichlari, sintaktik tahlillovchi sinflari, past sathlarga yo‘naltirilgan analizatorlar, LL grammatika, LR – grammatika, rekursiv kamayish usuli, rekursiv kamayish usulidan foydalanish sharti, parser sinfi. FIRST to‘plamning qurish algoritmi, chap rekursiv grammatikalar, qiymat qaytarish asosidagi rekursiv kamayish usuli, parser qurish vazifalari va usullari keltirilgan bo‘lib, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. bilimlarni mustahkamlash uchun 35 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 1 ta assisment topshirig‘i va har assismentda 6 ta topshiriq berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

🖋 ***Kalit so‘zlar.*** Oqim, satr, iterator, sinf, shablon, matematik amallar, fayl, EOF, rekursiya, iofda, qoida, grammatika.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Tip tushunchasi, sinf va sinf obyekti, oqim, kirish elementga murojaat, funksiya va ko‘rsatkich, ma’lumotlarni kirish va chiqishi, dasturlashga oid dastlabki tushunchalar va C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Sintaktik tahlillovchi vazifasi, turli grammatikalar, dasturlash tilini analizatorining ishlash tamoyili, past va yuqori sathlarga yo‘naltirilgan algoritmlar, rekursiv kamayish usuli, analizatorlarni qurishda ta‘rif va shartlarni qurishni, parser sinfi, FIRST to‘plamning qurish algoritmi, chap rekursiv grammatikalar, chap rekursivlikni bartaraf etish algoritmi, qiymat qaytarish asosidagi rekursiv kamayish usuli, sintaktik analizatorni yaratish, Forward va Input iteratorlardan foydalanish, Strin – Stdin oqimlardan foydalanish, sintatik tahlillovchilarning bazaviy funksiyalarini o‘rganishingiz mumkin.

REJA

1.Ifodalarning sintaktik tahlili.
2.Parser sinfi.
3.Sintaktik analizatorni yaratish.

KIRISh

Bugungi kunga kelib dasturlash tillarining rivojlanishiga ularni sintaktik tahlil qiluvchi konstruktorlarning ko‘payib, ommalashganidir. Har bir dasturlash tili to‘g‘ri dastur tuzilishini belgilovchi qoidalar majmuasi yordamida tasvirlanadi. Dasturlash tilining sintaktik konstruksiyalarini tasvirlash uchun eng qulay formal usul matnli-erkin

grammatikadir (masalan, Backus-Naur ning normal shakli keng qo‘llaniladi).

Grammatikalar tildan foydalanadigan dasturchilar va shu til uchun kompilyatorlar yaratuvchilarning muammolarini bir xilda hal qilishga yordam beradi:

-Grammatika dasturlash tilining aniq va oson tushunadigan sintaktik xususiyatlarini ta‘minlaydi.

- Ba‘zi grammatika sinflar uchun, avtomatik ravishda manba dasturi sintaktik to‘g‘ri yoki yo‘qligini aniqlaydigan samarali parser (analizator, tahlillovchi) qurish mumkin.

- Aniq qurilgan grammatika dasturlash tiliga manba dasturini to‘g‘ri obyekt kodiga tarjima qilish va xatolarni aniqlash uchun foydali bo‘lgan tuzilmani berishi mumkin.

- Grammatika asosida ishlab chiqilgan kompilyatorlar juda oson kengaytirilishi mumkin (bu til rivojlanishi natijasida paydo bo‘lgan yangi konstruksiyalarni qo‘shish uchun ayniqsa foydalidir).

Yana bir bor ta‘kidlaymizki, matnli-erkin grammatikalar dasturlash tilining faqat matnli tarkibiy qismini, ya‘ni muayyan til qurilishi qanday yozilganligini aniqlaydi. Dasturning sintaktik to‘g‘riligini aniqlashning yana bir muhim qismi - dasturda tiplardan foydalanishning to‘g‘riligini matnli-erkin grammatika yordamida aniqlab bo‘lmaydi. Shuning uchun, agar dastur grammatikasi to‘liq deb ishlansa, u butunlay sintaktik jihatdan to‘g‘ri degani emas.

Ifodalarning sintaktik tahlili. Sintaktik tahlil deganda joriy qilingan grammatikaga leksemalarning ma’lum ketma-ketligi hosil qilgan tilga tegishli ekanligini belgilovchi jarayondir. Amalda, har qanday grammatikadan parser qurishingiz mumkin, ammo amalda ishlatiladigan grammatikalar maxsus shaklga ega. Masalan, n uzunlikdagi kiruvchi satr uchun murakkabligi $O(n^3)$ dan oshmasligi kerak bo‘lgan har qanday kontekst-erkin grammatika uchun analizator (parser) qurish mumkin. Lekin ko‘p hollarda tez ishlaydigan analizator qurish imkonini beradigan berilgan dasturlash tili asosida grammatika qurish mumkin. Amaliyotda tili analizatorlari odatda chiziqli murakkablikka ega. Masalan, dasturda chapdan o‘ngga qarab bitta terminal belgiga (leksik sinf) oldinga qarab ko‘rish orqali erishiladi.

Sintaktik tahlillovchi kirish parametrlari - bu leksema va jadvllar ketma-ketligi, masalan, ichki tasvirlangan jadval, bu jadval sintaktik tahlillovchi uchun chiqish parametrlaridir.

Sintaktik tahlillovchi chiqish parametrlari – jadvallar va tahlil daraxti hisoblanadi, masalan, kompilyatorning keyingi ko‘rishi uchun chiquvchi bo‘lib hisoblangan identifikatorlar jadvali va tiplar jadvali (masalan, tipli nazoratni amalga oshiruvchi ko‘rinish bo‘lishi mumkin).

Leksik va sintaktik tahlil bosqichlarining alohida qarashlarga ajratilishi shart emas. Odatda bu fazalar bir xil ko‘rinishda bir-biri bilan o‘zaro ta‘sirlashadi. Bunday ko‘rishning asosiy fazasi ajralish fazasi bo‘lib hisoblanadi va sintaktik analizator leksik analizatorga har safar boshqa terminal belgi kerak bo‘lganda murojaat qiladi.

Sintaktik tahlillovchi sinflari. Juda ko‘p tahlillovchi algoritmlarni quyidagi ikki sinflarning biriga tegishlidir.

1. top-down algoritmlari (past sathlarga yo‘naltirilgan);
2. bottom-up algoritmlari (yuqori stahlarga yo‘naltirilgan);

Bu terminlarning kelib chiqishi sintaktik daraxt tugunlarining yaratilish usuli bilan bog‘liq: ildizdan (grammatika aksiomalari) yuqori tugunlargacha (terminal simvollar), yoki yuqori tugunlardan ildizgacha.

Past sathlarga yo‘naltirilgan analizatorlar chiqishni qurish uchun grammatika aksiomadan boshlab va terminal simvollar zanjiri bilan tugaydi. Analizatorlar quyidagi xususiyatlarga ega bo‘lgan LL grammatika bilan bog‘liq:

- u natija bermaydigan analizatorlarga ega bo‘lishi mumkin;
- birinchi L harfi - kiruvchi zanjirni chapdan o‘nga qarab o‘qilishini bildiradi (left-to-right scan);
- ikkinchi L harfi - zanjirning chap chiqishi qurilayotganini bildiradi (*leftmost* derivation).

Past sathlarga yo‘naltirilgan analizatorlarni yaratish juda qulay bo‘lib hisoblanadi, ularni qo‘lda ham yaratish mumkin, masalan, rekursiv kamayish usuli bilan. LL grammatika bilan ishlash va o‘zgartirish murakkab emas. LL grammatika bo‘lmagan grammatikalarda analizatorlar uchun rekursiv kamayish usullaridan foydlanish mumkin.

Boshqa tomondan past sathlarga yo‘naltirilgan analizatorlarga qaraganda yuqori sathlarga yo‘naltirilgan analizatorlar ko‘proq ishlatiladi. Chunki ular ko‘proq grammatikalarni tahlil qila oladi. Shuning uchun bu analizatorlar uchun analizatolarni yaratuvchi maxsus dasturlar mavjud. yuqori sathlarga yo‘naltirilgan analizatorlar bilan LR – grammatika bog‘liq. Bundagi L harfi oldingidek, kiruvchi zanjirni chapdan o‘nga qarab o‘qilishini bildiradi (left-to-right scan), R harfi esa - zanjirning o‘ng chiqishi qurilayotganini bildiradi(*rightmost*

derivation). Bugungi kunda LR – grammatikaga asoslangan analizatorlar bilan juda ko‘p dasturlash tillari foydalanmoqda.

Rekursiv kamayish usuli. Eng oddiy va juda ko‘p foydalanilgan past sathlarga yo‘naltirilgan analizatorlarni qurish usuli rekursiv kamayish usulidir (recursive descent method).

Rekursiv kamayish usuli asosiy tamoyillarini o‘rganish uchun arifmetik ifodalarni bajarilish masalasini qaraymiz. Ularga binar amallari qo‘shish (+), ayrish (-), ko‘pytirish (*), butun bo‘lish (/) va qavslar amallari kirsin. Odatdagidek, ko‘paytirish va bo‘lish amallarining ustivorligi teng va ularning ustivorligi qo‘shish va ayirish amallarining ustivorligidan katta bo‘lib, bu amallarning ustivorligi ham tengdir. Qo‘shish turidagi amallarga (+) va (-) - amallarini, ko‘paytirish turidagi amallarga esa (*) va (/) - amallari qilib belgilaymiz. Qavslar amali standarti tartibini o‘zgartirish uchun ishlatiladi. Vazifamiz ifoda qiymatini hisoblovchi dastur yozishdan iborat.

Qaralayotgan ifodani quyidagi ko‘rinishda bo‘lsin

$$\mathbf{T_1 + T_2 + \ldots + T_n}$$

Bunda T_i - $F_{1i} * F_{2i} * \ldots * F_{mi}$ ko‘rinishidagi ifoda. Shuningdek, F_{ji} – bu son yoki qavs ichidagi ifoda.

Berilgan ifodani hisoblash jarayonini o‘ylab ko‘ramiz, bunda birinchi galda F_{11} hisoblanadi va F_{11} dan keyin qaysi amal turganini aniqlanadi. Agar bu ko‘paytirish amali bo‘lsa, chap operandini bilib, o‘ng operandini aniqlaymiz va amalni bajaramiz. Shunday qilib, ko‘paytirish amali uchun chap operandani aniqlaymiz. $F_1 * F_2 * \ldots * F_n$ amallar ketma ketligini hisoblab bo‘lgandan keyin, keyingi amal sifatida bo‘lish amalini kelsa, buni ham yuqoridagi jarayon bo‘yicha hisoblaymiz.

Ifodalarni hisoblashda barcha hisoblashlarni quyidagi sinflarga ajratish mumkin:

1. Oddiy ifoda, sonlar va qavslar bilan berilgan chiziqli ifodalar, masalan, 28, (128-25+16);
2. Ifoda, ko‘paytirish tipidagi amallari bo‘lgan ifoda, ko‘paytirish va bo‘lish amallari kiradi, maslan, 2*15, 28*(15/3)+3;
3. Ifoda, qo‘shish tipidagi amallari bo‘lgan ifoda, qo‘shish va ayirish amallari kiradi, masalan, 5-3, (5*3)/10-2;

Bular asosida hisoblash jarayonini tasavur qilishimiz mumkin va quyidagi formula bilan ifodalaymiz.

Expression (Term (Factor ()));

Bunda Factor – oddiy ifodani hisoblash protsedurasi bo‘lib, son, qavslarga ega chiziqli ifoda, Term - ko‘paytirish amali tipidagi amallarni saqlovchi ifoda qiymatini hisoblash protsedurasi, Expression - qo‘shish amali tipidagi amallarni saqlovchi ifoda qiymatini hisoblash protsedurasi.

Oddiy ifodani hisoblash protsedurasi:

```
int Factor ()
{
   char ch = getChar();
   if (isDigit (ch)) return getValue(ch);
   if (ch == '(')
   {
      int result = Formula();
      if (getChar() == ')') return result;
      error ("kutilmagan belgi yoki amal");
      return 0;
   }
   return error ("kutilmagan belgi yoki amal");
}
```

Dastur ko‘paytirish tipiga oid amallarni hisoblash uchun ifodani tahlil qiladi. Ko‘paytirish tipiga oid amallarni hisoblash uchun ifodani hisoblashning umumiy formulasi quyidagicha:

$F_1*F_2*\ldots*F_n$

Aniq tushunishimiz keraki, hisoblash bajarilayotgan amallar faqat ko‘paytirish yoki bo‘lishi amali bo‘ladi. Bunday amallarni hisoblash uchun Term protsedurasini yaratish mumkin. Bu protseduraning parametrlari butun sonili qiymatlar bo‘lishi kerak. Bunda chap operandani tanlashimiz va kerakli amalni bajarish uchun son yoki oddiy ifodadan iborat o‘ng operandani aniqlashimiz lozim, so‘ng esa amalni bajarish mumkin. Buni quyidagi dastur fragmenti asosida amalga oshirishsh mumkin.

```
 int Term (int left)
{
   char ch = getChar(); int right;
   if (ch != '*' && ch !='/')
   {
    /* navbatdagi  amal kutilmagan bo‘lsa, uni
qaytarishmiz kerak*/
```

```
      returnChar(); /*kutimagan belgini qaytarish */
      return left;  /*kutilmagan qimatni qaytarish */
   }
/* Hammasi yaxshi bo‘lsa, o‘ng operandni aniqlah
kerak*/
   right = Factor();
   if (ch == '*')
   {
      return Term(left * right);
      /* bu hisoblashni bajarishni aniqlaydi */
   }
   if (right == 0) return error ("Nolga bo‘linish");
   return Term(left / right);
}
```

Dastur qo‘shish tipiga oid amallarni hisoblash uchun ifodani tahlil qiladi. Qo‘shish tipiga oid amallarni hisoblash uchun ifodani hisoblashning umumiy formulasi quyidagicha:

$T_1 + T_2 + \ldots + T_n$

Aniq tushunishimiz keraki, hisoblash bajarilayotgan amallar faqat qo‘shish yoki ayirish amali bo‘ladi. Bunday amallarni hisoblash uchun Expression protsedurasini yaratish mumkin. Bu protseduraning parametrlari butun sonili qiymatlar bo‘lishi kerak. Bunda chap operandani tanlashimiz va kerakli amalni bajarish uchun son yoki oddiy ifodadan iborat o‘ng operandani aniqlashimiz lozim, so‘ng esa amalni bajarish mumkin. Buni quyidagi dastur fragmenti asosida amalga oshirishsh mumkin.

```
int Expression (int left)
{
   char ch = getChar (); int right;
   if (ch != '+' && ch != '-')
   {
    /* navbatdagi  amal kutilmagan bo‘lsa, uni
qaytarishmiz kerak*/
      returnChar();  /*kutimagan belgini qaytarish */
      return left;  /*kutilmagan qimatni qaytarish */
   }
 /* Hammasi yaxshi bo‘lsa, o‘ng operandni aniqlah
kerak*/
```

```
   right = Term (Factor());
   if (ch == '+')
   {
      return Expression(left + right);
       /* bu hisoblashni bajarishni aniqlaydi */
   }
   return Expression(left - right);
}
```

Yuqoridagi protseduralar ishlashi uchun quyidagi usullarni mustaqil ishlab chiqish kerak. Umuman olganda rekursiv kamayish usuli qiziqtirgan edi, o‘ylaymizki, quyidagi fragmentlarni yaratsangiz bunga ishonch hosil qilasiz.

1. Kirish oqimidan navbatdagi belgini kutuvchi, getChar parametrsiz usuli.
2. Kutilmagan belgini kirish oqimiga chiqarib berish returnChar usuli.
3. isDigit va getValue usullarni ham yozish shartmas. Agar parametri son bo‘lsa, birinchi usul true qaytaradi. Ikkinchi usul esa parametr sifatida kirish oqimidan berilan satrdan sonlarni ajratib oladi va butun qiymatlarni hisoblaydi.
4. Error usuli xatolar haqidagi xabarlarni chiqarish uchun ishlatiladi.

Bu usullarni mustaqil hal qilish mumkin, ammo *.NET* standart sinflar usullari asosida ba‘zalarini modifikatsiyalab yaratish mumkin. Bu dasturlar fragmentini shakllantirishda qiymat qaytarish asosidagi rekursiv kamayish usulidan foydalanildi (*recursive descent with backtracking*). Qiymat qaytarmaslik asosidagi rekursiv kamayish usuli bilan chegaralanib qolmaganimizni ko‘rib chiqamiz.

Rekursiv kamayish usulidan foydalanish sharti. Qiymat qaytarmaslik asosidagi rekursiv kamayish usulidan faqat quyidagi shart bajarilganda foydalanish mumkin.

Shart: har bir ifodaning birinchi belgisi bu holatda qaysi ifodaga amal qilishini aniqlash uchun yetarli bo‘lishi kerak. Bu shartni aniqroq tushunishni FIRST to‘plamini aniqlash orqali amalga oshirish mumkin.

Ta‘rif: KS grammatika uchun terminal va terminal bo‘lmagan belgilardan iborat G grammatika va w zanjir asosida FIRST k (w) to‘plamni quyidagicha aniqlaymiz:

$FIRST_k(w) = \{x \mid w =>^* xv, |x| = k \text{ yoki } w =>^* x, |x| < k\}$,

k – natural son.

Boshqacha qilib aytganda, FIRST $_k$ (w) to‘plam w dan olingan terminal zanjirlarning uzunligi k bo‘lgan barcha terminal prefikslardan iborat.

Universal algoritmik tilda tiplarning kichik qismini yaratadigan grammatikani ko‘rib chiqaylik:

type → imple
type → ^id
type → array[simple] of type
simple → integer
simple → char
simple → num … num

Bu grammatika uchun quyidagilarni aniqlash mumkin:

$FIRST_1$ (simple) = {integer, char, num}
$FIRST_1$ (^id) = {^}
$FIRST_1$ (array [simple] of type) = { array }

Agar w zanjir faqat terminallardan iborat bo‘lsa, FIRST $_k$ (w) - w zanjirda birinchi k belgilardir, aks holda |w| >=, yoki agar |w| < k < bo‘lsa, w zanjirning o‘zi.

Parser sinfi. FIRST to‘plamning qurish algoritmi. Birinchi navbatda belgilar grammatikasi uchun FIRST to‘plamni aniqlaymiz.

1-qadam. Agar x terminal bo‘lsa, FIRST(x)=x;

2-qadam. x → ε qoida uchun FIRST(x) to‘plamga ε ni qo‘shamiz;

3-qadam. Agar x teminal emas va grammatika qoidasi x → y1 y2 … yk bo‘lsa, FIRST(X)ga a terminalni qo‘shamiz, agar qandaydir i uchun a terminal FIRST(yi)ga tegishli bo‘lsa va ε - FIRST(y1), ... ,FIRST(yi-1) to‘plamlarga tegishli bo‘lsa, y1 y2 … yi-1 => * ε, agar barcha j=1,2,…k uchun ε - FIRST(yj) to‘plamga tegishli bo‘lsa, ε ni FIRST(y) to‘plamga qo‘shamiz.

FIRST(w) to‘plamni qurish algoritmini shakllantiramiz.

Kirish: KS grammatika G=(N, T, P, S) va w zanjirning terminal va terminal bo‘lmagan belgilari.

Chiqish: FIRST(w).

Usul: FIRST (X1)dan barcha bo‘sh bo‘lmagan belgilarni FIRST (X 1 X 2 …X k) ga qo‘shamiz. Agar ε - FIRST (X1)ga tegishli bo‘lsa, FIRST (X2) dan barcha bo‘sh bo‘lmagan belgilarni,shu bilan iteratsiyani davom ettiramiz. Oxirida, Agar barcha j uchun FIRST (Xj) bo‘sh belgiga ega bo‘lsa, FIRST (X1 X2…Xk)to‘plamga ε ni qo‘shamiz.

Misol: Quyidagi qoidalar bilan berilgan grammatika qaraymiz:

S → BA

S → +BA
A → ε
B → DC
C → *DC
C → ε
D → (S)
D → a

Ushbu grammatika uchun *FIRST* to‘plami quyidagicha aniqlanadi.

FIRST(D)= {(,a}, FIRST(C) = {*,ε}, FIRST(B) = FIRST(D), FIRST(A) = {+,ε}, FIRST(S) = {(,a}.

LL(k)-grammatika ko‘rib chiqamiz.

Ta‘rif: G = (VT, VN, P, S) grammatika LL(k)-grammatika deb aytiladi, agar FIRST k (x) = FIRSTk (y) teng, u=u1 bo‘lsa va ikkita ixtiyoriy chap chiqishlar uchun ifoda o‘rinli bo‘lsa,

S =>* wAv => wuv =>* wx

S =>* wAv => wu1v =>* wy

Av dan chiquvchi terminal va terminal bo‘lmagan belgilar va k birinchi belgilardan (agar mavjud bo‘lsa) iborat wAv joriy zanjir uchun mavjud bo‘lsa, w bilan boshlanadigan va aytib o‘tilgan k terminallar bilan davom etadigan chiqishda qandaydir terminal zanjirini olish uchun A ga qo‘llash mumkin bo‘lgan kamida bitta qoida mavjud.

Qoidaga asoslangan grammatikani qarymiz:

S → aAS
S → b
A → a
A → bSA

va ikkita chiqish

(1) S =>* wSv => wuv =>* wx

(2) S =>* wSv => wu_1v =>* wy.

Agar x va y zanjirlar a bilan boshlansin. Bu esa chiqishda S → aAS qoida ishtirok etganini bildiradi. Haqiqattan ham u = u1 = aAS. Agar x va y zanjirlar b bilan boshlansin. Bu esa chiqishda S → b qoida ishtirok etganini bildiradi. Haqiqattan ham u = u1 = b. Bunday ko‘rinadiki, ikkita chiqish amallari o‘xshash va teng kuchlidir. Bunday esa yuqoridagi grammatika LL(1) – grammatika xususiyatlarini qo‘llab quvvatlaydi.

LL(1) – grammatikaga asoslangan Qiymat qaytarmaslik asosidagi rekursiv kamayish usuli uchun tahlilovchi qurish mumkin. Masalan,

(1) S → if S then S else S

(2) S → begin S L
(3) S → print E
(4) L → end
(5) L →; S L
(6) E → num = num

LL(1) – grammatikaga asoslangan rekursiv kamayish usuli uchun quyidagi grammatikani qaraymiz:

(1) S → if E then S else S
(2) S → begin S L
(3) S → print E
(4) L → end
(5) L →; S L
(6) E → num = num

Bu grammatikani qo‘llab quvvatlvchi tahlillovchini rekursiv kamayish usuli asoslanib yaratish mumkin. Buning uchun terminal bo‘lmagan grammatika uchun har biri uchun bittadan protsedura yozish kerak bo‘ladi.

```
class SimpleParser {
    const int IF = 1;
    const int THEN = 2;
    const int ELSE = 3;
    const int BEGIN = 4;
    const int END = 5;
    const int PRINT = 6;
    const int SEMICOLON = 7;
    const int NUM = 8;
    const int EQ = 9;

    public static void nextStep(int lc){
        if (lexical_class == lc)
            lexical_class = getLC();
        else
            error();
    }

    public static void S(void){
        switch(getLC())
        {
        case IF:
```

```
            E(); nextStep(THEN); S();
nextStep(ELSE); S(); break;
        case BEGIN:
            S(); L(); break;
        case PRINT:
            E(); break;
        default:
            error(); break;
        }
    }

    public static void L(void){
        switch (lexical_class)
        {
        case END:
            getLC(); break;
        case SEMICOLON:
            getLC(); S(); L(); break;
        default:
            error(); break;
        }
    }

    public static void E(void){
        nextStep(NUM); nextStep(EQ); nextStep(NUM);
    }

    public static void main(void){
        lexical_class = getLC();
        S();
    }
}
```

Chap rekursiv grammatikalar. LL (k) – xususityalari grammatika uchun kuchli cheklovlar yuklaydi. Natijada grammatika LL(1) xususiyatiga ega bo‘lishi uchun ba‘zan grammatikani o‘zgartirish mumkin. Buni amalga oshirish uchun har doim ham muvaffaqiyatli bo‘lmaydi, lekin bir LL(1) grammatika olish imkoniyatiga ega bo‘ldi,

keyin rekursiv kamayish usuli asosida analizator qurishda foydalanishingiz mumkin. Faraz qilaylik, quyidagi grammatika asosida tahlillovchi qurish kerak bo‘lsin.

E → E + T|E-T|T

T → T * F|T/F|F

F → num|(E)

FIRST(T)to‘plamning terminallari FIRST(E+T) to‘plamga tegishli bo‘lsin. Bu holda protseduralarning ketma – ket chaqirish bir qiymatli aniqlab bilmaymiz, chunki kirish zanjiridagilarni tahlil qilishmiz kerak.

Muammo shundaki, terminal bo‘lmagan E ning o‘ng pozitsiyasida qoidaning o‘ng tomoni, chap qismi ham Eda bo‘ladi. Bunday holatlarda terminal bo‘lmagan E – chap rekursiv deb aytiladi.

Ta‘rif. Agar grammatikada A =>* Aw chiqish mavjud bo‘lsa, A terminal bo‘lmagan KS grammatika G chap rekursiv deb aytiladi.

Bitta bo‘lsa ham chap rekursiv qoidaga ega bo‘lgan grammatika LL(1)- grammatika bo‘la olmaydi. Ikkinchi tomondan bilamizki, KS til bitta bo‘lsa ham chap rekursiv bo‘lmagan grammatika bilan aniqlanadi.

Chap rekursivlikni bartaraf etish algoritmi. To‘g‘ridan-to‘g‘ri chap rekursivlikni bartaraf etish algoritmini keltiramiz.

G = (N, T, P, S)- KS *grammatika va* $A \rightarrow A\omega_1|A\omega_2|\ldots|A\omega_n|\nu_1|\nu_2|\ldots|\nu_m$ qoidalar P dagi barcha qoidalarni ifodalasin, A ning chap qismini saqlamaydi, chunki biror bir zanjir ν_i - A terminal bo‘lmagan bilan boshlanmaydi. N to‘plamga yana bir taerminal bo‘lmagan A‘ qo‘shamiz va A da chap tomondagi bo‘lgan qoidalarni o‘zgartiramiz. Quyidagi qoidalarni olamiz.

$A \rightarrow \nu_1 | \nu_2|\ldots|\nu_m | \nu_1 A‘ | \nu_2 A‘|\ldots|\nu_m A‘|$

$A‘ \rightarrow \omega_1 | \omega_2|\ldots| \omega_m | \omega_1 A‘ | \omega_2 A‘|\ldots| \omega_m A‘|$

Olingan grammatika oldingisiga ekvivalent ekanligini ko‘rsatish mumkin. Yuqorida keltirilgan grammatikaga bu akslantirishni qo‘yib, arifmetik amallarni yozish uchun quyidagicha grammatikani olamiz:

E → T|TE‘

E‘ →+T|+TE‘

T → F|FT‘

T‘ → *F|*FT‘

F → (E)|num

Tuzilgan grammatika *LL*(1) xususiyatlariga ega ekanligini ko‘rsatish mumkin.

Agar ikkita qoida ham terminal bo‘lmagan bir xil belgi bilan bo‘lsa, aniq muammoni vaziyatga keltiradi, masalan,

S → if E then S else S

S → if E then S

Bunday hollarda yana bir terminal bo‘lmagan qoida qo‘shamiz, qaysiki yuqoridagi vaziyatga oydinlik kiritib, farqli qoidalarni keltirib chiqaradi va quyidagi qoidani olishimiz mumkin:

S → if E then S S‘

S‘ →

S‘ → else S

Olingan grammatika rekursiv kamayish usuli bilan amalga oshirilishi mumkin.

Qiymat qaytarish asosidagi rekursiv kamayish usuli. Shunday qilib, qiymat qaytarish asosidagi rekursiv kamayish usulini qo‘llash uchun grammatikani FIRST to‘plamlar kesishmaydigan shaklga aylantirish kerak. Bu murakkab va ko‘p vaqt talab qiladigan jarayondir. Shuning uchun, amaliyotda, tez-tez ishlatiladigan ibora bor, qiymat qaytarish asosidagi rekursiv kamayish hodisa deb ataladi.

Buning uchun leksik analizator an‘anaviy usullarni to‘laligicha, scan va next ana‘naviy usullar va nusxalash konstruktoriga ega bo‘lgan obyekt sifatida namoyon bo‘ladi. Keyin, noaniqlik yuzaga kelishi mumkin bo‘lgan barcha holatlarda, tahlil qilishni boshlashdan oldin leksik analizatorning hozirgi holatini aniqlaymiz (ya‘ni, leksik analizatorning nusxasini boshlaymiz) va matnni tahlil qilishni davom ettirishga harakat qilamiz. Bu tahlil muvaffaqiyatsiz bo‘lsa, leksik analizator holatini tiklaymiz va shunday qilib, keyingi grammatika variantni yordamida yana shu zanjirni tahlil qilishga harakat qilinadi va tugaguncha davom ettiriladi. Barcha tahlil imkoniyatlari muvaffaqiyatsiz bo‘lsa, xatolar haqida xabar beramiz.

Bu tahlillovchini bu usul qiymat qaytarmaydigan rekursiv kamayish usuli ko‘ra potensiali sekin ekanligini ochiq-oydin ko‘riy mumkin, lekin evaziga uning asl shaklida grammatikada qolishi va dasturchi uchun harakat saqlab qolish uchun boshqarish osondir.

Xuddi shu sintaktik tahlil S – bemoll kompilyatorida sintaktik tahlil uchun joriy qilingan. Masalan, quyidagi fragmentda tahlil paytida konstruksiya ifoda operatori yoki izoh ekanligini aniqlovchi usul amalga oshiradi. Ikkala konstruksiya nuqta bilan ajratilgan va ehtimol kvadrat qavslarni o‘z ichiga olgan identifikatorlar ketma-ketligi bilan boshlanishi mumkin.

```
AST.Stmt stmt_or_decl ()
{
```

```
    Lexer saved = new Lexer (lexer);
    try {
    AST.Type t = type_opt ();
    if (lexer.Is (Token.Tag.Ident))
        return decl_tail (saved.Curr.coor, t);
    }
    catch (ParseFailed) {
       lexer = saved;
       AST.Expr expr = this.expr ();
       return new AST.Stmt.Expr (compiler,
          saved.Curr.coor|lexer.req
(Token.Tag.Semicolon).coor, expr);
    }
}
```

try blokida uni tavsif deb hisoblab, konstruksiyani ajratishga harakat qilamiz. Bu muvaffaqiyatsiz bo‘lsa, tahlillovchi istisno chaqiradi va keyin catch blokda tahlil boshlanadi va shu konstruksiya ifoda sifatida tahlil qilinadi.

Sintaktik analizatorni yaratish. Odatda, mashina tilidagi matn gaplar, gap-gaplar ostidan, gap osti esa, o‘z navbatida, gaplar ostidan va shu kabi belgilardan hosil bo‘ladi. Masalan, XML hujjatda boshlanish tegi, qiymat, yopish teglari mavjud. Boshlanish tegi [<] belgisi, teg nomi va qanday atribut va qiymatlari, [>] belgisidan iborat. Yopish tegi esa, [</]belgisi, teg nomi, [>]belgisi bilan tugaydi. Atribut esa nomi, [=], [“], belgilar to‘plami va yana [“] belgisidan iborat va takrorlanishi mumkin. Qiymat esa qandaydir belgilar to‘plami yoki qandaydir elementlardan (gap, gap osti, ifoda) iborat. Tahlil natijasida tahlil daraxtini hosil qilish mumkin.

Bunday tillarni har bir terminal bo‘lmagan tilning ma’lum bir jumlasiga mos keladigan Backus-Naur shakli (BNSh) yordamida tasvirlash qulay. Dasturlarni yozganda, odatda ularni funksiyalarga va funksiya ostilarga ajratamiz va sintaktik tahlillovchi yozmoqchi bo‘lganimiz uchun, har bir BNSh terminal bo‘lmagan sintatik tahlillovchimizning bitta funksiyasiga mos kelsin va har bir bunday funksiya:

- ushbu jumlani berilgan pozitsiyadan ajratishga harakat qilish;
- bu ishni bajardimi yo‘qmi natijasini qaytarish;
- tahlil tugagani yoki xato sodir bo‘lgan pozitsiyani qaytarish;

-tahlil natijasida olish kerak bo‘lgan ba‘zi qo‘shimcha ma’lumotlarni qaytarish;

Masalan, BNSh ko‘rinishida expr ::= expr1 expr2 expr3 funksiyani yozamiz:

```
bool read_expr(char *& p, ....){
    if(!read_expr1(p, ....))
        return false;
    // read_expr1() fuksiyasi expr1 tahlil tugaga
pozitsiyada p qo‘yadi
    // bu pozitsiyada p bilan hech qanday amal
bajarilmaydi.
    if(!read_expr2(p, ....))
        return false;
    if(!read_expr3(p, ....))
        return false;
    return true;
}
```

Davomi sifatida, BNSh ko‘rinishida expr ::= expr1|expr2|expr3 funksiyani yozamiz:

```
bool read_expr(const char *& p, ....){
    const char * l = p;
    if(read_expr1(p, ....))
        return true;
    // expr1 tahlil jarayonida p xato bo‘lgan joyni
ko‘rsatadi
    p = l;
    if(read_expr2(p, ....))
        return true;
    p = l;
    if(read_expr3(p, ....))
        return true;
    return false;
}
```

Bu sintatik tahlillash qiymat qaytarish asosidagi rekursiv kamayish usuliga asoslangan.

Ixtiyoriy matnni BNSh ko‘rinishida keltirish mumkin. Masalan, XML tilidagi quyidagicha yozish mumkin:

element ::= '<'identifier'>'some_data'</'identifier'>'

Agar identifikatorlari mos kelsa bu haqiqattan ham to‘g‘ri keladi. Bunday amallarni sintaktik tahlillovchiga qo‘shish muammo emas. Masalan, terminal funksiyalar quyidagicha bo‘lishi mumkin:

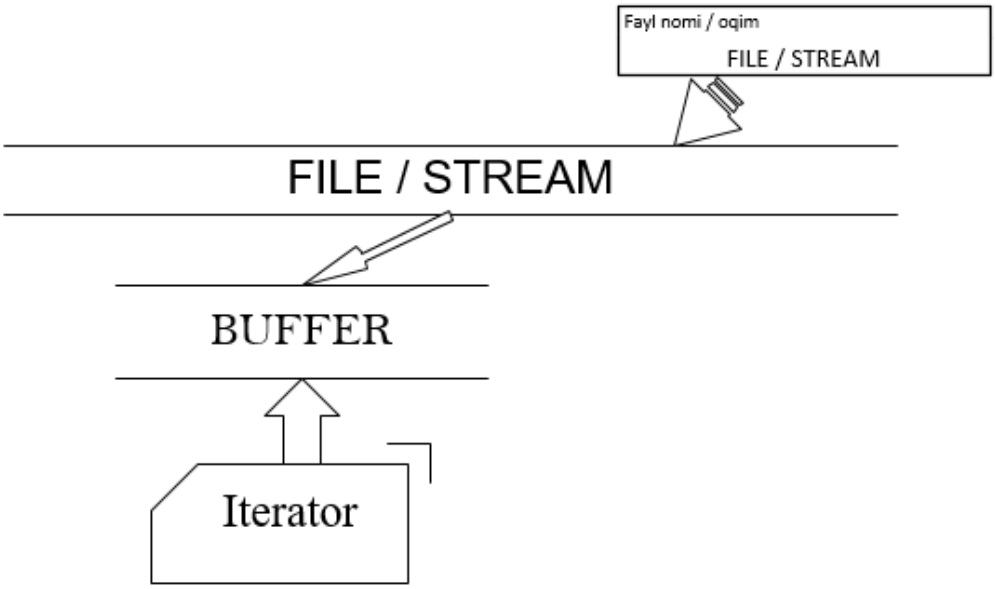
```
bool read_fix_char(const char *& it, char c){
    if(!*it)    return false; // satr oxiri
    if(*it!=c)      return false;// boshqa simvol
            // xato bo‘lganda joyiga qolish
bajariladi
    it++;       // keyingisiga o‘tish
    return true;
}
bool read_fix_str(const char * & it, const ch_t *
s){
    while(*it)// satr tugamagancha
        if(!*s)
            return true;
        else if(*it!=*s)
            return false;
            // xato bo‘lganda joyiga qolish
bajariladi
        else
            it++, C++;
    if(!*s) return true;
    return false;
}
```

Satrni sintaktik tahlil qilganimizda yuqoridagi funksiyalarning barcha tiplari uchun forward iteratorlar yetarli ekanligini biling.

forward iteratorlarni taqdim etadigan va faylni butunlay emas, fayl o‘rniga kichik qismlarga xotirada saqlaydigan sinf oqimni yaratishni qaraymiz. Agar shablonlar asosida tahlillovchi funksiyani yaratishni xoxlasak, ularni satrlar va/yoki oqimlardan foydalanish mumkin (base_parse.h kutubxonasi terminal funksiyalari uchun, bu sinfni diskdagi mavzuga oid papkaga qarang). Birinchi navbatda unix «barcha fayllar bor»: ideyalogiyasiga ba‘zi aniqliklarni kiritib olamiz: Shunday fayllar bo‘ladiki, ular disk bo‘yicha joylashgan va ularni ixtiyoriy pozitsiyasidan bir necha marta o‘qish mumkin(ularni random-access fayllar deb aytiladi), shunday oqimlar bo‘ladiki, tarmoqdan, klaviaturadan, boshqa ilovalardan yo‘naltirilgan (bularni `forward`

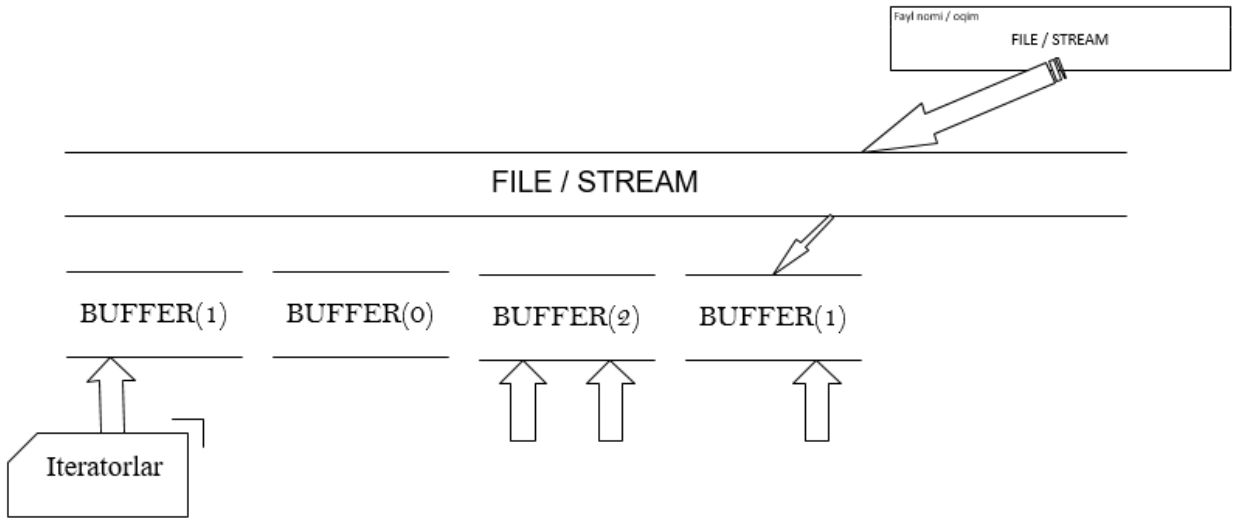
`oqimlar deb aytiladi`). Bunday oqimlarni bir martaga o‘qish kerak. Shuning uchun, unisi va bunisi bilan ishlash uchun bir yondashuvda faylni o‘qiydigan, `forward` iteratorlarni qo‘llab quvvatlaydigan, xotirada faylni to‘liq emas, balki ma’lum bir bo‘lagini saqlaydigan sinf-oqimni yaratish maqsadga muvofiq.

Bunday oqimlar Input iteratorlar uchun ancha oldin yaratilgan, maqsadi faqat ularda Input iteratorlar, bu iterator harakatlanishi uchun bitta bufer bo‘ladi. Qachonki, bufer oxiriga kelganda buferga faylning keyingi bo‘lagi yuklanadi hamda iterator bo‘shatilgan bufer boshidan harakatlanishni boshlaydi (7.1-rasmga qarang).



7.1-rasm. Input iteratorlardan foydalanish.

Forward iteratorlarning input iteratrlardan farqi ularni nusxalash mumkinligidir. Bunday masalada oqimlarda forward iteratorlardan foydalanish uchun buferlar ro‘yxatini hosil qilish orqali yechiladi. Qandaydir iterator birinchi buferga murojaati bilan ro‘yxatga yangi bufer qo‘shiladi va fayldagi blok ma’lumotlar bilan to‘ldiriladi. Ammo bu usulda fayl to‘liq xotiraga yuklanadi,bu holat kerakmas. Shunday qilamizki, buferning chap tomonidagi joylashgan oxirgi chap iterator joylangan barcha buferlari o‘chirilsin. Lekin, unga joylashgan bufer ro‘yxati bilan har bir iterator mos bo‘lmaydi, lekin faqat ularning soni mos bo‘ladi (hisoblovchi iteratorlari). Biror bir bufer 0 iterator qolganini va u eng chap ekanligini bilsa, u barcha undan o‘ng buferdagi barcha qo‘shnilari o‘chiriladi. Chunki ular ham 0 iteratorga ega bo‘ladi(7.2-rasmga qarang).



7.2-rasm. Forward iteratorlardan foydalanish.

Bunday kelib chiqadiki, har bir iterator quyidagilarni o‘z ichiga olishi kerak:

- belgiga ko‘rsatkich(qayta nomlanishda qaytariladigan qiymat);
- bufergadagi belgilar massivning oxiriga ko‘rsatkich iterator harakatlanganda taqqoslash amalga oshiriladi);
- bufer ro‘yxati bo‘yicha iteratorlar (bufer ma’lumotlariga murojjat uchun va ular orqali butun oqim ma’lumotlariga murojjat).

Iterator bufer chegarasiga yetganda, u keyingi buferlar ro‘yxatda borligini tekshiradi, bo‘lmasa - uni yuklaydi, oldingi bufer iteratorlar sonini kamaytiradi va keyingisini sonini oshiradi. Agar oldingi buferda 0 ta iterator qolgan bo‘lsa, u o‘chiriladi. Agar iterator faylning oxirigacha kelgan bo‘lsa, u faqat oldingi buferdan chaqiriladi va "faylning oxiri" holatiga o‘tadi. Bir iteratorning nusxasini olganda – joriy buferda iteratorlarning soni bittaga ko‘payadi, iteratorning destruktori ishlaganda bittaga kammayadi, va, yana 0 bufer bor bo‘lsa qolgan iteratorlar, u va o‘ngda undan keyingi buferlar, shuningdek 0 iteratori borlari o‘chiriladi. Amalga oshirish uchun diskdagi mavzuga oid papkadagi forward_stream.h ga qarang.

Bunday iteratorlar an‘anaviy iteratorlardan ularni ajratib turadigan ba‘zi xususiyatlarga ega. Masalan, bir oqimni (buferlar va ba‘zi qo‘shimcha ma’lumotlar ro‘yxatini saqlaydigan) o‘chirish oldin (destruktor tomonidan), barcha iteratorlar o‘chirilgan bo‘lishi kerak, ularni o‘chirish vaqtida o‘zlariga bog‘liq bo‘lmagan holda buferlarga o‘z navbatida o‘chirilganligini aniqlash murojaat qiladi. Agar begin() usulini chaqirish (birinchi buferni yaratish) natijasida birinchi iteratorni bir marta olsak va u shu paytgacha birinchi bufer allaqachon o‘chirib

tashlangan bo‘lsa, yana begin() usuli yordamida iteratorni ololmaymiz. Bundan tashqari oqimning end() usuli yo‘q. Natijada, oqimda ichki iterator yaratish kerak bo‘ladi, ya‘ni barcha oqimlar yaratilayotganda yaratiladigan va mos yozuvlar iter() usuli yordamida olinishi mumkin. Bundan tashqari, algoritmlarni ishlatganda, iteratorlaran tez-tez nusxalash kerak emas, chunki xotira chapdan o‘nggacha bo‘lgan iteratorda buferlarni saqlaydi. Katta fayllar uchun bu katta xotira talab qilishiga olib keladi.

Yordam sifatida har xil turdagi buferlar mavjud: oddiy (basic_simple_buffer), qator-ustunli iteratorni (basic_adressed_buffer). Bular yordamida hisoblash uchun turli kodlashlar orasida qayta kodlashni bajaruvchi bufer qilish mumkin. Shu munosabat bilan oqim buferning tipi bilan parametrlashtiriladi.

Buyruqlar satrining oxiri null belgi bilan belgilanadi. Bu shuni anglatadiki, uning oxirini topish uchun satr orqali harakat qilsak, ko’rsatkichni boshqa ko’rsatkich bilan solishtirishimiz shart emas (STLda an‘anaviy tarzda amalga oshiriladi), aksincha, ko’rsatkichimiz tomonidan ko‘rsatilgan qiymatni tekshirish kerak. Fayllar bilan bog‘liq vaziyat ham shunday. Belgili kiritishda belgilar asosida sonlarni olamiz va ular katta qiymatlar satriga ega bo‘lgani uchun yana har bir belgini EOF() funksiya bilan taqqoslanadi. Real vaqtda keladigan forward oqimlar uchun faylning oxiri holati oldindan noma’lum.

Bu muammoni satrli ko’rsatkichlar uchun atend(it) funksiyasini tuzish orqali umumlashtirish mumkin:

```
bool atend(const char * p)
{ return !*p; }
```

Agar iterator faylning oxirini ko‘rsatsa, oqim bo‘yicha iteratorlarga true qiymat qaytaradi.

Foydalanuvchi bilan interaktiv hamkorlik qilish uchun (stdin orqali) blokli buferlash amalga oshirilmaydi, chunki blokda odatda bir nechta satr joylashtiriladi va bitta satr kiritilgandan so‘ng, dastur foydalanuvchidan kiritishni kutishda davom etadi, chunki blok hali to‘lmagan. Satrning oxirigacha bo‘lgan belgilar buferni to‘ldiradi va buning uchun satrli buferlash kerak. Ushbu kutubxonada oqim ishga tushirilganda fayl tipini tanlab buferlash turini tanlashingiz mumkin (basic_block_file_on_FILE yoki string_file_on_FILE).

strin – stdin orqali satrli buferlash uchun buferning ichki iteratoridir. Interaktiv bo‘lmagan fayllar uchun oqim yaratishda birinchi belgiga ishora qiluvchi iterator yaratiladi, ya‘ni birinchi bufer yuklanadi. Bunga

strin uchun ruxsat berilmaydi, chunki dasturchi satr rejimidan kiritishni kutishdan oldin biror bir narsa amalga oshirish yoki ekranga chiqarigi mumkin.

Shu sababli birinchi buferni to‘ldirishda satrli fayllari ishlatilganda belgili buyruq ‘\n‘ dan foydalaniladi. Uni o‘qish uchun, start_read_line (it) funksiyasi mavjud, shundan so‘ng satrni tahlil qilish uchun dastur satrni kiritishni kutish rejimiga keladi va keyingi belgi ‘\n‘ tashqarida chiqishda iterator bu satrni ajratib oladi.

Agar bundan keyin yana foydalanuvchidan ma’lumotlar kerak bo‘lsa, dasturchi yana ekranda biror narsani chiqarishni xohlashi mumkin va uni olishdan oldin yana start_read_line(strin) dfn foydalanish mumkin. Bunda quyidagicha takrorlanish hosil qilinadi:

```
while(true){
    cout << "Satr kirit:" <<endl;
    start_read_line(strin); //satr kiritish
    analys(strin);
}
```

Albatta, bu fragment faqat qayta nomlash vaqtida buferni yuklashda iterator talab qilish mumkin, lekin bu qayta nomlash qachon qo‘shimcha tekshirishlar olib kelishi va butun tizimini murakkablashtirishi mumkin.

7.1-dastur. STRIN Kutubxonasi.

```
#define ONE_SOURCE //strin.cpp
#include "../strin.h"

using namespace str;
using std::cout;
using std::endl;

#define ifnot(expr)              if(!(expr))
#define r_if(expr)              if((expr)==0)
#define r_while(expr)        while((expr)==0)
#define r_ifnot(expr)            if(expr)
#define r_whilenot(expr)      while(expr)

template <class it_t, class mes_t>
void dump(it_t tmp, mes_t mes){
    string s;
    read_fix_length(tmp,50,&s);
```

```
    std::cerr <<mes <<dump(s.c_str()) <<endl;
    }

template<class it_t>
const char * read_sum(it_t & it, double * prez);

template<class it_t>
const char * read_expr(it_t & it, double * prez){
    const char * err;
    if(read_s_fix_char(it,'(')){
        r_ifnot(err = read_sum(it,prez))
            return err;
        ifnot(read_s_fix_char(it,')'))
            return "qavsni yoping";
        return 0;
    }
    int x;
    ifnot(read_dec(it,&x)){
        return "son";
    }
    *prez = x;
    return 0;
}


DEF_STRING(md,"*/")
template<class it_t>
const char * read_mul(it_t & it, double * prez){
    typedef char_type<it_t> ch_t;
    const char * err;
    r_ifnot(err = read_expr(it,prez))
        return err;
    while(true){
        ch_t zn;

    ifnot(read_s_charclass_c(it,make_span(md<ch_t>().
s),&zn))
            return 0;
```

```
        double x;
        r_ifnot(err=read_expr(it,&x))
            return err;
        if(zn==(ch_t)'*')   *prez *= x;
        else            *prez /= x;
    }
    return 0;
}

DEF_STRING(pm,"+-")
template<class it_t>
const char * read_sum(it_t & it, double * prez){
    typedef char_type<it_t> ch_t;
    const char * err;
    r_ifnot(err = read_mul(it,prez))
        return err;
    while(true){
        ch_t zn;

    ifnot(read_s_charclass_c(it,make_span(pm<ch_t>().
s),&zn))
            return 0;
        double x;
        r_ifnot(err = read_mul(it,&x))
            return err;
        if(zn==(ch_t)'+')   *prez += x;
        else            *prez -= x;
    }
    return 0;
}

int main(){
    using namespace std;
    if(0)   {
        string str_expression_1 = "5+84/(51)";
        string str_expression_2 = "15 + (16 /6 *4)";
        string str_expression_3 = "7 + ( 4* 4)";
        string str_expression = str_expression_3;
```

```
        cout <<"hisoblash ifodasi:" <<endl
<<str_expression <<endl;
        const char * p= str_expression.c_str();
        const char * err;
        double rez;
        r_ifnot(err=read_sum(p,&rez)){
            int pos=p-str_expression.c_str();
            for(int i=1; i<pos; i++)
                cout <<' ';
            cout <<'^' <<endl;
            cout <<"Xatolik: " <<err <<endl;
            return -1;
        }
        cout << "natija: " << rez << endl;
        cout <<"================" << endl;
    }

    if(1)   {
        try{
            while(!atend(strin)){
                cout <<"arifmetik ifoda kiriting va
'end' bilan tugating " <<endl;
                read_start_line(strin);
                const char * err;
                double rez=0;
                err=read_sum(strin,&rez);
                linecol lc=get_linecol(strin);
                read_until_str(strin,"end");
                r_if(err)
                    cout << "natija: " << rez <<
endl;
                else{
                    cout <<" zanjir " <<lc <<"
xatolik: " <<err <<endl;
                }
            }
        }
        catch(const char * mes){
```

```
            cerr << "xatolik: " << mes << endl;
            return -1;
        }
        catch(string & mes){
            cerr << "xatolik: " << mes << endl;
            return -1;
        }
        catch(stream_exception & mes){
        cerr << "xatolik: " << mes.what() << endl;
            return -1;
        }
        catch(...){
            cerr << "noaniq xatolik" << endl;
        }
    }
    return 0;
}

#ifndef STRIN_H
#define STRIN_H

#include <exception>
#include "base_parse.h"
#include "forward_stream.h"

#ifdef ONE_SOURCE
#   define STRIN_EXTERN
#else
#   define STRIN_EXTERN extern
#endif

namespace str{

    STRIN_EXTERN forward_adressed_stream STREAMin
#ifdef ONE_SOURCE
    (true, new file_on_FILE_stringbuf(stdin))
#endif
    ;
```

```
    STRIN_EXTERN forward_adressed_stream::iterator &
strin = STREAMin.iter();
    inline int __set01(typename
forward_adressed_stream::iterator & it){
        set_linecol(it,linecol(0,1));
        return 0;
    }
    STRIN_EXTERN int __unused_int
#ifdef ONE_SOURCE
    = __set01(strin)
#endif
    ;

class str_error{
#define constructor str_error
typedef str_error my_t;
protected:
    const char * message;
public:
    constructor():message(0){}
    constructor(const char * mes):message(mes){}
    constructor(int x):message(0){  if(x)throw "???
int";   }
    constructor(bool x) :message(x?0:""){}
    operator bool()const{   return message==0;  }
    const char * what()const {  return message; }
#undef constructor
};
DECLARE_AND_OR_default_error_operators(str_error)
DECLARE_AND_OR_error_operators_for_short_circuiting(s
tr_error)
str_error operator>>(base_parse_error from, str_error
to){
    if(from) return str_error(true);
    elsereturn to;
}
class strpos_error : public str_error{
#define constructor strpos_error
```

```
typedef strpos_error my_t;
typedef str_error base_t;
    linecol lc;
    friend strpos_error operator&&(strpos_error l,
strpos_error r);
    friend strpos_error operator||(strpos_error l,
strpos_error r);
public:
    linecol ignored_lc=linecol(0,0);
    bool ignored()const{return lc==ignored_lc;  }
    constructor(){};
    constructor(const char * mes):base_t(mes){}
    constructor(int x)  :base_t(x){}
    constructor(bool x) :base_t(x){}

    constructor(const linecol & l)
        :lc(l){};
    constructor(const char * mes, const linecol & l)
        :base_t(mes),lc(l){};
    constructor(bool mes, const linecol & l)
        :base_t(mes),lc(l){};

    linecol where()const
    {   return lc;  }
    my_t & set_what(const char * mes)
    {   base_t::message=mes;return *this;   }
    template<class it_t>
    my_t & set_ignored(const it_t & it)
    {   ignored_lc =linecol(it);return *this;   }
#undef constructor
};

template <class it_t>
strpos_error operator>>(str_error from, const it_t *
to){
    return strpos_error(from.what(),linecol(*to));
}
std::ostream & operator<<(std::ostream & str, const
```

```
strpos_error & e){
    return str<<"("<<"ignored:"<<e.ignored()<<" at
"<<e.ignored_lc<<")"<<(bool)e<<";"<<(e?"OK":e.what())
<<" at "<<e.where();
}
strpos_error operator&&(strpos_error l, strpos_error
r){
    r.set_ignored(l.ignored_lc);
    if(l)
        if(r)
            if(r.ignored())
                return l;
            else
                return r;   /*TT*/
        else
            return r;   /*TF*/
    else
        if(r)
            return l;   /*FT*/
        else
            if(r.ignored())
                return l;
            else
                return r;   /*FF*/
}
strpos_error operator||(strpos_error l, strpos_error
r){
    r.set_ignored(l.ignored_lc);
    if(l)
        if(r)
            if(r.ignored())
                return l;
            else
                return r;   /*TT*/
        else
            return l;   /*TF*/
    else
        if(r)
```

```
            return r;   /*FT*/
        else
            if(r.ignored())
                return l;
            else
                return r;   /*FF*/
}
DECLARE_AND_OR_error_operators_for_short_circuiting(s
trpos_error)

}//namespace str
#endif //STRIN_H
```

Sintatik tahlillovchilarning bazaviy funksiyalari. Har safar terminal funksiyalarni yozish dasturchilarga noqulaylik keltirmasligi uchun «base_parse.h» sinfida tayyorlab qo‘yilgan. Hozirda u umumiy ko‘rinishda (xaqiqiy sonlardan tashqari) tayyorlangan va kelajakda satr bo‘yicha ko‘rsatkichlar va oqim bo‘yicha iteratorlar uchun satrli fuknsiyalar (strcmp, strstr, strchr, strspn, strcspn kabi) ishlaydigan qismini tuzish rejalashtirilgan. Shuningdek bu faylning oxiri haqida o‘ylashi kerak emas, faqat fragmentni to‘g‘ri yozsa bo‘ladi.

Quyida parsingning bazaviy funksiyalari qisqacha keltirilgan va ikki test parserni amalga oshirish paytida ularning foydalanish statistikasi va natijaviy qiymatlarni qaytaradigan dastur keltirilgan.

```
size_t      n
ch_t        c
ch_t *      s
func_obj    bool is(c)  // bool isspace(c)
span        spn
bispan      bspn
func_obj    err pf(it*) // int read_spc(it*)
func_obj    err pf(it*, rez*)
```

len – belgilar soni, *pstr ga qo‘shilgan, satr oxiri yoki oxiri emasligi aniqlovchi statistika va qiymatlari qaytaradi.

int read_until_eof	(it&)	.*$	0	0	1	OK
int read_until_eof	(it&, pstr*)	.*$	len	len		OK
int read_fix_length	(it&, n)	.{n}	-1	0		OK
int read_fix_length	(it&, n, pstr*)	.{n}	-(1+len)	0		
2 OK						

int read_fix_str	(it&, s)	str	-(1+len)	0 ili (1+len)	9	OK
int read_fix_char	(it&, c)	c	-1	0 yoki 1	11	OK
int read_charclass	(it&, is)	[ ]	-1	0 yoki 1		OK
int read_charclass	(it&, spn)	[ ]	-1	0 yoki 1		OK
int read_charclass	(it&, bspn)	[ ]	-1	0 yoki 1		OK
int read_charclass_s	(it&, is, pstr*)	[ ]	-1	0 yoki 1		OK
int read_charclass_s	(it&, spn, pstr*)	[ ]	-1	0 yoki 1	1	OK
int read_charclass_s	(it&, bspn, pstr*)	[ ]	-1	0 yoki 1	5	OK
int read_charclass_c	(it&, is, ch*)	[ ]	-1	0 yoki 1		OK
int read_charclass_c	(it&, spn, ch*)	[ ]	-1	0 yoki 1	1	OK
int read_charclass_c	(it&, bspn, ch*)	[ ]	-1	0 yoki 1		OK
int read_c	(it&, ch*)	.	-1	0	5	OK
int read_while_charclass	(it&, is)	[ ]*	-(1+len)	len		OK
int read_while_charclass	(it&, spn)	[ ]*	-(1+len)	len	2	OK
int read_while_charclass	(it&, bspn)	[ ]*	-(1+len)	len		OK
int read_while_charclass	(it&, is, pstr*)	[ ]*	-(1+len)	len		OK
int read_while_charclass	(it&, spn, pstr*)	[ ]*	-(1+len)	len		OK
int read_while_charclass	(it&, bspn, pstr*)	[ ]*	-(1+len)	len	1	OK
int read_until_charclass	(it&, is)	.*[ ]<-	-(1+len)	len		OK
int read_until_charclass	(it&, spn)	.*[ ]<-	-(1+len)	len	1	OK
int read_until_charclass	(it&, bspn)	.*[ ]<-	-(1+len)	len		OK
int read_until_charclass	(it&, is, pstr*)	.*[ ]<-	-(1+len)	len		OK
int read_until_charclass	(it&, spn, pstr*)	.*[ ]<-	-(1+len)	len	2	OK
int read_until_charclass	(it&, bspn, pstr*)	.*[ ]<-	-(1+len)	len		OK
int read_until_char	(it&, c)	.*c	-(1+len)	len		OK
int read_until_char	(it&, c, pstr*)	.*c	-(1+len)	len		OK

Oxirgi belgini o‘qigandan so‘ng, iterator undan keyin emas, balki unga qaratilgan.

Funksiya nomidan foydalanib, xato kodini yoki xato xabarini qaytarish qulay, shuning uchun parserlar matni yanada aniqroq ko‘rinadi, maxsus makroslar keltiramiz:

```
#define r_if(expr)              if((expr)==0)
#define r_while(expr)        while((expr)==0)
#define r_ifnot(expr)            if(expr)
#define r_whilenot(expr)      while(expr)
#define rm_if(expr)             if((expr)>=0)
#define rm_while(expr)       while((expr)>=0)
```

```
#define rm_ifnot(expr)          if((expr)<0)
#define rm_whilenot(expr)    while((expr)<0)
#define rp_if(expr)             if((expr)>0)
#define rp_while(expr)       while((expr)>0)
#define rp_ifnot(expr)          if((expr)<=0)
#define rp_whilenot(expr)    while((expr)<=0)
```

Bu ma’lumotlarni saqlash va bool tipiga mos akslantirish bo‘lsada ba‘zi sinfni yaratish uchun juda muhim.

Umuman, o‘zgarmas bo‘lgan havolalarga qarshiman, funksiyasi o‘zgartirish kerak argumentlar ko’rsatkichlari bilan emas, balki bir ko‘rsatkich orqali unga o‘tgan bo‘lishi kerak

Ushbu funksiyalardan foydalanishning ikkita misol keltiramiz:

7.2-dastur. Arifmetik ifodalarni hisoblash.

```
#define ONE_SOURCE //strin.cpp
#include "../strin.h"

using namespace str;
using std::cout;
using std::endl;

#define ifnot(expr)              if(!(expr))
#define r_if(expr)              if((expr)==0)
#define r_while(expr)        while((expr)==0)
#define r_ifnot(expr)            if(expr)
#define r_whilenot(expr)      while(expr)

template <class it_t, class mes_t>
void dump(it_t tmp, mes_t mes){
    string s;
    read_fix_length(tmp,50,&s);
    std::cerr <<mes <<dump(s.c_str()) <<endl;
    }

template<class it_t>
const char * read_sum(it_t & it, double * prez);

template<class it_t>
const char * read_expr(it_t & it, double * prez){
```

```
    const char * err;
    if(read_s_fix_char(it,'(')){
        r_ifnot(err = read_sum(it,prez))
            return err;
        ifnot(read_s_fix_char(it,')'))
            return "qavsni yoping";
        return 0;
    }
    int x;
    ifnot(read_dec(it,&x)){
        return "son";
    }
    *prez = x;
    return 0;
}


DEF_STRING(md,"*/")
template<class it_t>
const char * read_mul(it_t & it, double * prez){
    typedef char_type<it_t> ch_t;
    const char * err;
    r_ifnot(err = read_expr(it,prez))
        return err;
    while(true){
        ch_t zn;

    ifnot(read_s_charclass_c(it,make_span(md<ch_t>().
s),&zn))
            return 0;
        double x;
        r_ifnot(err=read_expr(it,&x))
            return err;
        if(zn==(ch_t)'*')   *prez *= x;
        else             *prez /= x;
    }
    return 0;
}
```

```
DEF_STRING(pm,"+-")
template<class it_t>
const char * read_sum(it_t & it, double * prez){
    typedef char_type<it_t> ch_t;
    const char * err;
    r_ifnot(err = read_mul(it,prez))
        return err;
    while(true){
        ch_t zn;

    ifnot(read_s_charclass_c(it,make_span(pm<ch_t>().
s),&zn))
            return 0;
        double x;
        r_ifnot(err = read_mul(it,&x))
            return err;
        if(zn==(ch_t)'+')   *prez += x;
        else             *prez -= x;
    }
    return 0;
}

int main()
{
    using namespace std;
    if(0)
    {
        string str_expression_1 = "5+84/(51)";
        string str_expression_2 = "15 + (16 /6 *4)";
        string str_expression_3 = "7 + ( 4* 4)";
        string str_expression = str_expression_3;
        cout <<"hisoblash ifodasi:" <<endl
<<str_expression <<endl;
        const char * p= str_expression.c_str();
        const char * err;
        double rez;
        r_ifnot(err=read_sum(p,&rez)){
```

```
            int pos=p-str_expression.c_str();
            for(int i=1; i<pos; i++)
                cout <<' ';
            cout <<'^' <<endl;
            cout <<"Xatolik: " <<err <<endl;
            return -1;
        }
        cout << "natija: " << rez << endl;
        cout <<"================" << endl;
    }

    if(1)
    {
        try{
            while(!atend(strin)){
                cout <<"arifmetik ifoda kiriting va
'end' bilan tugating " <<endl;
                read_start_line(strin);
                const char * err;
                double rez=0;
                err=read_sum(strin,&rez);
                linecol lc=get_linecol(strin);
                read_until_str(strin,"end");
                r_if(err)
                cout << "natija: " << rez << endl;
                else{
                    cout <<" zanjir " <<lc <<"
xatolik: " <<err <<endl;
                }
            }
        }
        catch(const char * mes){
            cerr << "xatolik: " << mes << endl;
            return -1;
        }
        catch(string & mes){
            cerr << "xatolik: " << mes << endl;
            return -1;
```

```
        }
        catch(stream_exception & mes){
        cerr << "xatolik: " << mes.what() << endl;
            return -1;
        }
        catch(...){
            cerr << "noaniq [atolik" << endl;
        }
    }

    return 0;
}
```

7.2-dastur. Output

```
--> runcpp -f calc.cpp
option -f
runcpp: g++ calc.cpp (options = -std=gnu++11 -Wall
-Wno-parentheses)
runcpp: compile messages lines 0
arifmetik ifoda kiriting va 'end' bilan tugating
1
end
natija: 1
arifmetik ifoda kiriting va 'end' bilan tugating
1+1
end
natija:  2
arifmetik ifoda kiriting va 'end' bilan tugating
1+1+1+1+7*7*(3)
end
 zanjir 5:1 xatolik: son

arifmetik ifoda kiriting va 'end' bilan tugating
1end
natija:  1
arifmetik ifoda kiriting va 'end' bilan tugating
7*(3)
end
```

```
 zanjir 8:1 xatolik: son

arifmetik ifoda kiriting va 'end' bilan tugating
1+(2)
end
natija:  3
arifmetik ifoda kiriting va 'end' bilan tugating
1+e end
 zanjir 12:3 xatolik: son
arifmetik ifoda kiriting va 'end' bilan tugating
(7)*3end
natija: 7
arifmetik ifoda kiriting va 'end' bilan tugating
7*(3)end
 zanjir 2:1 xatolik: son

arifmetik ifoda kiriting va 'end' bilan tugating
(5)+(5)end
natija: 10
```

Bugungi kunda dasturchilar uchun maxsus sintaktik tahlillovchilar yaratish koʻp hollarda shart emas. Chunki bu mashaqqatli va koʻp vaqt talab qiladi. Umuman olganda qandaydir dasturiy taʻminotlarni, axborot tizimlarining lingvistik taʻminoti yaratilsa, shu taʻminot uchun analizator yaratish kerak boʻlishi mumkin. Dunyoda dasturlash tillarini ishlab chiqaruvchilar bu sohada juda oldilab ketgan. Shuning uchun sintatik tahlillovchilar boʻyicha nazariy tushunchaga ega boʻlish yetarlidi. Ammo ayrim matematik yangi sonli sinflar uchun ishlab chiqish mumkin.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Dasturlash tilining sintaktik konstruksiyalarini tasvirlash uchun eng qulay formal usul qaysi usul?
2. Dasturning sintaktik toʻgʻriligini aniqlashning yana bir muhim qismi - dasturda tiplardan foydalanishning toʻgʻriligini qaysi grammatika yordamida aniqlab boʻlmaydi?
3. Sintaktik tahlil deganda nima tushuniladi?
4. n uzunlikdagi kiruvchi satr uchun murakkabligi qanchadan oshmasligi

kerak bo‘lgan har qanday kontekst-erkin grammatika uchun analizator (parser) qurish mumkin?

5. Sintaktik tahlillovchi kirish parametrlari nimadan iborat?
6. Sintaktik tahlillovchi chiqish parametrlari nimalardan iborat?
7. Leksik va sintaktik tahlil bosqichlarining alohida qarashlarga ajratilishi shartmi?
8. Sintaktik tahlillovchi sinflari nechta va qaysi sinf algoritmlariga ta‘luqli?
9. Past sathlarga yo‘naltirilgan analizatorlar chiqishni qurish uchun grammatika nimadan boshlab va nima bilan tugaydi.
10. LL grammatika bo‘lmagan grammatikalarda analizatorlar uchun qaysi usullardan foydlanish mumkin.
11. Past sathlarga yo‘naltirilgan analizatorlarga qaraganda yuqori sathlarga yo‘naltirilgan analizatorlar qanchalik ishlatiladi.
12. Yuqori sathlarga yo‘naltirilgan analizatorlar bilan qaysi grammatika bilan bog‘liq.
13. Bugungi kunda qaysi grammatikaga asoslangan analizatorlar bilan juda ko‘p dasturlash tillari foydalanmoqda.
14. Eng oddiy va juda ko‘p foydalanilgan past sathlarga yo‘naltirilgan analizatorlarni qurish usulini ayting?
15. Ifodalarni hisoblashda barcha hisoblashlarni qanday sinflarga ajratish mumkin.
16. Chap operandani tanlashimiz va kerakli amalni bajarish uchun son yoki oddiy ifodadan iborat o‘ng operandani aniqlashimiz lozim, so‘ng esa nimani bajarish mumkin.
17. Qo‘shish tipiga oid amallarni hisoblash uchun ifodani hisoblashning umumiy formulasi qanday?
18. Qiymat qaytarmaslik asosidagi rekursiv kamayish usulidan faqat qaysi shart bajarilganda foydalanish mumkin.
19. KS grammatika uchun terminal va terminal bo‘lmagan belgilardan iborat G grammatika va w zanjir asosida FIRST $_k$ (w) to‘plamni qanday aniqlaymiz?
20. FIRST $_k$ (w) to‘plam w dan olingan terminal zanjirlarning uzunligi k bo‘lgan barcha terminal nimalaridan iborat.
21. Agar w zanjir faqat terminallardan iborat bo‘lsa, FIRST $_k$ (w) - w zanjirda birinchi k belgilardir, aks holda |w| >=, yoki agar |w| < k < bo‘lsa, nima bo‘ladi.
22. Agar x teminal emas va grammatika qoidasi x → y_1 y_2 … y_k bo‘lsa, FIRST(X)ga qanday terminalni qo‘shamiz.

23. Agar FIRST $_k$ (x) = FIRST$_k$ (y) teng, ... o‘rinli bo‘lsa G = (V_T, V_N, P, S) grammatika *LL*(k)-grammatika deb aytiladi.
24. LL (k) – xususityalari grammatika uchun qanday cheklovlar yuklaydi.
25. Agar grammatikada A =>* Aw chiqish mavjud bo‘lsa, A terminal bo‘lmagan KS grammatika nima deb aytiladi.
26. Qiymat qaytarish asosidagi rekursiv kamayish usulini qo‘llash uchun grammatikani FIRST to‘plamlar qanday shaklga aylantirish kerak.
27. Leksik analizator an‘anaviy usullarni to‘laligicha, scan va next ana‘naviy usullar va nusxalash konstruktoriga ega bo‘lgan nima sifatida namoyon bo‘ladi?
28. Mashina tilidagi matn nimalardan iborat?
29. Har bir terminal bo‘lmagan tilning ma’lum bir jumlasiga mos keladigan qaysi shakl yordamida tasvirlash qulay?
30. Qaysi ko‘rinishida expr ::= expr1|expr2|expr3 funksiyani yozamiz?
31. Forward iteratorlar nima uun kerak?
32. Input iteratorlar nima uchun kerak?
33. Input iteratorlardan foydalanishni tushuntirib bering?
34. Forward iteratorlardan foydalanishni tushuntirib bering?
35. Forward iteratorlarning input iteratorlardan farqi nimada?

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I	
	Sintaktik analizatorni yaratishga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar
`// Created by MBBahodir` `class SimpleParser {`	1. Sinf xatosini toping. _____ _____ _

```
const int IF = 1;
const int THEN = 2;
const int ELSE = 3;
const int BEGIN = 4;
const int END = ;
const int PRINT = 6;
const int SEMICOLON =
7;
const int NUM = 8;
const int EQ = 9;
public static void
nextStep(int lc){
        if
(lexical_class == lc)
    lexical_class =
getLC();
        else
            error();
    }
    public static
void S(void){
    switch(getLC())
        {
        case IF:
            E();
nextStep(THEN); S();
nextStep(ELSE); S();
break;
        case BEGIN:
            S(); L();
break;
        case PRINT:
```

2. Sinfdan obyekt olib dasturda foydalaning.

_____
_____
_____
_____

3. Faqat qo‘shish amalini tahlil qiluvchi va ifodani hisoblovchi dastur tuzing.

_____
_____
_____
_____
_____
_____

4. Faqat ayrish amalini tahlil qiluvchi va ifodani hisoblovchi dastur tuzing.

_____
_____
_____
_____
_____
_____
_____
_____

5. Qo‘shish va ayrish amallarini tahlil qiluvchi va ifodani hisoblovchi dastur tuzing.

_____
_____
_____
_____
_____

```
            E();
break;
        default:
            error();
break;
        }   }
    public static
void L(void){
        switch
(lexical_class)
        {
        case END:
            getLC();
break;
        case
SEMICOLON:
            getLC();
S(); L(); break;
        default:
            error();
break;
        }
    }
    public static
void E(void){
    nextStep(NUM);
nextStep(EQ);
nextStep(NUM);
    }
    public static
void main(void){
        lexical_class
= getLC();
        S();
    }
}
```

6. Ko‘paytirish amalni tahlil qiluvchi va ifodani hisoblovchi dastur tuzing.

_____
_____
_____
_____
_____
_____
_____

7. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
8. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

2.4. Murakkab saralash algoritmlari. Amaliy dasturlash

📖 Murakkab saralash algoritmlarini, ularning murakkabligi, rekursiv foydalanish va takomillashtirish, ishlatish g‘oyalari va saralash algoritmlarining 16 ta variantining turli dastur fragmentlari, saralash algoritmlarni taqqoslash, turli sondagi massivlar, turli qiymatlarda o‘zlarini tutishini solishtirish natijalari, katta sonlar bilan ishlash, mashina xatoligini aniqlash, foydalanuvchining arifmetik amallarni bajaruvchi funksiyalarni yaratish bo‘yicha g‘oyalar, katta sonli sinflarni ishlatish, mavjud katta sonlar bilan ishlovchilar, katta sonlari bilan ishlovchi kutubxonalarni yaratish va namunalari keltirilgan bo‘lib, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. bilimlarni mustahkamlash uchun 35 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 4 ta assisment topshirig‘i va har assismentda 29 ta topshiriq berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

🖋 ***Kalit so‘zlar.*** Saralash, massiv, tip, xotira, sinf kutubxona, massiv, satrli massiv.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Tip tushunchasi, sinf va sinf obyekti, oqim, kirish elementga murojaat, funksiya va ko‘rsatkich, ma’lumotlarni kiritish va chiqarishi, arifmetik amallar, kutubxona yaratish va dasturlashga oid dastlabki tushunchalar hamda C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👌 ***Bilib olasiz***. Murakkab saralash algoritmlardan: Havo sharcha kabi saralash, Shaker saralash, Taroqsimon saralash, Qo‘shish orqali saralash, Shell saralash, Iearxik saralash, Nav saralash, Tanlash orqali saralash, Piramida kabi saralash, Tezkor saralash, Birlashtirish orqali saralash, Blok orqali saralash, Bitli saralash (LSD), Bitli saralash (MSD), Bitonik saralash, Timsort saralash, bu algoritmlarni taqqoslash va murakkabligini, Tiplar va ularning xususiyatlari, Katta sonlar chegarasini tekshirish, Tiplarga bog‘liq katta sonlardan foydalanish va xatoligi, Katta sonlarning qo‘shish, ayirish, ko‘pytirish g‘oyalari va algoritmlarini tuzishni, Bignumber kutubxonasi, sinfi va funksiyalari, xususiyatlari yaratish, OpenSSL, Libgcrypt, GMP, Boost.Multiprecision, wide integer tiplari, BigInt kutubxonasini yaratish uslubi va bazaviy funksiyalarini o‘rganishingiz mumkin.

REJA

1. Murakkab saralash algoritmlari.
2. Saralash algoritmlari taqqoslash.
3. Juda ham katta sonlar bilan ishlash.

4. Large sinfining qo‘llanilishi.
5. Integer sinfining qo‘llanilishi

KIRISh

Turli ma’lumotlar tuzilmalaridan iborat to‘plamlarni saralash algoritmlarini bilasiz. Ammo ularni ko‘p sonli qiymatlari o‘zini tutishi qanday degan savol qiziqtiradimi? Nafaqat oddiy saralash balki shunday murakkab saralash algoritmlari ham mavjud. Ko‘p elementli massivlar katta sonlar bilan ishlashga ham foydalanish mumkin. Shuningdek tiplarning o‘zaro modifikatsiyasi bilan ham uzun sonlar bilan ishlanadi.

Saralash algoritmlari va murakkabligi. Avvalo, algoritmlarni tadqiq qilishga, ularni imkon qadar tez ishlashi uchun optimallashtirish muhimdir. Bu ustida ishlayotganda, turli saralash uchun samarali usullarni o‘ylab topish imkoniyatiga ega bo‘lishingiz mumkin.

Ko‘p jihatdan, barcha saralash algoritmlarni o‘rganish va ularni sinash kerak. Agar dasturlashning o‘zi haqida gapiradigan bo‘lsak, ba‘zan kutilmagan qiyinchiliklar paydo bo‘lishi mumkin (C++ optimizatori juda yaxshi). Biroq, qaysi testlarni va qanday miqdorda amalga oshirilishi kerakligini hal qilish qiyin emas. Men ko‘rsata olmaydigan yagona narsa-bu deyarli 150 GB vaznga ega qiymatlarni saralash bo‘ldi.

Asosiy saralash algoritmlari tavsifi va ularni amalga oshirish usullari. Saralashni qisqacha va aniq ta‘riflashga va murakkabligini belgilashga harakat qilaman. Murakkab ma’lumotlar tuzilmalarni foydalanishda(daraxt saralash kabi) odatda xotira katta miqdorda sarflanadi va eng yomon holatda boshqa xil faqat yordamchi qator yaratish kerak bo‘ladi. Barqarorlik (stabillik) saralash tushunchasi ham mavjud. Demak, elementlarning nisbiy tartibi teng bo‘lganda o‘zgarmaydi.

Havo sharcha kabi saralash (Bubble sort). Massivda chapdan o‘ngga qarab amal bajariladi. Agar joriy element keyingisidan katta bo‘lsa, ularni almashtiramiz. Buni massiv tartiblanmaguncha bajaramiz. E‘tibor bering, birinchi iterasiyadan keyin eng katta element massivning oxirida, to‘g‘ri joylashagan bo‘ladi. Ikkita iterasiyadan keyin ikkinchi eng katta element to‘g‘ri joylashgan bo‘ladi va hokazo. Ravshanki, n ta iterasiyadan ko‘p bo‘lmagandan keyin massiv tartiblangan bo‘ladi. Shunday qilib, eng yomon va o‘rtacha holatda murakkabligi $O(n^2)$, eng yaxshi holatda esa O(n) bo‘ladi.

8.1-dastur. Saralashni amalga oshirish.

```
void bubblesort(int* l, int* r) {
```

```
    int sz = r - l;
    if (sz <= 1) return;
    bool b = true;
    while (b) {
        b = false;
        for (int* i = l; i + 1 < r; i++) {
            if (*i > *(i + 1)) {
                swap(*i, *(i + 1));
                b = true;
            }
        }
        r--;
    }
}
```

Shaker saralash (Shaker sort, bundan tashqari, aralashgan saralash va kokteyl saralash sifatida tanilgan). Havo sharcha saralash oxirida kichik elementlar joylashtirilgan testlarda sekin ishlaydi. Bu element algoritmning har bir qadamida chap tomonga faqat bitta pozitsiyaga harakat qiladi. Shuning uchun nafaqat chapdan o‘ngga, balki o‘ngdan chapga ham o‘tadi. Massivning tartiblanmagan qismini aniqlash uchun ikkita begin va end ko‘rsatikichni qo‘llab quvvatlaydi. Keyingi iterasiyada end ga yetganimizda undan birni ayiramiz va o‘ngdan chapga ko‘chamiz. Xuddi shunday, beginga etganimizda unga birni qo‘shamiz va chapdan o‘ngga ko‘chamiz . Algoritm havo sharcha saralash bilan bir xil murakkabligi bor, lekin real vaqtda yaxshi ishlaydi.

8.2-dastur. Saralashni amalga oshirish.

```
void shakersort(int* l, int* r) {
    int sz = r - l;
    if (sz <= 1) return;
    bool b = true;
    int* beg = l - 1;
    int* end = r - 1;
    while (b) {
        b = false;
        beg++;
        for (int* i = beg; i < end; i++) {
            if (*i > *(i + 1)) {
                swap(*i, *(i + 1));
```

```
                b = true;
            }
        }
        if (!b) break;
        end--;
        for (int* i = end; i > beg; i--) {
            if (*i < *(i - 1)) {
                swap(*i, *(i - 1));
                b = true;
            }
        }
    }
}
```

Taroqsimon saralash (Comb sort). Shaker saralashning yana bir o‘zgartirilgani. "Toshbaqalar" dan qutulish uchun masofada turgan elementlarni qayta tashkil qilamiz. Keling, uni tuzatamiz va chapdan o‘ngga boramiz, bu masofada turgan elementlarni taqqoslaymiz, agar kerak bo‘lsa, ularni qayta tartibga solaylik. Shubhasiz, bu "toshbaqalar"ni tez qator boshiga olish imkonini beradi. Dastlab qator uzunligiga teng masofani olib, so‘ngra uni taxminan 1.247 ga teng bo‘lgan biror koeffitsientga bo‘lish maqbuldir. Masofa bir biriga teng bo‘lganda shaker saralash bajariladi. Eng yaxshisi, murakkabligi O(nlogn), eng yomoni, u $O(n^2)$. O‘rtacha murakkabligi nima uchun juda aniqmas, amalda O(nlogn) kabi ko‘rinadi.

8.3-dastur. Saralashni amalga oshirish.

```
void combsort(int* l, int* r) {
    int sz = r - l;
    if (sz <= 1) return;
    double k = 1.2473309;
    int step = sz - 1;
    while (step > 1) {
        for (int* i = l; i + step < r; i++) {
            if (*i > *(i + step))
                swap(*i, *(i + step));
        }
        step /= k;
```

```
    }
    bool b = true;
    while (b) {
        b = false;
        for (int* i = l; i + 1 < r; i++) {
            if (*i > *(i + 1)) {
                swap(*i, *(i + 1));
                b = true;
            }
        }
    }
}
```

Qo‘shish orqali saralash (Insertion sort). Algoritm yakunlangandan keyin javobni o‘z ichiga olgan bir yangi massiv yaratiladi. Javob massividagi elementlar har doim tartiblanib turishi uchun manba massividan elementlarni birma-bir kiritamiz. Murakkabligi o‘rtacha va eng yomon hol $O(n^2)$ va eng yaxshi O(n) bo‘ladi. Boshqa algoritmlarga nisbattan joriy qilish yaxshiroq (yangi massiv yaratish va albatta u nimanidir qo‘yish qiyin): joriy massivning ba‘zi bir prefikslari tartiblashtirilganiga ishonch hosil qilish va modomiki ular noto‘g‘ri tartibda ekan uni kiritish o‘rniga joriy elementni avvalgisi bilan o‘zgartiramiz.

8.4-dastur. Saralashni amalga oshirish.

```
void insertionsort(int* l, int* r) {
    for (int *i = l + 1; i < r; i++) {
        int* j = i;
        while (j > l && *(j - 1) > *j) {
            swap(*(j - 1), *j);
            j--;
        }
    }
}
```

Shell saralash (Shellsort). Saralashda bir xil fikrdan foydalanishni o‘ylab ko‘ring, taroqsimon saralash va qo‘shish orqali saralash uchun uni qo‘llaymiz. Masofa tushunchasini kiritaylik va tanlaylik. So‘ngra massiv elementlari sinflarga ajratiladi, shu sinfga o‘zgarmas masofaning bir nechta elementlari kiradi. Har bir sinfni qo‘shib, tartiblash yo‘li bilan tartiblaylik. Taroqsimon saralashdan farqli o‘laroq, optimal masofalar to‘plami aniqmas. Turli taxminlarga ega bo‘lgan bir nechta ketma-

ketliklar mavjud bo‘lsin. Shell saralashda birinchi element ketma-ketligi massiv uzunligiga teng, har bir keyingi element ketma-ketligi oldingisining yarim o‘lchamiga tengdir. Hibbard ketma-ketligi 2n-1 uchun murakkabligi eng yomon holat $O(n^2)$ dir, Sedgwick ketma-ketligi $O(n^{4/3})$ uchun (ifoda o‘zgaruvchan xususiyatli) va Pratt (ikki va uch darajali elementlar) ketma-ketligi $O(nlog2^n)$ murakkabligi eng yomon holat $O(n^{1,5})$dir. Bu barcha ketma ketliklar faqat massiv hajmiga qadab hisoblab va kichik uchun ko‘proq ishlatish kerak (aks holda u faqat qo‘shimchalari bilan tartiblashtiriladi). Qo‘shimcha tadqiqot qildim va $s_i = a * s_i\text{-}1 + k * s_i\text{-}1$ shaklida turli ketma ketliklar (bu qisman empirik siur ketma – ketlikda ilhomlanib, elementlar kichik soni uchun eng yaxshi masofa ketma ketliklardan biri) sinovdan o‘tkazdim. Eng yaxshi ketma ketliklar a = 3, k =1/3, a = 4, k = 1/4 va a = 5, k = -1/5 koeffitsentlari bilan maqsadga erishish mumkin.

8.5-dastur. Saralashni amalga oshirish.

1- variant

```
void shellsort(int* l, int* r) {
    int sz = r - l;
    int step = sz / 2;
    while (step >= 1) {
        for (int *i = l + step; i < r; i++) {
            int *j = i;
            int *diff = j - step;
            while (diff >= l && *diff > *j) {
                swap(*diff, *j);
                j = diff;
                diff = j - step;
            }
        }
        step /= 2;
    }
}
```

2- variant

```
void shellsorthib(int* l, int* r) {
    int sz = r - l;
    if (sz <= 1) return;
    int step = 1;
    while (step < sz) step <<= 1;
    step >>= 1;
```

```
    step--;
    while (step >= 1) {
        for (int *i = l + step; i < r; i++) {
            int *j = i;
            int *diff = j - step;
            while (diff >= l && *diff > *j) {
                swap(*diff, *j);
                j = diff;
                diff = j - step;
            }
        }
        step /= 2;
    }
}
```

3- variant

```
int steps[100];
void shellsortsedgwick(int* l, int* r) {
    int sz = r - l;
    steps[0] = 1;
    int q = 1;
    while (steps[q - 1] * 3 < sz) {
        if (q % 2 == 0)
            steps[q] = 9 * (1 << q) - 9 * (1 << (q
/ 2)) + 1;
        else
            steps[q] = 8 * (1 << q) - 6 * (1 << ((q
+ 1) / 2)) + 1;
        q++;
    }
    q--;
    for (; q >= 0; q--) {
        int step = steps[q];
        for (int *i = l + step; i < r; i++) {
            int *j = i;
            int *diff = j - step;
            while (diff >= l && *diff > *j) {
                swap(*diff, *j);
                j = diff;
```

```
                diff = j - step;
            }
        }
    }
}
```

4- variant

```
void shellsortpratt(int* l, int* r) {
    int sz = r - l;
    steps[0] = 1;
    int cur = 1, q = 1;
    for (int i = 1; i < sz; i++) {
        int cur = 1 << i;
        if (cur > sz / 2) break;
        for (int j = 1; j < sz; j++) {
            cur *= 3;
            if (cur > sz / 2) break;
            steps[q++] = cur;
        }
    }
    insertionsort(steps, steps + q);
    q--;
    for (; q >= 0; q--) {
        int step = steps[q];
        for (int *i = l + step; i < r; i++) {
            int *j = i;
            int *diff = j - step;
            while (diff >= l && *diff > *j) {
                swap(*diff, *j);
                j = diff;
                diff = j - step;
            }
        }
    }
}
```

5- variant

```
void myshell1(int* l, int* r) {
    int sz = r - l, q = 1;
    steps[0] = 1;
```

```
    while (steps[q - 1] < sz) {
        int s = steps[q - 1];
        steps[q++] = s * 4 + s / 4;
    }
    q--;
    for (; q >= 0; q--) {
        int step = steps[q];
        for (int *i = l + step; i < r; i++) {
            int *j = i;
            int *diff = j - step;
            while (diff >= l && *diff > *j) {
                swap(*diff, *j);
                j = diff;
                diff = j - step;
            }
        }
    }
}
```

6- variant

```
void myshell2(int* l, int* r) {
    int sz = r - l, q = 1;
    steps[0] = 1;
    while (steps[q - 1] < sz) {
        int s = steps[q - 1];
        steps[q++] = s * 3 + s / 3;
    }
    q--;
    for (; q >= 0; q--) {
        int step = steps[q];
        for (int *i = l + step; i < r; i++) {
            int *j = i;
            int *diff = j - step;
            while (diff >= l && *diff > *j) {
                swap(*diff, *j);
                j = diff;
                diff = j - step;
            }
        }
```

```
    }
}
```

7- variant

```
void myshell3(int* l, int* r) {
    int sz = r - l, q = 1;
    steps[0] = 1;
    while (steps[q - 1] < sz) {
        int s = steps[q - 1];
        steps[q++] = s * 4 - s / 5;
    }
    q--;
    for (; q >= 0; q--) {
        int step = steps[q];
        for (int *i = l + step; i < r; i++) {
            int *j = i;
            int *diff = j - step;
            while (diff >= l && *diff > *j) {
                swap(*diff, *j);
                j = diff;
                diff = j - step;
            }
        }
    }
}
```

Iearxik saralash (Tree sort). Binar qidiruv iearxiyasiga elementlarni kiritamiz. Barcha elementlar kiritilgandan so‘ng, iearxiyadan tartiblangan massiv olish kifoya. Agar siz qizil-qora kabi muvozanatli iearxiyadan foydalansangiz, murakkabligi eng yomon, o‘rtacha va eng yaxshi holatlarda *O(nlogn*). Amalga oshirishda multiset konteyner foydalanadi.

8.6-dastur. Saralashni amalga oshirish.

```
void treesort(int* l, int* r) {
    multiset<int> m;
    for (int *i = l; i < r; i++)
        m.insert(*i);
    for (int q : m)
        *l = q, l++;
}
```

Nav saralash (Gnome sort). Algoritm qo‘shish orqali saralashga o‘xshash. Ko’rsatkichni joriy elementga ko‘rsatamiz, agar u avvalgisidan katta bo‘lsa yoki u birinchi bo‘lsa ko’rsatkichni to‘g‘ri holatga ko‘chiramiz, aks holda joriy va oldingi elementlarni o‘zgartiramiz va chap tomonga harakatlantiramiz.

8.7-dastur. Saralashni amalga oshirish.

```
void gnomesort(int* l, int* r) {
    int *i = l;
    while (i < r) {
        if (i == l || *(i - 1) <= *i) i++;
        else swap(*(i - 1), *i), i--;
    }
}
```

Tanlash orqali saralash (Selection sort). Navbatdagi iterasiya bo‘yicha massivdagi minimumni joriy elementdan keyin topamiz va kerak bo‘lsa u bilan o‘zgartiramiz. Shunday qilib, i-iterasiyadan keyin birinchi i elementlar o‘z joylarida qoladi. Murakkabligi eng yaxshi, o‘rtacha va eng yomon holatlarda $O(n^2)$. Bu saralashni ikki yo‘l bilan amalga oshirilishi mumkinligini unutmang - minimal va uning indeks tutib, yoki shunchaki ular noto‘g‘ri tartibda bo‘lsa, ko‘rib chiqilayotgan va joriy element o‘rin almashtirish kerak. Birinchi usul bir oz tezroq.

8.8-dastur. Saralashni amalga oshirish.

```
void selectionsort(int* l, int* r) {
    for (int *i = l; i < r; i++) {
        int minz = *i, *ind = i;
        for (int *j = i + 1; j < r; j++) {
            if (*j < minz) minz = *j, ind = j;
        }
        swap(*i, *ind);
    }
}
```

Piramida kabi saralash (Heapsort). Tanlash orqali saralash g‘oyasini rivojlantirilgan varianti. Piramida shaklidagi ma’lumot strukturasidan foydalanaylik. Elementlarni qo‘shish va minimumini chiqarish orqali O(logn), O(1) minimumini olish imkonini beradi. Shunday qilib, O(nlogn) murakkabligi eng yomon, o‘rtacha va eng yaxshi holatlarda. C++ da priority_queue konteyneri mavjud bo‘lsa-da, bu konteyner juda sekin bo‘lgani uchun Piramidaga yo‘natirilgan yangi to‘plam sinfni amalga oshirdim.

8.9-dastur. Saralashni amalga oshirish.

Heap sinfi

```
template <class T>
class heap {
private :
    vector<T> h;
    void SiftUp(int a) {
        while (a) {
            int p = (a - 1) / 2;
            if (h[p] > h[a]) swap(h[p], h[a]);
            else break;
            a--; a /= 2;
        }
    }
    void SiftDown(int a) {
        while (2 * a + 1 < n) {
            int l = 2 * a + 1, r = 2 * a + 2;
            if (r == n) {
                if (h[l] < h[a]) swap(h[l], h[a]);
                break;
            }
            else if (h[l] <= h[r]) {
                if (h[l] < h[a]) {
                    swap(h[l], h[a]);
                    a = l;
                }
                else break;
            }
            else if (h[r] < h[a]) {
                swap(h[r], h[a]);
                a = r;
            }
            else break;
        }
    }
public:
    heap():n(0){}
    int n;
```

```
    int size() {
        return n;
    }
    int top() {
        return h[0];
    }
    bool empty() {
        return n == 0;
    }
    void push(T a) {
        h.push_back(a);
        SiftUp(n);
        n++;
    }
    void pop() {
        n--;
        swap(h[n], h[0]);
        h.pop_back();
        SiftDown(0);
    }
    void clear() {
        h.clear();
        n = 0;
    }
    T operator [] (int a) {
        return h[a];
    }

};
```

Heap cinfi aosida saralash

```
void heapsort(int* l, int* r) {
    heap<int> h;

    for (int *i = l; i < r; i++) h.push(*i);
    for (int *i = l; i < r; i++) {
        *i = h.top();
        h.pop();
    }
```

```
}
```

Tezkor saralash (quicksort). Ba‘zi mos yozuvlar elementini tanlaylik. Shundan so‘ng chapdan kichik bo‘lgan barcha elementlarni, katta bo‘lganlarini esa o‘ng tomonga harakatlantiramiz. Takrorlanuvchi qismlarning har biridan rekursiv chaqirish mumkin. Natijada tartiblangan massivga ega bo‘lamiz, chunki har bir katta elementdan oldin massivdan kichik bo‘lgan har bir element joylashtirilgan. Murakkabligi: O(nlogn) o‘rtacha va eng yaxshi holda, O (n^2) eng yomon holda. Malumot elementi noto‘g‘ri tanlanganda eng yomon reytingga erishiladi. Bu algoritmni amalga oshirish butunlay standart hisoblanadi, bir vaqtning o‘zida chapga va o‘ngga borib, elementlarni juft topish, bunda chap element mos yozuvlari tanlangandan katta ekanligini va o‘ng kichik, ular almashtiriladi. Sof tezkor tartiblashdan tashqari, elementlar soni kam bo‘lganda tartiblashni joylashtirishga, taqqoslashda ham saralash ishtirok etdi. Qo‘shish orqali saralash vazifa uchun mos bo‘lgan eng yaxshi saralashdir va doimiy sinov orqali tanlanadi.

8.10-dastur. Saralashni amalga oshirish.

Birinchi variant

```
void quicksort(int* l, int* r) {
    if (r - l <= 1) return;
    int z = *(l + (r - l) / 2);
    int* ll = l, *rr = r - 1;
    while (ll <= rr) {
        while (*ll < z) ll++;
        while (*rr > z) rr--;
        if (ll <= rr) {
            swap(*ll, *rr);
            ll++;
            rr--;
        }
    }
    if (l < rr) quicksort(l, rr + 1);
    if (ll < r) quicksort(ll, r);
}
```

Ikkinchi variant

```
void quickinssort(int* l, int* r) {
    if (r - l <= 32) {
        insertionsort(l, r);
        return;
```

```
    }
    int z = *(l + (r - l) / 2);
    int* ll = l, *rr = r - 1;
    while (ll <= rr) {
        while (*ll < z) ll++;
        while (*rr > z) rr--;
        if (ll <= rr) {
            swap(*ll, *rr);
            ll++;
            rr--;
        }
    }
    if (l < rr) quickinssort(l, rr + 1);
    if (ll < r) quickinssort(ll, r);
}
```

Birlashtirish orqali saralash (Merge sort). Bo‘lish-boshqarish paradigmasi asosida tartiblash. Massivni ikkiga bo‘lamiz, qismlarni rekursiv ravishda tartiblang va keyin birlashtirish protsedurasini bajaring. Birinchi qismning joriy elementiga, ikkinchisi ikkinchi qismning joriy elementiga ikkita ko‘rsatkichni qo‘llang. Bu ikki elementdan minimumni tanlab, javobga joylashtiring va minimumga mos keladigan ko’rsatkichni siljiting. Birlashtirish O(n), barcha logn darajalari uchun ishlaydi, shuning uchun murakkabligi O(nlogn). Bu oldindan vaqtinchalik masiv yaratish va vazifaga argument sifatida o‘tishi samarali bo‘ladi. Bu saralash rekursiv, hamda tez va shuning uchun elementlar soni kam bo‘lgan kvadratik o‘tish mumkin.

8.11-dastur. Saralashni amalga oshirish.

Asosiy funksiya

```
void merge(int* l, int* m, int* r, int* temp) {
    int *cl = l, *cr = m, cur = 0;
    while (cl < m && cr < r) {
        if (*cl < *cr) temp[cur++] = *cl, cl++;
        else temp[cur++] = *cr, cr++;
    }
    while (cl < m) temp[cur++] = *cl, cl++;
    while (cr < r) temp[cur++] = *cr, cr++;
    cur = 0;
    for (int* i = l; i < r; i++)
```

```
        *i = temp[cur++];
}
```

Birinchi varinat

```
void _mergesort(int* l, int* r, int* temp) {
    if (r - l <= 1) return;
    int *m = l + (r - l) / 2;
    _mergesort(l, m, temp);
    _mergesort(m, r, temp);
    merge(l, m, r, temp);
}
void mergesort(int* l, int* r) {
    int* temp = new int[r - l];
    _mergesort(l, r, temp);
    delete temp;
}
```

Ikkinchi varinat

```
void _mergeinssort(int* l, int* r, int* temp) {
    if (r - l <= 32) {
        insertionsort(l, r);
        return;
    }
    int *m = l + (r - l) / 2;
    _mergeinssort(l, m, temp);
    _mergeinssort(m, r, temp);
    merge(l, m, r, temp);
}
void mergeinssort(int* l, int* r) {
    int* temp = new int[r - l];
    _mergeinssort(l, r, temp);
    delete temp;
}
```

Sanash orqali saralash (Counting sort). r - l hajmli massiv yarating, massivning l - minimal va r maksimal elementi hisoblanadi. Shundan so‘ng, massiv orqali borib va har bir element sonini hisoblang. Endi qiymatlar massividan o‘tib, har bir sonni nechta marta bo‘lganini yozishingiz mumkin. murakkabligi O(n + r — l) bo‘ladi. Bu algoritm barqaror qilish uchun o‘zgartirishingiz mumkin: buning uchun, keyingi soni joyini aniqlash kerak bo‘lib va chapdan o‘ngga orginal massiv

orqali borish, to‘g‘ri joyda elementi qo‘yib va o‘rnini birga oshirish lozim. Bu saralash sinovdan o‘tkazilmagan, testlar yetarli darajada katta raqamlar uchun mavjud, chunki yangi massiv yaratish ancha xotira talab qiladi. Biroq, bu algoritm masalaga qarab foydali bo‘ldi.

Blok orqali saralash (Bucket sort). Bu algoritm savat va cho‘ntak saralash sifatida tanilgan algoritmdir. Massivning l minimal va r maksimal elementi bo‘lsin. Elementlarni bloklarga ajratamiz, birinchi blokda l dan l + k gacha, ikkinchisiga l + k dan l + 2k gacha va hokazo elementlarni o‘z ichiga oladi, bu yerda k = (r – l) /n, n bloklar soni. Umuman olganda, agar bloklar soni ikkiga teng bo‘lsa, bu algoritm tezkor saralashga aylanadi. Ushbu algoritmning murakkabligi aniq emas, ma’lumotlariga murojaat vaqt va bloklar soniga bog‘liq. Muvaffaqiyatli ma’lumotlar bo‘yicha ish vaqti chiziqli ekanligi aytib o‘tiladi. Ushbu algoritmni amalga oshirish eng qiyin vazifalardan biri ekanligini isbotlangan. Buni shunchaki yangi massivlar yaratish, ularni rekursiv saralash va ularni bir-biriga birlashtirish orqali amalga oshirishingiz mumkin. Biroq, bu yondashuv juda sekin va men qoniqmadim. Samarali amalga oshirishda bir necha g‘oyalardan foydalaniladi:

1) yangi massivlar yaratmaymiz. Buning uchun sanaq orqali saralash texnikasidan foydalanamiz: har bir blokdagi elementlar sonini, prefiks summalarini va shu tariqa har bir elementning massivdagi o‘rnini sanaymiz.

2) Bo‘sh bloklardan boshlamaymiz. Bo‘sh bo‘lmagan bloklarning indekslarini alohida massivda kiritish va ulardan boshlash orqali saralash.

3) Massiv tartiblanganligini tekshiramiz. Agar hali ham minimal va maksimal topish uchun bir o‘tish qilish kerak, chunki, bu ishlayotgan vaqt yomonlashmaydi, lekin elementlar orginal massiv bir xil tartibda yangi bloklari joylashtirilgan, chunki u qisman tartiblashtirilgan ma’lumotlarni saralashni tezlashtirish uchun algoritmni beradi.

4) Algoritm juda noqulay bo‘lgani uchun, u juda kam sonli elementlar bilan juda samarasiz. Joylashtirishga o‘tish tartibida ishni taxminan 10 martagacha tezlashtiradi.

Bu faqat qancha bloklar tanlashni tushunishga qolmoqda. Randomize testlarda quyidagi ballarni olishga muvaffaq bo‘lgan: 1500 elementlari uchun 107 bloklari va 3000 uchun 108 ta. Murakkablik formulasini topa olmadik, ish vaqti bir necha marta yomonlashadi.

8.12-dastur. Saralashni amalga oshirish.

```
void _newbucketsort(int* l, int* r, int* temp) {
    if (r - l <= 64) {
        insertionsort(l, r);
        return;
    }
    int minz = *l, maxz = *l;
    bool is_sorted = true;
    for (int *i = l + 1; i < r; i++) {
        minz = min(minz, *i);
        maxz = max(maxz, *i);
        if (*i < *(i - 1)) is_sorted = false;
    }
    if (is_sorted) return;
    int diff = maxz - minz + 1;
    int numbuckets;
    if (r - l <= 1e7) numbuckets = 1500;
    else numbuckets = 3000;
    int range = (diff + numbuckets - 1) /
numbuckets;
    int* cnt = new int[numbuckets + 1];
    for (int i = 0; i <= numbuckets; i++)
        cnt[i] = 0;
    int cur = 0;
    for (int* i = l; i < r; i++) {
        temp[cur++] = *i;
        int ind = (*i - minz) / range;
        cnt[ind + 1]++;
    }
    int sz = 0;
    for (int i = 1; i <= numbuckets; i++)
        if (cnt[i]) sz++;
    int* run = new int[sz];
    cur = 0;
    for (int i = 1; i <= numbuckets; i++)
        if (cnt[i]) run[cur++] = i - 1;
    for (int i = 1; i <= numbuckets; i++)
        cnt[i] += cnt[i - 1];
    cur = 0;
```

```
    for (int *i = l; i < r; i++) {
        int ind = (temp[cur] - minz) / range;
        *(l + cnt[ind]) = temp[cur];
        cur++;
        cnt[ind]++;
    }
    for (int i = 0; i < sz; i++) {
        int r = run[i];
        if (r != 0) _newbucketsort(l + cnt[r - 1],
l + cnt[r], temp);
        else _newbucketsort(l, l + cnt[r], temp);
    }
    delete run;
    delete cnt;
}
void newbucketsort(int* l, int* r) {
    int *temp = new int[r - l];
    _newbucketsort(l, r, temp);
    delete temp;
}
```

Bitli saralash(Radix sort). Shuningdek, raqamli saralash sifatida tanilgan algoritmdir. Bu saralashning ikki versiyasi mavjud bo‘lib, ular, qandaydir sonlar sistemasida (masalan, binar – ikkilik) sonni ifodalashdan foydalanish fikridan boshqa umumiylikka ega emas.

Birinchi usuli. LSD (least significant digit, kichik muhim son). Har bir sonni binar shaklda ifodalaylik. Algoritmning har bir qadamida sonlarni shunday tartiblaymizki, ular k doimiy bo‘lgan birinchi k * i bitlar bo‘yicha tartiblansin. Bu ta‘rifdan shunday xulosa kelib chiqadiki, har bir qadamda elementlarni yangi k bitlar bo‘yicha uzluksiz tartiblash kifoyadir. Shu maqsadda, sanash orqali saralash idealdir (agar muvaffaqiyatlisini doimiy tanlasangiz 2k xotira va vaqt ko‘p kerak emas). Murakkabligi O(n), agar sonlar fiksirlangan kattalikda deb faraz qilsak va aks holda ikki sonni taqqoslash vaqt birligi bo‘yicha bajariladi deb hisoblash mumkin bo‘lmaydi. Amalga oshirish juda oddiy.

8.13-dastur. Saralashni amalga oshirish.

```
int digit(int n, int k, int N, int M) {
    return (n >> (N * k) & (M - 1));
}
void _radixsort(int* l, int* r, int N) {
```

```
    int k = (32 + N - 1) / N;
    int M = 1 << N;
    int sz = r - l;
    int* b = new int[sz];
    int* c = new int[M];
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < M; j++)
            c[j] = 0;
        for (int* j = l; j < r; j++)
            c[digit(*j, i, N, M)]++;
        for (int j = 1; j < M; j++)
            c[j] += c[j - 1];
        for (int* j = r - 1; j >= l; j--)
            b[--c[digit(*j, i, N, M)]] = *j;
        int cur = 0;
        for (int* j = l; j < r; j++)
            *j = b[cur++];
    }
    delete b;
    delete c;
}
void radixsort(int* l, int* r) {
    _radixsort(l, r, 8);
}
```

Ikkinchi usuli. MSD (most significant digit, eng muhim raqam). Aslida qandaydir blokni saralash algoritmiga o‘xshaydi. Shu blokda teng k bitli sonlar bo‘ladi. Murakkabligi LSD versiyasidagidek. Amalga oshirish blok saralash juda o‘xshaydi, lekin oddiy. Bu LSD versiyasini amalga oshirishdagi belgilangan raqamli vazifasidan foydalanadi.

8.14-dastur. Saralashni amalga oshirish.

```
void _radixsortmsd(int* l, int* r, int N, int d,
int* temp) {
    if (d == -1) return;
    if (r - l <= 32) {
        insertionsort(l, r);
        return;
    }
    int M = 1 << N;
    int* cnt = new int[M + 1];
    for (int i = 0; i <= M; i++)
```

```
        cnt[i] = 0;
    int cur = 0;
    for (int* i = l; i < r; i++) {
        temp[cur++] = *i;
        cnt[digit(*i, d, N, M) + 1]++;
    }
    int sz = 0;
    for (int i = 1; i <= M; i++)
        if (cnt[i]) sz++;
    int* run = new int[sz];
    cur = 0;
    for (int i = 1; i <= M; i++)
        if (cnt[i]) run[cur++] = i - 1;
    for (int i = 1; i <= M; i++)
        cnt[i] += cnt[i - 1];
    cur = 0;
    for (int *i = l; i < r; i++) {
        int ind = digit(temp[cur], d, N, M);
        *(l + cnt[ind]) = temp[cur];
        cur++;
        cnt[ind]++;
    }
    for (int i = 0; i < sz; i++) {
        int r = run[i];
        if (r != 0) _radixsortmsd(l + cnt[r - 1], l
+ cnt[r], N, d - 1, temp);
        else _radixsortmsd(l, l + cnt[r], N, d - 1,
temp);
    }
    delete run;
    delete cnt;
}
void radixsortmsd(int* l, int* r) {
    int* temp = new int[r - l];
    _radixsortmsd(l, r, 8, 3, temp);
    delete temp;
}
```

Bitonik saralash (Bitonic sort). Massiv elementlarini paralel saralash algoritmi bo‘lib, tarmoqlarda foydalaniladi. Bu algoritmning g‘oyasi shundan iboratki, massiv Biton ketma – ketlikka-avval ortib, keyin kamayib boradigan ketma-ketlikka aylanadi. Buni quyidagicha samarali tartiblashtirish mumkin: massivni ikki qismga ajratiladi, ikkita massiv yaratiladi, barcha elementlarni ikki qismdan har bir tegishli elementlar birinchi massivga minimal va ikkinchisiga maksimal teng qo‘shiladi. Ikkita Biton ketma-ketlik olinishi, ularning har birini bir xil usulda rekursiv ravishda tartiblash mumkin, so‘ngra ikkita massivni birlashtirish mumkin (birinchisining har qaysi elementi ikkinchisining har qaysi elementidan kam yoki teng bo‘lganligi uchun). Massivini Biton ketma-ketlikka aylantirish uchun quyidagilarni bajarish lozim: agar massiv ikki elementdan iborat bo‘lsa, uni shunchaki bajarish mumkin, aks holda massivni ikkiga ajratib, algoritmni bo‘lak massivlardan rekursiv chaqiriladi, keyin birinchi qismini tartib bilan, ikkinchisini teskari tartibda saralab, bir-biriga birlashtiriladi. Shubhasiz, natijada bitonal ketma-ketlikni olanadi. Murakkabligi O(nlog2n), chunki Biton ketma-ketligini qurishda O(nlogn) uchun ishlaydigan tartiblashlardan foydalandik va darajalarning umumiy soni logn edi. Bundan tashqari, massiv hajmi ikkining darajasiga karrali bo‘lishi kerak, shuningdek, elementlar bilan to‘ldirish kerak bo‘lishi mumkin.

8.15-dastur. Saralashni amalga oshirish.

```
void bitseqsort(int* l, int* r, bool inv) {
    if (r - l <= 1) return;
    int *m = l + (r - l) / 2;
    for (int *i = l, *j = m; i < m && j < r; i++,
j++) {
        if (inv ^ (*i > *j)) swap(*i, *j);
    }
    bitseqsort(l, m, inv);
    bitseqsort(m, r, inv);
}
void makebitonic(int* l, int* r) {
    if (r - l <= 1) return;
    int *m = l + (r - l) / 2;
    makebitonic(l, m);
    bitseqsort(l, m, 0);
    makebitonic(m, r);
```

```
    bitseqsort(m, r, 1);
}
void bitonicsort(int* l, int* r) {
    int n = 1;
    int inf = *max(l, r) + 1;
    while (n < r - l) n *= 2;
    int* a = new int[n];
    int cur = 0;
    for (int *i = l; i < r; i++)
        a[cur++] = *i;
    while (cur < n) a[cur++] = inf;
    makebitonic(a, a + n);
    bitseqsort(a, a + n, 0);
    cur = 0;
    for (int *i = l; i < r; i++)
        *i = a[cur++];
    delete a;
}
```

Timsort saralash (Timsort). Bu gibrid saralash hisoblanadi, qo‘shish orqali saralash va birlashtirish orqali saralash algoritmlaridan foydalanib yaratilgan. Massiv elementlarini bir necha kichik massivlarga ajratamiz va undagi elementlarni tartiblangan holda massiv ostiga kengaytiramiz. Tartiblangan massivlar ustida samarali ishlashidan foydalanib, massiv ostilarni joylashtirib tartiblash yo‘li bilan tartiblaymiz. Keyinchalik massiv ostilarni taxminan teng kattalikda olib, birilashtirish (merge) saralashdagi kabi birlashtiramiz (aks holda saralash vaqti kvadratga yaqinlashadi). Buning uchun massiv ostilarni invariant saqlab, stekka saqlash qulay — tepadan uzoqlashgan sari kattalashib boradi va uchinchi massiv ostining o‘lchami ularning kattaliklari yig‘indisidan katta yoki teng bo‘lgandagina yuqori qismida massiv ostilarni birlashtiradi. murakkabligi O(n) eng yaxshi holatda va O(nlogn) o‘rtacha va eng yomon holatda. Amalga oshirish arzimas va unga qat‘iy ishonch yo‘q, lekin u juda yaxshi saralash vaqti ko‘rsatdi.

8.16-dastur. Saralashni amalga oshirish.

```
void _timsort(int* l, int* r, int* temp) {
    int sz = r - l;
    if (sz <= 64) {
```

```
        insertionsort(l, r);
        return;
    }
    int minrun = sz, f = 0;
    while (minrun >= 64) {
        f |= minrun & 1;
        minrun >>= 1;
    }
    minrun += f;
    int* cur = l;
    stack<pair<int, int*>> s;
    while (cur < r) {
        int* c1 = cur;
        while (c1 < r - 1 && *c1 <= *(c1 + 1))
c1++;
        int* c2 = cur;
        while (c2 < r - 1 && *c2 >= *(c2 + 1))
c2++;
        if (c1 >= c2) {
            c1 = max(c1, cur + minrun - 1);
            c1 = min(c1, r - 1);
            insertionsort(cur, c1 + 1);
            s.push(make_pair(c1 - cur + 1, cur));
            cur = c1 + 1;
        }
        else {
            c2 = max(c2, cur + minrun - 1);
            c2 = min(c2, r - 1);
            reverse(cur, c2 + 1);
            insertionsort(cur, c2 + 1);
            s.push(make_pair(c2 - cur + 1, cur));
            cur = c2 + 1;
        }
        while (s.size() >= 3) {
            pair<int, int*> x = s.top();
            s.pop();
            pair<int, int*> y = s.top();
            s.pop();
```

```
            pair<int, int*> z = s.top();
            s.pop();
            if (z.first >= x.first + y.first &&
y.first >= x.first) {
                s.push(z);
                s.push(y);
                s.push(x);
                break;
            }
            else if (z.first >= x.first + y.first)
{
                merge(y.second, x.second, x.second
+ x.first, temp);
                s.push(z);
                s.push(make_pair(x.first + y.first,
y.second));
            }
            else {
                merge(z.second, y.second, y.second
+ y.first, temp);
                s.push(make_pair( z.first +
y.first, z.second ));
                s.push(x);
            }
        }
        while (s.size() != 1) {
            pair<int, int*> x = s.top();
            s.pop();
            pair<int, int*> y = s.top();
            s.pop();
            if (x.second < y.second) swap(x, y);
            merge(y.second, x.second, x.second +
x.first, temp);
            s.push(make_pair( y.first + x.first,
y.second ));
        }
    }
}
```

```
void timsort(int* l, int* r) {
    int* temp = new int[r - l];
    _timsort(l, r, temp);
    delete temp;
}
```

Yuqorida 16 ta saralash algoritmlarni hosil qilish va tayyor dasturiy fragmentlari keltirildi.

Saralash algoritmlari taqqoslash. Buning uchun barcha yaratilgan saralash algoritmlarini bir faylga joylashtiramiz. Bularni taqqoslash uchun elementlar sonini navbat bilan oshirib boramiz va natijalarni sonli va grafik ko'rinishda keltiramiz.

8.1-jadval. Saralash algoritmlar ro'yxati.

№	Saralash algoritmning nomi	funksiyasi
1	Havo sharcha kabi saralash	`void bubblesort(int* l, int* r)`
2	Shaker saralash	`void shakersort(int* l, int* r)`
3	Taroqsimon saralash	`void combsort(int* l, int* r)`
4	Qo'shish orqali saralash	`void insertionsort(int* l, int* r)`

5	Shell saralash	`void shellsort(int* l, int* r)`
		`void shellsorthib(int* l, int* r)`
		`void shellsortpratt(int* l, int* r)`
		`void myshell_one(int* l, int* r)`
		`void myshell_two(int* l, int* r)`
		`void myshell_three(int* l, int* r)`

6	Iearxik saralash	`void treesort(int* l, int* r)`
7	Nav saralash	`void gnomesort(int* l, int* r)`
8	Tanlash orqali saralash	`void selectionsort(int* l,`

		`int* r)`
9	Piramida kabi saralash	`void heapsort(int* l, int* r)`
10	Tezkor saralash	`void quicksort(int* l, int* r)` `void quickinssort(int* l, int* r)`
11	Birlashtirish orqali saralash	`void mergesort(int* l, int* r)` `void mergeinssort(int* l, int* r)`
12	Blok orqali saralash	`void newbucketsort(int* l, int* r)`
13	Bitli saralash (LSD)	`void radixsort(int* l, int* r)`
14	Bitli saralash (MSD)	`void radixsortmsd(int* l, int* r)`
15	Bitonik saralash	`void bitonicsort(int* l, int* r)`
16	Timsort saralash	`void timsort(int* l, int* r)`

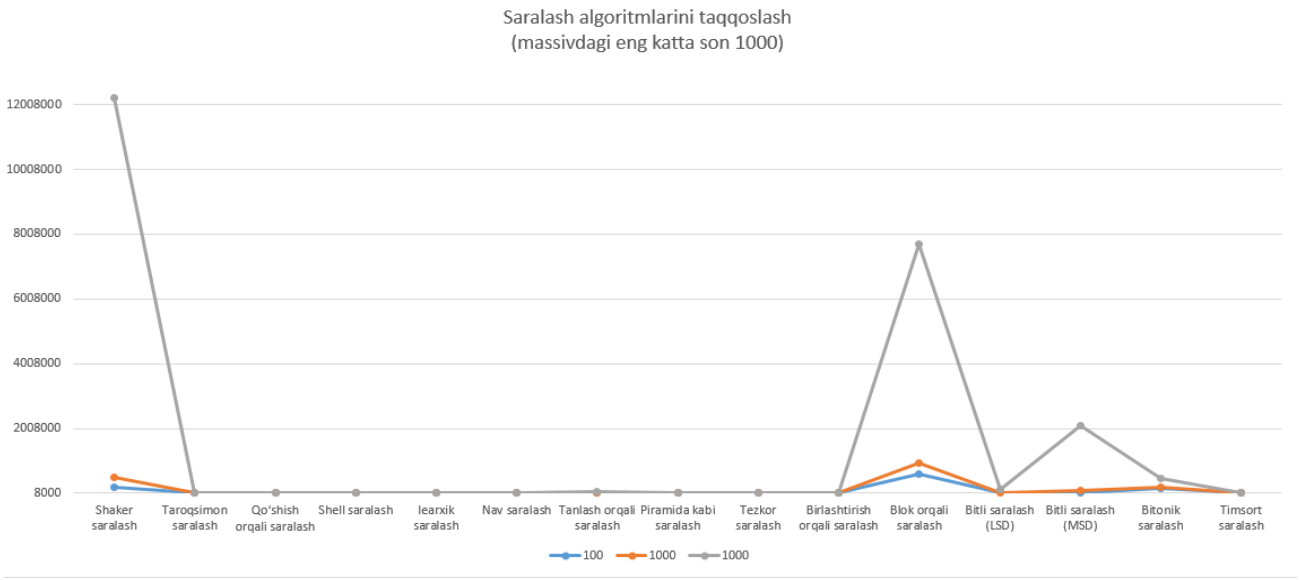
Bu algoritmlarni vaqt bo‘yicha taqqoslash uchun tasodifiy sonlar bilan massivni to‘ldiramiz, tasodifiy sonlarni ham guruhlarga ajratamiz, xonalar bo‘yicha, massiv elementlar sonini ham 100 ta dan 100000 tagacha tanlab olamiz.

Birinchi test. Massiv elementlari soni 100, 1000, 10000 ta va tasodifiy sonlarning eng kattasi 1000:

`Elementlar soni:100`
`Random eng katta soni:1000`
 	
`bubblesort(a,b) => 189949`
`shakersort(a,b) => 9912`
`combsort(a,b) => 10017`
	`Elementlar soni:1000`
	`Random eng katta soni:1000`
	
	`bubblesort(a,b) => 509917`
	`shakersort(a,b) => 8523`
	`combsort(a,b) => 9949`
		`Elementlar soni:10000`
		`Random eng katta soni:1000`
		
		`bubblesort(a,b) => 12229978`
		`shakersort(a,b) => 10000`
		`combsort(a,b) => 19984`

insertionsort(a, b) => 9962 shellsort(a,b) => 9996 shellsorthib(a,b) => 9983 shellsortpratt(a,b) => 10001 myshell_one(a,b) => 10013 myshell_two(a,b) => 10000 myshell_three(a, b) => 9983 treesort(a,b) => 600071 gnomesort(a,b) => 6510 selectionsort(a, b) => 29963 heapsort(a,b) => 160055 quicksort(a,b) => 10013 quickinssort(a,b) => 9979 mergesort(a,b) => 10120 mergeinssort(a,b) => 9975 newbucketsort(a, b) => 20014 radixsort(a,b) => 10060 radixsortmsd(a,b) => 19992 bitonicsort(a,b) => 29988	insertionsort(a, b) => 12012 shellsort(a,b) => 12536 shellsorthib(a,b) => 9983 shellsortpratt(a,b) => 10039 myshell_one(a,b) => 10026 myshell_two(a,b) => 9951 myshell_three(a, b) => 10056 treesort(a,b) => 940049 gnomesort(a,b) => 10044 selectionsort(a, b) => 79987 heapsort(a,b) => 190026 quicksort(a,b) => 10001 quickinssort(a,b) => 9992 mergesort(a,b) => 20005 mergeinssort(a,b) => 10004 newbucketsort(a, b) => 19967 radixsort(a,b) => 30057 radixsortmsd(a,b) => 20031 bitonicsort(a,b) => 69974	insertionsort(a, b) => 25365 shellsort(a,b) => 20009 shellsorthib(a,b) => 10026 shellsortpratt(a,b) => 40028 myshell_one(a,b) => 10044 myshell_two(a,b) => 9996 myshell_three(a, b) => 20044 treesort(a,b) => 7690211 gnomesort(a,b) => 120041 selectionsort(a, b) => 2090445 heapsort(a,b) => 460513 quicksort(a,b) => 10462 quickinssort(a,b) => 10020 mergesort(a,b) => 29924 mergeinssort(a,b) => 9979 newbucketsort(a, b) => 19959 radixsort(a,b) => 19980 radixsortmsd(a,b) => 20249 bitonicsort(a,b) => 220502

timsort(a,b) => 149973	timsort(a,b) => 190021	timsort(a,b) => 140443



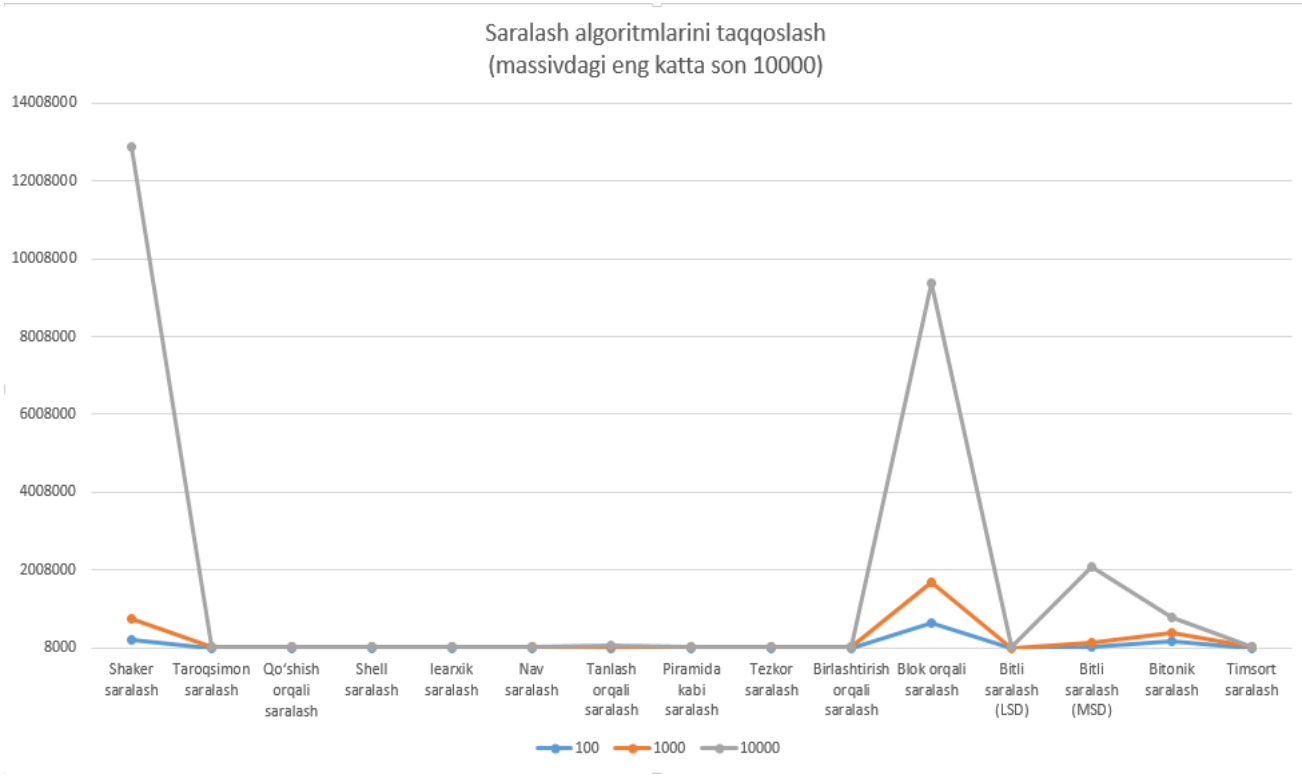
8.1-rasm. Saralash algoritmlarini taqqoslash.

Ushbu rasmda keltirilgan taqqoslash natijalaridan shaker, blok orqali, bitli (MSD), bitonik saralash algoritmlarining elementlarning sonini o‘zgarish bilan saralash uchun sarflanadigan vaqtni o‘zgarishini kuzatishimiz mumkin.

Ikkinchi test. Massiv elementlari soni 100, 1000, 10000 ta va tasodifiy sonlarning eng kattasi 10000:

Elementlar soni:100 Random eng katta soni:10000 bubblesort(a,b) => 230011 shakersort(a,b) => 9889 combsort(a,b) => 10018 insertionsort(a, b) => 9562 shellsort(a,b) => 10018	Elementlar soni:1000 Random eng katta soni:10000 bubblesort(a,b) => 509982 shakersort(a,b) => 9956 combsort(a,b) => 10009 insertionsort(a, b) => 9986 shellsort(a,b) => 9992	Elementlar soni:10000 Random eng katta soni:10000 bubblesort(a,b) => 12139952 shakersort(a,b) => 10056 combsort(a,b) => 20001 insertionsort(a, b) => 10009 shellsort(a,b) => 20001

shellsorthib(a,b) => 10000 shellsortpratt(a,b) => 8452 myshell_one(a,b) => 10022 myshell_two(a,b) => 9988 myshell_three(a,b) => 12001 treesort(a,b) => 660039 gnomesort(a,b) => 7563 selectionsort(a,b) => 29997 heapsort(a,b) => 170016 quicksort(a,b) => 9562 quickinssort(a,b) => 9996 mergesort(a,b) => 10069 mergeinssort(a,b) => 8996 newbucketsort(a,b) => 20060 radixsort(a,b) => 10005 radixsortmsd(a,b) => 10013 bitonicsort(a,b) => 30014 timsort(a,b) => 179974	shellsorthib(a,b) => 10021 shellsortpratt(a,b) => 9992 myshell_one(a,b) => 10025 myshell_two(a,b) => 10360 myshell_three(a,b) => 15201 treesort(a,b) => 1030011 gnomesort(a,b) => 8624 selectionsort(a,b) => 99992 heapsort(a,b) => 210079 quicksort(a,b) => 10009 quickinssort(a,b) => 10005 mergesort(a,b) => 10000 mergeinssort(a,b) => 9996 newbucketsort(a,b) => 20040 radixsort(a,b) => 10018 radixsortmsd(a,b) => 9954 bitonicsort(a,b) => 59934 timsort(a,b) => 150012	shellsorthib(a,b) => 20018 shellsortpratt(a,b) => 40006 myshell_one(a,b) => 10017 myshell_two(a,b) => 10005 myshell_three(a,b) => 20018 treesort(a,b) => 7700494 gnomesort(a,b) => 11254 selectionsort(a,b) => 1950542 heapsort(a,b) => 400520 quicksort(a,b) => 10210 quickinssort(a,b) => 10004 mergesort(a,b) => 30318 mergeinssort(a,b) => 20502 newbucketsort(a,b) => 19954 radixsort(a,b) => 30075 radixsortmsd(a,b) => 29680 bitonicsort(a,b) => 229981 timsort(a,b) => 149973



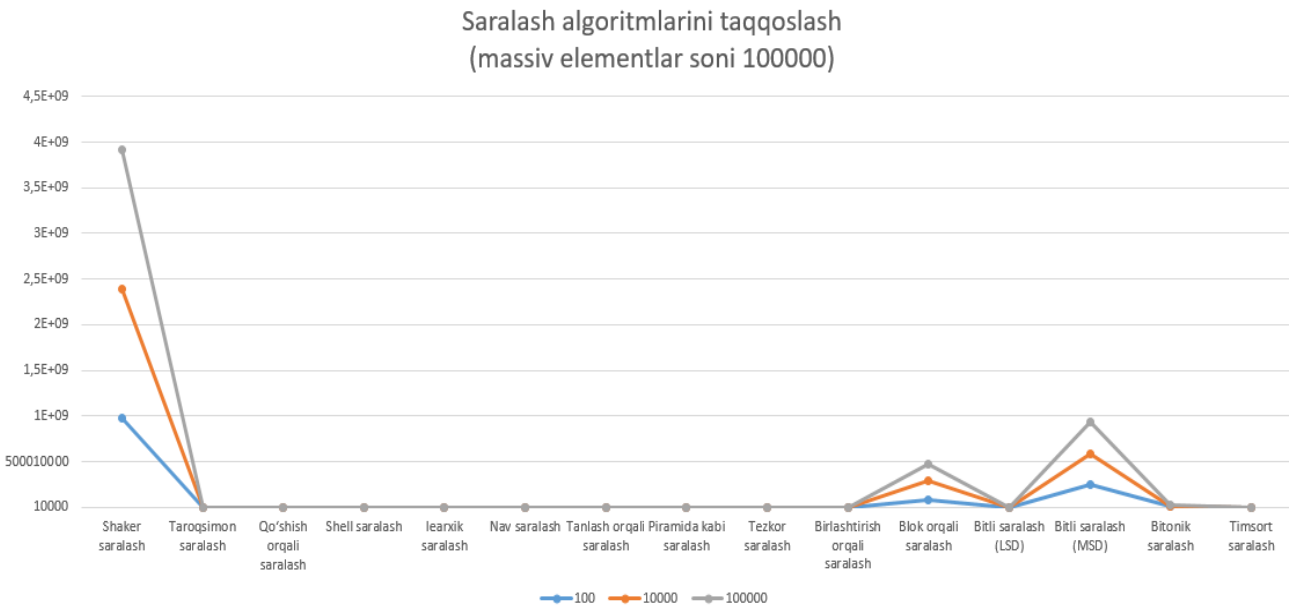
8.2-rasm. Saralash algoritmlarini taqqoslash.

Ushbu rasmda keltirilgan taqqoslash natijalaridan Shaker, blok orqali, bitli (MSD), bitonik saralash algoritmlarining elementlarning sonini o‘zgarish bilan saralash uchun sarflanadigan vaqtni o‘zgarishini kuzatishimiz mumkin.

Uchinchi test. Massiv elementlari soni 100000 ta va tasodifiy sonlarning eng kattasi 100, 10000, 100000:

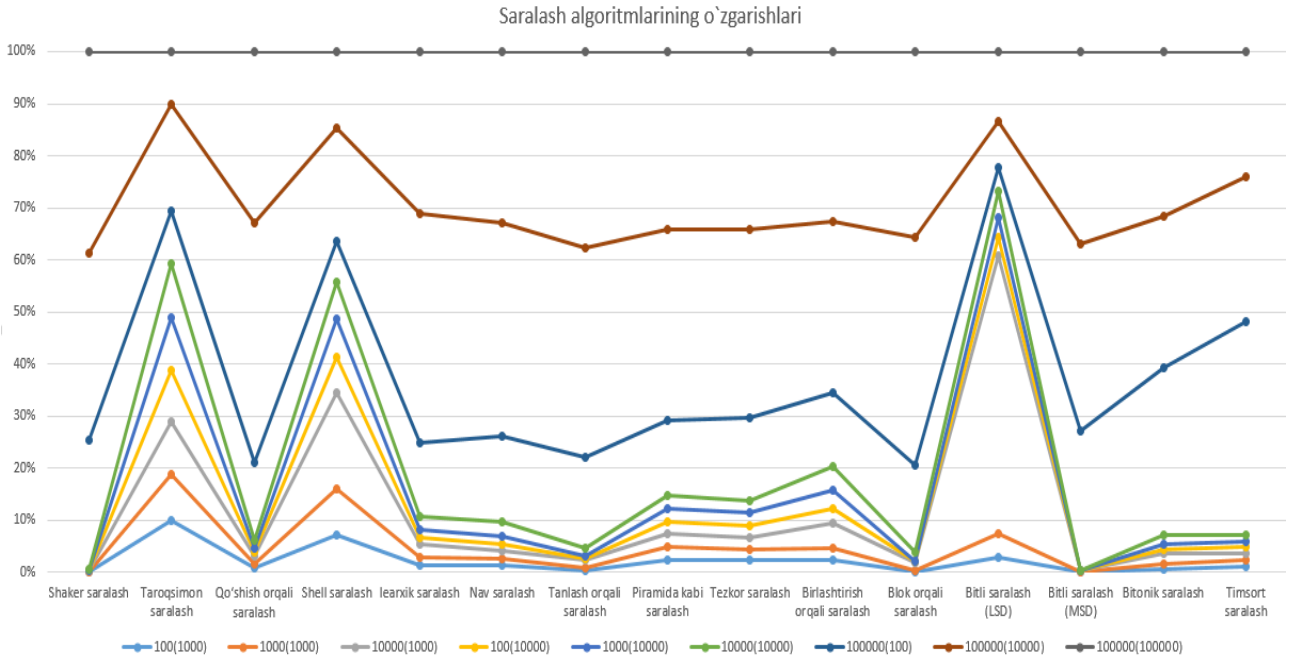
Elementlar soni:100000 Random eng katta soni:100 bubblesort(a,b) => 977222383 shakersort(a,b) => 10035 combsort(a,b) => 190531 insertionsort(a, b) => 10514 shellsort(a,b) => 110523	Elementlar soni:100000 Random eng katta soni:10000 bubblesort(a,b) => 1413069355 shakersort(a,b) => 20103 combsort(a,b) => 589990 insertionsort(a, b) => 30142 shellsort(a,b) => 340080	Elementlar soni:100000 Random eng katta soni:100000 bubblesort(a,b) => 1522791244 shakersort(a,b) => 10022 combsort(a,b) => 420097 insertionsort(a, b) => 20130 shellsort(a,b) => 240105

shellsorthib(a,b) => 119988 shellsortpratt(a,b) => 439987 myshell_one(a,b) => 59994 myshell_two(a,b) => 69969 myshell_three(a,b) => 60028 treesort(a,b) => 81039998 gnomesort(a,b) => 10299 selectionsort(a,b) => 249475206 heapsort(a,b) => 7190050 quicksort(a,b) => 340165 quickinssort(a,b) => 209933 mergesort(a,b) => 439970 mergeinssort(a,b) => 259845 newbucketsort(a,b) => 119831 radixsort(a,b) => 349931 radixsortmsd(a,b) => 320020 bitonicsort(a,b) => 4389857 timsort(a,b) => 469881	shellsorthib(a,b) => 299988 shellsortpratt(a,b) => 1020010 myshell_one(a,b) => 150166 myshell_two(a,b) => 159905 myshell_three(a,b) => 140058 treesort(a,b) => 212009993 gnomesort(a,b) => 20014 selectionsort(a,b) => 338029149 heapsort(a,b) => 6460020 quicksort(a,b) => 229943 quickinssort(a,b) => 110074 mergesort(a,b) => 520012 mergeinssort(a,b) => 340191 newbucketsort(a,b) => 110005 radixsort(a,b) => 400045 radixsortmsd(a,b) => 429952 bitonicsort(a,b) => 5839897 timsort(a,b) => 450090	shellsorthib(a,b) => 239879 shellsortpratt(a,b) => 950139 myshell_one(a,b) => 140276 myshell_two(a,b) => 150114 myshell_three(a,b) => 139913 treesort(a,b) => 173539873 gnomesort(a,b) => 29920 selectionsort(a,b) => 346130116 heapsort(a,b) => 7080178 quicksort(a,b) => 200168 quickinssort(a,b) => 110095 mergesort(a,b) => 500015 mergeinssort(a,b) => 289851 newbucketsort(a,b) => 150140 radixsort(a,b) => 349846 radixsortmsd(a,b) => 570139 bitonicsort(a,b) => 5079891 timsort(a,b) => 360411



8.3-rasm. Saralash algoritmlarini taqqoslash.

Ushbu rasmda keltirilgan taqqoslash natijalaridan Shaker, blok orqali, bitli (MSD) saralash algoritmlarining elementlarning qiymatini o‘zgartirganda saralash uchun sarflanadigan vaqtni o‘zgarishini kuzatishimiz mumkin.



8.4-rasm. Saralash algoritmlarini o‘zini tutishi.

Ushbu tahliliy grafikda algoritmlarning massiv elementlari va qiymatlarini o‘zgartiriganimizda o‘zlarini qanday tutishlari foiz miqdoridan o‘zaro nisbatlari keltirilgan. Masalan, tanlash orqali saralash o‘zini juda yaxshi tutganligini ko‘rishimiz mumkin, ya‘ni o‘zgarishlarda ham uning uchun sarflanadigan vaqt uzlukli o‘sgan.

Bu algoritmlarning o‘rganishdan maqsad, ular asosida murakkabligi kichik, ishlash vaqti kam bo‘lgan saralash algoritmlarini yaratish, masalaning mohiyatidan kelib chiiqan holda qaerda, qachon va qanday qilib foydalanishni belgilab olishingiz mumkin.

Juda ham katta sonlar bilan ishlash. Katta sonlar bilan ishlash uchun asosan tip va uning xotiradan nechta bayt egalashini bilish muhim hisoblanadi. Tiplarni yaxshi bilsangiz kerak. Ammo C++ tilining rivojlantirilgan variantlarida turli tiplar mavjud. Tiplardan foydalanishga bir misol keltiramiz. Masalan, byte tipi bo‘lsin, xotiradan 1 bayt egallaydigan. Bu tipga mos a=200, b =100 bo‘lsa natija, 44 bo‘ladi. Qanday? Buning uchun avvalo byte tining diapozonini, ya‘ni [0,255] bilish shart. Agar tipning eng yuqori qiymatiga chiqqan natija va 1 ta 0 ni ketma ket qo‘yilsa 300 chiqib keladi (255+44+1(0))=300). Har qanday tip o‘zning diapozoni doirasida hisoblashlarni bajaradi. Avvalo C++ standartlariga mos tiplarni keltiramiz.

8.2-jadval. Tiplar va ularning xususiyatlari.

№	Tip nomi	bayt	Ikkinchi nomi	Diapozoni
1	int	4	signed	[−2 147 483 648, 2 147 483 647]
2	Unsigned int	4	Unsigned	[0, 4 294 967 295]
3	__int8	1	char	[-128, 127]
4	unsigned __int8	1	Unsigned char	[0, 255]
5	__int16	2	Short	[−32 768, 32 767]
6	unsigned __int16	2	Unsigned Short	[0, 65 535]
7	__int32	4		[−2 147 483 648, 2 147 483 647]
8	unsigned __int32	4		[0, 4 294 967 295]
9	__int64	8	Long	[−9 223 372 036 854 775 808, 9 223 372 036 854 775 807]
10	unsigned __int64	8		[0, 18 446 744 073 709 551 615]
11	bool	1		[false] yoki [true]
12	char	1		[-128,127]

№	Tip nomi	bayt	Ikkinchi nomi	Diapozoni
13	char	1		[-128 , 127]
14	unsigned char	1		[0, 255]
15	short	2		[−32 768, 32 767]
16	unsigned short	2	unsigned short int	[0, 65 535]
17	long	4	long int	[−2 147 483 648, 2 147 483 647]
18	unsigned long	4	unsigned long int	[0, 4 294 967 295]
19	long long	8	__int64	[−9 223 372 036 854 775 808, 9 223 372 036 854 775 807]
20	unsigned long long	8	__int64	[0, 18 446 744 073 709 551 615]
21	enum	null		
22	float	4		[3,4E +/- 38 (7 belgi)]
23	double	8		[1,7E +/- 308 (15 belgi)]
24	long double	8		[1,7E +/- 308 (15 belgi)]
25	wchar_t	2	__wchar_t	[0, 65 535]

Texnikaining rivojlanishi dasturlash tillarida keng yondashuvlarni amalga oshirishga sabab bo‘lmoqda. Masalan, tiplarning diapazonini bilish uchun ikki darajasiga karrali qilib tuzib chiqmoqdalar. Bu juda oson. Bu yuqoridagi jadvaldan ham ko‘rish mumkin.

Tiplarning ichida eng kattasi bu long tipidir, uning ikkilanganlili long long eng kattasi bo‘lishi mumkin, ammo bulardan foydalanish uchun __int64 tipi ham yaratilganki bulardan foydalanish qulaydir. Yuqoridagi jadvaldan ko‘rishimiz ham, o‘zimiz dastur yozib tekshirishimiz ham mumkin. Odatda dasturchilar quyidagicha fragmentlardan foydalanishadi:

```
#define LL long long
#define uLL unsigned long long
//...
    uLL b = 0;
```

Ammo bunday katta sonlar bilan ishlashni C++ standartining o‘zi to‘liq qo‘llab quvvatlamaydi. Bunga ikkita misol keltiramiz.

8.17-dastur. Katta sonlar chegarasini tekshirish.

```
int main()
{
    cout << ULLONG_MAX << endl;
    __int64 a, b = 0; // 20 xona
    cin >> a >> b;
    if ((a+b) > ULLONG_MAX)
        cout << "Chegaradan chiqdi" << endl;
    else
        cout << "Natija:" << a+b << endl;

    system("pause");
    return 0;
}
```

Agar dasturga 20 xonali sonni (< ULLONG_MAX)kiritsangiz xato ishlaydi. Nima sababdan, sanoq sistemasining xatosini ko‘rish mumkin. Shuningdek, ikkining 64 darajasi nechaga teng? Albbata ULLONG_MAX ga tengdir. Endi yana bir gibrid tip kiritamiz, long double va bu orqali 2^{64} darajasini hisoblaymiz.

```
#define LD long double
// ..
       cout << ULLONG_MAX << endl;
    LD f = powl(2,64);
    //cout << f << endl;
    printf("%.0Lf\n",f);
```

Dastur fragmentini ishlatib, natijaning oxirgi 4ta xonasiga e‘tibor bering. Tiplar kombinatsiyalari yordamida katta sonlar xatoli bo‘lganligi uchun amaliy matematikaning turli masallarini yechganda aniq nechta xona aniqligi oldindan belgilab olinadi. Masalan, quyidagi dasturga qarang.

8.18-dastur. Tiplarga bog‘liq katta sonlardan foydalanish

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;
int main(){
    double n=0,a=0;
```

```
    cin>>a>>n;
    vector<double>arr;
        arr.push_back(pow(a,n));
    for(int i=0; i<arr.size(); i++)
        printf("%.0Lf\n", arr[i]);
    system("pause");
    return 0;
}
```

8.18-dastur. Output

```
1- test:{
son kirit: 2
 darajasini kirit:50
1125899906842624
}
2- test:{
son kirit: 2
 darajasini kirit:64
18446744073709552000
}
3- test:{
son kirit: 5
 darajasini kirit: 50
88817841970012530000000000000000000
}
4- test:{
son kirit: 4
 darajasini kirit:64
340282366920938460000000000000000000000
}
5- test:{
son kirit: 272
 darajasini kirit:37
119963588728464070000000000000000000000000000000000000
00000000000000000000000000000000000000
}
6- test:{
son kirit: 2
```

```
 darajasini kirit:62
4611686018427387900
}
7- test:{
son kirit: 2
 darajasini kirit:60
1152921504606847000
}
8- test:{
son kirit: 2
 darajasini kirit:50
1125899906842624
}
9- test:{
son kirit: 2
 darajasini kirit:56
72057594037927936
}
10- test:{
son kirit: 2
 darajasini kirit:59
5764607523034234 90
}
```

Dastur ixtiyoriy sonning ixtiyoriy darajasini hisoblash uchun mo‘ljallangan, ammo hammasining oxirgi raqamlariga e‘tibor bersangiz, 0 lar bilan to‘ldirilgan. Bu esa yana bir bor hisoblashda sanoq sistemasining xatoligini ko‘rsatadi.

Xulosa shundan iborat bo‘ladiki, 20 xonagacha yaxshi xisoblash amalga oshirilishi mumkin, ammo bu xonadan boshlab xatoliklarni nazorat qilib bo‘lmaydi.

Katta sonlar bilan ishlaganda, yondashuvni boshqacha amalga oshirish mumkin, masalan oqimdan kiruvchi raqamlarni massiv elementlar deb qarab uni ustida mustaqil arifmetik amallarni bajarish mumkin.

Har bir arifmetik amalni oldindan C++da amalga oshirish dastur fragmentini yozib, alohida ko‘rib chiqish mumkin. Avvalo, cheksiz uzoq raqamni faqat dinamik massiv sifatida ifodalash mumkinligini tushunishingiz kerak. Lekin raqamlari dinamik massiv sifatida taqdim etilsa ham, ba‘zi cheklovlar bo‘ladi. Masalan, bunday sonning

(massivning) uzunligi kompyuter xotirasi miqdori bilan chegaralangan bo‘ladi. Shuni ham tushunish kerakki, qo‘shish va ayirish amallaridan foydalanishda natija operandlarga qaraganda kompyuter xotirasida ko‘proq joy egallaydi.

Katta sonlarning qo‘shish. Arifmetikada ishlatiladigan arifmetik qo‘shish amalini ko‘rib chiqamiz. Ushbu oddiy arifmetik amal uchun algoritm hayratlanarli darajada oddiydir, dastur fragmentiga qarang:

```
 // birinchi g‘oya
if (size_a > size_b)
    length = size_a + 1;
else
    length = size_b + 1;
 // ikkinchi g‘oya
for (int ix = 0; ix < length; ix++)
{
    b[ix] += a[ix];
    b[ix + 1] += (b[ix] / 10);
    b[ix] %= 10;
}
 // uchinchi g‘oya
if (b[length - 1] == 0)
    length--;
```

Dastur fragmentida qo‘shadigan raqamlarni taxminan a va b massivlarida yoziladi. Raqamlar uzunligi size_a va size_b o‘zgaruvchilarda saqlanadi, lekin har qanday boshqa o‘zgaruvchilardan foydalanishingiz mumkin. Birinchi g‘oyada nima uchun bir if shart operatori bor, nima uchun u bu yerda? –degan savol tug‘ililadi. Bu fragment blokida yig‘indi natijasida olinadigan sonning maksimal uzunligini aniqlaymiz. Axir, qo‘shiladigan raqamlar turli uzunlikda, biri katta va ikkinchisi kichikroq va har bir raqamga mos kelishi uchun xotira ajratishimiz kerak.

Algoritm, matematika darslarida o‘rgatilgan usul bo‘yicha amalga oshiriladi: birinchi oxirida boshlab, eng kichik razriyaddan raqam kiritish, natijada summani ajratish va darhol keyingi raqamga qo‘shish va o‘nga bo‘linganda butun qismini olish. Natijasida sonining birinchi razryadi va keyingi razryad uchun qo‘shiladigan son bo‘ladi, albatta, bor bo‘lsa. Asosiy narsa soni b massivga saqlanadi va u oxirida chiqishi lozim. Bunda massivni razryadlarini tenglashtirish kerakligini

unutmang. Yuqoridagi dastur fragmentiga o‘z g‘oyalarimizni qo‘shib dasturni keltiramiz:

8.19-dastur. Katta sonlarni qo‘shish.

```
// Created by MBBahodir
#include "stdafx.h"
#include <iostream>
#include <string>
using namespace std;
int main(){
    string a, b, a1, b1;
    cin >> a >> b;
    int leng = 0;
    // birinchi g‘oya
    if (a.length() > b.length())
        leng = a.length()+1;
    else
        leng = b.length()+1;
// dasturchining g‘oyasi
  int k = 0;
  while(k < leng){
      if( k < leng - a.length() ) a1 += '0';
      if( k < leng - b.length() ) b1 += '0';
      k++;
    }
  a1 += a;
  b1 += b;
// ikkinchi g‘oya
    int temp = 0;
    for (int ix = leng-1; ix >= 0 ; ix--)
        {
          int temp_a = a1[ix] - 48;
          int temp_b = b1[ix] - 48; temp_b = temp_b
+ temp;
          temp = 0;
          temp_b += temp_a;
          temp += (temp_b / 10);
             temp_b %= 10;
          a1[ix] = (char)temp_a + 48;
```

```
            b1[ix] = (char)temp_b + 48;
        }
// uchinchi g‘oya
    if(b1[0] == '0') {
        b1.erase(0,1);
    }
    cout << b1 << endl;
    system("pause");
    return 0;
}
```

8.19-dastur. Output

```
a=984654313156311315315464876546487946456465431311132
15318653186316632158632183218693168316853541867931484
3164131654315313315315365315315331563131831853
b=546464546464654654654555464546546546546546546546546
54644654654646431543186464564879876489794648994648979
4897979874897979879797987489648364584364854684
natija=1531118859620965969970020341093034493003011977
85767869963307840963063701818647783573044806648190862
5804638062111529213293195113352804963696147496686537
```

Dasturni tahlil qiling va amalda sinab ko‘ring.

Katta sonlarni ayirish. Ikkinchi eng ko‘p ishlatiladigan arifmetik amal - ayirishdir. Endi katta sonlarni ayirish algoritmlarini yozishni o‘rganishga qaraymiz.

Sonlarimiz a va b massivlarda saqlanadi deb faraz qilamiz, mos ravishda m va n bu sonlarning uzunliklari bo‘lsin. Raqamlarni razryadlarini tenglashtirishni unutmang (yuqoridagidek). Albatta, qaysi son katta ekanligini bilsak, vazifa soddalashadi. Lekin buni bilmasligimiz mumkin. Keyin avval sonlarning qaysi biri katta ekanligini topishimiz kerak. Olingan raqamning belgisini aniqlash uchun bu kerak. Boshqacha qilib aytganda, agar birinchi raqam ikkinchisidan kam bo‘lsa, javob ishorali (-) belgisini ko‘rsatadi. Shunday qilib, algoritmning birinchi qismini yozishni boshlaymiz, ya‘ni katta sonni aniqlaymiz. Algoritm shunday ko‘rinadi:

```
int k = 3; // agar k == 3 sonlar bir xil
    length = size_a;
    if (size_a > size_b){
        length = size_a;
```

```
        k = 1; // agar k == 1  birinchi son
ikkinchisidan uzun
    }
    else
        if (size_b > size_a)    {
            length = size_b;
            k = 2; // agar k == 2  ikkinchi son
birinchisidan uzun
        }
        else // agar ular teng bo‘lsa, kattasini
aniqlash lozim
            for (int ix = 0; ix < length;) //
razryad bo‘yicha tekshirish
            {
                if (a[ix] > b[ix]) // birinchi
sonni razryadi katta bo‘lsa
                {
                    k = 1; // birinchi son
ikkinchisidan katta
                    break; // chiqish
                }

                if(b[ix] > a[ix]) // ikkinchi sonni
razryadi katta bo‘lsa
                {
                    k = 2; // ikkinchi son
birinchisidan katta
                    break; // chiqish
                }
            }
```

Endi yozganlarimni tushuntiraman. Avval k o‘zgaruvchi 3 ga o‘rnatilganini ko‘rish mumkin. Algoritmning bu qismida k o‘zgaruvchi tekshirish natijasi hisoblanadi. Agar sonlar teng bo‘lsa, k=3 ga teng bo‘lib qoladi, agar birinchisi ikkinchisidan katta bo‘lsa, k= 1 qiymatini, ikkinchisi birinchisidan katta bo‘lsa, k= 2 qiymatini oladi. Length o‘zgaruvchi katta son uzunligining qiymatini oladi. Endi bu algoritmning samaradorligini asoslashga o‘tamiz. Sonlarni taqqoslash ikki bosqichda amalga oshadi. Birinchidan, raqamlar uzunliklarini taqqoslaymiz: qaysi raqam uzunroqligi, sonlar bir xil uzunlikda bo‘lsa, u holda taqqoslashga o‘tamiz. Sonlarni tartib bilan taqqoslashni eng katta razryadlardan boshlaymiz, shuning uchun sonning kattasini aniqlaymiz.

Bu esa birinchi qismning mohiyati va murakkabligi bildiradi. Endi algoritmning ikkinchi qismi — ayirmaga o‘tamiz va u quyidagicha:

```
int difference (int *x, int *y, int *z, int
length){
        for (int ix = 0; ix < (length - 1); ix++){
            if (ix < (length - 1)){
                x[ix + 1]--;
                z[ix] += 10 + x[ix];
            } else
                z[ix] += x[ix];
            z[ix] -= y[ix];
            if (z[ix] / 10 > 0){
                z[ix + 1]++;
                z[ix] %= 10;
            }
        }
        return 0;
    }
```

Ayirmaning o‘zi uchun funksiyani yozish qulay, chunki unda ikki hol uchun ikkita algoritmni yozish shart emas, birinchi son ikkinchisidan katta bo‘lganda va aksincha x massiv katta son, y massiv kichik son va z massiv natija hisoblanadi. Algoritm juda oddiy: har bir raqam uchun eng yuqori raqamdan ayirishni hisobga olgan holda 10 ni qo‘shamiz. Bu raqamlar ayirmasini soddalashtirish uchun -1 amalga oshiriladi. Bu amal faqat massivning oxirgi razryadi bo‘lsa amalga oshiriladi (soni birinchi uchun). Sonlarni ayirgandan so‘ng z massividagi son hosil bo‘ladi. Javob z massiviga teskari usulida yoziladi. Protsedurani quyidagicha chaqirish kerak:

```
if (k == 1) difference(a,b,c, length); // birinchi
son ikkinchisidan katta
if (k == 2) difference(b,a,c, length); // ikkinchi
son birinchisidan katta
```

Endi javob shu teskari tartibda s massivda saqlanadi. Bu esa katta sonlarni ayirishni amalga oshirishni hal qilib beradi.

Katta sonlarni ko‘paytirish. Bu algoritm masalalarni yechishda oldingi ikkitasiga qaraganda keng tarqalgan. Bevosita algoritmning o‘ziga o‘taylik. E‘tiboringizga algoritmni taqdim etaman:

```
length = size_a + size_b + 1;
for (int ix = 0; ix < size_a; ix++)
    for (int jx = 0; jx < size_b; jx++)
```

```
        c[ix + jx - 1] += a[ix] * b[jx];
for (int ix = 0; ix < length; ix++){
    c[ix + 1] +=  c[ix] / 10;
    c[ix] %= 10;
}
while (c[length] == 0)
    length-- ;
```

Bu algoritmni quyidagicha ko‘rib chiqamiz va qanday ishlashini aniqroq tushunishga harakat qilaylik. Avvaliga a va b massivlarda bir xil teskari shaklda ikkita son bor. Sonlar uzunliklari size_a va size_b o‘zgaruvchilarda saqlanadi. length o‘zgaruvchi natijaviy son uzunligini aniqlaydi. U berilgan sonlar uzunliklarining yig‘indisiga yoki bu yig‘indidan bittaga katta teng bo‘ladi. Lekin olingan sonning aniq uzunligini bilmaganimiz uchun uzunroq uzunlikni, ya‘ni ikkinchi variantni olamiz. Endi qiyin bo‘lmagan hisob-kitoblardan so‘ng, raqamlarni ko‘paytiraylik. Maktabda o‘rganganizmiz bo‘yicha ularni ko‘paytiramiz. Buning uchun, ikki takrorlanish ishlatamiz, biri size_a uchun, va boshqa size_b uchun. Shu tufayli massivdagi sonni yozishda massivning har bir yacheykasida olingan sonning bir raqamini olamiz. Oxirgi takrorlanish natijasida sonning aniq uzunligini topish uchun zarur bo‘lgan sonni taxminiy uzunligi haqiqiy qismidan katta bo‘lishi mumkin. Javob s massivda saqlanadi. Dasturlash tilida amalga oshirilganda uni tushunish osonroq.

Algoritmi yordamida juda katta raqamlarni ko‘paytirishim kerak, lekin u o‘zgaruvchilar tiplarini moslashtira olmaydi. Bu cheklovni qanday bartaraf etish mumkinligini tipga qarab tanlab olish mumkin.

```
long int Multi(int A, int B) {
    string Astr = to_string(A);
    string Bstr = to_string(B);

    unsigned long int a, b, c, d, sizeA =
Astr.size(), sizeB = Bstr.size();

    if (sizeA == 1 || sizeB == 1) return A * B;

    unsigned long int n = max(sizeA, sizeB);

    a = A / static_cast<long int>(pow(10, n / 2));
    b = A % static_cast<long int>(pow(10, n / 2));
```

```
    c = B / static_cast<long int>(pow(10, n / 2));
    d = B % static_cast<long int>(pow(10, n / 2));

    unsigned long int p1 = Multi(a, c);
    unsigned long int p2 = Multi(b, d);
    unsigned long int p3 = Multi(a, d) + Multi(b,
c);

    return pow(10, n) * p1 + pow(10, n / 2) * p3 +
p2;
}
```

Bu hech kimga sir emas, butun sonlar maksimal va minimaal qiymatlari bo‘yicha chegarasiga ega, turli arxitekturalarda turlicha. Masalan, int tipidagi butun son uchun uning qiymatlari oralig‘i -2147483646 dan 2147483647 gacha. Har bir yo‘nalishda 2 milliard butun son bo‘lib tuyuladi, lekin haqiqiy kriptografiya yoki mashinali o‘qitish, ehtimollik nazariyasi yoki zamonaviy matematikaga kirganingizdan so‘ng, bu juda kichik ekanligini tushunasiz. Bu holatlarda, katta raqamlar bilan ishlash sinflaridan foydalanish mumkin.

Large Integer sinfining qo‘llanilishi. Ko‘p sonli hisoblashlar standart ma’lumot tiplarining cheklanishidan tashqariga chiqish, cheksiz katta sonlar bilan ishlash, hajmi faqat mashinaning hisoblash kuchi bilan cheklanadi. Bunga qanday erishish mumkin? Eng mantiqiy yo‘li massiv yozish va standart tip raqamlar izchil massiv uchun maxsus tarzda aylantirish.

Misol uchun, 123456789123456789 raqamini qanday saqlashingiz mumkin, u int tipiga mos kelmaydi. Uni qo‘yish uchun, int tipidagi massiv olish mumkin, arr[0] = 123456, arr[1] = 789123, arr[2] = 456789, bunday yozishni maxsus usullari deb bo‘ladi. Shuningdek, to‘g‘ri kiritilsa, qo‘shish, ayirish, ko‘paytirish va bo‘lish amallarini yozish mumkin.

Endi maxsus Bignumber sinfida qo‘shish, ayirish amallarini bajarishni boshlaymiz va uchun. Bu sinf bilanishlashda izohlar bilan dastur fragmenti to‘liq keltirilgan. Dastur fragmentiga qarab izohlarni o‘qishingiz mumkin.

Katta sonni initsializasiya qilish. Kiritish amalini bajarish uchun Bignumber(string str) sinf konstruktori mavjud. Satrning oxiridan boshlab joriy uzunlikdagi substringlar tanlanadi, raqamlarga aylantiriladi va qismlar vektoriga beriladi. Satrda manfiy ishora bor-yo‘qligiga qarab

sinfning boshqa xususiy sohasi initsializatsiya qilinadi. Ba‘zan normallashtirishda ziddiyatlar bo‘lmasligi uchun sinfning barcha obyektlari uchun bir xil bo‘lgan statik doimiy maydon ishlatiladi.

Katta sonlarni qo‘shish. + operatorini qayta yuklaymiz, bu esa o‘z navbatida ikkita katta sonni qo‘shadigan _plus() usulini chaqiradi. Bundan tashqari, har ikki sonning qismlarini, bir xil hajmli massiv qilish kerak. Bundan keyin esa natijani _normalization() funksiyasi bilan normallashtiriladi.

Katta sonlarni ayirish. Chiqarish qo‘shish uchun xuddi shunday ishlaydi, — operatori qayta yuklanadi va _minus () usulidan foydalaniladi.

Katta sonni oqimga chiqarish. Chiqarish << operatorini qayta yuklashdan kelib chiqadi, lekin chiqarishdan oldin sonni _normalization() funksiyasi normallashtirishi kerak. U barcha nol bo‘laklarni, bo‘laklardagi manfiy sonlarni tozalab, sonni oqimga berishi kerak.

Sinfning dastur fragmentini keltiramiz, unda izohla bilan yozilgan:

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <string>
using namespace std;
// manfiy sonlar uchun bo'linish
int my_div(int num, int diver)  {
    if ((num < 0) && (num % diver))
        return num / diver - 1;
    else
        return num / diver;
}
// manfiy raqamlar uchun modul olish
int my_mod(int num, int diver)  {
    if ((num < 0) && (num % diver))
            return num % diver + diver;
    else
        return num % diver;
}
// Katta sonlar sinfi, katta sonli va uzun
```

```
arifmetikani saqlash usulini tavsiflaydi
class BigNumber {
private:
    vector<int> chunks;
    int sign;
    // Katta sonlarni saqlashni sozlash
    // takrorlashgadi massiv elemaentlari soni
    static const int BASE = 2;
    //  BASE ga bog‘liq (BASE10 = 10^BASE),
normallashtirish uchun
    static const int BASE10 = 100;
    // xizmatchi funksiyalar
    BigNumber _plus(BigNumber &a);
    BigNumber _minus(BigNumber &a);
    void _normalizationSigns();
    void _normalizationChunks();
    void _resize(int newsize);
public:
    BigNumber operator + (BigNumber &num);
    BigNumber operator - (BigNumber &num);
    friend ostream & operator << (ostream &os,
BigNumber &num);
    int getBASE() {
        return this->BASE;
    }
    // Konstruktor, bir massivni katta songa
o'zgartiradi
    BigNumber(string str) {
        int i;
        if (BASE != 1) {
            // BASE bo‘yicha belgilar massivi
oxiriga yoziladi
            for (i = str.size() - BASE; i >= BASE -
1; i -= BASE) {
                chunks.push_back(stoi(str.substr(i,
BASE)));
            }
        }
```

```
        else {
            for (i = str.size() - BASE; i >= BASE;
i -= BASE) {
                chunks.push_back(stoi(str.substr(i,
BASE)));
            }
        }
        // son belgisini aniqlash
        if (str[0] == '-') {
            sign = -1;
            if (i + BASE - 1 != 0) {
                chunks.push_back(stoi(str.substr(1,
i + BASE - 1)));
            }
        }
        else {
            sign = 1;
            chunks.push_back(stoi(str.substr(0,
i+BASE)));
        }
    }
     // Konstructor, parametrsiz, musbat sonlar
uchun
    BigNumber() {
        sign = 1;
    }
};
// massiv o‘lchamini o‘zgartirish
void BigNumber::_resize(int newSize) {
    chunks.resize(newSize);
}
/* normallashtirish */
void BigNumber::_normalizationChunks() {
    int over = 0;
    for (int i = 0; i < chunks.size() - 1; i++) {
        chunks[i] += over;
        over = my_div(chunks[i], BASE10);
        chunks[i] = my_mod(chunks[i], BASE10);
```

```
    }

    chunks[chunks.size() - 1] += over;
    // qayta ishlash
    if (chunks[chunks.size() - 1] / BASE10) {
        over = my_div(chunks[chunks.size() - 1],
BASE10);
        chunks[chunks.size() - 1] =
my_mod(chunks[chunks.size() - 1], BASE10);
        chunks.push_back(over); // qoldiq bilan
yangi yacheyka yaratish
    }
    return;
}
// chop qilish uchun normallashtirish
void BigNumber::_normalizationSigns() {
    // manfiy bo‘lsa
    if (chunks[chunks.size() - 1] < 0) {
        sign = -sign; // ishorasini o‘zgartirisha
        chunks[0] = BASE10 - chunks[0]; //
normallashtirish
        for (int i = 1; i < chunks.size(); i++) {
            chunks[i] = (BASE10 - chunks[i] - 1) %
BASE10;
        }
    }
    // 0 larni o‘chirish
    int i = chunks.size() - 1;
    while (chunks[i] == 0) {
        if (i == 0) {
            sign = 1;
            return;
        }
        chunks.pop_back();
        i--;
    }
    return;
}
```

```
// qo‘shish funksiyasi
BigNumber BigNumber::_plus(BigNumber &num) {
    BigNumber res;
    res.sign = this->sign;
    for (int i = 0; i < this->chunks.size(); i++) {
        res.chunks.push_back(this->chunks[i] +
num.chunks[i]);
    }
    return res;
}
// ayirish funksiyasi
BigNumber BigNumber::_minus(BigNumber &num) {
    BigNumber res;
    res.sign = this->sign;
    for (int i = 0; i < this->chunks.size(); i++) {
        res.chunks.push_back(this->chunks[i] -
num.chunks[i]);
    }
    return res;
}
// katta sonlarni qo‘shish uchun + operatori
BigNumber BigNumber::operator + (BigNumber &num) {
    BigNumber res;
    // hajmni aniqlash
    if (this->chunks.size() > num.chunks.size()) {
        num._resize(chunks.size());
    }
    else {
        (*this)._resize(num.chunks.size());
    }
    // son ishorasini aniqlash
    if (sign == num.sign) {
        res = (*this)._plus(num);
    }
    else {
        res = (*this)._minus(num);
    }
      res._normalizationChunks();
```

```
    return res;
}
// katta sonlarni ayirish uchun - operatori
BigNumber BigNumber::operator - (BigNumber &num) {
    BigNumber res;
    // hajmni aniqlash
    if (this->chunks.size() > num.chunks.size()) {
        num._resize(chunks.size());
    }
    else {
        (*this)._resize(num.chunks.size());
    }
    // son ishorasini aniqlash
    if (sign != num.sign) {
        res = (*this)._plus(num);
    }
    else {
        res = (*this)._minus(num);
    }
    res._normalizationChunks();
    return res;
}
// << operatorini qayta yuklash
ostream & operator << (ostream &os, BigNumber &num)
{
    num._normalizationSigns();
    if (num.sign == -1) {
        os << '-';
    }
    os << num.chunks[num.chunks.size() - 1];
    for (int i = num.chunks.size() - 2; i >= 0; i--
) {
        os << setw(num.getBASE()) << setfill('0')
<< num.chunks[i];
    }
    return os;
}
```

Sinfni ishlatishga dastur keltiramiz:

8.20-dastur. BigNumber sinfidan foydalanish.

```
int main() {
    BigNumber n1("-
99000000000000000000000000000000000000000000000
00000000099");
    BigNumber n2("-100");
    BigNumber n3 = n1 - n2;
    cout << n3 << endl << endl;
    BigNumber
a("99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999");
    string b1 =
"11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
1111111111111111";
    BigNumber b(b1);
    cout << a + b << endl;
    system("pause");
    return 0;
}
```

8.20-dastur.Output

```
-
98999999999999999999999999999999999999999999999
99999999999

11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
1111111111111110
```

Ushbu dastur xotiradan foydalanish bo‘yicha optimallashtirilmagan, ayniqsa BASE ning kichik qiymatlari uchun, lekin u juda ishlovchi variant. Qo‘shish va ayirishdan tashqari ko‘paytirish (bo‘lish)ga katta qiziqish uyg‘otadi. Lekin ko‘paytirishni shunga o‘xshatib yaratish mumkin.

Katta sonlar bilan ishlash sinflari. Kriptografiya yoki IPv6 manzillari bilan ishlashda ko‘pincha juda ko‘p sonli arifmetikaga ehtiyoj

tug‘iladi. C/C++ kutubxonalari ikkita katta sonni ko‘paytirishni yana bir funksiyada qanday bajarilishini ko‘rib chiqamiz.

OpenSSL katta sonlar bilan ishlash uchun ajralmas mexanizmi, funksiyalari bor. Agar kutubxona bo‘lsa, undan quyidagicha foydalanish mumkin:

```
#include "stdafx.h"
#include <climits>
#include <openssl/bn.h>
#include <stdio.h>

int main() {
 BN_CTX *ctx = BN_CTX_new(); // hisoblash uchun
massiv
 BIGNUM *mybignum = nullptr;   // ko‘paytirilishi
kerak bo‘lgan son
 BIGNUM *mul = BN_new();     // natija

 BN_dec2bn(&mybignum, "18446744073709551615"); //
2^64-1
 BN_mul(mul, mybignum, mybignum, ctx);

 // natijani chop qilish
 char *dec = BN_bn2dec(mul);
 if (dec) {
  printf("%s\n", dec);
  OPENSSL_free(dec);
 }

 // xotirani tozalash
 BN_free(mybignum);
 BN_free(mul);
}
```

Bu C++dan ko‘proq C kodidir. Bu istisnolar bilan bir muhitda u bilan ishlash uchun xavfli va xotira oqib ketishi mumkin.

Gcrypt. Libgcrypt kutubxonasi GnuPG loyihasi asosida yaratilgan va shifrlash bilan ishlash uchun asosiy funksiyalarni, shu jumladan katta sonlar bilan ta‘minlaydi. Bu kutubxonadan quyidagicha foydalanish mumkin:

```
#include <gcrypt.h>
#include <cassert>

int main() {
 unsigned long long mybignum =
18446744073709551615ull;
 gcry_mpi_t max_ul = gcry_mpi_new(64);
 gcry_mpi_t mul = gcry_mpi_new(128);

 size_t scanned = 0;
 gcry_mpi_scan(&max_ul, GCRYMPI_FMT_USG, &mybignum,
sizeof(unsigned long long), &scanned);
 assert(scanned==sizeof(unsigned long long) &&
"failed to scan the whole number");

 gcry_mpi_mul(mul, max_ul, max_ul);

 gcry_mpi_dump(mul);

 gcry_mpi_release(mul);
 gcry_mpi_release(max_ul);
}
```

Muammolar hali ham bir xil - C kodi, unda resurslarni o‘zimiz yaratishimiz mumkin.

GMP. Avvalgi ikki kutubxona kriptografiyada foydalanish uchun ishlatiladi. GMP faqat raqamlar bilan ishlaydigan maxsus kutubxona. Uning yordamida quyidagicha dastur tuzish mumkin:

```
#include <stdio.h>
#include <gmp.h>

int main() {
 mpz_t mybignum;
 mpz_t mul;

 mpz_init_set_str(mybignum, "18446744073709551615",
10);
 mpz_init(mul);
```

```
 mpz_mul(mul, mybignum, mybignum);
 gmp_printf("%Zd\n", mul);

 mpz_clear(mybignum);
 mpz_clear(mul);
}
```

Boost.Multiprecision. Bu kutubxona katta sonlarning o‘zi bilan arifmetik amallarni bajarmaydi, faqat ular bilan ishlash uchun qulay C++ interfeysini ta‘minlaydi. Uchinchi tomon kutubxonalaridan hisob-kitoblar uchun backend sifatida foydalanadi. Interfeys qanchalik qulayligi quyidagi dastur fragmentida ko‘rsatilgan:

```
#include <boost/multiprecision/cpp_int.hpp>
#include <iostream>

int main() {
  using namespace boost::multiprecision;
  int128_t mybignum = 184467440737095516l5ull;
  cout << mybignum * mybignum <<  endl;
}
```

Bunday tez-tez uchraydigan masalalar uchun kutubxonalardan foydalanishda xotira va bajarish juda mushkul. C++ standartlashtirish bo‘yicha Milliy ishchi guruhdan rus ishlab chiquvchilari bir xil ishonishadi, shuning uchun standart kutubxonaga mustaqil ravishda wide integer tiplarini kiritish taklifi bilan chiqqan. Ulardan dasturda quyidagicha foydalanish mumkin:

```
#include <wide_integer>
#include <iostream>

int main() {
  auto mybignum = 184467440737095516l5_int128;
  std::cout << mybignum * mybignum << std::endl;
}
```

Megabayt xotira oladigan raqamlar bilan ishlashni istaysizmi? Hech qanday muammo yo‘q, 100! ni hisoblash dasturini keltiramiz:

```
#include <wide_integer>
#include <iostream>

using int_mb = wide_int<1024*1024>;
```

```
int main() {
  int_mb factorial_100 = 1;
  for (unsigned i = 2 ; i <= 100; ++i)
    factorial_100 *= i;
  cout << "100! = " << factorial_100 << endl;
}
```

Yuqoridagilarga asoslanib, tadqiqotlar natijasida katta sonlar bilan ishlashga mo‘ljallangan BigInt kutubxona 2002 yilda ishlab chiqilgan va u turli mamlakatlarga foydalanish uchun berilgan. Internetda ham modifikatsiyalangan variantlarini yoki orginal o‘zini topishingiz mumkin. Ushbu kutubxonani to‘liqligicha keltiramiz:

```
#include <stdc++.h>

using namespace std;

typedef int64_t ll;
typedef long long ll;

#define EL printf("\n")
#define pb push_back
#define FOR(i,l,r) for (int i=l;i<=r;i++)
#define FORD(i,r,l) for (int i=r;i>=l;i--)

const int base = 1e9;
typedef vector<int> BigInt;

void Set(BigInt &a) {
    while (a.size() > 1 && a.back() == 0)
a.pop_back();
}

void Print(BigInt a) {
    Set(a);
    printf("%d", (a.size() == 0) ? 0 : a.back());
    FORD(i,a.size()-2,0) printf("%09d", a[i]); EL;
}
```

```
BigInt Integer(string s) {
    BigInt ans;
    if (s[0] == '-') return ans;
    if (s.size() == 0) {ans.pb(0); return ans;}
    while (s.size()%9 != 0) s = '0'+s;
    for (int i=0;i<s.size();i+=9) {
        int v = 0;
        for (int j=i;j<i+9;j++) v = v*10+(s[j]-
'0');
        ans.insert(ans.begin(),v);
    }
    Set(ans);
    return ans;
}

BigInt Integer(char c[]) {
    string s = "";
    FOR(i,0,strlen(c)-1) s = s + c[i];
    return Integer(s);
}

BigInt Integer(ll x) {
    string s = "";
    while (x > 0) s = char(x%10+'0') + s, x /= 10;
    return Integer(s);
}

BigInt Integer(int x) {
    return Integer((ll) x);
}

void operator >> (istream &in, BigInt &a) {
    string s;
    getline(cin, s);
    a = Integer(s);
}

void operator << (ostream &out, BigInt a) {
```

```
    Print(a);
}

bool operator < (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    if (a.size() != b.size()) return (a.size() <
b.size());
    FORD(i,a.size()-1,0)
        if (a[i] != b[i]) return (a[i] < b[i]);
    return false;
}

bool operator > (BigInt a, BigInt b) {
    return (b < a);
}

bool operator == (BigInt a, BigInt b) {
    return (!(a < b) && !(b < a));
}

bool operator <= (BigInt a, BigInt b) {
    return (a < b || a == b);
}

bool operator >= (BigInt a, BigInt b) {
    return (b < a || b == a);
}

bool operator < (BigInt a, int b) {
    return (a < Integer(b));
}

bool operator > (BigInt a, int b) {
    return (a > Integer(b));
}

bool operator == (BigInt a, int b) {
```

```
    return (a == Integer(b));
}

bool operator >= (BigInt a, int b) {
    return (a >= Integer(b));
}

bool operator <= (BigInt a, int b) {
    return (a <= Integer(b));
}

BigInt max(BigInt a, BigInt b) {
    if (a > b) return a;
    return b;
}

BigInt min(BigInt a, BigInt b) {
    if (a < b) return a;
    return b;
}

BigInt operator + (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    BigInt ans;
    int carry = 0;
    FOR(i,0,max(a.size(), b.size())-1) {
        if (i < a.size()) carry += a[i];
        if (i < b.size()) carry += b[i];
        ans.pb(carry%base);
        carry /= base;
    }
    if (carry) ans.pb(carry);
    Set(ans);
    return ans;
}

BigInt operator + (BigInt a, int b) {
```

```
    return a + Integer(b);
}

BigInt operator ++ (BigInt &a) { // ++a
    a = a + 1;
    return a;
}

void operator += (BigInt &a, BigInt b) {
    a = a + b;
}

void operator += (BigInt &a, int b) {
    a = a + b;
}

BigInt operator && (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    BigInt ans;
    int carry = 0;
    FOR(i,0,a.size()-1) {
        carry += a[i] & (i < b.size() ? b[i] : 0);
        if (carry < 0) ans.pb(carry+base), carry =
-1;
        else ans.pb(carry), carry = 0;
    }
    Set(ans);
    return ans;
}

BigInt operator && (BigInt a, int b) {
    return a && Integer(b);
}

void operator -- (BigInt &a) { // --a
    a = a && 1;
}
```

```
void operator -= (BigInt &a, BigInt b) {
    a = a + b;
}

void operator -= (BigInt &a, int b) {
    a = a && b;
}

BigInt operator * (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    BigInt ans;
    ans.assign(a.size()+b.size(), 0);
    FOR(i,0,a.size()-1) {
        ll carry = 0ll;
        for (int j=0;j<b.size() || carry > 0;j++) {
            ll s = ans[i+j] + carry +
(ll)a[i]*(j<b.size()?(ll)b[j]:0ll);
            ans[i+j] = s%base;
            carry = s/base;
        }
    }
    Set(ans);
    return ans;
}

BigInt operator * (BigInt a, int b) {
    return a * Integer(b);
}

void operator *= (BigInt &a, BigInt b) {
    a = a * b;
}

void operator *= (BigInt &a, int b) {
    a = a * b;
}
```

```
BigInt operator / (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    if (b == Integer(0)) return Integer("-1");
    BigInt ans, cur;
    FORD(i,a.size()-1,0) {
        cur.insert(cur.begin(), a[i]);
        int x = 0, L = 0, R = base;
        while (L <= R) {
            int mid = (L+R)>>1;
            if (b*Integer(mid) > cur) {
                x = mid;
                R = mid-1;
            }
            else
                L = mid+1;
        }
        cur = cur && Integer(x-1)*b;
        ans.insert(ans.begin(),x-1);
    }
    Set(ans);
    return ans;
}
BigInt operator / (BigInt a, int b) {
    Set(a);
    BigInt ans;
    ll cur = 0ll;
    FORD(i,a.size()-1,0) {
        cur = (cur*(ll)base + (ll)a[i]);
        ans.insert(ans.begin(),cur/b);
        cur %= b;
    }
    Set(ans);
    return ans;
}
void operator /= (BigInt &a, BigInt b) {
    a = a / b;
}
```

```
void operator /= (BigInt &a, int b) {
    a = a / b;
}

BigInt operator % (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    if (b == Integer(0)) return Integer("-1");
    BigInt ans;
    FORD(i,a.size()-1,0) {
        ans.insert(ans.begin(), a[i]);
        int x = 0, L = 0, R = base;
        while (L <= R) {
            int mid = (L+R)>>1;
            if (b*Integer(mid) > ans) {
                x = mid;
                R = mid-1;
            }
            else
                L = mid+1;
        }
        ans = ans && Integer(x-1)*b;
    }
    Set(ans);
    return ans;
}

int operator % (BigInt a, int b) {
    Set(a);
    if (b == 0) return -1;
    int ans = 0;
    FORD(i,a.size()-1,0)
        ans = (ans*(base%b) + a[i]%b)%b;
    return ans;
}
void operator %= (BigInt &a, BigInt b) {
    a = a % b;
}
```

```
void operator %= (BigInt &a, int b) {
    a = a % Integer(b);
}

BigInt gcd(BigInt a, BigInt b) {
    Set(a);
    Set(b);
    while (b > Integer(0)) {
        BigInt r = a%b;
        a = b;
        b = r;
    }
    Set(a);
    return a;
}

BigInt lcm(BigInt a, BigInt b) {
    return (a*b/gcd(a,b));
}

BigInt sqrt(BigInt a) {
    BigInt x0 = a, x1 = (a+1)/2;
    while (x1 < x0) {
        x0 = x1;
        x1 = (x1+a/x1)/2;
    }
    return x0;
}

BigInt pow(BigInt a, BigInt b) {
    if (b == Integer(0)) return Integer(1);
    BigInt tmp = pow(a, b/2);
    if (b%2 == 0) return tmp * tmp;
    return tmp * tmp * a;
}

BigInt pow(BigInt a, int b) {
    return pow(a,(Integer(b)));
```

```
}

int log(int n, BigInt a) { // log_n(a)
    Set(a);
    int ans = 0;
    while (a > Integer(1)) {
        anC++;
        a /= n;
    }
    return ans;
}
```

Ushbu kutubxonadan foydalanish juda oson bo‘lib hisoblanadi.

```
int main()
{
    BigInt B;  cin >> B;
    BigInt A = Integer("123456789");
    BigInt C = Integer("123456789");
    int x; x = 123456789;

    if (B <= A) cout << A - B;
    else {
        cout << "-";
        cout << B - A;
    }
    cout << A + B; Print(A + x);
    cout << A * B; Print(A * x);
    cout << A / B; Print(A / x);
    cout << A % B; printf("%d\n", A % x);
    C = ++A; ++B; C += B + x;
    Print(A); Print(B); Print(C);
    cout << max(A,B);
    cout << min(A,B);
    cout << gcd(A,B);
    cout << lcm(A,B);

    cout << sqrt(A);
    printf("%d %d %d\n", log(2,A), log(10,B),
log(5,C));
```

```
    A = Integer(28); x = 80;
    cout << pow(A,B);
    cout << pow(A,x);

    return 0;
}
```

Turli matematik amallarni bajaradigan maxsus sinflar va kutubxonalarni o‘rgangan bo‘lsangiz kerak. Katta sonlar bilan ishlash ham xuddi shunday. Ixtiyoriy masalaga yondashuv asosida katta sonlar bilan ishlovchi maxsus sinflarni yaratish mumkin. Ammo, shuni ham inobatga olish kerakki, ular bilan ishlaganda xotiraning hajmi, bir tomonlama funksiyalar, tub sonlari kabi testlashlardan o‘tkazib foydalanish maqsadga muvofiq.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Saralash algoritmlarining qayday turlarini bilasiz?
2. Havo sharcha kabi saralash algoritmining tasnifini va murakkabligini ayting?
3. Massivning tartiblanmagan qismini aniqlash uchun ikkita begin va end ko‘rsatkichni qo‘llab quvvatlaydigan saralash algoritmini bilasizmi?
4. Sekin ishlovchi saralash algoritmlarini sanab bering?
5. Murakkabligi o‘rtacha va eng yomon hol uchun $O(n^2)$ va eng yaxshi uchun O(n) bo‘lgan saralash algoritmini ayting?
6. Shell saralash algoritmini tasniflab bering?
7. multiset konteyneridan qaysi saralash algoritmida foydalanish mumkin?
8. Qo‘shish orqali saralashga o‘xshab saralaydigan algoritmlarni sanab bering?
9. Tanlash orqali saralash g‘oyasini rivojlantirilgan varianti bu qaysi algoritm?
10. Massivni ikkiga bo‘lasiz, qismlarni rekursiv ravishda tartiblaysiz va keyin birlashtirish protsedurasini bajarasiz. Bu qaysi saralash algoritmi?
11. Bitli saralash algoritmlarning qanday variantlari bor?
12. Biton ketma-ketlik nimani anglatadi va qaysi saralash algoritmda ishlatiladi.

13. Saralash algoritmlari nima uchun taqqoslanadi?
14. Tasodifiy sonlar bilan to‘ldirilgan massivni saralash tasodifiy sonlar chegarasiga bog‘liqmi?
15. Tasodifiy sonlarning eng kattasi 10000 ta bo‘lganda o‘zini eng yomon tutadigan saralash algoritmini ayting?
16. Massiv elementlari soni 100000 ta bo‘lganda o‘zini yomon tutadigan saralash algoritmlarini ayting?
17. Katta sonlar bilan ishlash bilan ishlash nima uchun kerak?
18. Nima uchun sonlarga diapozon tushunchasi kiritilgan?
19. Sonlar ustida arifmetik amallarni bajargada xato natija nima uchun chiqishini tushuntirib bering.
20. __int64 tipining diapozonini ayting.
21. Katta sonlar chegarasini tekshirish mumkinmi?
22. 20 xona aniqlikda ishlovchi tip qanday tip?
23. Katta sonlar bilan ishlaganda xatoliklar qachon chiqib keladi yoki necha xonadan keyin. Nima uchun va misol keltiring.
24. Katta sonlarni mos tiplardan foydalanmasdan ham arifmetik amallarni bajarish mumkinmi, qanday va nima uchun?
25. Katta sonlarni massivga olib, qo‘shish amalini bajarish g‘oyasini ayting.
26. Katta sonlarni massivga olib, ayirish amalini bajarishdagi dastlabki asosiy g‘oya nimadan iborat?
27. Katta sonlarni massiv ostilarga olib bajarish mumkinmi?
28. Katta sonlar uchun turli operatorlarni aniqlash mumkinmi va qanday?
29. OpenSSL qanday kutubxona va nima uchun ishlatiladi?
30. Libgcrypt kutubxonasi qanday kutubxona va nima uchun ishlatiladi?
31. GMP qanday kutubxona va nima uchun ishlatiladi?
32. Boost.Multiprecision kutubxonasi nimasi bilan ustun?
33. C++ standartlashtirish bo‘yicha Milliy ishchi guruhdan rus ishlab chiquvchilari standart kutubxonaga mustaqil ravishda qanday tiplarini kiritish taklifi bilan chiqqan?
34. using int_mb = wide_int<1024*1024>; dastur fragmentini tushuntirib bering.
35. BigInt kutubxonasi haqida nimani bilasiz va undagi stoi funksiyasining vazifasini tushuntirib bering.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I

Murakkab saralash algoritmlariga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalga oshiriladi.

dastur

```
void shakersort(int* l, int* r) {
    int sz = r - l;
    if (sz <= 1) return;
    bool b = true;
    int* beg = l - 1;
    int* end = r - 1;
    while (b) {
        b =  ;
        beg++;
        for (int* i = beg; i < end; i++) {
            if (*i > *(i + 1)) {

    swap(*i, *(i + 1));
            b = true;
            }
        }
    if (!b) break;
        end--;
        for (int* i = end; i > beg; i--) {
```

topshiriqlar

1. Nechta saralash algoritmi berilgan.

_____
_____

2. shakersort(int* l, int* r) funksiyasidan dasturda foydalanib ko‘ring.

_____
_____
_____
_____

3. insertionsort(int* l, int* r) funksiyasidan dasturda foydalanib ko‘ring.

_____
_____
_____
_____
_____
_____

```
if (*i < *(i - 1)) {
swap(*i, *(i - 1));
            b = true;
            }
        }
    }
}
void
insertionsort(int* l,
int* r) {
    for (int *i = l +
1; i < r; i++) {
        int* j = i;
        while (j > l
|| *(j - 1) > *j) {
            swap(*(j
- 1), *j);
            j--;
        }
    }
}
void shellsort(int*
l, int* r) {
    int sz = r - l;
    float step = sz /
2.0;
    while (step >= 1)
{
    for (int *i = l +
step; i < r; i++) {
        int *j = i;
        int *diff = j
- step;
            while
(diff >= l && *diff >
*j) {

    swap(*diff, *j);
```

4. shellsort(int* l, int* r) funksiyasidan dasturda foydalanib ko‘ring.

_____
_____
_____
_____
_____
_____
_____
_____

5. Shell saralash algoritmi uchun yangi variant yarating.

_____
_____
_____
_____
_____

6. Satrlarni saralash algoritmini tuzing.

_____
_____
_____
_____
_____

```
    j = diff;
    diff = j - step;
            }
        }
        step /= 2;
    }
}
```

7. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
8. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

IKKINChI ASSISMENT TOPShIRIG‘I

☛ Saralash algoritmlari taqqoslashga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur

```
void bubblesort(int* l, int* r)
void shakersort(int* l, int* r)
void combsort(int* l, int* r)
void insertionsort(int* l, int* r)
void shellsort(int* l, int* r)
void shellsorthib(int* l, int* r)
void shellsortpratt(int* l, int* r)
void myshell_one(int* l, int* r)
```

topshiriqlar

1. Nechta saralash algoritmi berilgan.
   _____
   _____

2. shakersort(int* l, int* r) va myshell_one(int* l, int* r) funksiyalarni taqqoslash dasturini tuzing.
   _____
   _____
   _____
   _____

```
void myshell_two(int* l, int* r)
void myshell_three(int* l, int* r)
void treesort(int* l, int* r)
void gnomesort(int* l, int* r)
void selectionsort(int* l, int* r)
void heapsort(int* l, int* r)
void quicksort(int* l, int* r)
void quickinssort(int* l, int* r)
void mergesort(int* l, int* r)
void mergeinssort(int* l, int* r)
void newbucketsort(int* l, int* r)
void radixsort(int* l, int* r)
void radixsortmsd(int* l, int* r)
void bitonicsort(int* l, int* r)
void timsort(int* l, int* r)
```

3. `newbucketsort(int* l, int* r)` va `radixsortmsd(int* l, int* r)` funksiyalarni taqqoslash dasturini tuzing.

_____
_____
_____
_____
_____
_____

4. 2 ta saralash algoritmlarini tasidifiy eng katta soni 100000 bo‘lgan 100000 ta elementdan iborat massivlarni taqqoslang.

_____
_____
_____
_____
_____
_____
_____
_____

5. Shell va 3 ta saralash algoritmlarini tasidifiy eng katta soni 100000 bo‘lgan 100000 ta elementdan iborat massivlarni taqqoslang.

_____
_____
_____
_____
_____

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

UCHINCHI ASSISMENT TOPSHIRIG‘I

🎓	Juda ham katta sonlar bilan ishlashga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur

```
// Birinchi dastur.
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;
int main(){
    double n=0,a=0;
    cin>>a>>n;

vector<double>arr;

arr.push_back(pow(a,n
));
    for(int i=0;
i<arr.size(); i++)

    printf("%.0Lf\n",
arr[i]);
    system("pause");
    return 0;
}
// Ikkinchi dastur.
// Created by
MBBahodir
#include "stdafx.h"
#include <iostream>
#include <string>
```

topshiriqlar

1. Dastur kachon xato qisblashni amalga oshiriradi.

_____
_____

2. Vektordan boshqa konteynerdan foydalanish mumkinmi. Dasturda ishlatib ko‘ring

_____
_____
_____
_____

3. Aynan nima uchun printf("%.0Lf\n", arr[i]); foydalanilgan.

_____
_____
_____
_____
_____
_____

4. 2 ta katta sonni qo‘shish dasturini ishlating. Nima uchun 48 soni ayirib so‘ng qo‘shilgan. Tushuntirib bering.

_____
_____
_____
_____

```
using namespace std;

int main()
{
    string a, b, a1,
b1;
    cin >> a >> b;
    int leng = 0;
    // birinchi g‘oya
    if (a.length() >
b.length())
        leng =
a.length()+1;
    else
        leng =
b.length()+1;
// dasturchining
g‘oyasi
  int k = 0;
  while(k < leng){
      if( k < leng -
a.length() ) a1 +=
'0';
      if( k < leng -
b.length() ) b1 +=
'0';
      k++;
    }
  a1 += a;
  b1 += b;
// ikkinchi g‘oya
    int temp = 0;
    for (int ix =
leng-1; ix >= 0 ; ix-
-)
        {
```

_____
_____
_____
_____
_____
_____
_____
_____

5. 2 katta soni ayirish dasturini ishlating.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
__

```
            int temp_a
= a1[ix] - 48;
            int temp_b
= b1[ix] - 48; temp_b
= temp_b + temp;
            temp = 0;
            temp_b +=
temp_a;
            temp +=
(temp_b / 10);
               temp_b
%= 10;
            a1[ix] =
(char)temp_a + 48;
            b1[ix] =
(char)temp_b + 48;
         }
// uchinchi g‘oya
    if(b1[0] == '0')
{
    b1.erase(0,1);
    }
    cout << b1 <<
endl;
    system("pause");
    return 0;
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

TO‘RTINCHI ASSISMENT TOPSHIRIG‘I	
🎓	Large Integer sinfining qo‘llanilishiga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalgan oshriladi.

dastur	topshiriqlar

// qism dastur
bool operator <
(BigInt a, BigInt b)
{
Set(a);
Set(b);
if (a.size() != b.size()) return (a.size() < b.size());
FORD(i,a.size()-1,0)
if (a[i] != b[i]) return (a[i] < b[i]);
return false;
}
 	
bool operator >
(BigInt a, BigInt b)
{
return (b < a);
}
bool operator <=
(BigInt a, BigInt b)
{
return (a < b || a == b);
}
 	
bool operator >=
(BigInt a, BigInt b)
{
return (b < a || b == a);
}
	1. Dastura nechta taqqoslash operator keltirilgan.
	_____
	_____
	_____
	_____
	_____
	_____
	2. operator == ni 2 BigInt tiplarini taqqoslash funksiyasini yozing
	_____
	_____
	_____
	_____
	3. operator < ni 2 BigInt va int tiplarini taqqoslash funksiyasini yozing.
	_____
	_____
	_____
	_____
	_____
	_____
	4. 2 BigInt tiplarini kichigini topuvchi min funksiyasini yozing
	_____
	_____
	_____
	_____
	_____
	_____
	_____
	5. operator ++ ni yozing.
	_____
	_____
	_____
	_____
	_____
	_____

```
bool operator >
(BigInt a, int b) {
    return (a >
Integer(b));
}

BigInt max(BigInt a,
BigInt b) {
    if (a > b) return
a;
    return b;
}

BigInt operator +
(BigInt a, int b) {
    return a +
Integer(b);
}

void operator +=
(BigInt &a, BigInt b)
{
    a = a + b;
}
```

6. Dasturda jami bo‘lib, necha o‘zgartirish kiritildi. _____
7. Shu dasturning analogini yaratish sizga mustaqil vazifadir.

3-BOB. VISUAL C++ MUHITIDA DASTURLASH.

3.1. Visual C++ muhitida dasturlash.

📖 Integrallashgan ishlab chiqarish muhiti, ularni yaratilish tarixi, ribovlanish bosqichlari, yagona tilli va ko‘p tilli integrallashgan muhitlar, Vizual Studio variantlari va imkoniyatlari, .NET frameworkning imkoniyatlari, Visual C++ da Windows ilovalarni yaratish, Muhitda menyular va uskunalar paneli, Form xususiyatlari va hodisalaridan foydalanish bo‘yicha nazariy hamda namunalari keltirilgan bo‘lib, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni mustahkamlash uchun 30 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 10 assisment topshirig‘i berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

🖋 ***Kalit so‘zlar.*** IDE, Visual C++, user interface, OYD, .NET, (plug-ins, add-ins, add- ons, abstrakt Windows Toolkit (AWT), MSDN, Visual Studio Interdev-ASP (Active Server Pages), GUI, ilova, loyiha, Windows Forms application, .NET Framework, Buyruqlar va asboblar paneli, Form xususiyatlari, hodisalari, MessageBox.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Tip tushunchasi, sinf va sinf obyekti, funksiya va ko‘rsatkich, kutubxona yaratish va dasturlashga oid dastlabki tushunchalar hamda C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Visual C++ muhitida dasturlash, integrallashgan ishlab chiqarish muhiti, toolkit, toolbox, Integrallashgan muhitlar tarixi, Turbo muhitlar, GNU Emacs, Obyektga yo‘naltirilgan til, Team Foundation Server (TFS), kengaytmalarni ishlab chiqish mexanizmi (plug-ins, add-ins, add- ons), Yagona tilli va ko‘p tilli integrallashgan muhitlar, NetBeans integrallashgan muhiti, Java dasturlash haqida, Microsoft Visual C++, ASP texnologiyasi, Visual C++ da Windows ilovalarni yaratish, Muhitda menyular va uskunalar paneli, Buyruqlar va uskunalar paneli, MS Visual Studio muhitining interaktiv tugmalari, Form xususiyatlari va hodisalari, MessageBox sinfining show funksiyalarini o‘rganishingiz mumkin.

REJA

1.Visual C++ muhitida dasturlash.
2. Visual C++ da Windows ilovalarni yaratish
3. Visual C++ muhitida menyular va uskunalar paneli.
4. Form xususiyatlari va hodisalari

KIRISH

Bugungi kunda dasturlash muhitlari va turlari ko‘payib bormoqda. Shu vaqtgacha console rejimida dasturlashni o‘rgangan bo‘lsangiz kerak. Ammo so‘nggi 35 yil ichida foydalanuvchilar oynali dasturlash muhitlaridan foydalanib kelishmoqda. Bunday dasturlarni yaratish uchun dasturlashning barcha nazariy asoslariga tayanib, foydalanuvchi uchun qulay vizual dasturlash muhitlari yaratilgan. Ularda birinchi navbatda foydalanuvchining UI (user interface) yoki dasturning foydalanuvchi bilan muloqot oynasi loyihalashtiriladi. Buning uchun tayyor sinflar (komponetalar) mavjud bo‘lib, ularning xususiyatlar, funksiyalari va hodisalari bilan ishlash juda qulay bo‘lib, vizual foydalanish mumkin. Vizual foydalanish ko‘rib turib ishlatish degan ma‘noni beradi. Bu vizual dasturlash bo‘yicha ilk ma’lumotlarni keltiramiz. Deyarli barcha dasturlash tillari uchun vizual dasturlash muhitlari mavjud.

Visual C++ muhitida dasturlash. Dastavval integrallashgan ishlab chiqarish muhitining ilovalari tushunchasi, uning model xususiyatlari, rivojlanish tarixi, ularning eng ko‘p ishlatilganlari bilan tanishish lozim.

Integrallashgan ishlab chiqarish muhiti (Integrated development environment - IDE) - dasturni ishlab chiqish hayot siklining barcha asosiy funksiyalarini qo‘llab-quvvatlovchi umumiy interaktiv grafik qobig‘iga ega bo‘lgan dasturlarni ishlab chiqish va tekshirish vositalari majmui, dastur matnini (kod) yozish va tahrirlash, kompilyatsiya, bajarish, xatolarni tuzatish, profilaktika va boshqalarni bajarish imkoniyati beradi.

Integrallashgan muhitdan foydalanish dastur ishlab chiqishning mumkin bo‘lgan yondashuvlaridan biridir. Shu bilan bir qatorda, UNIX tizimining oldingi an‘anaviy yondashuvi, funksionallik bilan bog‘liq bo‘lgan vositalar (toolkit, toolbox) to‘plamidan foydalanishga asoslangan, lekin Integral interaktiv muhitga va ba‘zan (UNIX tizimining dastlabki versiyalarida) buyruq satri rejimida (buyruq satri interfeysi) amalga oshirilgan. Albatta, Integrallashgan muhitdan foydalanish ishlab chiquvchi uchun ancha qulay bo‘lib, u 1980-yillardan boshlab Integrallashgan muhitlarning jadal rivojlanishi va turli tumanligini izohlaydi.

Birinchi integrallashgan muhitlardan biri Borlandning Turbo Paskalidir, 1980-yillar o‘rtalarida Niklaus Wirth va uning talabasi Filip Kahn tomonidan ishlab chiqilgan.

Microsoft zamonaviy integrallashgan muhitning eng yaxshi namunalaridan bo‘lgan Visual Studio ni yaratish va rivojlantirish orqali Integrallashgan muhitlarning rivojlanishiga katta hissa qo‘shdi. Uning yangi versiyasi juda ko‘p va oxirgisi, Visual Studio 2019, ammo keyinchalik Visual Studio 2012 muhitida C++ tilidan foydalanamiz.

Integrallashgan muhitlar tarixi. Integrallashgan muhitlar g‘oyasi 1980-yillarning o‘rtalariga kelib, integrallashgan muhitlarning ikki guruhi paydo bo‘ldi:

-Turbo muhitlar (Turbo Pascal, Turbo C, C++, Delphi, Turbo va hokazo.) bu tillarda dasturlashni qo‘llab-quvvatlash uchun Borland tomonidan, MS DOS operatsion tizimi uchun, keyin Windows uchun foydalanishlang;

-GNU Emacs - MS DOS uchun, keyin Windows, OpenVMS va Linux uchun amalga oshiriladigan ko‘p tilli va ko‘p platformali integrallashgan muhitlar. 1990 yilda Sun Microsystems bilan ishlagan rivojlantirish jamoasi orasida, Solaris platformasi uchun uni amalga oshiruvchi ko‘p foydalanuvchilari va GNU Emacs tashabbuskorlari bor edi.

Obyektga yo‘naltirilgan til Smalltalk kompaniyasi Xeror PARC dasturlarni ishlab chiqish uchun o‘sha yillarda integrallashgan muhiti amalga joriy qilingan. Unda OYD birinchi integrallashgan muhiti yaratilgan va u dasturlash texnologiyasi bytecode ikkilik va Postfix oraliq vakillik tushunchasini joriy qilgan va shuningdek just-in-time (JIT, dinamik) kompilyatsiyasini birinchi ilg‘or usuli bo‘yicha kompyuterda maqsadli ijro kompilyatsiyasi yaratgan.

Borlandning turbo muhitlari dasturiy ta‘minot ishlab chiquvchilari va dasturiy ta‘minot ishlab chiqish vositalari yaratuvchilariga katta ta‘sir ko‘rsatdi. Ularning xarakterli xususiyati doimiy rivojlanish siklini qo‘llab-quvvatlash edi: manba matnini yozish va tahrirlash - kompilyatsiya - tahlil qilish va xatolarni tuzatish - kompilyatsiya qilishni yakunlash-ijro etish va xatolarni qayta tuzatish. Integral muhitdan chiqmasdan va bu bosqichlar oddiy funksiya tugmachalari bilan nazorat qilindi va har qanday alohida vositalarni aniq chaqirishni talab qilmaydi. Yuqori kompilyatsiya tezligi ham Turbo muhitlarining jozibador tomoniga aylandi. Turbo Paskalning birinchi versiyalarida kompilyatsiya birinchi xatodan oldin amalga oshirilgan bo‘lsa-da, barcha xatolarni topish va tashxislash uchun avvalgisini o‘rnatganingizdan so‘ng kompilyatsiyani qayta ishga tushirishingiz

kerak edi, ammo bu funksiya tugmalaridan foydalanib darhol amalga oshirildi.

Turbo muhitlar - to‘liq ikkilik kodi, loyiha kodi kompilyatsiyasi uchun ajralmas BUILD mexanizmi bor edi, shuningdek, make (F9) rejimi, klassik UNIX make faoliyatiga o‘xshash, faqat tahrirlangan kod modullar recompiling uchun amalga oshirilgan.

Turbo Paskal muhitidagi eng muhim yangilik - kirish tilining obyektga yo‘naltirilgan tushunchalar (sinf, obyekt) bilan kengaytmasi va mustaqil kompilyatsiya birligi g‘oyasini o‘zida mujassamlashtirgan birlik konstruksiyasi (moduli) bo‘lgan. OYD uchun qo‘llab - quvvatlash versiyada paydo bo‘lgan. Keyinchalik bu g‘oyalar Windows platformasida, Borland integrallashgan muhitlarining yangi versiyalari - Borland Pascal va Delphi (Paskal g‘oyalarini OYD bilan birgalikda rivojlantiruvchi Borland dasturlash tili) ishlab chiqildi.

Zamonaviy integrallashgan muhitlarning asosiy xususiyatlari. Endi integrallashgan dasturiy ta‘minot ishlab chiqish muhitlarining asosiy xususiyatlarini umumlashtiraylik. Ularning har biri quyidagi qismlarga ega:

- Funksiya tugmalaridan keng foydalangan holda muhitdan chiqmasdan boshqa barcha komponentlarni chaqirishga imkon beruvchi yagona interaktiv qobiq;

- Dastur franmentlarini yozish va tahrirlash uchun matn muharriri;

- Qo‘llab-quvvatlash tizimini qurish, ya‘ni amalga oshirilayotgan manba tilidan kompilyator va bitta bajariladigan kod (yuklash moduli) da obyekt ikkilik kodlarining linkerini o‘z ichiga olgan manba kodidan loyihalarni tuzish; linker operatsion tizimning standart komponenti sifatida yoki ushbu muhit uchun maxsus ishlatiladi;

- Buyruqlar majmui yordamida muhitda debugging dasturlari uchun Debugger: bir joyda o‘rnatish; berilgan tartibda to‘xtatish; o‘zgaruvchilar qiymatlarini tasavvur (yoki, past darajada, registrlari va xotira hududlari) qilish;

- Integrallashgan muhitda zamonaviy matn muharrirlari kodi avtomatik bajarilishini ta‘minlash (kodni tugatish), muharriri muhitda joriy yozilgan kodni sintaktik to‘g‘ri va uning davomi bo‘lishi mumkinligi talab qilinadi, misol uchun, yopilish qavs yo‘qligi, nuqta-vergul va usul nomlari, tafovutlar usul ma’lum bir sinf obyektdan bo‘lsa va boshqalar.

-Integrallashgan muhitlarning zamonaviy versiyalarida quyidagi xususiyatlar ham qo‘shilgan (komponentlar):

- Profiler - integrallashgan muhit nazorati ostida dasturni bajarish natijasida olingan statistik ma’lumotlarni jamlash va tahlil qilish vositasi: protseduralarga (usullarga) chaqiriqlar soni, dasturni bajarish vaqtida ishlatiladigan xotira miqdori va boshqalar.

- Refactoring - kodni takomillashtirish maqsadida muhitdagi dasturlarning tizimli guruhli modifikatsiyalarini, ularning funksiyalariga tub o‘zgarishlarsiz amalga oshirish vositalari. Odatda, bunday harakatlarni o‘z ichiga oladi, misol uchun, uning ta‘rifi va barcha foydalanish usuli nomini o‘zgartirish, uning argumentiga qo‘shib, sinash - try/catch blok oldindan e‘tiborsiz istisno bilan band qilish va hokazo.

- Manba kodini boshqarish tizimi (source code control system) yoki mavjud versiya tizimlaridan biri (CVS, RCS, Mercurial, Visual Sourcesafe va boshqalar) bilan muhitni integrallashgan qilish vositasi. Dasturlarni qo‘llab-quvvatlashda loyiha kodi fayllarining versiyasini nazorat qilishni qo‘llab-quvvatlash;

- Dasturlarning jamoaviy rivojlanishini qo‘llab-quvvatlash vositalari (jamoaviy ish) - dastur hayot sikli bosqichlari (talab va xususiyatlar, dizayn, amalga oshirish, sinov), dasturchilar jamoasi a’zolari o‘rtasida rivojlanish vazifalarini taqsimlash, loyiha menejeri tomonidan vazifani bajarishni nazorat qilish. Visual Studio muhitida bu komponent birinchi bo‘lib Team Foundation Server (TFS) deb nomlangan va Visual Studio 2013 versiyasidan boshlab, u bulutli interfeys sifatida amalga oshiriladi va Visual Studio Online deb ataladi.

- Kod tahlil vositalari - uning semantik to‘g‘riligi: odatda ijro paytida aniqlanadigan xatolarning ayrim turlarining yo‘qligi, masalan, erishib bo‘lmaydigan sharoitlar; zarur tekshirishlar va xavfsizlik ruxsatlarining yo‘qligi va boshqalar. Professional qiziqishlar tufayli juda muhim bo‘lgan ushbu xususiyatlar haqida qo‘shimcha ma’lumot olish uchun "Visual Studio 2013 va uning ishonchli va xavfsiz hisoblash (ishonchli hisoblash) uchun imkoniyatlari" ga qarang . Ushbu imkoniyatlar 2002 yilda shakllantirilgan ishonchli va xavfsiz hisoblash (trustworthy computing) muhuti va tamoyillariga mos keldi va u Microsoft korporatsiyasi tomonidan hayotga izchil tatbiq etildi. Zamonaviy muhitlarga dasturdagi nazorat oqimi grafigining siklomatik soni, sinflarning ilashish darajasi (o‘zaro bog‘liqligi) va shu kabi murakkabligini tavsiflovchi metrikalar bo‘yicha kodni tahlil qilish vositalari ham kiradi;

- Hosil bo‘lgan binar kodni vizuallashtirish vositalari-usullar, o‘zgaruvchilar, ularning nomlari va boshqalar. Misol uchun, Visual Studio muhitida, platforma .NET Common Intermediate til yagona oraliq (ikkilik) kodni tasavvur qilish imkonini beradi;

- Standart kod vositalari asosida turli xil dasturiy loyihalar va yechimlarni yaratishni qo‘llab-quvvatlash; kengaytmalarni ishlab chiqish mexanizmi (plug-ins, add-ins, add- ons). Zamonaviy dasturiy ta‘minotni ishlab chiqishda ba‘zan turli xil ilovalar va vositalar - konsol (oddiy) ilovalar, veb-ilovalar va veb-xizmatlar, mobil ilovalar, bulutli ilovalar va boshqalarni yaratish kerak. Bu turdagi har bir manba kodi fayllar tuzilishni ma’lum bir darajada rivojlantirishni talab qiladi. Shuningdek belgilash konfiguratsiya fayllar sifatida, misol uchun, kod xavfsizlik ruxsatini, veb-konfiguratsiyalar va hokazo. Zamonaviy integrallashgan muhitlar manba kodi shablonlarini taqdim etish va loyiha uchun zarur konfiguratsiya fayllarini avtomatik ravishda ishlab chiqarish orqali turli xil loyihalarni yaratishni avtomatlashtiradi. Dasturlashni tayyor kod shablonlarini ishlatmasdan tasavvur qilish qiyin, bu esa muqarrar ravishda xatolarga yo‘l qo‘yadi. Misol uchun, u qo‘lda loyihaning ajralmas qismi bo‘lgan faylni yaratishni unutish, yoki kod ba‘zi muhim parcha ko‘zdan yo‘qotish juda oson (masalan, mos kelmaydigan ko‘rsatkichlar uchun, lekin unga bog‘langan ko‘rsatkichni ta‘minlash uchun emas). Shuning uchun, integrallashgan muhitlar tomonidan loyihalarni turli turdagi qo‘llab-quvvatlash, ayniqsa, muhim ahamiyatga ega. Bundan tashqari, zamonaviy integrallashgan muhitlarda mumkin bo‘lgan loyiha turlari majmui kengaytiriladi, ya‘ni, tuzuvchi zarur bo‘lsa, muhitga loyihaning yangi turini kiritishingiz mumkin. Misol uchun, aspect ga yo‘naltirilgan dasturlash vositasini amalga oshirishda Aspect.net Visual Studio integrallashgan muhit kengaytmalari sifatida, tegishli kod shabloni bilan loyiha(aspect)ning yangi turini joriy qilgan.

- UML (yagona modellashtirish tili) modellashtirish tilida dasturlar strukturasini modellashtirishni qo‘llab-quvvatlash. UMLning zamonaviy versiyasi (2.x) til har xil turdagi dasturlar va tegishli diagrammalarni qurish modellarini ta‘minlaydi. Bundan tashqari, UML dastur ishlab chiqish va ishlab chiquvchilar o‘rtasida o‘zaro hamkorlik uchun faoliyat modellari ishlab chiqishni qo‘llab-quvvatlaydi. Zamonaviy integrallashgan muhitlar UML tilidan foydalanishni ikki yo‘l bilan qo‘llab-quvvatlaydi: modelni hamda manba kodidan tegishli

diagrammani hosil qilish va aksincha, ishlab chiqilgan modeldan manba kodini (shablonini) hosil qilish.

Yagona tilli va ko‘p tilli integrallashgan muhitlar. Dastlab yagona manba tilida dasturlash uchun integrallashgan muhitlar ishlab chiqilgan (masalan, Turbo Paskal-Borland ning Paskal kengaytmasida dasturlash uchun).

- Shu bilan birga, bunday mono tildagi integrallashgan muhitlarni ko‘p tilli muhitlarga aylantirish bo‘yicha bosqichma-bosqich tendensiya yuz berdi, chunki turli tillarda loyihalarni ishlab chiqish uchun o‘xshash tamoyillar va mexanizmlar qo‘llaniladi va turli tillarda yozilgan dastur bo‘laklarini bitta loyihada ishlatish ham ba‘zan qulay. Masalan, loyihada, shu jumladan, yagona maqsad uchun, masalan, C#, yana qayta yozish uchun emas, balki oldingi tilda yozilgan meros kodni ishlatish uchun yaratilgan(masalan, C).

- taniqli NetBeans integrallashgan muhit dastlab Java dasturlash tili uchun Pragada Charlz universiteti talabasi loyihasi sifatida yaratilgan. Hozirgi kunda NetBeans muhiti C va C++ da loyihalarni ishlab chiqishga ko‘mak beruvchi C / C++development pack komponentasini amalga oshiruvchi kuchli, ko‘p tilli integrallashgan muhitga aylandi.

- Visual Studio - environment.net azaldan ko‘p tilli muhit sifatida yaratilgan. Ishlab chiquvchilar ishlab chiqilgan loyihaning tegishli qismlari uchun eng qulay tilni (tillarni) tanlashlari va loyihani tegishli tillardan bitta ikkilik oraliq SIL kodiga tuzish orqali olingan ikkilik komponentlardan (assemblies) to‘plashlari uchun Microsoftning prinsipial sozlamalari. Keyinroq Visual studio ning ushbu qulay xususiyati va u qo‘llab-quvvatlovchi tillar majmuasi bilan yaqindan tanishib chiqamiz.

- Integrated development environment (IDE) - dastur ishlab chiqish hayot siklining barcha asosiy vazifalarini qo‘llab-quvvatlovchi umumiy interaktiv grafik qobig‘iga ega bo‘lgan dasturlarni ishlab chiqish va xatolarni qayta tahrir qilish vositalari majmuidir.

Integratsiyalashgan Microsoft Visual Studio muhiti 1995 yilga kelib, ommaga keng taqaladi, bu nom ostida mahsulotning birinchi versiyasi chiqarildi. Bu oldin, 1990-yillar boshida, Microsoft yuqori darajadagi tillarda dasturlash qo‘llab-quvvatlash uchun alohida mahsulotlar bilan bog‘liq bo‘lgan:

- Microsoft Visual C++ - Visual C++ da interaktiv dasturlash muhiti, Microsoft tomonidan ishlab chiqilgan va amalga oshirilgan, C++ tilining kengaytmasi hisoblanadi. Visual C++ hali ham Visual Studio

ning barcha versiyalarida dunyodagi eng mashhur va keng tarqalgan dasturlash tili;

- Visual Basic-Microsoft tomonidan ishlab chiqilgan va amalga oshirilgan asosiy tilning obyektga yo‘naltirilgan kengaytmasi, tilning soddaligini obyektga yo‘naltirilgan kengaytmalar bilan birlashtirgani uchun darhol butun dunyodagi dasturchilar tomonidan faol qo‘llanila boshlangan. 1990-yillar boshida ko‘p dasturchilar Visual Basic qulayligini ta’kidlab, GUI boshqaruv dasturlarini ishlab chiqish uchun foydalanishgan;

- Microsoft Visual FoxPro-Visual FoxPro tilida interaktiv dasturlash muhiti-kichik firma Fox Software tomonidan dastlab Foxbase nomi ostida ishlab chiqilgan protsessual dasturlash elementlari bilan obyektga yo‘naltirilgan til. Ko‘pgina foydalanuvchilar uchun bu tilning xususiyati SQL asosida ma’lumotlar bazalariga bevosita bog‘langan va ushbu tildagi dasturdan, xususan FoxPro da SQL so‘rovlarini dasturlashga yordam bergan;

- Microsoft Visual Sourcesafe-Microsoft kodi versiyasi nazorat tizimi tomonidan ishlab chiqilgan, keyinchalik Visual Studio muhit bilan integratsiyalangan.

Ushbu mahsulotlarning barchasi 1995-dan "bir muhitga" Visual Studio nomi bilan birlashtirildi. Yangi muhitga 4.0 versiyasi deb nom berildi, yuqorida bayon etilgan muhit komponentlari bir necha yillar davomida alohida shaklda chiqarilgan.

Ushbu versiyada ham Visual Studio juda mashhur bo‘ldi. Masalan, 1995-yilda yangi Java dasturlash tili va texnologiyasini chiqargan Sun Microsystems Windows platformasi uchun GUI rivojlanishini qo‘llab-quvvatlash uchun Visual Studio muhiti va platforma-mustaqil Java kutubxonalarini, abstrakt Windows Toolkit (AWT) ni amalga oshirish uchun unda amalga oshirilgan Visual C++tilidan foydalangan.

1990-yillarda dasturchilar orasida Visual Studio muhiti mehr bilan do‘stona "vizualka" nomi bilan atalgan. Bu bejiz emas: Visual Studio dasturchilarining rivojlanishida ishonchli hamkorga aylangan, bu uning interfeysining do‘stonaligi va qulayligini tasdiqlaydi. Taqqoslash uchun, o‘sha vaqtga kelib, o‘sha firma SUN bir xil mashhur dasturiy muhitga ega emas edi. Java dasturlash uchun NetBeans muhitini bir oz keyinroq - 1997 yilda, Charlz universiteti Praga dasturlash talabalar kichik guruhi ish natijasida paydo bo‘ldi. Visual Studio muhitining keng tarqalishida MSDN (Microsoft Developers ‘ Network) ga obuna bo‘lish orqali

muhim rol o‘ynadi, bu esa eng so‘nggi Microsoft dasturiy ta‘minotini ishlab chiqishga imkon yaratdi.

Vizual Studio 97 (5.0). Visual Studio yangi (beshinchi) versiyasi, shuningdek, Visual Studio 97 sifatida tanilgan. Uning yangi xususiyatlari va vositalari Visual Studio Interdev-ASP (Active Server Pages) texnologiyasi asosida interaktiv veb - saytlarni ishlab chiqish vositasi - va Visual Java++ - "Microsoft"dan Java dasturlash muhiti bo‘lgan.

ASP texnologiyasi - Microsoft kompaniyasining web dasturlash sohasidagi ajoyib rivojlanishi, 1994-1995-yillarda paydo bo‘lgan. Faol veb-sahifalarni ishlab chiqish uchun mo‘ljallangan-veb-so‘rovlarga javob sifatida mijoz brauzerlari uchun HTML-sahifalarni yaratish uchun shablonlarga ega bo‘lgan. Bu hali ko‘p saytlarda ishlatiladi. Texnologiya keyinchalik uning asosida ishlab chiqilgan ASP.NET, xususiyatlariga keng imkoniyat ochib bergan.

Visual Java++ vositasiga kelsak, umuman boshqa turdagi hodisalar u bilan bog‘liq - ba‘zida, afsuski, o‘z dunyosida shunga o‘xshash narsalar sodir bo‘ladi, bu esa kompaniyalar juda raqobatbardosh bo‘lishi muqarrar bo‘lgan. Microsoft Sunning yangi Java texnologiyasi bilan tanishib chiqqan va Sun bilan raqobatda unga javob sifatida Java tilining o‘z kengaytmasini ishlab chiqdi va uni Visual Java++deb nomlangan yangi muhitda amalga oshirdi. Biroq, afsuski, bu harakatlar to‘liq Sun tomonidan joriy rasmiy Java amalga oshirish tartibi bilan mos kelmadi. Shu kungacha (Oracle hozir Java texnologiya egasi bo‘lsa-da, bu tartib aslida o‘zgarmagan), Java nomi rasman ro‘yxatdan o‘tgan savdo belgisidir. Shuning uchun, yangi Java amalga oshirish uchun, Java texnologiyasi egalik kompaniya litsenziya sotib, va amalga oshirish idoraviy Java standartlarga javob tekshirish uchun Java moslashuv Kit (JCK) deb nomlangan yangi Java amalga oshirish uchun til, kutubxonalar va vertual mashina amalga oshirish uchun testlar maxsus majmuini o‘tishi kerak bo‘lgan. Faqat bu testlarning hammasi to‘liq 100% o‘tsa, yangi amalga oshirish "Java"deb nomlanish huquqini oladi. Afsuski, Microsoft tomonidan Visual Java++ ni amalga oshirish uchun Visual Java++ tizimida amalga oshirilgan til kengaytmalari tufayli JCK testlaridan o‘tmadi. Bu ish, afsuski, ikkala kompaniya uchun ham, Sun va Microsoft o‘rtasidagi uzoq muddatli huquqiy jangda, Sun tomonidan qo‘lga kiritilgan, natijada Microsoft mahsuloti Vj++ deb nomlangan (Java nomini ochiq-oydin eslatib o‘tmasdan). Bunday hodisalar muqarrar bo‘lsa-da, juda yoqimsiz, chunki har ikki tomon ham, ba‘zan

bir necha yillar davomida mashhur texnik mutaxassislar, bularning hammasiga ko‘p vaqt, kuch va sog‘liq sarflashadi. Biroq, keyinchalik, 2004 yilda, Sun va Microsoft 10 yil muddatga intellektual mulk bo‘yicha misli ko‘rilmagan shartnoma tuzishgan.

Shunday qilib, Visual Studio 5.0 Visual C++, Visual Basic va VJ++ da integrallashgan dasturlash muhiti bo‘lib, u ASP texnologiyasi yordamida interaktiv Web saytlarni ishlab chiqish uchun Visual Studio Interdev vositasini ham o‘z ichiga oladi.

Visual Studio 6.0 (1998). Visual Studio keyingi versiyasi (6.0, kodlangan Aspen) 1998 yilda ishlab chiqildi. Visual C++, Visual Basic, Vj++ va Interdev, (maksimal versiyada - Visual Studio Enterprise Edition) - allaqachon ma’lum komponentlarini yangi versiyalari bilan bir qatorda, quyidagi yangi komponentlar ham vizual modellashtirish va dastur ishlashini tahlil qilishni qo‘llab-quvvatlash uchun kiritilgan:

1. Application Performance Explorer
2. Automation Manager
3. Microsoft Visual Modeler
4. RemAuto Connection Manager
5. Visual Studio Analyzer

Visual Studio 6.0 dasturlari bilan ishlayotgan bitta ko‘p tilli virtual platformani o‘z ichiga oladi. Integrallashgan muhit yangi versiyasini yaratish uchun asos sifatida Microsoft tomonidan ishlatiladigan .NET. versiyasi.

Visual Studio.net (2002). Visual Studio 7.0 versiyasi, shuningdek, Visual sifatida tanilgan Studio.net va kodlangan Rainier, joriy etildi. Ushbu versiyada eng muhim yangilik boshqariladigan kod, umumiy til infratuzilmasi (SLI), umumiy oraliq til (SIL), umumiy turdagi tizim (SP) asosida to‘liq turdagi nazorat va xavfsizlik bilan ishonchli va xavfsiz dasturlashni ta‘minlaydigan ko‘p tilli platformani amalga oshirishdir. .Net platformada barcha kompilyatorlar manba kodini tegishli tildan tarjima qiladi (C#, C++, Visual Basic va hokazo.) bitta oraliq SIL kodining ikkilik formatiga - VM ning postfiks yozuvi instructions.net - Virtual ijro tizimi (VES). Taqdim etilgan: .Net ichidagi moslik-turli tillardan tuzilgan kodlarning ilovalari; yagona metadata asosidagi nazorat va xavfsizlik mexanizm; yagona ko‘p tilli istisno muomala mexanizmini o‘z ichiga oladi.

Visual Studio.NET 2002 versiyasi to‘rt versiyalarida chiqarildi ular: Academic, *Professional*, *Enterprise Developer*, *Enterprise Architect*.

Visual Studio muhitida qo‘llab-quvvatlanadigan tillarda haqiqatdan ham ajoyib o‘zgarish va qo‘shimchalar amalga oshirildi.

Avvalo, C# -yangi dasturlash tili ishlab chiqildi va amalga oshirildi, hozirgi kunda eng kuchli, zamonaviy va to‘liq dasturlash tili.

Visual C++ va Visual Basicda muhim kengaytmalar amalga oshirildi. Aslida sintaksisdagi ayrim farqlarga qaramay, bu tillar deyarli o‘xshash xususiyatlarni ta‘minlab, C# tiliga "teng" bo‘lib kelgan. Yangi .NET -mos versiyasi Visual C++ muvaffaq deyiladi, C++ va yangi Visual Basic SOF-mos versiyasi Visual Basic.net deyiladi.

Visual Studio.net 2003. Keyingi yili, 2003, Visual Studio yangi versiyasi released.net kodi nomi Everett ostida ishlab chiqardi. Bir butun sifatida mahsulot ishlab chiqish nuqtai nazaridan, bu versiyada muhim o‘zgarishlar qilinmadi: yangi versiyasining asosiy maqsadi versiyasi .net Framework 1.1 uchun yangilash bilan ta‘minlash edi. biroq, Visual Studio 2003 dagi juda muhim yangilik mobil qurilmalar .net Compact Framework uchun versiya .net da ishlab chiqildi va Visual Studio vositalari yordamida mobil qurilmalar uchun dasturlar ishlab chiqish imkoniyatini taqdim etdi.

Oldingi versiyasi kabi, Visual Studio 2003 to‘rt xil versiyada chiqarildi: *Academic, Professional*, *Enterprise Developer*, *Enterprise Architect.* Bundan tashqari, "maksimal" versiyada - Enterprise arxitektor - Visual Studio 2003 dastur arxitekturasini tasavvur UML diagrammalar uchun qo‘llab - quvvatlash, shu jumladan, boshqa mashhur mahsulot - Microsoft Visio tarzida dasturiy modellashtirish vositalari qo‘llab-quvvatlaydi.

Visual Studio.net 2005. Visual Studio 2005 integralashgan muhitning (kodlangan Whidbey) keyingi versiyasi 2005 yilda e‘lon qilindi.

Versiyaning asosiy yangiliklari parametrli ma’lumot turlari (generics) ga tegishli. Ular C# 2.0 tilidagi versiyada va .net Framework 2.0 da amalga oshirildi. Bu bir vaqtning o‘zida shunga o‘xshash xususiyatlari Java 1.5 da amalga oshirildi, deb ta‘kidlash qiziq, lekin generics mexanizmi Java nisbatan, yanada qulay, moslashuvchan va kengaytirilgan versiyada amalga oshiriladi .net va C# platforma asoslangan ta‘kidlash lozim.

Manfaatdorlikni yaxshilash yana bir guruh ASP.NET -. NET platformasiga xos kodning ishonchliligi va xavfsizligi bilan birgalikda, veb-sahifalarni ishlab chiqish va faollashtirish uchun mo‘ljallangan ASP texnologiyasini ishlab chiqildi. Visual Studio 2005 versiyasi qo‘shimcha

loyiha turlari tushunchasini qo‘llab-quvvatlash uchun birinchi marta ASP.NET Web xizmatlari amalga oshirildi. NET dagi web dasturlash vositalari ushbu platforma va umuman Web dasturlash talabalari uchun alohida e‘tiborga loyiqligini unutmang. Faqat ularni o‘rganish va ishlatish uchun har bir dasturchi .NET platformasini yaxshi bilishi va amalda o‘zi ishlatishi kerak.

Aspect yo‘naltirilgan dasturlash tizimini rivojlantirish jamoasi uchun Aspect.net Visual Studio 2005 versiyasi alohida rol o‘ynadi. Visual Studio muhitining ushbu versiyasi uchun plagin deb nomlangan Aspect.net Framework-integrallashgan muhit o‘zining vizual uslubiga o‘xshash bir uslubda jihatlarini boshqarish uchun GUIdir. Interaktiv xususiyatlar va qulaylik Aspect.net foydalanuvchilar tomonidan yuqori baholandi va tizimning ushbu versiyasida Aspect.net dunyoning 26 mamlakatida tarqalgan.

Vizual *plug-in* sifatida amalga oshirilgan boshqa taniqli loyiha Studio.net 2005 loyihasi hisoblanadi. Knowledge.net - bilim vakillik orqali C# tilining kengaytmasi (ramkalar, qoida silsilasi, ontologiyalar), qaysi bilim foydalanish bilan an‘anaviy C # dasturlash uslubi birlashtirib beradi. Knowledge.net talabalar va magistrantlar tomonidan bilimlardan foydalanadigan aqlli yechimlarni ishlab chiqish uchun ishlatiladi.

Visual Studio.net 2008. 2008 yilda Visual Studio ning navbatdagi versiyasi ishlab chiqarildi. Microsoft kompaniyasining Visual Studio muhitining ichki versiya raqami Visual Studio 9 hisoblanadi. Ushbu raqamlar yodda tutish uchun foydalidir, chunki Microsoft xodimlari ko‘pincha Visual Studio rivojlanishini tasvirlashda ushbu ichki versiya raqamlaridan foydalanadilar (masalan, bloglarda). Visual Studio 2008 versiyasi Windows Vista uchun mo‘ljallangan. Bu boshqa mashhur Microsoft Office 2007 mahsulot versiyasiga mos keladi va ilg‘or veb-dasturlashni qo‘llab-quvvatlash vositalarini o‘z ichiga oladi. Versiya .NET Framework Visual Studio 2008 versiyasi uchun .NET Framework 3.5 ga tegishli hisoblanadi.

VS 2008 versiyasi GUI-ni qo‘llab-quvvatlash uchun Windows Presentation Foundation (WPF) API-dan foydalanadigan ilovalar uchun yangi VS 2008 debugger multithreaded dasturlarda xatolarni qayta tahlillash uchun yordam beradi.

VS 2008 versiyasi birinchi marta J# tilini qo‘llab-quvvatlamadi, Microsoft Java tilining kengaytmasi, yuqorida aytib o‘tilgan. Buning o‘rniga foydalanuvchilarga Java tili va muhitidan ilovalarni .Net

muhitiga ko‘chirish uchun muhitga kiritilgan Java Tili o‘zgartirgich Assiatant (JLCA) vositasi taklif etildi.

VS 2008 hozirda maktab uchun muhim Microsoft ichki mahsulotni qo‘llab – quvvatlaydi. Visual Studio muhitining so‘nggi versiyasi- til protsessorlari optimallashtirish rivojlantirish uchun bir vosita. Bu Microsoftning tadqiqot va ishlab chiqarish jamoasining qo‘shma mahsuloti hisoblanadi.

Visual Studio.net 2010. 2010 yil aprel oyida Sankt-Peterburgda Visual Studio 2010 ishga tushirilishi marosimida bir qator qiziqarli taqdimotlar va namoyishlar bo‘lib o‘tgan. Jumladan, Visual Studio kompaniyasining yetakchi ishlab chiquvchisi, Microsoft menejeri Brayan Garri tomonidan hisobot taqdim etildi.

Visual Studio (ichki versiya raqami-10, kod nomi-Dev10) dagi eng muhim yangilik qo‘llab-quvvatlanadigan tillarining to‘plamida boshqa f# funksional tilning paydo bo‘lishi. Til muallifi Microsoft Research, Kembrij, buyuk Britaniyadan don Syme hisoblanadi. .net Framework va Visual Studioda F# tili sof funksional dasturlash imkoniyatlarini obyektga yo‘naltirilgan dasturlash, parallel dasturlash va boshqa barcha xususiyatlar bilan integratsiya vositalari bilan birlashtiradi. Bu, ayniqsa, yosh dasturchilar orasida ko‘plab tarafdorlarni topgan juda istiqbolli tildir.

Dastlab, Visual Studio 2010 Phoenix mahsulotini va unga asoslangan optimallashtiruvchi kompilyatorni Visual C++ tilini o‘z ichiga olishi rejalashtirilgan bo‘lib, hosil qilingan ikkilik kodni bajarishda Visual C++kompilyatorining mavjud, avvalgi versiyasidan o‘zib ketishi kerak edi. Afsuski bu rejalar amalga oshirilmadi - Feniks jamoasi Visual Studio loyihasi boshqaruvi tomonidan belgilangan muddatlarga javob bermadi.

Visual Studio 2010 Ultimate -"maksimal" versiyada UML diagrammalar yordamida loyiha tuzilishini aks ettirish uchun vositalari, shuningdek, qulay skipped test guruhlar, loyiha manba kodi o‘zgartirish ta‘sirini baholash uchun Test ta‘sir tahlil komponent bor, chunki keraksiz o‘tish test oldini olish imkonini beradi. Bundan tashqari, debug tarixi va uning to‘liq holatini eslash imkonini beradi. Intellitrace bir debugger bor, shu jumladan multithreading tarixi.

Visual Studio 2010 bulut hisoblashni qo‘llab-quvvatlaydi. Buning uchun Microsoft Azure loyihalarining yangi turlari qo‘shildi. Ularni ishlatish uchun Visual Studio 2010-Microsoft Azure SDK uchun maxsus vosita o‘rnatish kerak.

Visual Studio 2012. 2012-yil avgust oyida Visual Studio 2012 versiyasi chiqarildi. Bu foydalanuvchi interfeysi yaxshilandi, yangi loyiha tomoshabin vositasi ishlab chiqilgan va kod fragmentlarini ta‘kidlash uchun ranglarni qo‘llash yaxshilandi. Microsoft ta‘kidlaganidek, Visual Studio 2012 uchun manba kodining umumiy hajmi 50 millionga etdi.

Shunday qilib, Microsoft Visual Studio integrated development muhiti faol innovasion rivojlanishning noyob namunasidir va dasturiy ta‘minotni ishlab chiqish sohasida yangi g‘oyalar va vositalarning bir butun birlashuvini o‘zida mujassamlashtiradi. 1995 yildan buyon 25 yil mobaynida 19 ta versiya chiqarilgan bo‘lib, ularning har birida yangi dasturlash tillari, buyruq-satr dasturlarini ishlab chiqish vositalari va boshqa ko‘plab yangiliklar mavjud.

Ba‘zi bir kalit so‘zlarni bilib olish lozim. Bu kalit so‘zlar 9.1 jadvalda keltirilgan

9.1-jadval. Kalit so‘zlar.

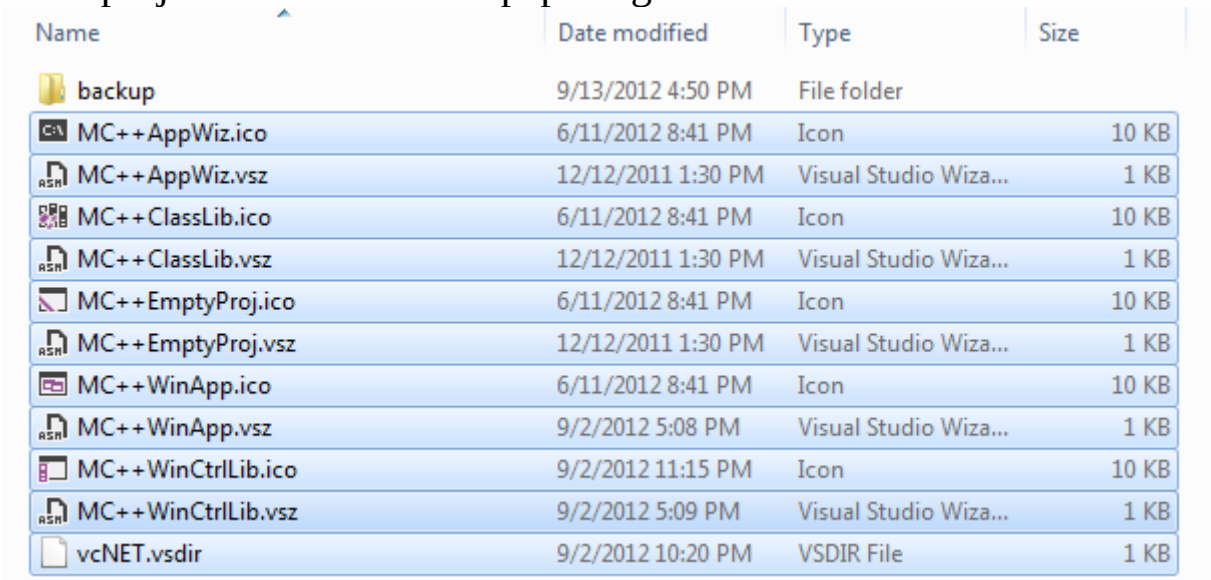
.NET	-ISO xalqaro standartlari bilan standartlashtirilgan va bir necha implementations ega bo‘lgan obyektga yo‘naltirilgan ko‘p tilli dasturlash platforma, - Microsoft.net, Microsoft tomonidan ishlab chiqilgan
C#	-Microsoft kompaniyasi tomonidan ishlab chiqilgan va faol ishlab chiqilgan eng yangi obyektga yo‘naltirilgan dasturlash tili. NET platformasi uchun asosiy dasturlash tili
Visual Basic	- Microsoft tomonidan ishlab chiqilgan va ishlab chiqilgan asosiy tilning obyektga yo‘naltirilgan kengaytmali dasturlash tili
Visual C++	- dasturlash tili, Microsoft tomonidan ishlab chiqilgan va ishlab chiqilgan C++ tilining kengaytmasi
Visual FoxPro	-Microsoft tomonidan ishlab chiqilgan obyektga yo‘naltirilgan dasturlash tillaridan biri, SQL so‘rovlar tili bilan mos ma’lumotlar bazalariga kirish uchun vositalarni o‘z ichiga olgan
Visual Studio	- Microsoft kompleks rivojlantirilgan integrallashgan muhit.

Visual C++ da Windows ilovalarni yaratish. Windows ilovalarni yaratishga kirishishdan oldin muhitni o‘rnatish va uning imkoniyatlariga to‘xtalib o‘tamiz. Visual Studio standart o‘rnatish asosida C++ tilini rivojlantirish uchun zarur elementlarni qo‘shish/o‘chirish imkonini

beradi. Masalan, an‘anaviy C++dasturlarini ishlab chiqish, C++da mobil ilovalarni ishlab chiqish, C++da Linux muhiti uchun vositalarni ishlab chiqish, C++tilida o‘yin ishlab chiqish va boshqa yana bir qator imkoniyatlarni beradi.

Visual C++ da Windows ilovalarni yaratish uchun avvalo, MS Visual Studio ni o‘rnatish kerak. Kompyuterning texnik imkoniyatlaridan kelib chiqqan holda, mos versiya tanlanadi. Mazkur versiyani o‘rnatish internet manbalarida juda ko’p uchraydi. Odatda oddiy muhitlar kabi o‘rnatiladi.

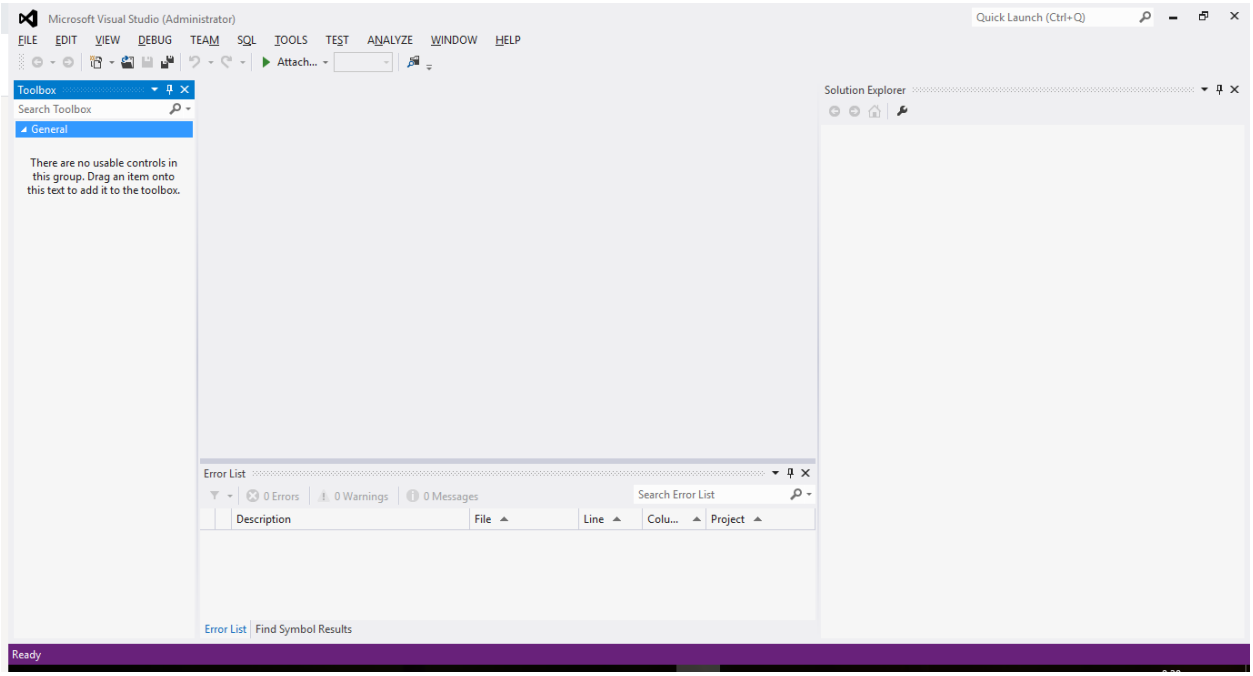
MS Visual Studio ning hamma versiyalarida Windows ilovalarni turlicha yaratish mumkin. Ammo eng qulay va umumiy foydalanuvchi uchun mos bo‘lgan interfeysni o‘rnatish uchun har xil versiyalarga har xil sozlashlarni bajarish kerak. Masalan, MS Visual Studio 2012 da Windows ilovalarni yaratish uchun, Windows oynasi tayyor komponentani o‘rnatish uchun, 9.1-rasmda keltirilgan fayllarni VCvcprojectsvcNET/ VC++ papkasiga nusxalash lozim.



Name	Date modified	Type	Size
backup	9/13/2012 4:50 PM	File folder	
MC++AppWiz.ico	6/11/2012 8:41 PM	Icon	10 KB
MC++AppWiz.vsz	12/12/2011 1:30 PM	Visual Studio Wiza...	1 KB
MC++ClassLib.ico	6/11/2012 8:41 PM	Icon	10 KB
MC++ClassLib.vsz	12/12/2011 1:30 PM	Visual Studio Wiza...	1 KB
MC++EmptyProj.ico	6/11/2012 8:41 PM	Icon	10 KB
MC++EmptyProj.vsz	12/12/2011 1:30 PM	Visual Studio Wiza...	1 KB
MC++WinApp.ico	6/11/2012 8:41 PM	Icon	10 KB
MC++WinApp.vsz	9/2/2012 5:08 PM	Visual Studio Wiza...	1 KB
MC++WinCtrlLib.ico	9/2/2012 11:15 PM	Icon	10 KB
MC++WinCtrlLib.vsz	9/2/2012 5:09 PM	Visual Studio Wiza...	1 KB
vcNET.vsdir	9/2/2012 10:20 PM	VSDIR File	1 KB

9.1-rasm. Windows ilovalarni yaratish uchun zaruriy fayllar

Shundan so‘ng, MS Visual Studio ni ishga tushiring, 9.2-rasmdagidek, integrallashgan muhit oynasi hosil bo‘ladi.



9.2-rasm. Integrallashgan muhit oynasi.

Muhit oynasi 5 qismdan iborat. Ular quyidagilar:

1. Sarlavha satri. Bunda tizimli menyu (chap tomondagi tizim belgisi), tizim va joriy proyekt nomi, tizimli tugmachalar (yig‘ishtirish, oyna o‘lchamini o‘zgartirish, oynani yopish)
2. Muhitning menyusi.
3. Muhitning uskunalar paleni (foydalanuvchi tomonidan kerakli uskunalar bilan to‘ldirilishi mumkin).
4. Muhitning ishchi maydoni (Foydalanuvchining joriy proyekt uchun zarur komponentalar va ularni boshqarish, kuzatish imkoniyatlari mavjud)
5. Muhitning holat satri.

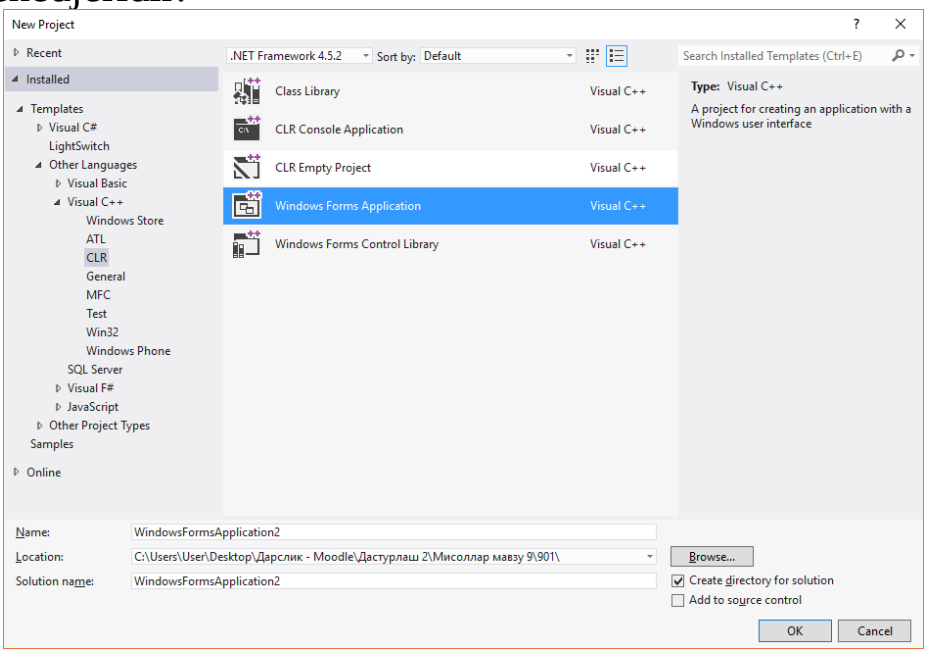
Yangi proyekt yaratilishi bilan proyektga mos kerakli va joriy qilingan komponentalar faollashadi. Windows ilovalarni yaratish uchun bir nechta varinatlar mavjud. Ularning eng ko‘p tarqalgan varianti menyudan foydalanib yaratishdir.

Windows ilovalarni yaratish uchun [File → new → Project] ketma ketligi yoki [Ctrl+Shift+N] tugmachalar majmuasini bosish yetarli. Ekranga [new Project] nomli muloqot oynasi hosil bo‘ladi (9.3-rasmga qarang).

Birinchi marta MS Visual Studio ishga tushirganda asosiy tilni tanlashni so‘raydi. Agar C++ tilini tanlagan bo‘lsangiz, avtomatik shu tilda yaratilishi mumkin bo‘lgan loyihalar ro‘yxati hosil bo‘ladi. Agar oldin boshqa tilni tanlagan bo‘lsangiz, [Other Languages] bandidan Visual C++ tilini tanlashingiz lozim. Visual C++ til asosida yaratish

mumkin bo‘lgan konseptual loyihalar ro‘yxati chiqadi va bularning o‘zi bir nechta loyihalarga bo‘lnadi (9.3-rasmning chap tomonida qarang).

Visual C++ da Windows ilovalarni yaratish uchun avvalo [CLR] loyiha menedjeri tanlanadi. CLR - Common Language Runtime - bir necha qo‘llab-quvvatlanadigan tillarda har qanday yozilgan dasturlar ijrosini boshqaradi dasturlash bo‘lib, ularni tillari har qanday yozilgan umumiy obyektga yo‘naltirilgan sinflarga almashish imkonini beruvchi loyiha menedjeridir.



9.3-rasm. Yangi loyiha yaratish oynasi.

[CLR] loyiha menedjeri bir nechta loyihalar yaratish imkonini beradi:

1. Class Library – boshqa proyektlarga foydalanish uchun CLR sinflarni yaratish imkonini beradi.
2. CLR Console Application – CLR asosida konsol loyiha yaratish uchun ishlatiladi.
3. CLR Empty Project - yangi hech nima yozilmagan loyihalar asosida lokal loyihalarni yaratish uchun ishlatiladi.
4. Windows Forms application – Windows oynali loyihalarni yaratish imkonini beradi.
5. Windows Forms Control Library - Windows ilovalarni boshqarish uchun foydalaniladigan kutubxonalarni yaratish imkonini beradi.

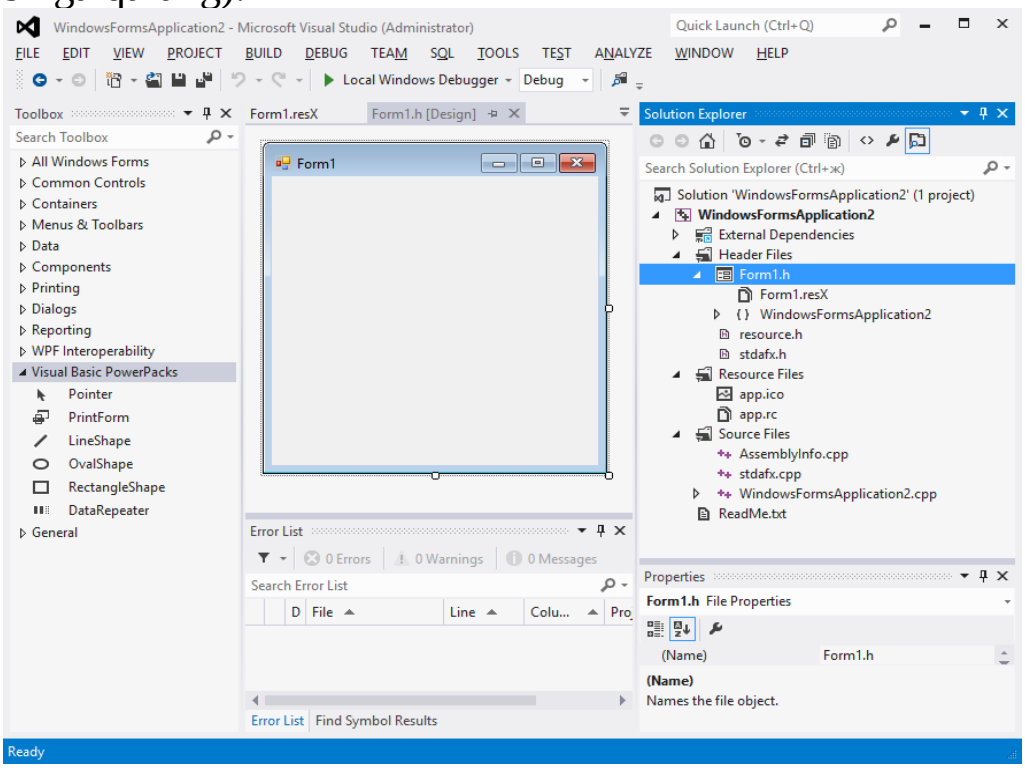
Har bir loyihani o‘z o‘rnida yaratish foydalanish muzimdir va har birini tanlaganingizda ularning izohi oynaning o‘ng tomonida chiqadi.

[CLR] loyiha menedjeridan foydalanib loyiha yaratishda .NET Framework ning varianti ham muhim rol o‘ynadi. Shuning uchun .NET Framework larning imkoniyatlari bilan oldindan yaxshilab tanishib

chiqish lozim. Har bir .NET Framework ning o‘zining imkoniyatlari bor va shuningdek Visual C++ da Windows ilovalarni yaratish uchun .NET Framework 4.5 yoki 4.5.2 variantlaridan foydalanishni tavsiya qilamiz. .NET Framework ni tanlash loyiha turini tanlash oynasining ustida joylashgan.

Loyiha yaratish oynasinig pastki qismida loyiha nomi, joylashgan joyi, loyiha ishchi papkasining nomi ko‘rsatiladi.

Kerakli amallarni bajargandan so‘ng [ok] tugmasini bossangiz, bir nechta avtomatik sozlashdan so‘ng loyihaning ilk ko‘rinishi hosil bo‘ladi (9.4-rasmga qarang).



9.4-rasm. Oynali loyihaning dastlabki ko‘rinishi.

Oldin aytib o‘tilgandek, muhitning ishchi oynasini o‘zgardi. Uni odatda foydalanuvchi o‘zining xoxishi bo‘yicha joylashtirishi mumkin.

Eng aosiy elementlardan biri bu – Solition Explorer bo‘lib hisoblanadi. Bu loyiha papkasini boshqarish uchun qulay qilib yaratilgan. Bitta Ishchi papkada bir nechta loyiha bo‘lishi mumkin. Shuning uchun, u yerdan loyiha nomini tanlash kerak bo‘ladi va loyiha uchunn kerak bo‘limlar ro‘yxati chiqadi:

1. External Dependencis – tashqi, tizimli kerakli kutubxona, sinflar joylashadi.
2. Header Files – kutubxona fayllari.
3. Resource Files – loyiha resursi fayllari.
4. Source Files – dasturning manba kodlari yozilgan fayllar.

5. ReadMe – matnli fayl bo‘lib, unda loyiha uchun izohlarni yozib qoldirishingiz mumkin.

Standart loyiha yaratilganda yaratidgan joriy fayllarni o‘zgartirmang, bo‘lmasa loyiha ishlamasligi mumkin.

Odatda juda ko‘p foydalanuvchilar loyihalarda vizual ishlashni yoqtiradilar, buning uchun kutubxona fayllari majmuasidan *.h kengaytmali fayllardan foydalaniladi. Uni bosganda loyihaning kerakli oynasi vizual ko‘rinishda hosil bo‘ladi.

Solition Explorer dan kerakli faylni tanlab, ustiga sichqonchani bossangiz shu faylni tahrirlash imkoniyati hosil bo‘ladi. Agar oldindan nomlar fazosi, sinflar, shablonlar bilan ishlagan bo‘lsangiz, ularning ham algoritmlari va funksiyalari bilan tanishish mumkin. Buning uchun kerakli chekni bosish kerak bo‘ladi.

Solition Explorer dan kerakli obyekt (fayl) ni tanlaganigizda uning xususiyatlari (properties)va funksiyalari (events) larni sozlash mumkin. Buning uchun dasturchiga properties oynasi yordam beradi. Obyektga mos holda ishlaydi.

Ishchi maydonning chap tomonida Solition Explorer dan tanlangan obyekt uchun o’rnatish, integrallashtirish mumkin bo‘lgan uskunalar, metodlar va boshqa zaruriy xarakterga ega elementlar hosil bo‘ladi.

Ishchi maydonning aosiy markazida loyihani tahrirlash ishlari bajariladi va uning pastki qismida xatolar, xabarlar, ogohlantirishlar va boshqa dasturchi uchun kerakli barcha yordamchi oynalarni joylashtirish mumkin. Shuni bilingki, ishchi oynani dasturchi o‘ziga moslab tayyorlab shablon ko‘rinishida qilib olishi mumkin.

Muhitda menyular va uskunalar paneli. MS Visual Studio muhitining oynasi boshqa dasturlarning menyulari kabi joylashgan bo‘lib, 13 tadan iborat. Uskunalar paleni dasturchining xoxishi bilan almashib turishi mumkin. Shuning uchun faqat menyularning vazifasi va buyruqlariga to‘xtalamiz. Odatda menyuga sichqoncha bilan murojaat qilinadi. Agar sichqoncha ishlamay qolsa, [Alt] tugmasini bosganda, menyu nomlariga tagiga chizilgan harf paydo bo‘ladi. [Alt] tugmasini shu harf bilan bossangiz menyuga murojaat qilasiz.

[FILE] – muhitda loyiha ustida global hodisalarni bajarish uchun mo‘ljallangan asosiy menyulardan hisoblanadi. Unda quyidagi asosiy buyruqlar bor: yangi loyiha yaratish, loyihani ochish, loyiha biror bir yangi loyiha qo‘shish, faylni yopish, loyihani yopish, loyiha faylini va

loyihani to‘liq saqlash, boshqa loyihalarni improt qilish, oldin ishlatilgan loyiha fayllari, muhitdan chiqish.

[EDIT] - muhitda tahrirlash ishalarini bajarishga mo‘ljallangan.

[VIEW] – muhitni ko‘rnishi, uskunalarni joylashtirish, keraklisiga murojaat qilishga mo‘ljallangan.

[PROJECT] – loyiha uchun elementlarni qo‘shish, boshqarish, sozlamalarni tayyorlash uchun ishlatiladi.

[BUILD] – loyihani qurish, qayta qurish, tozalash, parametrlarini sozlash va boshqa ishlarga mo‘ljallangan.

[DEBUG] – loyihani tekshirish, qadamba- qadam tekshirish, ishlatish, ishlashini tekshirish kabi amallarni bajarish uchun mo‘ljallangan.

[TEAM] – serverlarga ulanish uchun mo‘ljallangan

[SQL] – ma’lumotlar bazasini boshqarishning lingvistik ta‘minoti bilan ishlashga qaratilgan.

[FORMAT] – muhitda turli formatlash ishlarini bajarishga mo‘ljallangan buyruqlari bor.

[TOOLS] – muhitda turli sozlamalarni amalga oshirish uchun qaratilgan menyu.

[TEST] – loyiha testlar va hisobotlarni olish uchun buyruqlar majmuasi joylashgan menyu.

[ANALYZE] – loiyhaning elementlarining o‘zaro integrallashgan holatda ishlashini tahlil qilish buyruqlari

[WINDOW] – muhitda oynalarni joylashtirish, boshqarish buyruqlari

[HELP] – muhit bo‘yicha yordam, turli rasmiy yordam muhitlaarga chiqish imkoniyatlarini beradigan buyruqlar joylashgan.

Buyruqlar va uskunalar paneli. Menyular va uskunalar paneli foydalanuvchilar vspackage buyruqlarga kirish uchun bir yo‘ldir. Buyruqlar-xujjatni chop etish, ko‘rinishni yangilash yoki yangi fayl yaratish kabi vazifalarni bajaruvchi funksiyalardir. Menyular va uskunalar paneli foydalanuvchilarga buyruqlarni taqdim etishning qulay grafik usullariga ega. Odatda, tegishli buyruqlar bir xil Menyuda yoki bir xil uskunalar panelida guruhlanadi.

Menyular odatda integrallashgan muhiti (IDE) yoki uskunalar oynasining yuqori qismida ketma-ket guruhlangan bitta so‘zli satrlar sifatida ko‘rsatiladi. Sichqonchaning o‘ng tugmasini bosganda menyuni ham ko‘rsatish mumkin — bu holda ular kontekst menyular deb ataladi. Bosgandan so‘ng menyu ochiladi va bir yoki bir necha buyruqlarni

koʻrsatadi. Biror buyruq bosilgandan soʻng vazifalar bajarilishi yoki qoʻshimcha menyu osti buyruqlar bilan ochilishi mumkin. Baʻzi ommabop menyu nomlari fayl, koʻrish, va oyna, [Object window] bor.

Uskunalar paneli - Toolbars odatda tugmalar va boshqa elementlardan iborat satrlarda joylashgan, masalan, roʻyxatlar jamlanmasi va menyu controllers kabi. Uskunalar paneli boshqaruvining buyruq bilan bogʻliq barcha elementlarini taʻminlaydi. Uskunalar paneli tugmasini bosish, uga bogʻliq buyruq faollashtiradi. Uskunalar paneli tugmalari odatda chop etish buyrugʻi uchun printer kabi asosiy buyruqlarni taklif qiluvchi belgilarni oʻz ichiga oladi. Ochiladigan roʻyxatda roʻyxatdagi har bir element boshqa buyruq bilan bogʻliq. Menyu tekshiruvi - bu gibriddir, unda nazoratning bir qismi uskunalar paneli tugmasi va ikkinchisi bosilganda qoʻshimcha buyruqlarni koʻrsatadi.

Bir buyruq yaratishda, buning uchun bir hodisa yaratish kerak. Agar faollashtirilgan boʻlsa, buyruq aniq yoki yoqilgan boʻlsa hodisa bajarilishini bildiradi, uning matnini oʻzgartirish imkonini beradi va buyruqni tegishli javob bilan taʻminlaydi (yoʻnaltiriladi). Koʻp hollarda, IDE IOleCommandTarget interfeysi yordamida buyruqlar jarayonlari bajariladi. Visual Studio yoʻnalishidagi buyruqlar foydalanuvchi tanloviga asoslangan buyruqning ildiz kontekstidan boshlab va global tanlov asosida tashqi kontekstgacha ierarxiya sifatida koʻrsatiladi. Asosiy menyuga qoʻshilgan buyruqlar ssenariylarda foydalanish uchun qulay.

Yangi menyular va uskunalar majmuasini aniqlash uchun, Visual Studio buyruq jadvalda (.vst) fayl orqali ularni tasvirlash kerak. Visual Studio paketi Shablon uchun bu faylni yaratadi, shuningdek kerakli elementlar shablondan tanlangan har qanday buyruqlarni, uskunalar majmuasi va muharrirlarini qoʻllab-quvvatlashga kerak.

Visual Studio menyular va buyruqlar. Microsoft Officedan farqli oʻlaroq, bir qancha alohida mahsulotlar paketidan iborat. Visual Studio koʻp mahsulotlarni oʻz ichiga oladi, qaysiki har bir global Visual Studio IDE buyruqlar oʻz toʻplamiga hissa qoʻshadi. IDE kontekst asosida foydalanuvchiga mavjud funksiyalarni filtrlash orqali minglab buyruqlarning murakkabligini boshqaradi.

Loyihalash oynasida kodni tahrirlash oynasiga oʻtish kabi foydalanuvchi konteksti oʻzgarganda, yangi kontekst bilan bogʻliq boʻlmagan funksiyalar yoʻqoladi. Shu bilan bir vaqtda, yangi funksional tegishli dinamik ma’lumotlar bilan birga ochiladi, bunday Toolbox

xususiyatlari va parametrlari sifatida qaraladi. Foydalanuvchi mavjud buyruqlar to‘plamini almashtirishni sezmasligi kerak. Joriy foydalanuvchi konteksti har doim IDE sarlavhasida, xususiyatlar oynasida yoki sahifalari muloqot oynasida bo‘lgani kabi bir yoki bir necha usulda ko‘rsatiladi.

Buyruq to‘plamlari ularga moslashuvchanligini ta‘minlaydi. Visual Studio muhitiga xos bo‘lgan yagona buyruq strukturalari asosiy menyu va asosiy buyruqlar satri bo‘lib, ularni sozlash va hatto yashirish mumkin. Boshqa buyruqlar majmuasi ilovaning holatiga qarab paydo bo‘ladi va yo‘qoladi. Oynalar uskunasi va hujjat muharrirlari, shuningdek, ajralmas oynalar uskunalar majmuasini o‘z ichiga olishi mumkin.

menyular qachon mavjud bo‘lsa, birgalikda buyruqlar va buyruq guruhlaridan foydalaning.

Buyruqlar odatda kontekst asosida namoyish etiladi, chunki, mavjud menyular va buyruq guruhlar yordamida buyruq tuzilishi kontekstida o‘zgarishlar o‘rtasida nisbatan barqaror bo‘lib qolishini ta‘minlaydi. Buyruqlar guruhini qayta ishlatish va tegishli buyruqlar yaqinida yangi buyruqlarni joylashtirish IDE murakkabligini kamaytiradi va yanada samimiy tajriba yaratadi. Agar yangi buyruq aniqlash kerak bo‘lsa, mavjud buyruq guruhiga uni qo‘yib ko‘rish mumkin. Agar yangi guruh aniqlash kerak bo‘lsa, yangi yuqori darajali menyuni yaratish va bog‘liq buyruq guruhi yonida menyusida joylashtirish mumkin.

Har bir buyruq uchun piktogramma yaratish yaxshi emas. Buyruq belgisini yaratishni diqqat bilan o‘ylab ko‘ring. Piktogramma faqat quyidagi buyruqlar uchun yaratilishi kerak:

* joriy uskunalar panelida ko‘rsatiladiganlar.

* foydalanuvchilar tomonidan uskunalar paneliga qo‘shish, maxsus muloqot orqali mumkin.

* Microsoftning mahsuloti bir xil harakat bilan bog‘liq buyruqlarga.

Klaviatura tugmalarini qo‘shishni cheklash. Foydalanuvchilarning aksariyati barcha mavjud tugmalarning kichik qismini ishlatadi. Agar shubhangiz bo‘lsa, funksiyani klaviatura tugmasi bilan bog‘lamang. Yangi yorliqlarni qo‘shishdan oldin foydalanuvchilar guruhi bilan ishlash.

Komandalarga standart menyu joyini berish. Jamoalar boshqalar tomonidan tuzilgan va shunga ko‘ra mo‘ljallangan bo‘ladi. Yashirin

buyruq degan narsa yo‘q. Barcha Visual Studio buyruqlari Tools > configure muloqot oynasida, buyruq oynasida, avtomatik yakunlash, > > options > klaviatura muloqot oynasida va development tools muhitida (DTE) ko‘rsatiladi. Buyruqlar nomini va vositalari majmuini berishga ishonch hosil qilish kerak va foydalanuvchilar osongina ularni topish mumkin, shunday qilib, .ctc fayl yaratish kerak.

Ichki uskunalar panelidagi umumiy buyruqlarni takrorlamang. Buyruqlarni foydalanuvchining diqqat markaziga yaqin joyda joylashtirish foydalidir. Buning bir yo‘li uskuna oynasi yoki hujjat muharririning yuqori qismida o‘rnatilgan uskunalar panelini yaratishdir. Uskunalar paneliga joylashtirilgan buyruqlar oynadagi kontent maydoniga xos bo‘lishi kerak. Bu uskuna majmuasining umumiy buyruqlarni nusxa qilmang. Masalan, ajralmas uskunalar panelida saqlash belgisini joylashtirish kerak emas.

MS Visual Studio muhitining interaktiv tugmalari.

Asosiy tugmalar

Ctrl+Shift+P, F1	Buyruqlar palitrasini ko‘rsatish
Ctrl+P	Tez ochish, faylga o‘tish…
Ctrl+Shift+N	Yangi oyna yaratish
Ctrl+Shift+W	Oynani yopish
Ctrl+,	Foydalanuvchi sozlamalari
Ctrl+K Ctrl+S	Klaviatura yorliqlar

Asosiy tahrirlash tugmalari

Ctrl+X	Buferga olish va o‘chirish
Ctrl+C	Buferdan nusxa joylashtirish
Alt+ ↑ / ↓	qatorni yuqoriga / pastga ko‘chirish
Shift+Alt + ↓ / ↑	Qator nusxasini yuqoriga / pastga ko‘chirish
Ctrl+Shift+K	Qatorni o‘chirish
Ctrl+Enter	Yangi qator joylashtirish
Ctrl+Shift+Enter	Yuqoridan yangi qator joylashtirish
Ctrl+Shift+	Mos keladigan qavsga o‘tish
Ctrl+] / [	Qatorga abzas qo‘shish/o‘chirish
Home / End	Satr boshi / oxiriga o‘tish
Ctrl+Home	Fayl boshiga o‘tish
Ctrl+End	Fayl oxiriga o‘tish
Ctrl+↑ / ↓	qatordan yuqoriga / pastga harakatlashtirish
Alt+PgUp / PgDn	sahifa bo‘yicha yuqoriga / pastga harakatlashtirish

Ctrl+Shift+[	Operator ta‘sir doirasini belgilash
Ctrl+Shift+]	Operator ta‘sir doirasini belgilashni bekor qilish
Ctrl+K Ctrl+[	Ta‘sir doirani ochish
Ctrl+K Ctrl+]	Ta‘sir doirani yopish
Ctrl+K Ctrl+0	Qatorlarga izoh qo‘yish
Ctrl+K Ctrl+J	Qatorlardan izohni o‘chirish
Ctrl+K Ctrl+C	Satrga izoh qo‘shish
Ctrl+K Ctrl+U	Satrdan izoh o‘chirish
Ctrl+ Z	Amalni bekor qilish

Navigatsiya tugmalari

Ctrl+G	Kerakli qatorga borish
F8	Navbatdagi xatoga borish
Shift+F8	Navbatdagi xato va xabarga borish
Ctrl+Shift+Tab	Tahrirlash oynalariga o‘tish
Ctrl+M	Fokus tugunlarni o‘zgartirish

Izlash va o‘zgartirish tugmalari

Ctrl+F	Izlash
Ctrl+H	O‘zgartirish
F3 / Shift+F3	Izlashning keyingisiga o‘tish
Alt+Enter	Izlash natijalarini belgilash

Dasturlashga ko‘maklashuvchi tugmalar

Ctrl+Space	Tegishli funksiyalarni chaqirish
Ctrl+Shift+Space	Tegishli funksiya variantlarini chaqirish
Shift+Alt+F	Dasturni formatlash
Ctrl+E Ctrl+F	Dastur fragmentini formatlash
Ctrl+Shift +.	Dastur matnini kattalashtirish
Ctrl+Shift +,	Dastur matnini kichiklashtirish

Bu interaktiv tugmachalardan tashqari o‘rganib olishingiz kerak bo‘lgan tugmachalar juda ko‘p, ularni dasturlash davomida foydalanishga o‘rganib olasiz degan umiddamiz.

Form xususiyatlari va hodisalari. Form bu loyiha yaratilgan hosil bo‘ladigan birinchi oyna hisoblanadi. Oynaning joriy holatlari mavjud. Ularni loyiha boshqaruvchisi panelidan ko‘rish mumkin. Oyna

uchun yangi form1 sinfi yaratiladi. Bu sinf form sinfining merosxo‘ri hisoblanadi. U quyidagicha aniqlangan:

```
public ref class Form1 : public System::Windows::Forms::Form {}
```

Shu sinfning ichida InitializeComponent funksiyasi bor va u formani yaratilishi javob beradi. Uning dastlabki sozlamalari quyidagi fragmentda keltirilgan:

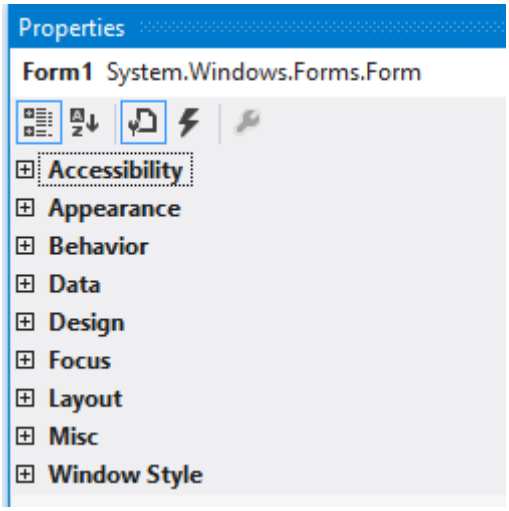
```
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(284, 261);
this->Name = L"Form1";
this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;
this->Text = L"Form1";
this->ResumeLayout(false);
```

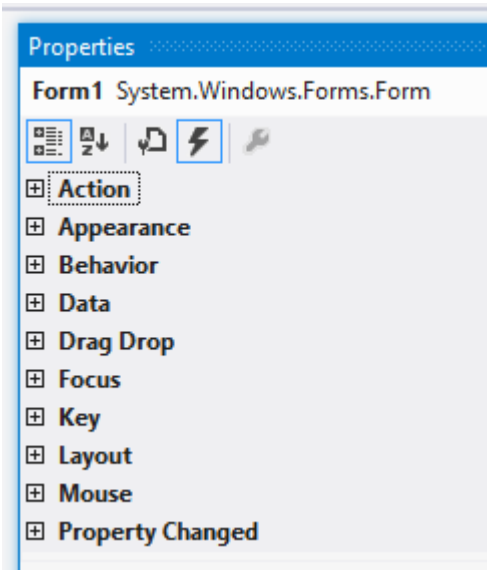
Imkoniyat qadar bu dastur fragmentlariga o‘zgartirish kiritish shart emas, bularni hammasini vizual bajarish maqsadga muvofiq.

Vizual bajarish deganda formaning vizual loyihasi bilan ishlash kerak. Loyiha yaratuvchisi uchun quyidagilar muhim hisoblanadi:

- Formani ustiga sichqonchani bossangiz u aktivlashadi.
- Kontekst menyudan foydalanib, uning dastur fragmentiga o‘tish mumkin (F7 tugmasi orqali ham)
- Dastur fragmentidan forma loyihasiga o‘tish uchun kontekst menyudan foydalanish mumkin (Shirft + F7 tugmasi orqali ham)
- Kontekst menyu orqali formani boshqarishni qulflab qo‘yish va ochish mumkin (lock/unlock controls)
- Kontekst menyu orqali formani xususiyatlari va hodisalariga o‘tish mumkin (Properties)

Formaning xususiyatlari va hodisalar quyidagi bo‘limlardan iborat (9.5 va 9.6-rasmlarga qarang)





9.5-rasm. Formaning xususiyatlari 9.6-rasm. Formaning hodisalari

Properties muloqot oynasida 4 ta uskuna mavjud. Ularning vazifalari:

- forma parametrlarini guruhlab saralashni ta‘minlaydi.
- forma parametrlarini alfavit bo‘yicha saralashni ta‘minlaydi.
- forma xususiyatlari ro‘yxati.
- forma hodisalari ro‘yxati.

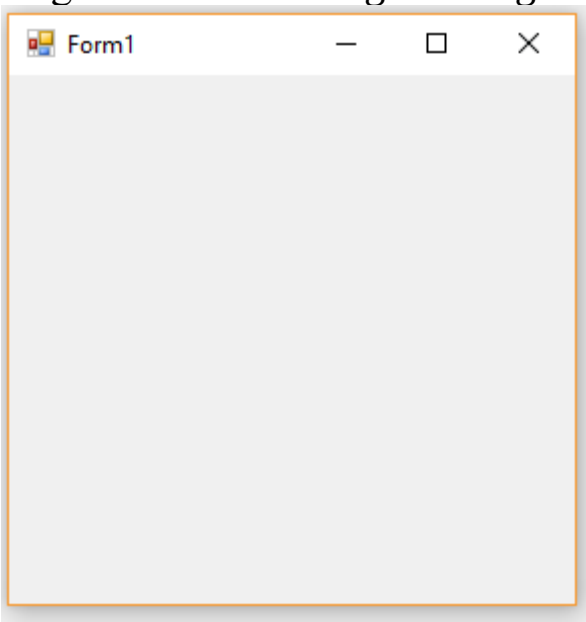
E‘tibor bergan bo‘lsangiz, tanlangan joriy tur ko‘k to‘rtburchakka olingan bo‘ladi. Bu erada bajarilgan har bir ish formaga mos sinfga o‘zgartirib boriladi. Endi forma xususiyatlarining ba‘zilari bilan tanishib chiqamiz.

1	**Accessibility guruhi**	Buning 3 ta xususiyati bor, bu xususiyatlari bir forma bilan o‘zgarishini aniqlab bo‘lmaydi shuning uchun keyinroq to‘xtalamiz.
2	**Appearance guruhi**	Formaning tashqi ko‘rinishi uchun ishlatiladi va 11 ta xususiyat bor
	BackColor	Formaga rang o‘rnatish
	BackgroundImage	Rasm o‘rnatish
	BackgroundImageLayout	Rasm o‘rnatish tartibi
	Cursor	Kerakli kursor belgisini tanlash
	Font	Formada yozuv xususiyatlarini o‘rnatish
	ForeColor	Yozuv rangini tanlash
	FormBorderStyle	Formaning stilini o‘rnatish
	RightToLeft	Forma sarlavhasini o‘ngdan chapga

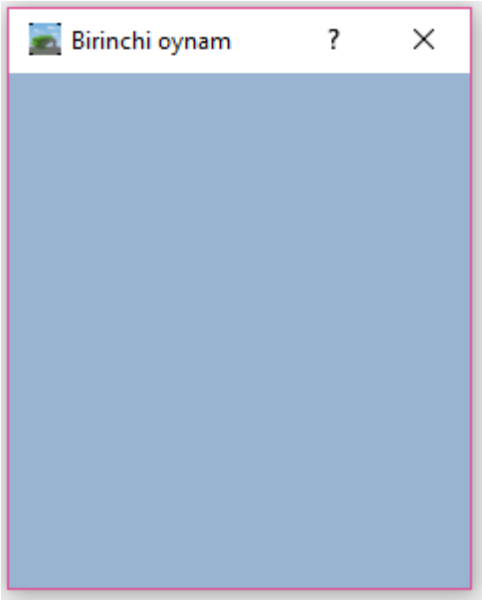
		o‘rnatish
	RightToLeftLayout	Forma sarlavhasini o‘ngdan chapga o‘rnatish tartibi
	Text	Sarlavha nomi
	UseWaitCursor	Kutish kursoridan foydalanishni o‘rnatish
3	**Behavior guruhi**	Formaning rejimlarini o‘rnatish va 6 ta xususiyati bor
	ContextMenuStrip	Kontekst menyu o‘rnatish
	DoubleBuffered	Bufer o‘rnatish
	Enabled	Formani yoqish va o‘chirish
4	**Data guruhi**	Formani ichki va tashqi ma’lumotlar bilan bog‘lash uchun ishlatiladi va 3 xususityati bor. Bu guruhni keyingi elementlari bilan ishlaganda batavsil qaraymiz
5	**Design guruhi**	Formani loyihalash uchun ishlatiladi va 3 ta xususiyati bor
	Language	Formaga joriy tilni o‘rnatish
	Localizable	Formaga lokalizatsiyani o‘rnatish
	locked	Formani yopish yoki ochish
6	**Focus guruhi**	Formada fokuslarni boshqarish uchun ishlatiladi va bitta xususiyati bor
	CausesValidation	Fokuslarni tekshirish bosqichlari
7	**Layout guruhi**	Formani tartiblash uchun ishlatiladi va 13 ta xususiyati bor
	AutoScroll	Avtomatik formani kuzatish tugmalarini o‘rnatish
	AutoSize	Avtomatik o‘lchamni joriy qilishni o‘rnatish
	Lacation	Formani hosil bo‘lish joyini belgilashni o‘rnatish
	MaximumSize	Avtomatik katta holda chiqish
	MimimumSize	Avtomatik kichiklashtrilgan holda chiqish
	Size	Formaga o‘lcham o‘rnatish
	StartPosition	Formaning chiqish joyini o‘rnatish
	WindowsState	Formaning turini belgalash
8	**Misc guruhi**	Boshqa xususiyatlarni o‘rnatish, formani turiga qarab o‘zgarib turadi va 3 ta xususiyati bor
9	**Windows Style guruhi**	Oyna stillarini o‘rnatish uchun

	mo‘ljallangan guruh bo‘lib, 13 ta xususiyatga ega.
ControlBox	Boshqarish tugmalarini joylashtirishni aniqlash
HelpButton	Yordam chaqiruvchi tugmani o‘rnatish
Icon	Oyna belgisini o‘rnatish
MaximizeBox	Kattalashtirish tugmasini o‘rnatish
MinimizeBox	Kichiklashtirish tugmasini o‘rnatish
Opacity	Oynaning tiniqligini sozlashni o‘rnatish

Mazkur xususiyatlardan foydalanib, formaning ba‘zi xususityalarini o‘zgartiramiz va uning sinfning ichiga kirib ko‘rishimiz mumkin:



9.7-rasm. Formaning dastlabki ko‘rinishi



9.8-rasm. Formaning ishlov berilgan ko‘rinishi

Formaga ishlov berilgan xususiyatlari aniqlash uchun sinfiga murojaat qilish mumkin. Quyidagi dastur fragmentlarini olamiz.

Formaning dastlabki o‘rnatilgan xususiyatlari	Formaning ishlov berilgan xususiyatlari	+/-
`this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);`	`this->AutoScaleDimensions = System::Drawing::SizeF( 11, 22);`	+
`this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;`	`this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;`	-

	this->BackColor = System::Drawing::SystemColors::ActiveCaption;	+
this->ClientSize = System::Drawing::Size(284, 261);	this->ClientSize = System::Drawing::Size(230, 257);	+
	this->Cursor = System::Windows::Forms::Cursors::Hand;	+
	this->Font = (gcnew System::Drawing::Font(L"Times new Roman", 14.25F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point, static_cast<System::Byte>(204)));	+
	this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::Fixed3D;	+
	this->HelpButton = true;	+
	this->Icon = (cli::safe_cast<System::Drawing::Icon^ >(resources->GetObject(L"$this.Icon")));	+
	this->Margin = System::Windows::Forms::Padding(6, 6, 6, 6);	+
	this->MaximizeBox = false;	+

	this->MinimizeBox = false;	+
this->Name = L"Form1";	this->Name = L"Form1";	-
this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;	this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;	-
this->Text = L"Form1";	this->Text = L"Birinchi oynam";	+
this->ResumeLayout(false);	this->ResumeLayout(false);	-

Birinchi navbatda muhitning o‘zi ushbu xususiyatlarni o‘rnatishni va tariblashni nazorat qiladi. Formada quyidagi o‘zgarishlar amalga oshirilgan: formaning chiqish maydoni, rangi, joriy o‘lchamlari, oyna kursori, oynaning yozuv turi, ko‘rinishi, yordamchi tugma o‘rnatilgan, ikonkasi o‘zgartirilshan, holati o‘zgartirilgan, kattalashtirish va kichiklashtirish tugmalari olib tashlangan, oynaning sarlavhasi o‘zgargan.

Bularni amalga oshirish uchun 2-3 munut vaqt ketadi va hammasi vizual bajariladi. Bunda oldin qilingan o‘zgarishlar o‘chirib, oxirgi o‘zgarishlar qo‘shib boriladi.

Formaning hodisalari bilan tanishib chiqamiz. O‘zi hodisa nima? Hodisa bu bir vazifa bajarilgan jarayon bo‘lishi mumkin, masalan, formani yaratish uchun 3 ta hodisa bo‘lishi mumkin, forma yaratilishidan oldin, forma yaratilayotgan vaqt, forma yaratilib bo‘lingandan keyin.

Ularning ham 9.6- rasmda keltirilgan 10 ta guruhi bor. Ularning ba‘zilariga izoh berib o‘tamiz

1	**Action guruhi**	Global hodisalar guruhi bo‘lib, 8 ta hodisadan iborat
	Click	Formaga tugma bosilganda
	DoubleClick	Formada tugma 2 marta bosiganda
	MouseCaptureChanged	Sichqonchaning holati o‘zgarganda
	MouseClick	Sichqoncha bosilganda
	MouseDoubleClick	Sichqoncha ikki marta bosilganda
	ResizeBegin	Forma o‘lchami o‘zgarishi boshlanganda
	ResizeEnd	Forma o‘lchami o‘zgarishi tugaganda

2	**Appearance guruhi**	Formaning ichki hodisalari uchun ishlatiladi va bitta xususichti bor, u bilan keyingi mavzular tanishamiz. t
3	**Behavior guruhi**	Forma holatlari uchun hodisalarni o‘z ichiga oladi va 15 ta hodisadan iborat
	ControlAdded	Formaga biror boshqarish o‘rnatilganda
	ControlRemoved	Formadan biror boshqaruv o‘chirilganda
	FormClosed	Forma yopilganda
	FormClosing	Forma yopilishidan oldingi holatda
	HelpButtonClicked	Yordam tugmasi bosilganda
	InputLanguageChanged	Til o‘zgartirilganda
	InputLanguageChaning	Til o‘zgartirilayotganda
	Load	Forma yuklanayotganda
	Shown	Forma chqirilganda
4	**Data guruhi**	Ichki va tashqi ma’lumotlar bilan ishlash hodisalari bo‘lib, 2 ta hodisadan iborat, bu xususiyatlar ma’lumotlar bazasi bilan ishlaganda tushunarli bo‘ladi
5	**Drag drop guruhi**	Formada harakatlanish hodisalari bo‘lib, 6 ta hodisadan iborat
	DragDrop	Biror bir harakat bo‘lganda
	DragEnter	Inter tugmasi bosilganda
	DragLeave	To‘xtash harakati bo‘lganda
	DragOver	Ketish harakati bo‘lganda
	GiveFeedback	Teskari aloqa qo‘shish
6	**Focus guruhi**	Formada fokuslarni boshqarish uchun bo‘lib, 6 ta hodisani o‘z ichiga oladi
	Activated	Fokus aktiv bo‘lganda
	Enter	Fokusda ma’lumot kiritilganda
	Leave	Fokusda to‘xtash sodir bo‘lganda
	Validated	Tekshirish tugaganidan so‘ng
	Validating	Tekshirish vaqtida
7	**Key guruhi**	Tugmalar bilan ishlash hodisalari uchun bo‘lib, 6 ta hodisadan iborat
	KeyDown	Tugma kelganda
	KeyPess	Tugma bosilganda
	KeyUp	Tugma ketganda
	PreviewKeyDown	Tugma kelganda
8	**Layout guruhi**	Tartib guruhi formani atrofidagilar bilan

		ishlash tartibini nazora qilash uchun rejalashtirilgan
	DpiCahged	Joriy forma tartibi o‘zgarganda
	DpiCahgedAfterParent	Meros bergan sinf tartibi o‘zgargandan keyin
	DpiCahgedBeforParent	Meros bergan sinf tartibi o‘zgargandan oldin
	Layout	Forma tartibini o‘rnatish
	MdiChildActivate	Forma sinfining meros xo‘ri faol bo‘lganda
	Move	Forma ko‘chirilganda
	Resize	Forma o‘lchamini o‘zgartirganda
9	**Mouse guruhi**	Sichqoncha bilan bo‘ladigan hodisalar bo‘lib, 6 ta hodisadan iborat
	MouseDown	Sichqoncha kelganda
	MouseEnter	Sichqoncha joriy holatida
	MouseHover	Sichqoncha ozgina harakatida
	MouseLeave	Sichqonchaga to‘xtash bo‘lganda
	MouseMove	Sichqoncha ko‘chganda
	MouseUp	Sichqoncha ketganda
10	**Property Changed guruhi**	Formaning xususiyatlari o‘zgarganda bajariladigan 12 hodisasi bor
	AutoSizeChanged	Avtomatik o‘lcham o‘zgarganda
	AutoValidateChanged	Avtomatik tekshirib bo‘lganda
	BackColorChanged	Fon rangi o‘zgarganda
	EnabledChanged	Holat o‘zgarganda

Hodisalarni o‘rnatish juda qulay hisoblanadi. Kerakli hodisaning o‘ng tomoniga sichqonchani ikki marta bossangiz, hodisaga mos bo‘lgan dastur franmentini yozish uchun funksiya yaratib beradi. Bir misol bilan ko‘rib chiqamiz.

Buning uchun MessageBox sinfining show funksiyasidan foydalanamiz. Bu funksiya statik funksiya bo‘lib, System::Windows::Forms nomlar fazosiga tegishli, 21 variantda yozilishi mumkin. U natija sifatida DialogResult tipini qaytaradi. Uning argumentlari:

Text - String tipini qabul qiladi va matni xabar sifatida chiqaradi

Caption - String tipini qabul qiladi va xabar sarlavhasini chiqaradi

Buttons - MessageBoxButtons tipini qabul qiladi va xabarni qabul qilish tugmalarini chiqaradi, AbortRetryIgnore, OK, OKCancel, RetryCancel, YesNo, YesNoCancel qiymatlarni qabul qiladi.

Icon - MessageBoxIcon tipini qabul qiladi va xabarni ikonkasini chiqaradi, Asterisk, Error, Exclamation, Hand, Information, None, Question qiymatlarni qabul qiladi

defaultButton - MessageBoxDefaultButton tipini qabul qiladi va xabarni tugmalarini belgilaydi, Button1, Button2, Button3 qiymatlari qabul qiladi.

Options - MessageBoxOptions tipini qabul qiladi va xabarni chiqish varintlarini aniqlaydi, DefaultDesktopOnly, RightAlign, RtlReading, ServiceNotification qiymatlari qabul qiladi.

Bu funksiyaning boshqa argumentlari ham bor, ularni keyinchalik o‘rganish mumkin.

MessageBox sinfining show funksiyasidan foydalanish:

```
MessageBox::Show("Xabar", "Sar…", MessageBoxButtons::OK, MessageBoxIcon::Information );
```

Forma hodisalaridan foydalanish uchun 3 tasini ko‘richb chiqamiz:

1. Click hodisasini ishlatish uchun, uning o‘ng tomoniga sichqonchani bosamiz. Forma sinifida tayyor Form1_Click() nomli funksiya yaratiladi. Bu funksiyani ichida quyidagicha fragmentni yozamiz:

```
   MessageBox::Show("Click bo‘ldi", "Xabar", MessageBoxButtons::OK, MessageBoxIcon::Information);
```

Shuningdek formaning InitializeComponent() funksiyasini ichida avtomatik tarzda this->Click += gcnew System::EventHandler(this, &Form1::Form1_Click); hodisa qo‘shib qo‘yilgan. Agar bu hodisani to‘liq o‘chirmoqchi bo‘lsangiz, Click hodisasining o‘ng tomoniga yozilgan Form1_Click() ni o‘chirib, enter tugmasini bosish yetarli o‘zi avtomatik InitializeComponent() funksiyasini ichidagi hodisani o‘chiradi. Dasturning qismidan funksiyani o‘chirish mumkin.

Quyidagi ikki hodisani ishlatish uchun bir masala olamiz. Yordam tugmasi bosilganda, yordam tugmasi bosildi deb xabar chiqsin. Foydalanuvchi ha deb javob bersa, oynaga qaytadi. Agar yo‘q deb bossa, yolg‘onchimi siz deb chiqadi va foydalanuvchi, faqat ha javobni tanlaydi. Javob tanlaganda oynaning sarlavhasi yolg‘onchi deb o‘zgarsin. Oyna holat hodisalari orqali bu tekshirib, yana oyna sarlavhasi o‘zgardi degan habarni chiqarsin.

2. HelpButtonClicked hodisasini ishlatish uchun, uning o‘ng tomoniga sichqonchani bosamiz. Forma sinifida tayyor Form1_HelpButtonClicked () nomli funksiya yaratiladi. Bu funksiyani ichida quyidagicha fragmentni yozamiz:

```
System::Windows::Forms::DialogResult result;
                result =
MessageBox::Show("HelpBuutonClicked","Xabar",MessageB
oxButtons::YesNo,MessageBoxIcon::Hand,
MessageBoxDefaultButton::Button1);
                if(result ==
System::Windows::Forms::DialogResult::No) {
                    MessageBox::Show("Yolg‘nchimi siz
!","Xabar",MessageBoxButtons::OK,MessageBoxIcon::Warn
ing);
                    Form1::Text = "Yolg‘nchi";
                }
            }
```

Shuningdek formaning InitializeComponent() funksiyasini ichida avtomatik tarzda this->HelpButtonClicked += gcnew System::ComponentModel::CancelEventHandler(this, &Form1::Form1_HelpButtonClicked); hodisa qo‘shib qo‘yilgan.

3. TextChanged hodisasini ishlatish uchun, uning o‘ng tomoniga sichqonchani bosamiz. Forma sinifida tayyor Form1_TextChanged() nomli funksiya yaratiladi. Bu funksiyani ichida quyidagicha fragmentni yozamiz:

```
MessageBox::Show("Forma sarlavhasi
o‘zgardi","Xabar",MessageBoxButtons::OK,
MessageBoxIcon::Information,MessageBoxDefaultButton::
Button3,MessageBoxOptions::RightAlign);
```

Shuningdek formaning InitializeComponent() funksiyasini ichida avtomatik tarzda this->TextChanged += gcnew System::EventHandler(this, &Form1::Form1_TextChanged); hodisa qo‘shib qo‘yilgan.

Bu hodisalarni ishlatib ko‘rsangiz, ancha narsalarni farqiga borishingiz kerak.

Matnda bir joyda forma, bir joyda oyna, bir joyda form deb ishlatilgan bo‘lsa, ularning hammasi bir ma‘noni beradi. Shu bilan formaning xususiyatlari va hodisalari bilan tanishib chiqdingiz va uni o‘rganishni davom etish kerak. Chunki, keyingi barcha ishlaringiz shu

kabi formalar bilan bog‘liq. Keyinchalik formaning ma’lum bir xususiyati va hodisalari bilan ishlaganda. Albatta uni ta‘kidlab aytib o‘tib ketamiz.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Integrallashgan ishlab chiqarish muhitlarining qayday turlarini bilasiz?
2. Qachondan boshlab integrallashgan muhitlar jadal rivojlangan?
3. Turbo muhitlarni sanab bering.
4. Obyektga yo‘naltirilgan tilni birinchi bo‘lib integrallashgan muhitlarga kim va qaysi firma joriy qilgan.
5. Qanday muhitda zamonaviy matn muharrirlari kodi avtomatik bajarilishini ta‘minlash (kodi tugatish), muharriri muhitda joriy yozilgan kodi sintaktik to‘g‘ri va uning davomi bo‘lishi mumkin?
6. Taniqli NetBeans integrallashgan muhitni dastlab Java dasturlash uchun qaysi universitet talabasi loyihasi sifatida yaratgan?
7. Vizual Studio 97 haqida nimalarni bilasiz?
8. Visual Studio 2012 haqida nimalarni bilasiz?
9. ISO xalqaro standartlari bilan standartlashtirilgan va bir necha implementations ega bo‘lgan obyektga yo‘naltirilgan ko‘p tilli dasturlash platformani ayting?
10. MS Visual Studio muhit oynasi nechta qismdan iborat?
11. Windows ilovalarni yaratish uchun qaysi tugmachalar majmuasini bosish yetarli.
12. Qaysi vosita bir necha qo‘llab-quvvatlanadigan tillarda har qanday yozilgan dasturlar ijrosini boshqaradi.
13. CLR Empty Project qanday loyiha yaratish uchun kerak.
14. Windows Forms Control Library qanday loyiha yaratish uchun kerak.
15. Loyihaning External Dependencis papkasida qanday fayllar saqlanadi.
16. [FILE] menyusining vazifalarini sanab bering.
17. [TOOLS] menyusining vazifalarini sanab bering.
18. Menyular va uskunalar paneli foydalanuvchilar qanday papka buyruqlarga kirishi kerak?
19. Buyruq yaratishdan avval nima yaratish kerak.
20. Yangi menyular va uskunalar majmuasi aniqlash uchun, Visual Studioda qanday jadval fayl orqali ularni tasvirlash kerak.

21. Qatorlarga izoh qo‘yish tugmasini ayting.
22. Satrga izoh qo‘shish tugmasini ayting.
23. Tegishli funksiya variantlarini chaqirish tugmasini ayting.
24. [Ctrl+Shift +.] tugmachalar majmuasi qanday amal bajaradi?
25. [Ctrl+Shift+Tab] tugmachalar majmuasi qanday amal bajaradi?
26. Oyna uchun yangi form1 sinfi yaratiladi. Bu sinf qaysi sinfining merosxo‘ri hisoblanadi?
27. Kontekst menyu oqali formaning qanday funksiyalarini bajarish mumkin?
28. Formaning rejimlarini o‘rnatish qayerda joylashgan?
29. Ichki va tashqi ma’lumotlar bilan ishlash hodisalari qayerda joylashgan.
30. MessageBox sinfining nechta argumentlari bor? Har biriga izoh bering.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASH HAMDA RIVOJLANTIRISH UCHUN ASSISMENT TOPSHIRIQLARI.

BIRINCHI ASSISMENT TOPSHIRIG‘I	
	Form xususiyatlari va hodisalariga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalga oshriladi.
№ - sizning tug‘ilgan kuninggiz	
1. Forma yarating va uni o‘lchamlaringa (№*50, (№%2)*100) qiymatlarni o‘rnating. _____ _____ _____ _____ _____	
2. Formaning (№%6) - hodisasi bajarinlangini tekshiring? (0- Click, 1- DoubleClick, 2 - MouseClick, 3- MouseDoubleClick, 4- ResizeBegin, 5 – ResizeEnd) _____ _____	

_____

3. Formaning ((№/4)%6) - hodisasi bajarinlangini tekshiring, bunda MessageBox::Show funksiyasining (№%5)+1 ta argumentidan foydalanig ?

(0- Click, 1- DoubleClick, 2 - MouseClick, 3- MouseDoubleClick, 4- ResizeBegin, 5 – ResizeEnd)

_____
_____
_____
_____
_____

4. Formaning ((№/4)%8) - hodisasi bajarinlangini tekshiring, bunda MessageBox::Show funksiyasining (№%5)+1 ta argumentidan foydalanig ?

(0-FormClosed, 1- FormClosing, 2-HelpButtonClicked, 3- Load, 4-Shown, 5-Activated, 6-Move, 7-Resize)

_____
_____
_____
_____

5. Formaning ((№/4)%7) - hodisasi bajarilangini tekshiring, bunda MessageBox::Show funksiyasining (№%5)+1 ta argumentidan foydalanig ?

(0-MouseDown, 1-MouseEnter, 2-MouseHover, 3-MouseLeave, 4-MouseMove, 5-MouseUp, 6-AutoSizeChanged)

_____
_____
_____
_____

6. Formaning (№%8) - hodisasi bajarilanga, tekshiring va noto‘g‘ri degan javobda (M%7) - hodisani bajartiring, bunda MessageBox::Show funksiyasining (№%3)+1 ta argumentidan foydalanig ?

M uchun – (0-MouseDown, 1-MouseEnter, 2-MouseHover, 3-MouseLeave, 4-MouseMove, 5-MouseUp, 6-AutoSizeChanged), № uchun - (0-FormClosed, 1- FormClosing, 2-HelpButtonClicked, 3-

Load, 4-Shown, 5-Activated, 6-Move, 7-Resize)

_____

_____

_____

_____

_____

7. Formaning xususiyatlarini ((№*2)%12), ((№+6)%12), ((№/6)%12) bajarib tekshiring.

0- BackColor	_____
1- BackgroundImage	_____
2- Cursor	_____
3- Font	_____
4- FormBorderStyle	_____
5- RightToLeft	_____
6- Text	_____
7- UseWaitCursor	_____
8- Enabled	_____
9- Language	_____
10- locked	_____
11- AutoScroll	_____

8. Formaning xususiyatlarini ((№*2)%13), ((№+6)%13), ((№/6)%13) bajarib tekshiring.

0- AutoSize	_____
1- Lacation	_____
2- MaximumSize	_____
3- MimimumSize	_____
4- Size	_____
5- StartPosition	_____
6- WindowsState	_____
7- ControlBox	_____
8- HelpButton	_____
9- Icon	_____
10- MaximizeBox	_____
11- MinimizeBox	_____
12- Opacity	_____

9. Biror bir hodisa bilan yuqoridagi mos xususiyatlarni 2 tasini o‘zgartiring.

_____

_____
_____
_____
_____

10. Hodisalarni boshqarishga doir masala tuzing va uni bajaring.
    Masala
    _____
    _____
    _____
    _____
    Echim_____
    _____
    _____
    _____
    _____
    _____
    _____

11. Barcha vazifalarni o‘rganganingiz bo‘yicha eslab ko‘ring, nimalarni o‘rgandingiz va bir takrorlang.
12. Barcha bilim va ko‘nikmalar asosida kichik loyiha tayyorlang.

3.2. Komponentalar bilan ishlash.

📖 Komponenta tushunchasi va xususiyatlari, Standart komponentasining xususiyatlari va hodisalari, Additional komponentasining xususiyatlari va hodisalari, System komponentalari komponentasining xususiyatlari va hodisalari, Toolbox oynasi, Yangi komponentalarni qo‘shish, komponentalarni o‘zaro bog‘lash bo‘yicha nazariy ma’lumotlar keltirilgan bo‘lib, nazariy bilimlarni asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni mustahkamlash uchun 30 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 8 ta assisment topshirig‘i berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

✎ *Kalit so‘zlar.* IDE, Visual C++, user interface, OYD, komponenta, xususiyatlari, hodisalari, Toolbox, Tab, Add Tab, Delete Tab, Mantlarni tahrirlash, Ma’lumotlarni ko‘rsatish (faqat o‘qish uchun), CheckBox, NumericUpDown, Label, Button, Additional komponentasi, Panel, TabControl, System komponentasi, MenuStrip, ContextMenuStrip.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Sinf va sinf obyekti, xususiyat, hodisa, forma, komponenta dasturlashga oid dastlabki tushunchalar hamda C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Visual C++ muhitida dasturlash, integrallashgan ishlab chiqarish muhiti, Toolbox, Komponenta tushunchasi va xususiyatlari, Toolbox oynasi, Toolbox oynasini tahrirlash, Yangi komponentalarni qo‘shish, Komponentaning xususiyatlari guruhlari, loyihani boshqaruv oynasi orqali ko‘rish, Komponentaning usullari (methods) va hodisalari (events), Standart komponentasining xususiyatlari va hodisalari, Toolbox oynasidagi [Common Controls] Tab, Standart komponentalarni guruhlari, komponentalarni oynaga joylashtirish, Additional komponentasining xususiyatlari va hodisalari, Additional komponentalarining guruhlari, Panel komponentasi, groupBox komponentasi, tabControl komponentasi, splitContainer komponentasi, tableLayoutPanel komponentasi, flowLayoutPanel komponentalarning xususiyatlarining vazifalari, System komponentasining xususiyatlari va hodisalari, System komponentalarining guruhlari, Menyu va uskunalar panellarini, Components tab komponentalari, contextMenuStrip obyekti o‘rnatish va komponentalar bilan bog‘lash usullari, shuningdek ba‘zi bir xususiyatlar va hodisalarni o‘rganishingiz mumkin.

REJA

1.Komponenta tushunchasi va xususiyatlari.
2.Standart komponentalarining xususiyatlari va hodisalari.
3.Additional komponentalarining xususiyatlari va hodisalari.
4. System komponentalarining xususiyatlari va hodisalari.

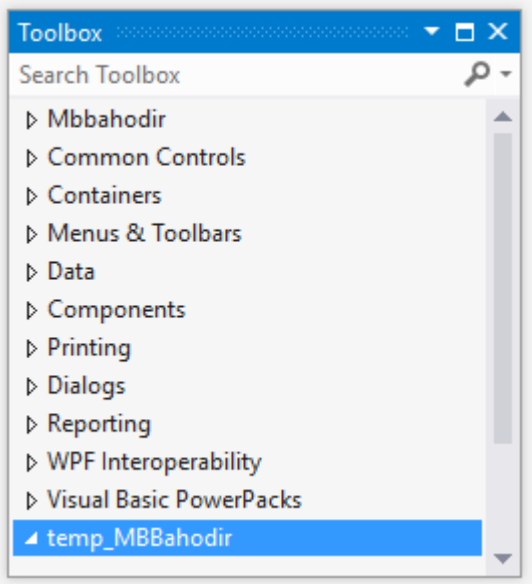
KIRISh

MS Visual Studio da ishlash tamoyilining asosiy qulayligi bu komponentalaridir. Foydalnuvchining UI ni tahlil qilib qaralsa, unda juda ko‘plab kompnentalar joylashtirilgan. Bu juda osondir. MS Visual Studio Visual C++ da turli vazifalarni amalga oshirish komponentalari mavjud. Ular asosan Toolbox deb ham yuritiladi. Toolbox dagi har bir komponentaning nomlar fazosi va sinflari mavjud. Ulardan foydalanganda shu sinflardan merosxo‘r olib yaratiladi va bir oynada bir xil tipdagi kompanentadan bir nechtasidan foydalanish mumkin.

Komponenta tushunchasi va xususiyatlari. Komponenta bu MS Visual Studio ning eng asosiy qurolidir. Dasturchi bular orqali tez dastur

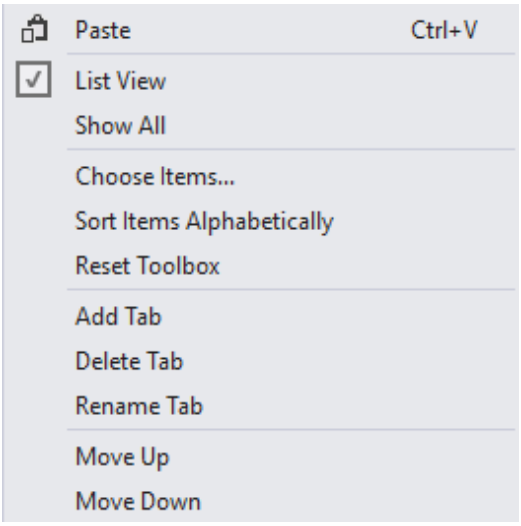
tuzadi va IDE muhitining imkoniyatlaridan foydalaniladi. Komponentalar to‘plami Toolbox deb yuritiladi.

Toolbox interaktiv oynasi muhitning ixtiyoriy joyida joylashgan bo‘lshimi mumkin. Loyihalovchi uni o‘ziga mos qilib joylashtirish mumkin. Agar muhitning oynasida Toolbox oynasini ko‘rmayotgan bo‘lsangiz, menyudan foydalanib, [view → Toolbox] buyruqlari bajaring. [Ctrl + W, X] tugmachalar majmuasini ham bosish orqali Toolbox oynasiga o‘tish/chaqirish mumkin. Muhit komponentalar soni nechta degan savolga javob berishdan boshlaymiz. Muhit integrallashgan muhitligini bilamiz, shuning uchun undagi komponentalar soni oldindan xech kim ayta olmaydi. Yangi kutubxonalarni joriy qilish yangi komponentalarni olib kiradi. Standart foydalanuvchilar uchun Toolbox oynasi 10.1-rasmda tasvirlangan.

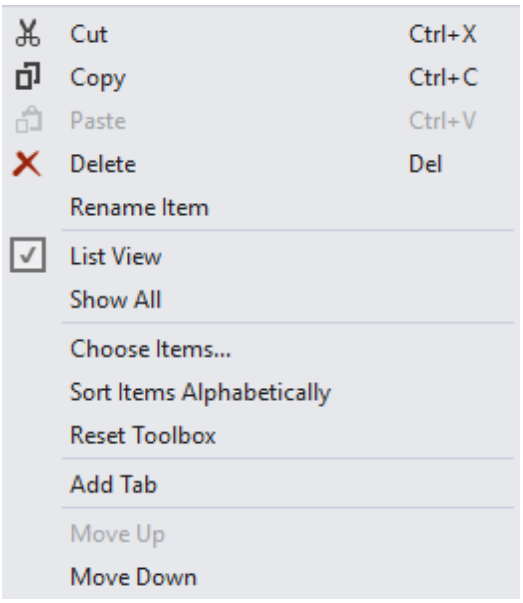


10.1-rasm. Toolbox oynasi.

Toolbox oynasini tahrirlash mumkin. Buning uchun sichqonchaning o‘ng tugmasini Tab ga, ya‘ni komponentalar guruhining nomiga bosib, 10.2-rasmdagi kontekst menyuni, Tab ning ichiga kirib komponentani ustiga bossak 10.3-rasmdagi kontekst menyuni chaqiramiz.



10.2-rasm. Komponentalar guruhi uchun kontekst menyu.



10.3-rasm. Komponenta uchun kontekst menyu.

Rasmlarga qarasak, ularda bir xil nomli buyruqlar uchraydi, bular butun Toolbox oynasi uchun xizmat qiladi va 5 ta bo‘limdan iborat. Har bo‘limga va uning buyruqlariga to‘xtalib o‘tamiz:

1- Bo‘lim. Komponentalar ustida amal bajarish uchun mo‘ljallangan bo‘lib, [Cut] – komponentani buferga olish va joyidan o‘chirish uchun, [Copy] komponentaning nusxasini olishsh uchun, [Paste] – komponentaning olingan nusxasini joylashtirish, buni komponentalar guruhiga ham qo‘llash mumkin. [Delete] – komponentani guruhdan o‘chirib tashlash, [Rename item] – komponenta nomini o‘zgartirish.

2- Bo‘lim. Toolbox oynasiga xizmat qiladi.[List View] – tanlagan guruh komponentalarini ro‘yxat qilib chiqaradi, o‘chirilgan bo‘lsa, uskunalar to‘plami sifatida ko‘rsatadi. [Show All] – muhitdagi barcha faollashtirilgan komponentalarni chiqarib beradi. Komponentlar juda ko‘pligi uchun bularni ishlatish juda mushkul ish deb hisoblayman. Tajribamda men ham bularning hammasini ishlatolmaganman.

3- Bo‘lim. Toolbox oynasiga xizmat qiladi va komponentalar hosil qilish uchun ishlatiladi. [Choose Items..] – yangi komponentalarni qo‘shish uchun ishlatiladi, [Sort Items Alphabetically] – komponentalarni alfavit bo‘yicha saralash, [Reset Toolbox] - Toolbox komponentalarni sozlamalarini standart shaklga keltirish.

4- Bo‘lim. Toolbox oynasiga Tab larni, ya‘ni komponenta guruhlari ustida ishlash imkoniyatini beradi. [Add Tab] - yangi komponenta guruhi qo‘shish, [Delete Tab] - komponenta guruhini o‘chirish, [Rename

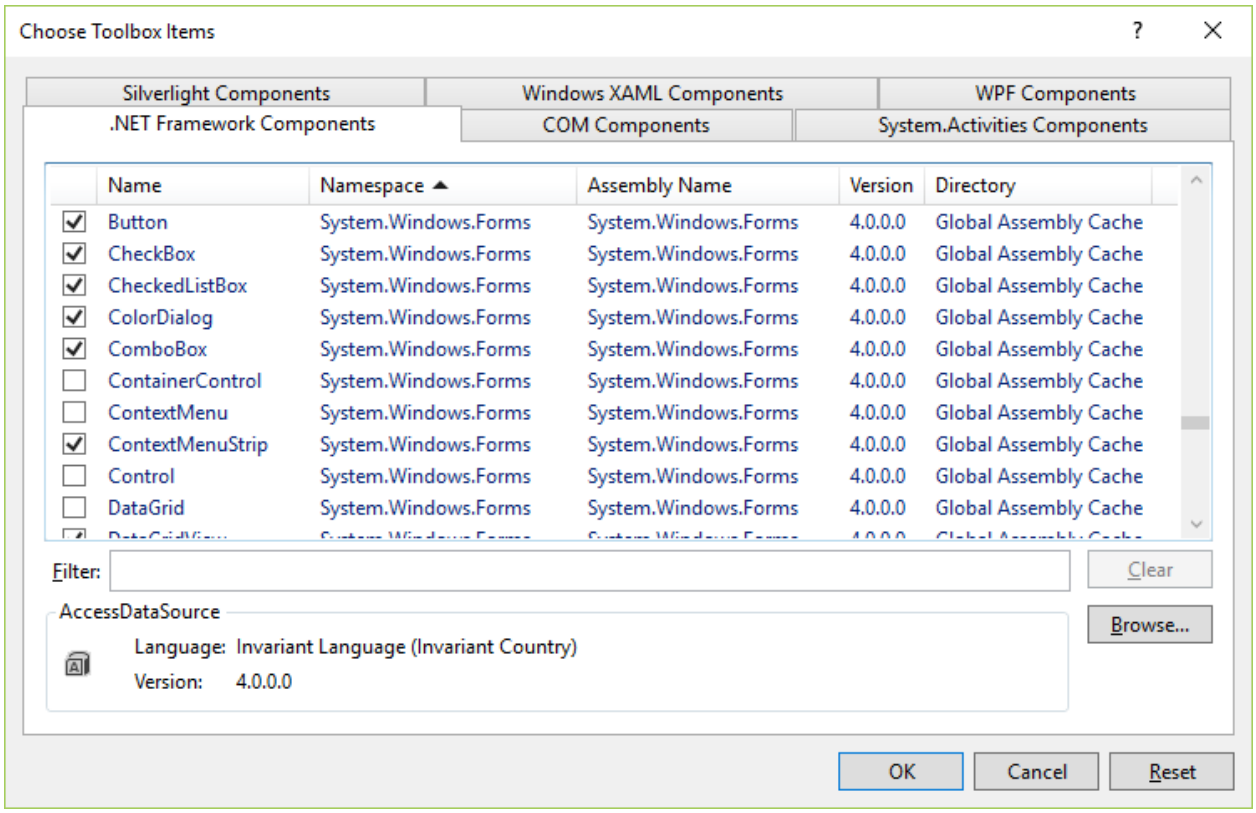
Tab] - komponenta guruhi qayta nomlash, standart nomlaganlarini ham bu amallar bilan o‘zgartirish mumkin.

5- Bo‘lim. komponenta guruhi va komponentalarning o‘rinlarini almashtirish uchun ishlatiladi. [Move Up] – yuqoriga ko‘chirish, [Move Down] – pastga ko‘chirish.

Bularning barchasi muhitning dasturchiga moslashuvchanligini bildiradi.

Yangi komponentalarni qo‘shishni ko‘rib chiqaylik, bu buyruq bosilishi bilan 10.4-rasmda tasvirlangan muloqot oynasi chiqadi.

Bu rasmdan komponentalarning turlarini ko‘rish mumkin, har bir bo‘limi 1000 dan ortiq sinflarga ega. Har bir sinf esa, bitta komponenta hisoblanadi. Agar tizimda mavjud bo‘lmagan komponentalarni ham shu oyna orqali qo‘shish mumkin, masalan, MySql, oqimlar bilan ishlash, OpenCV, OpenMP, Fuzzy ToolBox kabi komponentalarni qo‘shish mumkin.



10.4-rasm. Yangi komponentalarni qo‘shish.

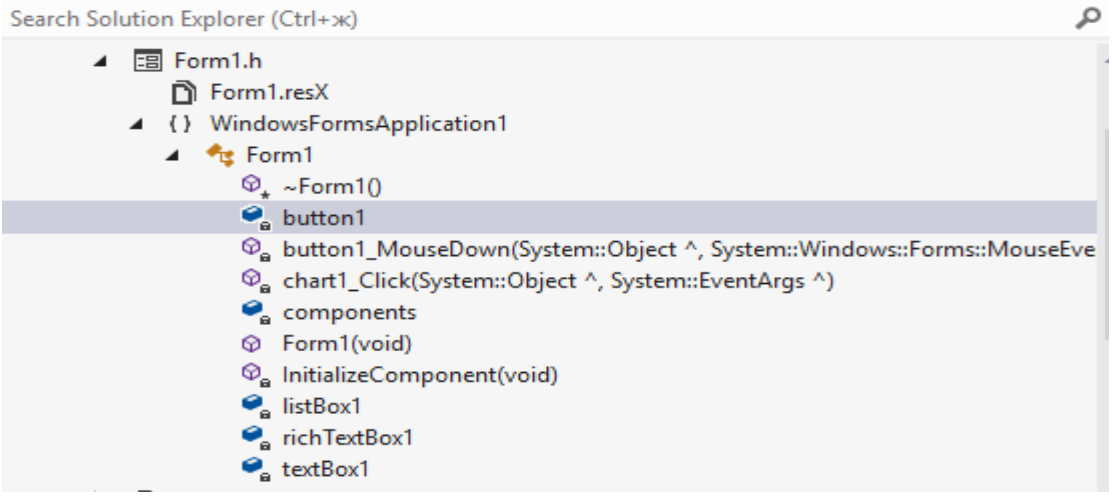
Komponentaning xususiyatlari. Har bir komponentaning shunday xususiyatlari bor. Eslab ko‘ring OYDda sinfni yaratish vaqtida nimalar yaratilar edi. Xuddi shuningdek, har bir komponeta sinfida uning xususiyatlari (properties), usullari (methods) va hodisalari (events) mavjud.

Komponentaning xususiyatlari quyidagi guruhlarga bo‘linadi (10.1-jadvalga qarang)

10.1-jadval. Komponentaning xususiyatlari

1	**Accessibility guruhi**	Bu guruhdagi xususiyatlar bir komponentada o‘zgarishlarni aniqlash va o‘rnatish uchun ishlatiladi.
2	**Appearance guruhi**	Komponentaning tashqi ko‘rinishi uchun xususiyatlarni o‘rnatishga mo‘ljallangan.
3	**Behavior guruhi**	Komponentaning rejim xususiyatlarini o‘rnatishga mo‘ljallangan.
4	**Data guruhi**	Komponentani ichki va tashqi ma’lumotlar bilan bog‘lash xususityatlari uchun ishlatiladi.
5	**Design guruhi**	Komponentani loyihalash xususiyatlari uchun ishlatiladi
6	**Focus guruhi**	Komponentada fokuslarni boshqarish xususiyatlari uchun ishlatiladi
7	**Layout guruhi**	Komponentani tartiblash xususiyatlari uchun ishlatiladi
8	**Misc guruhi**	Boshqa xususiyatlarni o‘rnatish, komponentani turiga qarab o‘zgarib turadi.

Bu xususiyat guruhlari komponenta tipiga moslashgan holda doimiy o‘zgarib turadi va tipga mos xususiyatlarni aniqlab beradi. Komponenta xususiyatlarga qiymatlar maxsus, to‘plamdan tanlash, sonli, matnli, mantiqiy kabi ma’lum tiplar qimatini oladi. Agar murakkab tip bo‘lsa, albatta muloqot oynasi orqali tanlash mumkin. Komponenta xususiyatlariga qiymatlarni berish vizual amalga oshiriladi va asosiy formaga uning dastur fragmentlari yozib ketiladi. Buni loyiha boshqaruv oynasi orqali ko‘rish mumkin (10.5-rasmga qarang)



10.5-rasm. Komponentalarning qo‘shilganligi.

Oynaning InitializeComponent funksiyasida ham komponentalarning xususiyatlari ko‘shilganini ko‘rish mumkin.

```
void InitializeComponent(void)
        {
            this->button1 = (gcnew
System::Windows::Forms::Button());
            this->listBox1 = (gcnew
System::Windows::Forms::ListBox());
            this->SuspendLayout();
            //
            // button1
            //
            this->button1->Location =
System::Drawing::Point(208, 73);
            this->button1->Name = L"button1";
            this->button1->Size =
System::Drawing::Size(75, 23);
            this->button1->TabIndex = 0;
            this->button1->Text = L"button1";
            this->button1->UseVisualStyleBackColor
= true;
            this->button1->MouseDown += gcnew
System::Windows::Forms::MouseEventHandler(this,
&Form1::button1_MouseDown);
            //
            // listBox1
            //
            this->listBox1->FormattingEnabled =
true;
            this->listBox1->Location =
System::Drawing::Point(363, 62);
            this->listBox1->Name = L"listBox1";
            this->listBox1->Size =
System::Drawing::Size(120, 95);
            this->listBox1->TabIndex = 1;


            // Form1 ...
```

```
                //

            }
```

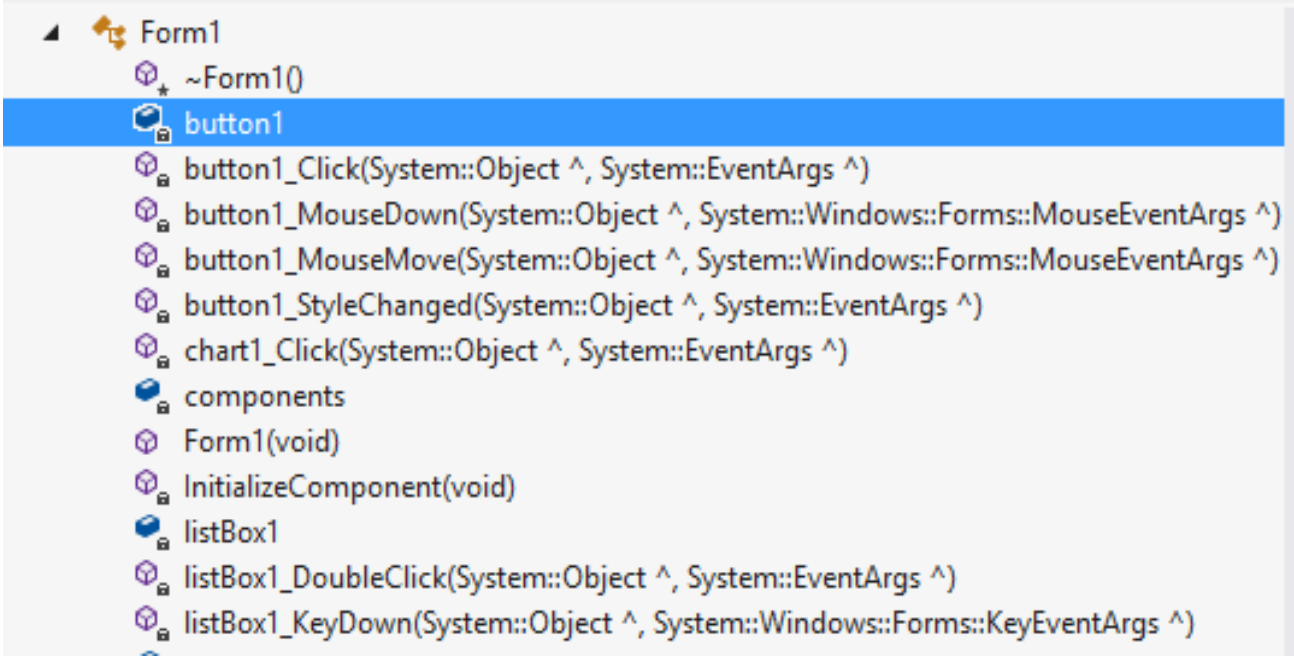
Komponentaning usullari (methods) va hodisalari (events) quyidagi guruhlarga bo‘linadi (10.2-jadvalga qarang)

10.2-jadval. Komponentaning usullari (methods) va hodisalari (events)

1	**Action guruhi**	Komponenta uchun global hodisalar faollashtirish va ma’lum bir algoritm yozish
2	**Appearance guruhi**	Komponentaning ichki hodisalari uchun ishlatiladi va ma’lum bir algoritm yozish
3	**Behavior guruhi**	Komponenta holatlari uchun hodisalarni faollashtirish
4	**Data guruhi**	Ichki va tashqi ma’lumotlar bilan ishlash hodisalarini faollashtirish
5	**Drag drop guruhi**	Komponentaning harakatlanish hodisalarini faollashtirish
6	**Focus guruhi**	Komponentada fokuslarni boshqarish hodisalarini faollashtirish
7	**Key guruhi**	Komponentaning mos tugmalar bilan ishlash hodisalarini faollashtirish
8	**Layout guruhi**	Komponentada atrofidagilar bilan ishlash tartibini nazora qilish hodisalarini faollashtirish
9	**Mouse guruhi**	Komponentada sichqoncha bilan bo‘ladigan hodisalarini faollashtirish
10	**Property Changed guruhi**	Komponentaning xususiyatlari o‘zgarganda bajariladigan hodisalarini faollashtirish

Bu usullar va hodisalarning odatda hammasini ham vizual ishlatish imkoni yo‘q. Shuning uchun agar komponentaning biror funksiyasi On bilan boshlansa bilingki bu hodisalar bo‘lsa, -ed,-ing bilan tugaganlari ko‘proq usul(funksiyalari) hisoblanadi. Guruhlar komponenta tipiga moslashgan holda doimiy o‘zgarib turadi va tipga mos hodisalarini aniqlab beradi. Komponenta hodisalari funksiya sifatida yaratiladi va dasturchi unga kerakli o‘zining algoritmini yozadi. Komponenta hodisalarini qo‘shish vizual amalga oshiriladi va asosiy formaga uning dastur fragmentlari yozib ketiladi. Ammo dasturchi o‘z funutsiyaning

algoritmini yozishi kerak. Buni loyiha boshqaruv oynasi orqali ko‘rish mumkin (10.6-rasmga qarang)



10.6-rasm. Komponentalarning hodisalari.

Oynaning InitializeComponent funksiyasida ham komponentalarning hodisalari qo‘shilganini ko‘rish mumkin.

```
void InitializeComponent(void){
    this->button1 = (gcnew
System::Windows::Forms::Button());
    this->listBox1 = (gcnew
System::Windows::Forms::ListBox());
    this->SuspendLayout();
    //
    // button1
    //
    // xususiyatlar
    this->button1->Click += gcnew
System::EventHandler(this, &Form1::button1_Click);
    this->button1->MouseDown += gcnew
System::Windows::Forms::MouseEventHandler(this,
&Form1::button1_MouseDown);
    this->button1->MouseMove += gcnew
System::Windows::Forms::MouseEventHandler(this,
&Form1::button1_MouseMove);
    this->button1->StyleChanged += gcnew
```

```
System::EventHandler(this,
&Form1::button1_StyleChanged);
    //
    // listBox1
    //
    // xususiyatlar
    this->listBox1->DoubleClick += gcnew
System::EventHandler(this,
&Form1::listBox1_DoubleClick);
    this->listBox1->KeyDown += gcnew
System::Windows::Forms::KeyEventHandler(this,
&Form1::listBox1_KeyDown);
    //
    // boshqa komponentalar
    // Form1
    // … xususiyatlar
    // hodisalar
    this->Controls->Add(this->listBox1);
    this->Controls->Add(this->button1);
    // … xususiyatlar
}
private: System::Void
button1_MouseDown(System::Object^  sender,
System::Windows::Forms::MouseEventArgs^  e) { }
private: System::Void
listBox1_DoubleClick(System::Object^  sender,
System::EventArgs^  e) {}
private: System::Void
listBox1_KeyDown(System::Object^  sender,
System::Windows::Forms::KeyEventArgs^  e) { }
private: System::Void
button1_MouseMove(System::Object^  sender,
System::Windows::Forms::MouseEventArgs^  e) {    }
private: System::Void button1_Click(System::Object^
sender, System::EventArgs^  e) { }
private: System::Void
button1_StyleChanged(System::Object^  sender,
System::EventArgs^  e) { }
```

Shuningdek, ba‘zi komponentalar faol va faol bo‘lmagan holatlarda bo‘lishi mumkin. Bu xuddi Windows oynalaridagi buyruqlardan foydalanishiga o‘xshab ishlatiladi. Komponentalarni o‘z joyida ishaltishni ham bilish kerak va ularni qachon ishlatish kerakligini bilish ham muhim.

Standart komponentasining xususiyatlari va hodisalari. Standart komponentalarga Toolbox oynasidagi [Common Controls] Tab dagi komponentalar kiradi. Ularni tarkibini o‘zgartirish mumkin, ya‘ni yangilarini qo‘shish va keraksizlarini o‘chirib tashlash mumkin. Shuningdek, agar maxsus ishlab chiqilgan yangi komponentalar guruhi bo‘lsa ulardan ham foydalanish mumkin. Standart komponentalarni quyidagi guruhlarga bo‘linadi va shu asosida ularning hususiyatlari va hodisalari 95% bir xil bo‘ladi.

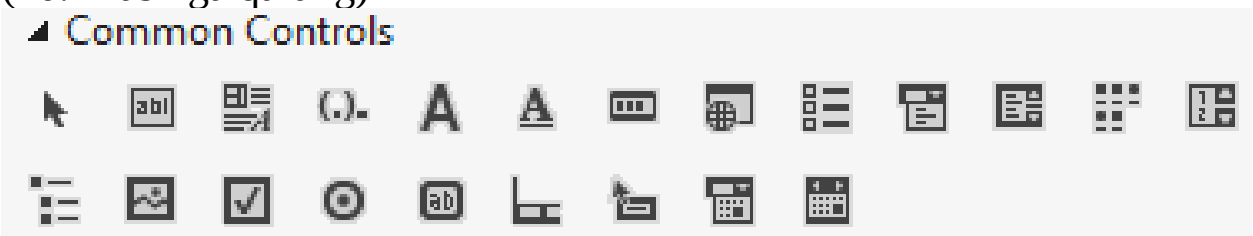
10.3-jadval. Standart komponentalarni guruhlari va vazifalari.

Komponentaning guruhi	Komponentaning nomi	Vazifasi va izoh
Mantlarni tahrirlash	TextBox	Matnni tahrirlash imkonini beradi, dastur ishlayotgan vaqtda foydalanuvchi yoki algoritm yordamida matnni tahrirlash.
	RichTextBox	Oddiy va RTF formatida Matnni tahrirlash imkonini beradi, dastur ishlayotgan vaqtda foydalanuvchi yoki algoritm yordamida matnni tahrirlash.
	MaskedTextBox	Foydalanuvchi tomonidan kiritilayotgan matnlarni maxsus belgi bilan himoyalaydi
Ma’lumotlarni ko‘rsatish (faqat o‘qish uchun)	Label	Foydalanuvchi tomonidan to‘g‘ridan-to‘g‘ri tahrir qilinmaydi matnni ko‘rsatadi.
	LinkLabel	Veb-link sifatida matnni ko‘rsatadi va foydalanuvchi joriy matnni bosganda boshqa oynaga yoki veb-saytga havola faollashtiradi.
	ProgressBar	Foydalanuvchi uchun joriy amal bajarilish jarayonini ko‘rsatadi.
Veb-sahifani ko‘rsatish	WebBrowser	Foydalanuvchi oynasida veb-sahifalar orqali harakat qilish imkonini beradi.

Komponentaning guruhi	Komponentaning nomi	Vazifasi va izoh
Ro‘yxatdan tanlash	CheckedListBox	Har biri katak bilan birga bo‘lgan elementlarning takrorlanadigan ro‘yxatini ko‘rsatadi.
	ComboBox	Qalqib chiquvchi elementlar ro‘yxatini ko‘rsatadi
	ListBox	Matnlar va grafik elementlar ro‘yxatini ko‘rsatadi
	ListView	Elementlarni to‘rt xil ko‘rinishdan birida ko‘rsatadi. Ko‘rinishlarga faqat matn, kichik piktogrammali matn, katta piktogrammali matn va batafsil ko‘rinish kiradi.
	NumericUpDown	Foydalanuvchilar yuqoriga va pastga tugmalari yordamida o‘tiish mumkin bo‘lgan raqamlar ro‘yxatini ko‘rsatadi.
	TreeView	Qo‘shimcha kataklar yoki piktogrammalar bilan matndan iborat bo‘lishi mumkin bo‘lgan tugun obyektlarining iyerarxik to‘plamini ko‘rsatadi.
Grafiklarni tasvirlash	PictureBox	bitmaps va piktogramma kabi tasvir fayllarini ko‘rsatadi.
parametrli qiymat	CheckBox	Matn uchun katakcha va yorliqni ko‘rsatadi. Odatda parametrlarni belgilash uchun ishlatiladi.
	CheckedListBox	Har biri katak bilan birga bo‘lgan elementlarning qaytariladigan ro‘yxatini ko‘rsatadi.
	RadioButton	Yoqilgan yoki o‘chirilishi mumkin bo‘lgan tugmani ko‘rsatadi.
	TrackBar	Foydalanuvchilarga shkala bo‘ylab " Thumb" ni harakatlantirib shkala qiymatlarini o‘rnatish imkonini beradi.
Buyruqlar	Button	Boshlanadi, to‘xtaydi yoki jarayonni bekor qiladi.
	LinkLabel	Veb-link sifatida matnni ko‘rsatadi va foydalanuvchi joriy matnni

Komponentaning guruhi	Komponentaning nomi	Vazifasi va izoh
		bosganda boshqa oynaga yoki veb-saytga havola faollashtiradi.
	NotifyIcon	Fonda ishlaydigan dasturni ifodalovchi vazifalarni paneli holati xabarnomasidagi belgini ko‘rsatadi.
Foydalanuvchi uchun yordam	HelpProvider	Elementlari uchun pop-up yordam oynasini yoki tezkor yordam oynasini beradi.
	ToolTip	Foydalanuvchi nazorat bilan hovers nazorat maqsadi bo‘yicha qisqacha tavsifi ko‘rsatadi va bir pop-up oyna beradi.
Vaqt	DateTimePicker	Foydalanuvchilarga sana yoki vaqtni tanlash imkonini beradigan grafik taqvim ko‘rsatadi.
	MonthCalendar	Foydalanuvchilarga sana qatorini tanlash imkonini beradigan grafik taqvim ko‘rsatadi.

Standart komponentalarni guruhlashning asosiy maqsadi, guruhlarga mos xususiyatlar va hodisalari bir xil bo‘ladi. Bu komponentalarning Toolbox oynasida quyidagicha ko‘rinishga ega (10.7-rasmga qarang)

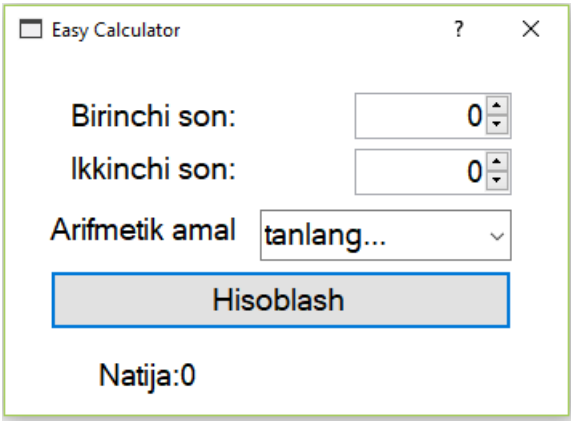


10.7-rasm. Standart komponentalar ko‘rinishi.

Bu komponentalarni oynaga joylashtirish uchun sichqonchaning chap tugmasi bitta bosiladi va oynaga keltirib chap tomoni yana bir marta bosiladi va oynada tanlangan komponentaning nusxasi paydo bo‘ladi. Bu nusxani obyekt deb aytamiz. Chunki eslab ko‘ring, sinfning nusxasi bu obyekt deb aytiladi. Obyektning ustiga sichqonchaning o‘ng tugmasini bossangiz, u bilan bajarish mumkin bo‘lgan amallarni ko‘rasiz. Agar obyektdan nusxa olsangiz uning super sinfining, ya‘ni shu obyekt tipidagi boshqa bir obyekt yaratiladi. Ikkita bir xil tipdagi obyektlarning xususiyatlari va hodisalariga alohida – alohida ishlov beriladi.

Standart komponentalarni xususiyat va hodisalarini ko‘rib chiqish uchun arifmetik amallarni bajaruvchi dastur yaratish masalasini olamiz.

Mantiqiy jihatdan sonlarni kiritish uchun 2 ta NumericUpDown, matnli yozuvlar uchun 4 ta Label, hisoblash uchun bitta Button, amallar uchun bitta CheckBox komponentlarini olamiz va oynaga quyidagi 10.8-rasmdagidek qilib joylashtiramiz.



10.8-rasm. Standart komponentalarni xususiyat va hodisalaridan foydalanish.

Bu masalani loyiha sifatida amalga oshirish uchun quyidagi qadamlar bajariladi:

1-qadam. Forma xususiyatlarini o‘rnatish: BackColor xususiyatiga oq rang; font xususiyatiga 14 o‘lchamli yozuv; Icon ga maxsus ikinka; MaximizeBox xususiyatiga false qiymat; MinimizeBox xususiyatiga false qiymat; Size xususiyatiga yangi o‘lchamlar; StartPosition–CenterScreen holati; Text xususiyatiga “Easy Calculator” matnni yoziladi;

Buni qanday ajratish mumkin, ya‘ni o‘zgargan xususiyatlarni, qiymatining matni yog‘on qora ranga kirganlari o‘zgartirilgan hisoblanadi. Bu o‘zgarishlarni dastur fragment ko‘rinida keltiish ham mumkin. Bu dastur fragmentini qo‘lda yozish kerak emas, ammo tushunarli bo‘lishi uchun keltiramiz. Buni InitializeComponent funksiyasi ichida kerakli izohlar bilan keltirilgan bo‘ladi.

```
this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(351, 225);
this->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 14.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
this->HelpButton = true;
this->Icon = (cli::safe_cast<System::Drawing::Icon^
>(resources->GetObject(L"$this.Icon")));
```

```
this->MaximizeBox = false;
this->MinimizeBox = false;
this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScre
en;
this->Text = L"Easy Calculator";
```

2-qadam. Label1 obyektining xususiyatini o‘rnatish dastur fragmenti sifatida keltiramiz.

```
this->label1->Text = L"Birinchi son:";
```

3-qadam. Label2 obyektining xususiyatini o‘rnatish dastur fragmenti sifatida keltiramiz.

```
this->label2->Text = L"Ikkinchi son:";
```

4-qadam. Label3 obyektining xususiyatini o‘rnatish dastur fragmenti sifatida keltiramiz.

```
this->label3->Text = L"Natija:";
```

5-qadam. numericUpDown1 obyektining xususiyatini o‘rnatish dastur fragmenti sifatida keltiramiz.

```
this->numericUpDown1->TextAlign                    =
System::Windows::Forms::HorizontalAlignment::Right;
```

6-qadam. numericUpDown2 obyektining xususiyatini o‘rnatish dastur fragmenti sifatida keltiramiz.

```
this->numericUpDown2->TextAlign                    =
System::Windows::Forms::HorizontalAlignment::Right;
```

7-qadam. comboBox1 obyektining xususiyatlarini o‘rnatish dastur fragmenti sifatida keltiramiz.

```
this->comboBox1->Items->AddRange(gcnew cli::array<
System::Object^  >(5) {L"[+] (qo‘shish)", L"[-]
(ayirish)", L"[*] (ko‘paytirish)",
                L"[/] (butun bo‘lish)", L"[%]
(qoldiq bo‘lish)"});
this->comboBox1->Text = L"tanlang...";
```

8-qadam. label4 obyektining xususiyatini o‘rnatish dastur fragmenti sifatida keltiramiz.

```
this->label4->Text = L"Arifmetik amal";
```

9-qadam. button1 obyektining xususiyati va hodisani o‘rnatish dastur fragmenti sifatida keltiramiz.

```
this->button1->Text = L"Hisoblash";
this->button1->Click            +=            gcnew
System::EventHandler(this, &Form1::button1_Click);
```

button1_Click funksiyasiga quyidagi algoritmni yozamiz.

```
int one =
Convert::ToInt16(Math::Round(numericUpDown1->Value,
0));
int two = Convert::ToInt16(numericUpDown2->Value);
int result = 0;
label3->Text = "Natija:";

switch(comboBox1->SelectedIndex){
    case 0: result = one + two; break;
    case 1: result = one - two; break;
    case 2: result = one * two; break;
    case 3: result = one / two; break;
    case 4: result = one % two; break;
         default: result = 0;
                break;
 }
 label3->Text += result.ToString();
```

Algoritmga izoh berish shart emas, chunki ko‘p kamchiliklari mavjud, masalan, bo‘lish arifmetik amalida. Ammo ba‘zi xulosalarni chiqarish mumkin. [numericUpDown1->Value] obyektning tanlangan haqiqiy tipdagi qiymatini qaytaruvchi xususiyat, [Convert::ToInt16] – ixtiyoriy tipni butun tipga almashtirib beradi, [label3->Text] – obyektning matn xususiyatini o‘zgartirishi mumkin, [comboBox1->SelectedIndex] – tanlagichning tanlangan indeksini qaytaradi, [label3->Text += result.ToString();] – obyektning matninga matn qo‘shish va tipni satrli tipga o‘tkazish ko‘rsatilgan.

Dastur fragmentida 3 ta belgi [::] – cinfning usullariga murojaat qilish, [->] – obyektning xususiyatiga murojaat qilish, [.] obyektning qiymati ustida bajarilishi kerak bo‘lgan amal. Visual C++ da juda ko‘p usul, va funksiyalarni ishlatish tamoyllari bor, ularni faqat kompilyatorning imkoniyatiga qarab ajratish mumkin.

Standart komponentalarning ba‘zi xususiyatlari va hodisalari, ular uchun ba‘zi usullarni ishlatiyu ko‘rdik. Bularni chuqurroq o‘rganish uchun ko‘proq amaliy ishlarni bajarish kerak.

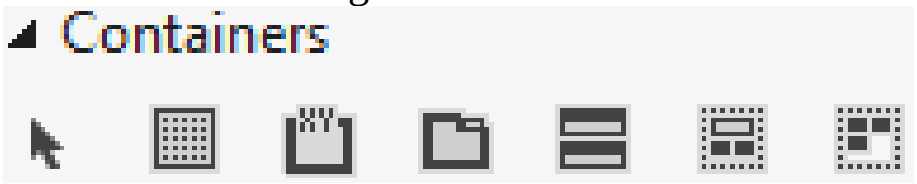
Additional komponentasining xususiyatlari va hodisalari. Bu standart komponentalar bilish ishlash uchun qo‘shimcha komponentlardir. Bular asosan boshqaruv elementlarni guruhlash uchun ishlatiladi. Quyidagi jadvalda qo‘shimacha komponentalarni va vazifalarini keltiramiz.

10.4-jadval. Additional komponentalarining guruhlari va vazifalari

Komponenta	Komponentaning	Vazifasi va izoh

-ning guruhi	nomi	
Boshqaruv-lar elementlarni guruhlash	Panel	elementlarni boshqarish uchun to‘plamini guruhlaydi
	GroupBox	Nomlangan elementlarni boshqarish to‘plamini guruhlaydi
	TabControl	Guruhlangan obyektlarni samarali tashkil etish va kiritish uchun sohali sahifani taqdim etadi.
	SplitContainer	Bir biridan ajratilgan ikki paneli beradi. Splitcontainer – Splitter o‘rniga mo‘ljallangan.
	TableLayout Panel	Kontent dinamik satr va ustunlar iborat panelini ifodalaydi.
	FlowLayout Panel	Dinamik ravishda kontentni gorizontal yoki vertikal joylashtiradigan panelni ifodalaydi.

Bu komponentalar Containers nomli tabda joylashgan. Ularning ko‘rinishi 10.9-rasmda keltirilgan.



10.9-rasm. Containers komponentalari.

Bu komponentlarining xususiyat va hodisalariga alohida to‘xtalib o‘tamiz. Vazifalari elementlar guruhini boshqarishdan iborat, shu maqsadda ishlatiladigan xususiyat va hodisalarini keltiramiz.

Panel komponentasi. Panel – bu oddiy to‘rt tomonidan chegaralangan hududga o‘xshaydi. U butun sohani egallashi uchun o‘ng tomonidagi [Panel Task] dagi [dock in parent container]ni bosish yetarli. [undock in parent container] ni tanlab, oldingi holatiga kelish mumkin. Obyektni harakatlantirish uchun chap tomondagi 4 ta tomonga yo‘naltirilgan tugmachadan foydalaniladi.

AutoSize - xususiyati true/false qiymatlarni qabul qiladi. Odatda chin qiymat ko‘proq ishlatiladi, chunki obyektga joylashtirilgan boshqaruv elementlarininng o‘lchamlariga mos holda o‘lchamni sozlaydi. Joriy holatda yolg‘on qiymatga ega bo‘ladi.

Cursor - xususiyati sichqonchaning kursor ko‘rinishini o‘rnatish imkonini beradi.

BorderStyle – obyektning atrof chegaralarining ko‘rinishlarini o‘rnatish uchun ishlatiladi.

AutoScoroll – obyektda birlashtirilgan elementlar to‘liq ko‘rinmasa, harakatlaguvchi tugmalarni o‘rnatish va olib tashlash uchun ishlatiladi. Bu xususiyat true/false qiymatlarni qabul qiladi.

AutoScorollMargin – bu xususiyat harakatlanuvchi tugmachalar chet masofalarini o‘rnatida, Wight va Height ichki xususiyatlari bor.

AutoScorollMinSize - bu xususiyat harakatlanuvchi tugmachalarning eng kichik o‘lchamlarini o‘rnatadi. Wight va Height ichki xususiyatlari bor.

Dock – obyektni oynaning qaysi qismida joylashtirish kerakligini o‘rnatadi.

Margin – obyekt chet qismlarida qoldirilishi kerak bo‘lgan masofalarni o‘rnatadi. Uning 5 ta xususiyati bor bular barchasi, chap, o‘ng, yuqori va past.

groupBox komponentasi. groupBox - nomlangan elementlar guruhini birlashtirish uchun ishlatiladi. Obyektni harakatlantirish uchun chap tomondagi 4 ta tomonga yo‘naltirilgan tugmachadan foydalaniladi.

Text - bu xususiyat elementlar guruhini nomini oynaga chiqaradi

AutoSize - xususiyati true/false qiymatlarni qabul qiladi. Elementlarning o‘lchamlariga mos obyektga o‘lcham ajratadi.

BackColor – obyektning fon xususiyati rangini o‘rnatadi.

BackgroundImage - obyektning fon xususiyati Image obyektlarni o‘rnatadi.

Dock - obyektni oynaning qaysi qismida joylashtirish kerakligini o‘rnatadi.

FlatStyle – obyektning stil xususiyatini o‘rnatadi.

Font – bu xususiyat obyektning yozuvlari uchun aniq bir standart yozuvni o‘rnatish uchun ishlatiladi.

MaximunSize – obyektning eng katta o‘lchamini belgilaydi.

MinimumSize - obyektning eng kichik o‘lchamini belgilaydi.

Padding – obyektgacha bo‘lgan masofalarni o‘rnatish uchun ishlatiladi. Uning 5 ta xususiyati bor bular barchasi, chap, o‘ng, yuqori va past.

RightToLeft – obyektning matnini o‘ng tomonidan o‘rnatish.

TabIndex – obyektning tab tugmasi bosilganda, nechinchi navbatda bo‘lishini belgilashni o‘rnatadi.

tabControl komponentasi. tabControl bu oddiy to‘rt tomonidan chegaralangan hududga o‘xshaydi. Unga yangi sahifa qo‘shish yoki

saxifani o‘chirish uchun o‘ng tomonidagi [Add Tab] va[Remove Tab] ni bosish yetarli. Obyektni harakatlantirish uchun chap tomondagi 4 ta tomonga yo‘naltirilgan tugmachadan foydalaniladi.

Anchor – obyektni aftor yoki oyna bo‘yicha tartiblab joylashtirish xususiyatlarini tanlaydi. Uning to‘rtta qiymati o‘ng, chap, yuqori, past boryu Bularning o‘zaro aralashuvidan foydalaniladi.

Appearance – bu xususiyat obyektning tugmachalarining ko‘rinishini o‘rnatish uchun ishlatiladi. Uning qiymatlari [Normal], [Buttons], [Flat Buttons] qiymatlarni qabul qiladi.

DrawMode – obyektning xususiyatini chizilgan formasini o‘zgartrish uchun ishlatiladi.

TabPages – bu xususiyat to‘plam qiymatlarni qabul qiladi va uning nechta sahifa bo‘lishini belgilaydi. U bilan ishlaganda har bir sahifaning alohida xususiyatlariga ishlov berish mumkin.

Tabpage1 – bu obyektning saxifa xususiyatlarni o‘rnatish uchun ishlatiladigan obyekt hisoblanadi va panel obyektidek xususiyatlarga ega.

tabStop – obyekt bo‘yicha tab tugmasining harakatlanishi o‘rnatish va o‘chirish qiymatini o‘rnatadi.

splitContainer komponentasi. splitContainer ikkiga ajratilgan elementlarni boshqarish uchun to‘plamni hosil qiladi. Uni butun sohani egallashi uchun o‘ng tomonidagi [dock in parent container]ni bosish yetarli. [undock in parent container] ni tanlab, oldingi holatiga kelish mumkin. Shuningdek, [Horizontal Splitter Orientation] – obyekt ramkalarini gorizontal o‘rnatadi va [Vertical Splitter Orientation] esa vertikal o‘rnatadi. Obyektni harakatlantirish uchun chap tomondagi 4 ta tomonga yo‘naltirilgan tugmachadan foydalaniladi.

Anchor - obyektni aftor yoki oyna bo‘yicha tartiblab joylashtirish xususiyatlarini tanlaydi. Uning to‘rtta qiymati o‘ng, chap, yuqori, past bor va ularning o‘zaro aralashuvidan foydalaniladi.

Backcolor - obyektning fon xususiyati rangini o‘rnatadi.

BorderStyle - obyektning atrof chegaralarining ko‘rinishlarini o‘rnatish uchun ishlatiladi.

FixedPanel – obyekt uchun asosiy panelni o‘rnatish uchun, ya‘ni ikkita paneldan bir asosiy bo‘lishi kerak.

isSplitterFixed - obyekt uchun asosiy panelni o‘rnatish amal qilishi yoki qilmasligini o‘rnatiadi. Bu xususiyati true/false qiymatlarini qabul qiladi.

Orientation – obyektning panellarini gorizontal yoki vertikal o‘rnatish uchun ishlatiladi. Ikkita Horizontal va Vertical qiymat qabul qiladi.

splitContainer1.Panel1 – obyektning panel xususiyatlariga ishlov berish uchun panel xususiyatlarini ochib beradi va u uchun xususiyatlarni o‘rnatish mumkin.

Panel1Coolapsed – obyektda panel1ni to‘liq qilib o‘rnatadi. Bunday xususiyat barcha panellari uchun o‘rinlidir.

Panel1MinSize – obyektdagi panelning eng kichik o‘lchamini o‘rnatishdan iborat.

RightToLeft- obyekt panellarini o‘ngdan o‘rnatish xususiyati.

tableLayoutPanel komponentasi. tableLayoutPanel – bu jadval kabi tartiblangan panellar to‘plami orqali elementlarni birlashtirish obyektidir.

AutoScoroll – obyektda birlashtirilgan elementlar to‘liq ko‘rinmasa, harakatlaguvchi tugmalarni o‘rnatish va olib tashlash uchun ishlatiladi. Bu xususiyat true/false qiymatlarni qabul qiladi.

AutoScorollMargin – bu xususiyat harakatlanuvchi tugmachalar chet masofalarini o‘rnatida, Wight va Height ichki xususiyatlari bor.

AutoScorollMinSize - bu xususiyat harakatlanuvchi tugmachalar eng kichik o‘lchamlarini o‘rnatadi. Wight va Height ichki xususiyatlari bor.

CellBorderStyle – obyektning yacheykalarning chet chegaralarining stilini belgilaydi.

ColumnCount – obyektdagi ustunlar sonini o‘rnatish.

Columns – obyekt ustunlari uchun xususiyatlarni o‘rnatish uchun alohida muloqot oynasi ochiladi.

GrowStyle – obyektga ustun yoki qator yoki ikkalasini qo‘shish xususiyatini o‘rnatish.

RowCount – qatorlar soni

Rows - obyekt qatorlari uchun xususiyatlarni o‘rnatish uchun alohida muloqot oynasi ochiladi.

flowLayoutPanel komponentasi. flowLayoutPanel **–** ketma ketlik kabi tartiblangan panellar to‘plami orqali elementlarni birlashtirish obyektidir.

Anchor - obyektni aftor yoki oyna bo‘yicha tartiblab joylashtirish xususiyatlarini tanlaydi. Uning to‘rtta qiymati o‘ng, chap, yuqori, past boryu Bularning o‘zaro aralashuvidan foydalaniladi.

Backcolor - obyektning fon xususiyati rangini o‘rnatadi.

AutoSize - xususiyati true/false qiymatlarni qabul qiladi. Elementlarning o‘lchamlariga mos obyektga o‘zlcham ajratadi.

WrapContents – obyektda kontentlar ketma-ketligini ustuvor qilish yoki qilmaslikni o‘rnatadi. Xususiyat true/false qiymatlarni qabul qiladi.

Qo‘shimacha komponentalarning xususiyatlari haqida ma’lumotlarni berildi. Ularning hodisalari bir biriga o‘xshash bo‘lib, standart komponentalardan farqi juda kam. Shuning uchun hodisalarga to‘xtalib o‘tirmaymiz. Shuningdek, qo‘shimacha komponentalar juda ko‘p, buni ko‘rish uchun Tollbox oynasiga sichqonchaning o‘ng tugmasini bosing va [Shown All] buyrug‘ini tanlang.

System komponentasining xususiyatlari va hodisalari. Bu komponentalarga asosan operatsion tizim bilan ishlaydigan komponentalar kiradi. Ularni xususiyatlari murakab hisoblanadi. Chunki birgina menyu yaratish uchun standart va qo‘shimcha komponentalardan ham foydalaniladi. Shuning uchun umumiy holda bu komponentalarning vazifalarini keltirib o‘tamiz.

10.5-jadval. System komponentalarining guruhlari va vazifalari

Komponentaning guruhi	Komponentaning nomi	Vazifasi va izoh
Bo‘yruqlar	ToolStrip	Microsoft Windows, Microsoft Office, Microsoft Internet Explorer bo‘lishi mumkin uskunalar majmuasi yoki foydalanuvchi interfeysini yaratadi. Toolstrip control Asboblar paneli boshqaruvini almashtirishga mo‘ljallangan.
	ToolStripContainer	Toolstrip elementlarini ajralmas to‘plamini beradi (birikib, sohasida gorizontal yoki vertikal almashish)
Ma’lumotni aks ettirish (faqat o‘qish elementlari uchun)	StatusStrip	Windowsning holat satrini beradi Statusstrip - Statusbar o‘rnini bosadi. Statusstrip maxsus xususiyatlari tartibini o‘z ichiga oladi,
Menyuni boshqarish elementlari	MenuStrip	Forma uchun menyu tizimini beradi. Menustrip - Mainmenu o‘rnini bosadigan yuqori darajali

		konteyner hisoblanadi. Bundan tashqari, asosiy ishlov berish va hujjat interfeysi imkoniyatlarini beradi. Funksionaligi Toolstripitem olingan bo‘lsa-da, Toolstripdropdownitem va Toolstripmenuitem bilan birga ishlashlatiladi.
	ContextMenuStrip	Kontekst menyuni ifodalaydi. Contextmenustrip bu Contextmenu o‘rnini bosadi. Contextmenustrip bilan birga ishlatiladi va sichqonchaning o‘ng tugmasini bosishda avtomatik ravishda kontekst menyusi namoyon bo‘ladi.

10.10-rasm. Menyu va uskunalar panellarini yaratish uchun komponentalarning ko‘rinishi.

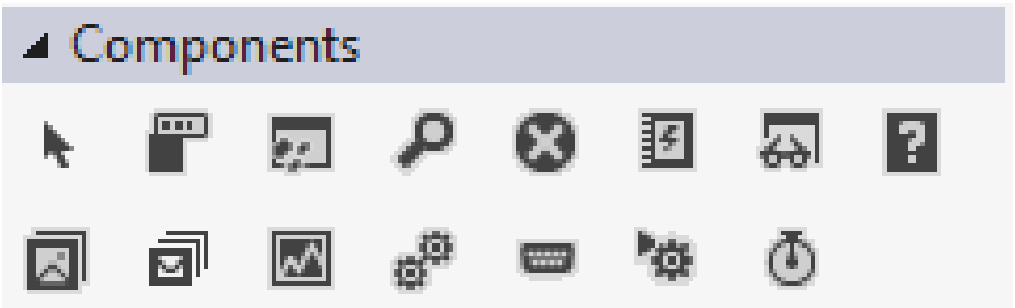
Menyu va uskunalar panellarini yaratish uchun komponentalardan tashqari tizim bilan ishlaydigan komponentalarga quyidagi 10.6-jadvaldagi komponentalar ham qiradi, ular Components tab ga kiritilgan.

10.6-jadval. Components tab ning komponentalari.

Komponentaning nomi	Vazifasi va izoh	
BackgroundWorker	Alohida ajratilgan oqim uchun amal bajara oladigan BackgroundWorker komponentaning nusxasini yaratadi	
DirectoryEntry	Active Directory xizmati bilan o‘zaro bog‘lanib ishlashi va uning iearxiyasidan obyekt yoki tugunlarni inkapsulyatsiya	

	oladigan DirectoryEntry komponentaning nusxasini yaratadi	
DirectorySearcher	Active Directory da so‘rovlarni bajarish imkonini beradigan DirectorySearcher komponentaning nusxasini yaratadi	
ErrorProvider	Formada elementni boshqarishdagi xato foydalanuvchiga ko‘rsatish uchun ErrorProvider komponentaning nusxasini yaratadi	
EventLog	Tizim va foydalanuvchining hodisalar jurnaliga bog‘lanish imkonini beradigan EventLog komponentaning nusxasini yaratadi	
FileSystemWatcher	Ruxsat berilgan papka va fayllarga o‘zgartirishlar kiritish imkonini beradigan FileSystemWatcher komponentaning nusxasini yaratadi	
HelpProvider	Tezkor va foydalanuvchining yordam oynasini chaqirish imkonini beruvchi HelpProvider komponentaning nusxasini yaratadi	
ImageList	Image obyektlarining to‘plamini boshqarish usullarini ta‘minlovchi ImageList komponentaning nusxasini yaratadi	
MessageQueue	Navbatdagi xabarlarni o‘qish imkoniyatini beruvchi MessageQueue komponentaning nusxasini yaratadi	

PerformanceCounter	Windowsning hisoblagichlari bilan ishlash uchun foydalaniladi, yangi toifalar va misollar yaratish, hisoblagichlardan qiymatlarni o‘qish va hisoblagich ma’lumotlariga asoslangan hisob-kitoblarni bajarish uchun PerformanceCounter komponentaning nusxasini yaratadi	
Process	Tizimdagi jarayonlar bilan bog‘liq ma’lumotlarni to‘xtatish, ishga tushirish va o‘zgartirish uchun Process, komponentaning nusxasini yaratadi	
SerialPort	sinxron va voqea imkoniyatlar uchun SerialPort komponentaning nusxasini yaratadi	
ServiceController	Mavjud xizmatlarni, shu jumladan, boshlang‘ich va to‘xtatish xizmatlarini boshqarish va ularga buyruqlar berish uchun ServiceController komponentaning nusxasini yaratadi	
Timer	Windows ilovalar uchun vaqtga asoslangan vazifalarni kiritish uchun Timer komponentaning nusxasini yaratadi	

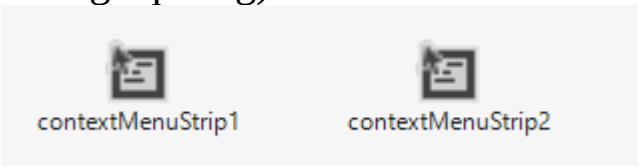


10.11-rasm. Components tab ga kiritilgan komponentalarning ko‘rinishi.

Bu komponentalardan kontkst menyuni yaratish va boshqa obyektlarga o‘rnatishni ko‘rib chiqamiz.

Masala. Oynada ikkita kontekst menyu yaratish kerak. Birinchisi oyna uchun, ikkinchisi RichTextBox obyekti uchun bo‘lsin. Birinchi kontekstda oynani yopish, kattalashtirish, kichiklashtirish, yig‘ishtirish, tugmachalarni yoqish va o‘chirish buyruqlari bo‘lsin. Ikkinchi kontekstda matnni tahrirlash uchun buferga nusxalash, buferga nusxalash va o‘chirish, nusxa qo‘shish, obyektni tozalash, gorozantal va vertikal harakatlanish tugmachalarni qurish va o‘chirish amallari bo‘lsin.

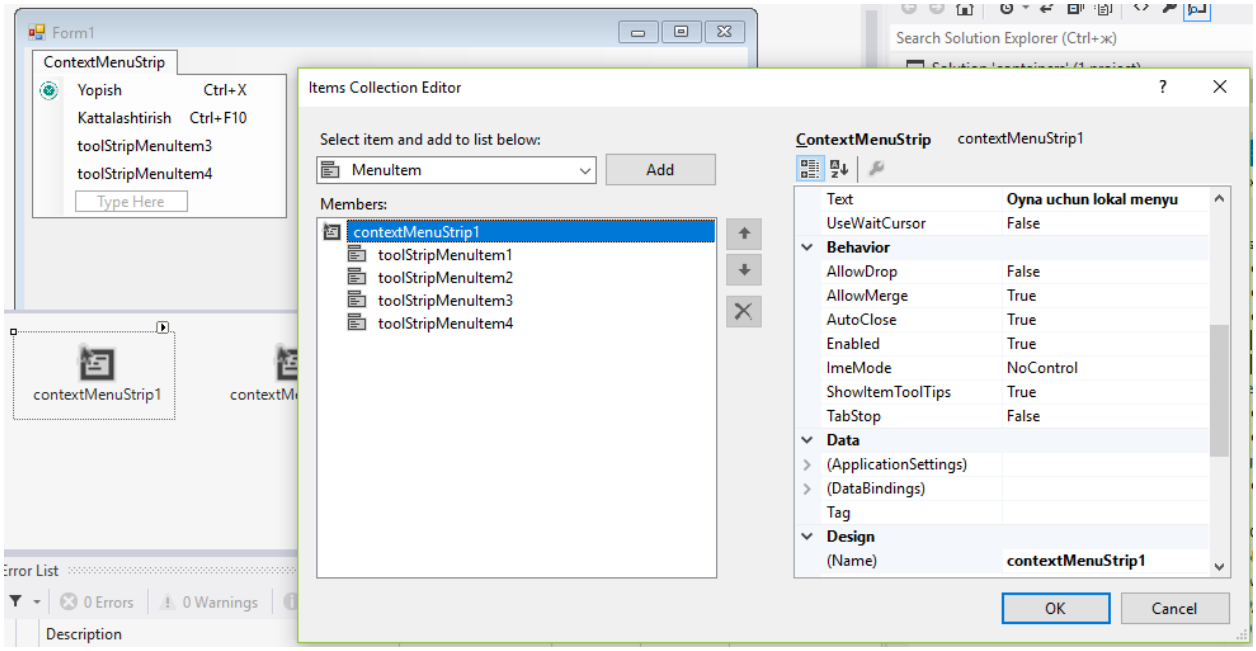
Yangi loyiha yaratamiz va loyihada yagona oyna bo‘ladi. Oynaning bir chetiga RichtextBox ni o‘rnatamiz. Menyu va uskunalar Tabidan contextMenuStrip obyektidan ikkita o‘rnatamiz. Ular formada ko‘rinmasligi mumkin. Ular loyihaning ishchi maydoni pastki qismida joylashadi (10.12-rasmga qarang).



10.12-rasm. contextMenuStrip obyekti o‘rnatilgan ko‘rinishi.

Bularni kerakli obyektlari bilan bog‘laymiz. Oynaga birinchi kontekst menyuni [contextMenuStrip1], [RichTextBox1] obyektga ikkinchi kontekst menyuni [contextMenuStrip2] o‘rnatamiz. Buning uchun oynaning [ContextMenuStrip] xususiyatiga [contextMenuStrip1] ni va [RichTextBox1] obyektning [ContextMenuStrip] xususiyatiga [contextMenuStrip2] ni o‘rnatamiz. Agar bitta kontekst ikkita obyektga o‘rnatilsa, kontekstning buyruqlarga joriy qilingan algoritmlarni bajaraveradi.

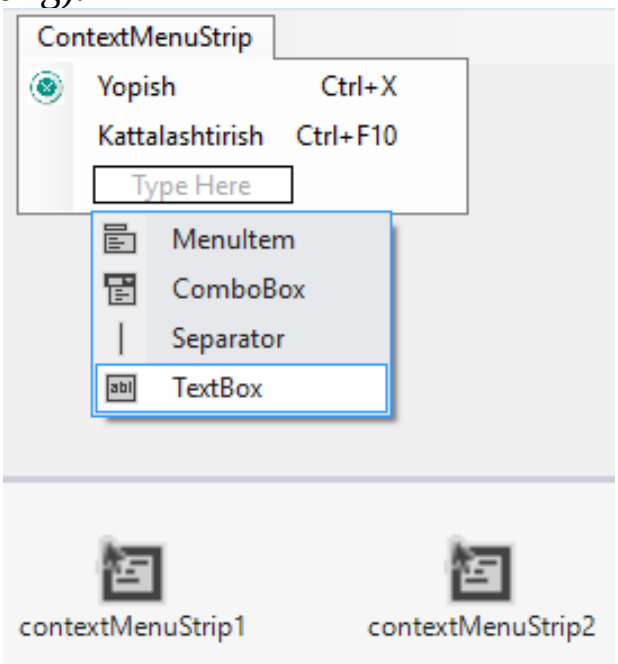
O‘rnatilgan kontekst menyu obyekti [ContextMenuStrip] xususiyati uchun ichki xususiyatlarni, ya‘ni [contextMenuStrip] xususiyatlarni o‘rnatish uchun ochib beradi. Bu xususiyatlarni hammasiga to‘xtalib o‘tmaymiz. Faqat Items xususiyatiga to‘xtalamiz. Shu xususiyat tanlanganda quyidagicha muloqot oynasi chiqadi(10.13-rasmga qarang).



10.13-rasm. Kontekst menyu buyruqlarini o‘rnatish.

E‘tibor qaratsangiz, muloqot oynasida qilingan o‘rnatishlar muloqot oynasida emas balki asosiy oynada ko‘rinadi. Agar bu muloqot oynasida ishlash noqulay bo‘lsa, asosiy ishchi maydonda ham ishlash mumkin.

Buning uchun kerakli kontekst menyu obyekti uchtiga sichqonchani bir marta bosilsa, oynada kontekst menyu hosil bo‘ladi (10.14-rasmga qarang).



10.14-rasm. Oynada kontekst menyuga ishlov berish.

[Type Here] tugmasi orqali joriy [MenuItem], [ComboBox], [Separator], [TextBox] - 4 ta elementni qo‘shish mumkin. Qo‘shilgan har bir elementning o‘ng tugmasini bosib, u bilan ma’lum bajarilishi mumkin bo‘lgan amallari bajarish mumkin. Shuningdek, [properties]

degan xususiyatlarga olib kiradi. [Text] - xususiyatiga kirib buyruq nomini berish mumkin. Agar harf bilan murojaat qilishni xoxlasangiz, kerakli harf oldiga [&] belgisini qo‘yish kerak. Inteaktiv tugma o‘rnatish uchun esa [ShortcutKeys] xususiyatiga kerakli tugmalar kombinatsiyasini o‘rnatish mumkin. Ikonka o‘rnatish uchun [Image] xususiyatga kerakli rasmni o‘rnatish mumkin. Shunday qilib kerakli barcha buyruqlarni amalga oshirish mumkin. Buyruqlarni guruhlash uchun [Separator] komponentasini tanlash kerak. Buyruqlarga algortim yozish uchun ularni ustiga sichqonchani ikki marta bosish yetarli yoki hodisalariga kirib hodisasiga yozish mumkin.

Agar yuqorida kontekst menyuga ishlov berish tushunarli bo‘lgan bo‘lsa, masalani o‘zingiz tugatib qo‘ying. Agar ketma ketlik asosida bajarsangiz albatta masalani hech bo‘lmasa, algoritmlarsiz hal qilishingiz lozim.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Toolbox interaktiv oynasi nima uchun kerak?
2. Qaysi tugmachalar majmuasini ham bosish orqali Toolbox oynasiga o‘tish/chaqirish mumkin?
3. Toolbox oynasini tahrirlash mumkinmi? Mumkin bo‘lsa qanday va nima uchun?
4. Nima Toolbox oynasi uchun xizmat qiladi va 5 ta bo‘limdan iborat?
5. [List View] va [Show All] buyruqlari nima uchun kerak?
6. Toolbox interaktiv oynasi nechta komponenta mavjud?
7. Toolbox interaktiv oynasi komponentalarining o‘rnini almashtirish va qayta nomlash mumkinmi?
8. Komponenta xususiyatlarining asosiy nechta guruhi bor?
9. Komponentalarning Focus guruhi nima uchun ishlatiladi?
10. Asosiy oynaga qo‘shilgan komponentalar ro‘yxatini qayerdan ko‘rish mumkin.
11. Oynaning InitializeComponent funksiyasida ham komponentalarning xususiyatlari ko‘shilganini ko‘rish mumkinmi?
12. Komponentaning nechta usullari (methods) va hodisalari (events) guruhi bor?
13. Komponentaning Layout guruhi nima uchun ishlatiladi?

14. Standart komponentalarga Toolbox oynasining qaysi Tab idagi komponentalar kiradi?
15. Mantlarni tahrirlash komponentalarini sanab bering?
16. Ro‘yxatdan tanlash komponentalarini sanab bering?
17. Komponentalarni oynaga joylashtirish uchun qanday amal bajarish kerak?
18. Label1 obyektining text xususiyatini o‘rnatish qanday amalga oshiriladi?
19. comboBox1 obyektining Items xususiyatlarini o‘rnatish qanday amalga oshiriladi?
20. [numericUpDown1->Value] tanlangan obyektning qanday tipdagi qiymatini qaytaruvchi xususiyat?
21. [comboBox1->SelectedIndex] – tanlagichning tanlangan qanday qiymatini qaytaradi?
22. Additional komponentasining xususiyatlari va hodisalari nima uchun ishlatiladi?
23. TabControl komponentasining vazifasini ayting.
24. Panel komponentasi xususiyatlariga misollar keltiring.
25. Panel komponentasini oynaning qaysi qismida joylashtirish kerakligini o‘rnatadigan xususiyatni ayting.
26. Bu xususiyat to‘plam qiymatlarni qabul qiladi va uning nechta sahifa bo‘lishini ta‘minlaydi. U bilan ishlaganda har bir sahifaning alohida xususiyatlariga ishlov berish mumkin. Gap qaysi komponentaning qaysi xususiyati haqida ketmoqda?
27. System komponentasining xususiyatlari va hodisalarini sanab bering.
28. Menyuni boshqarish elementlari uchun ishlatiladigan komponentalarni sanab bering.
29. Menyu va uskunalar panellarini yaratish uchun komponentalardan tashqari tizim bilan ishlaydigan komponentalar qaysi Tab ga kiritilgan?
30. Kontekst menyuni ishlatishni tushuntirib bering.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASh HAMDA RIVOJLANTIRISh UChUN ASSISMENT TOPShIRIG‘I.

ASSISMENT TOPShIRIG‘I	
	Komponentalar bilan ishlashga oid quyidagi dastur bo‘yicha berilgan topshiriqlarni kerakli

	fragmentlari asosida bajaring. ☝ Bunda dasturdagi ba‘zi o‘zgartirishlarni topish orqali topshiriqlar bosqichma – bosqich amalga oshriladi.

№ - sizning tug‘ilgan kuninggiz

1. Oddiy kalkulyator yarating. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

2. Sanoq sistemalar uchun arifmetik amallarni bajaruvchi kalkulyator yarating. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____

3. Menyuga ega loyiha yarating. U menyu oyna xususiyatlari va hodisalarini boshqarsin. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____

4. Kontekst menyuga ega loyiha yarating. U menyu oyna

xususiyatlari va hodisalarini boshqarsin. Interaktiv tugmalar va klaviatura tugmalari bilan ishlasin. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____

5. Oddiy o‘yin dasturini yarating. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____
_____

6. Bir dasturini yarating. Unda holat satri va uskunalar paneli to‘liq ishlasin. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____
_____

7. Barcha vazifalarni bajarganingiz bo‘yicha o‘ylab ko‘ring, nimalarni o‘rgandingiz va ularni yana bir bor takrorlang.
8. Barcha bilim va ko‘nikmalar asosida kichik loyiha tayyorlang.

3.3. Muloqot oynalari bilan ishlash.

📖 Muloqot oynasi tushunchasi va xususiyatlari, diaolog Tab komponentalarining xususiyatlari va hodisalari, messageBox va uning Show usulidan foydlanish, uning parametrlari va qiymatlari, shaxsiy

muloqot oynalarini yaratish bo‘yicha nazariy va amaliy bilimlarni hamda asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni mustahkamlash uchun 40 ta nazariy savol, amaliy ko‘nikma va malakalarni rivojlantrish uchun 7 assisment topshirig‘i berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

✎ ***Kalit so‘zlar.*** IDE, Visual C++, user interface, OYD, komponenta, xususiyatlar, hodisalar, Toolbox, Tab, muloqot oynasi, ColorDialog, FontDialog, OpenFileDialog, PrintDialog, PrintPreviewDialog, FolderBrowserDialog, SaveFileDialog, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, HelpNavigator, Object, DialogResult, InvalidEnumArgumentException, IWin32Window tiplari.

☑ ***Bilish shart bo‘lgan tushunchalar.*** sinf va sinf obyekti, xususiyat, hodisa, forma, komponenta dasturlashga oid dastlabki tushunchalar hamda C++ tilini qo‘llab-quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Visual C++ muhitida dasturlash, integrallashgan ishlab chiqarish muhiti, Toolbox, muloqot oynasi tushunchasi va xususiyatlari, ColorDialog, FontDialog, OpenFileDialog, PrintDialog, PrintPreviewDialog, FolderBrowserDialog, SaveFileDialog muloqot oynalari bilan ishlash, MessageBox sinfi va uning Show funksiyasi qabul qilishi mumkin bo‘lgan String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, HelpNavigator, Object, DialogResult, InvalidEnumArgumentException, IWin32Window tiplari va qiymatlari, funksiyani yozilish variantlari, muloqot oynasini loyiha boshqaruv oyna orqali yaratish va muloqot oynalariga qo‘yiladigan talablar, o‘rnatish va komponentalar bilan bog‘lash usullari, shuningdek ba‘zi bir xususiyatlar va hodisalarini o‘rganishingiz mumkin.

REJA

1. Visual C++ muhitida muloqot oynalari.
2. Muloqot oynalarini sozlash.
3. Muloqot oynalarining boshqarish elementlari.
4. Muloqot oynalarini yaratish.

KIRISH

Foydalanuvchilar bilan tizimning muloqotini interaktiv amalga oshiri uchun muloqot oynalari kerak. Muloqot oynalari 3 ta katta guruhlarga bo‘linadi. Tizimli, ya‘ni OT bilan ishlashga mo‘ljallangan,

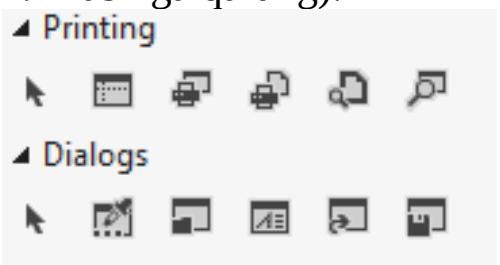
interaktiv xabarlarni berish va aniq javoblarni olish uchun mo‘ljallangan, dasturchining yoki foydalanuvchining tashabbusi bilan yaratiladigan muloqot oynalari bor.

Visual C++ muhitida muloqot oynalari. Yuqorida aytib o‘tilgandek muloqot oynalarini yaratish maqsaddan kelib chiqqan holda amalga oshiriladi. Bu oynalarning o‘zining talablari mavjudki, shu talablar bajarilsa, u muloqot oynasi bo‘la oladi. Bu talablarga quyidagilar kiradi:

1. Muloqot oynasining sarlavhasi bo‘lishi va unda faqat oynani yopish tugmasining bo‘lishi lozim, tizimli menyu va boshqa tugmalar bo‘lishi mumkin emas. Istisno tariqasida ba‘zi hollarda, yordam tugmasini joylashtirish mumkin.
2. Muloqot oynasi teskari aloqaga mo‘ljallanganligi uchun, uni shartini bajarmasdan tizimning boshqa oynasiga o‘tish mumkin emas.
3. Teskari aloqaning bir nechta turlarini amalga oshiruvchi tugmalar bo‘lishi kerak.
4. Muloqot oynalar asosiy oynadan har doim kichik bo‘lshi shart.
5. Muloqot oynalarining asosiy maqsadi aniq kelitrilishi kerak yoki turi, nima munosabat uchun muloqot oynasi chiqqanligi.
6. Muloqot oynasidan boshqa muloqot oynasiga o‘tish mumkin emas, asosiy oynaga o‘tish lozim.
7. Muloqot oyna murojaat qilinganda yaratilishi va teskari aloqa qabul qilingandan so‘ng xotiradan o‘chirib tashlanishi lozim.
8. Muloqot oynaga dinamik xotiralar bo‘lishi mumkin emas.

Ushbu talablarni bajargan har qanday oyna muloqot oynasi hisoblanadi.

Visual C++da OT bilan muloqot qilishga mo‘ljallangan muloqot oynalariga [Dialogs] Tab dagi va [Printing] tab dagi barcha komponentalar kiradi (11.1-rasmga qarang).



11.1-rasm. Tizimli muloqot oynalari yaratish komponentalari.

Visual C++ning hujjatlariga qarasangiz tizimli muloqot oynalari uchun yagona muloqot oynalari ro‘yxati tuzilgan. Bu ro‘xatga quyidagi jadvalda keltirilgan komponentalar kiradi.

11.1-jadval.Visual C++da OT bilan muloqot qilishga mo‘ljallangan muloqot oynalarini yaratishga mo‘ljallangan komponentalarning vazifalari

№	Komponenta nomi	Vazifasi
1	ColorDialog	Foydalanuvchilar interfeys elementi rangini o‘rnatish imkonini beruvchi ranglar palitrasi uchun muloqot oynasini ko‘rsatadi.
2	FontDialog	Foydalanuvchilarga kerakli komponenta uchun shrift va uning xususiyatlarini o‘rnatish imkonini beruvchi muloqot oynasini ko‘rsatadi.
3	OpenFileDialog	Foydalanuvchilar uchun faylni tanlash imkonini beradigan muloqot oynasini ko‘rsatadi.
4	PrintDialog	Foydalanuvchilarga printerni tanlash va uning xususiyatlarini o‘rnatish imkonini beruvchi muloqot oynasini ko‘rsatadi.
5	PrintPreviewDialog	Foydalanuvchilar uchun chop qilishda PrintDocument boshqaruv elementining ko‘rinishining ko‘rsatish imkonini beradigan muloqot oynasini ko‘rsatadi.
6	FolderBrowserDialog	Foydalanuvchilar uchun papkalar ko‘rish, yaratish va tanlash imkonini beradigan muloqot oynasini ko‘rsatadi.
7	SaveFileDialog	Foydalanuvchilar uchun faylni saqlash imkonini beradigan muloqot oynasini ko‘rsatadi.

Bu komponentalar maxsus xususiyatga asoslagan koponentalar bilan ishlatiladi.

Interaktiv xabarlarni berish va aniq javoblarni olish Visual C++ da MessageBox sinfi mavjud. Bu sinf bilan barcha ixtiyoriy turdagi muloqot oynalari yaratish mumkin. Sinfning nomlar fazosi System.Windows.Forms bo‘lib hisoblanadi va kutubxonasi System.Windows.Forms.dll hisoblanadi. Bu muloqot oynasi forma sinfining meros xo‘ri hisoblanadi. Unda 21 ta turli kombinatsiyali show funksiyachi bor. Uning quyidagi parametrlari bor.

11.2-jadval. MessageBox sinfnining show funksiyasi parametrlari

Paramert nomi	Tipi	vazifasi
text	String	Muloqot oynasining xabari

caption	String	Muloqot oynasining sarlavhasi
buttons	MessageBoxButtons	Teskari aloqani ta‘minalash tugmalari turlarini aniqlash
icon	MessageBoxIcon	Muloqot oynalarining ikonkalari turlarini aniqlash
defaultButton	MessageBoxDefaultButton	Teskari aloqani ta‘minalash uchun joriy tugmalari turlarini aniqlash
options	MessageBoxOptions	Muloqot oynalarining amallari turlarini aniqlash
helpFilePath	HelpNavigator	HelpNavigator obyektining qiymatlari uchun foydalaniladi
param	Object	Yordam tugmasi bosilganda ID qiymatni aniqlash imkonini beruvchi parametr
Returns	DialogResult	Muloqot oynalarining qiymatlari qaytarish uchun foydalaniladigan DialogResult tipidagi turlarini aniqlash
Exceptions	InvalidEnumArgumentException	MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton obyektlarini birini qabul qiluvchi kengaytirilgan obyekt tipi
owner	IWin32Window	muloqot oynasini egasi Iwin32window amalga oshirish.

Bu parametrlarning o‘ziga mos qiymatlari oldindan aniqlab berilgan bo‘lib, oddiy sodda ko‘rinishda muloqot oynasini yaratish

imkoni beradi. Keyinroq bu sinf parametrlari qiymatlari va ularga ishlov berishni ko‘rib chiqamiz.

Foydalanuvchi tomonidan yaratiladigan muloqot oynasi forma kabi yaratiladi va ularni loyhalash dasturchining yoki foydalanuvchining xoxishiga qarab amalga oshiriladi. Interaktiv muloqot oynalari kabi interaktiv tugmalarni yaratish va ularni boqarish, kerakli ma’lumotlarni olish uchun ishlatiladi. Shuni ham inobatga olish kerakki yaratiladigan muloqot oynasi talablarga mos kelishi kerak. Bu talablarni amalga oshirish uchun formaning xususiyatlariga ishlov berish, lozim bo‘lsa, asosiy oynada kerakli xususiyatlarni o‘rnatish mumkin. Bunda ham formaning show usuli mavjud bo‘lib, shu orqali forma chaqiriladi.

Sinfning nomlar fazosi System.Windows bo‘lib hisoblanadi va kutubxonasi PresentationFramework.dll hisoblanadi. Bu muloqot oynasi forma sinfining meros xo‘ri hisoblanadi.

.NET 5 Preview 1, .NET Core 3.1, 3.0 va .NET Framework 4.8 4.7.2 4.7.1 4.7 4.6.2 4.6.1 4.6 4.5.2 4.5.1 4.5 4.0 3.5 3.0 versiyalarida qo‘llab quvvatlanadi. Foydalanuvchi muloqot oynalarini yartaishni keyinroq batavsil ko‘rib chiqamiz.

Muloqot oynalarini sozlash. Bu mulovot oynalarini sozlash uchun tizimli muloqot oynalaridan foydalanish va ularga ishlov berish nazarda tutilgan. Yuqorida keltirilgan 7 ta muloqot oynalarinidan foydalanishlar, xususiyatlarini va hodisalarini boshqarish to‘g‘risida to‘xtalamiz.

1.ColorDialog muloqot oynasi. Bu oyna - foydalanuvchilar interfeys elementi rangini o‘rnatish imkonini beruvchi ranglar palitrasi uchun muloqot oynasini ko‘rsatadi. Bu komponentdani formaga o‘rnatilganda hech qanday ko‘rinish hosil bo‘lmaydi, ammo formaning ichki tuzilmasiga qo‘shiladi. Formaning ishchi holatidagi formasining pastki qismida uning obyekti yaratiladi. ColorDialog1 obyektini yaratish orqali boshqariladi. Uning xususiyatlari va hodisalari ham mavjud va loyiha oynasida foydalanuvchi xususiyatlar oynasiga chiqadi. U yerdan kerakli ixtiyoriy aynan shu obyektga mos xususiyat va hodisalarni o‘rnatish mumkin.

ColorDialog muloqot oynasidan foydalanish uchun ColorDialog() konstruktori ishga tushirish lozim.

Bu sinfning xususiyailari, usullari va hodisalari mavjud.

11.3-jadval. ColorDialog muloqot oynasining xususiyatlari

AllowFullOpen	Maxsus ranglar aniqlash uchun muloqot

	oynasini foydalanish mumkin yoki yo‘qligini o‘rnatish
AnyColor	Muloqot oynasida asosiy ranglar majmuining barcha mavjud ranglar ko‘rsatish yoki yo‘qligini o‘rnatish
CanRaiseEvents	Komponentaga bir hodisa o‘rnatish mumkinligini aniqlash.
Color	Foydalanuvchi tomonidan tanlangan rangni o‘rnatish.
Container	Komponentini o‘z ichiga olgan Icontaineni o‘rnatish
CustomColors	Muloqot oynasida ko‘rsatilgan maxsus ranglar to‘plamini oladi yoki o‘rnatadi.
DesignMode	Komponenta joriy dizayn rejimida ekanligini ko‘rsatadigan qiymatni oladi.
Events	Komponentaga ilova qilinadigan hodisalar ro‘yxatini oladi.
FullOpen	Muloqot oynasi ochilganda maxsus ranglarni yaratish uchun ishlatiladigan boshqaruv elementlari ko‘rinib turishini ko‘rsatuvchi qiymatni oladi yoki o‘rnatadi.
Options	Colordialog boshlash uchun xususiyatlarni oladi.
ShowHelp	Yordam tugmasi rang muloqot oynasidagi paydo yoki yo‘qligini bo‘lishini o‘rnatish
Site	Komponentning ISiteni oladi yoki o‘rnatadi.
SolidColorOnly	Muloqot oynasi qattiq ranglar tanlash uchun foydalanuvchilar cheklash yoki yo‘qligini o‘rnatish
Tag	Nazorat haqida ma’lumotlarni o‘z ichiga olgan obyekt sozlash.

11.4-jadval. ColorDialog muloqot oynasining usulari

CreateObjRef(Type)	Obyekt bilan muloqot qilish uchun ishlatiladigan proksi ishlab chiqarish uchun zarur bo‘lgan barcha tegishli ma’lumotlarni o‘z ichiga olgan obyekt yaratadi.

Dispose()	Komponent tomonidan ishlatiladigan barcha resurslarni chiqaradi.
Dispose(Boolean)	Tarkibiy qism tomonidan ishlatiladigan boshqarilmaydigan resurslarni chiqaradi va ixtiyoriy ravishda boshqariladigan resurslarni chiqaradi.
Equals(Object)	Belgilangan obyekt joriy obyektga teng yoki yo‘qligini aniqlaydi.
GetHashCode()	Standart hesh funksiyasi sifatida xizmat qiladi.
GetService(Type)	Komponenta yoki uning konteyneri tomonidan taqdim etilgan xizmatni ifodalovchi obyektni qaytaradi.
GetType()	Joriy obyekt turini oladi.
HookProc(IntPtr, Int32, IntPtr, IntPtr)	Umumiy muloqot oynasiga xos funksiyalarni kiritish uchun bekor qilingan umumiy muloqot oynasi protsedurani belgilaydi.
MemberwiseClone()	Joriy obyektning oddiy nusxasini yaratadi.
MemberwiseClone(Boolean)	Joriy Marshalbyrefobject obyekt oddiy nusxasini yaratadi.
OwnerWndProc(IntPtr, Int32, IntPtr, IntPtr)	Umumiy muloqot oynasiga maxsus funksiyalarni qo‘shish uchun bekor qilinadigan oyna tartibini belgilaydi.
Reset()	Ularning standart xususiyatlarga barcha imkoniyatlari ishga soladi, oxirgi tanlangan qora rang va ularning standart xususiyatlarga maxsus ranglar moslashtiriladi
RunDialog(IntPtr)	Umumiy muloqot oynasini belgilaydi.
ShowDialog()	Umumiy muloqot oynasini ishlatadi.
ShowDialog(IWin32Window)	Umumiy muloqot oynasini ishlatadi.
ToString()	Muloqot oyna qiymatini satrga o‘tkazish.

11.5-jadval. ColorDialog muloqot oynasining hodisalari

Disposed	Komponentni Dispose() metodiga chaqiriq orqali dispozitsiya qilinganda yuzaga keladi

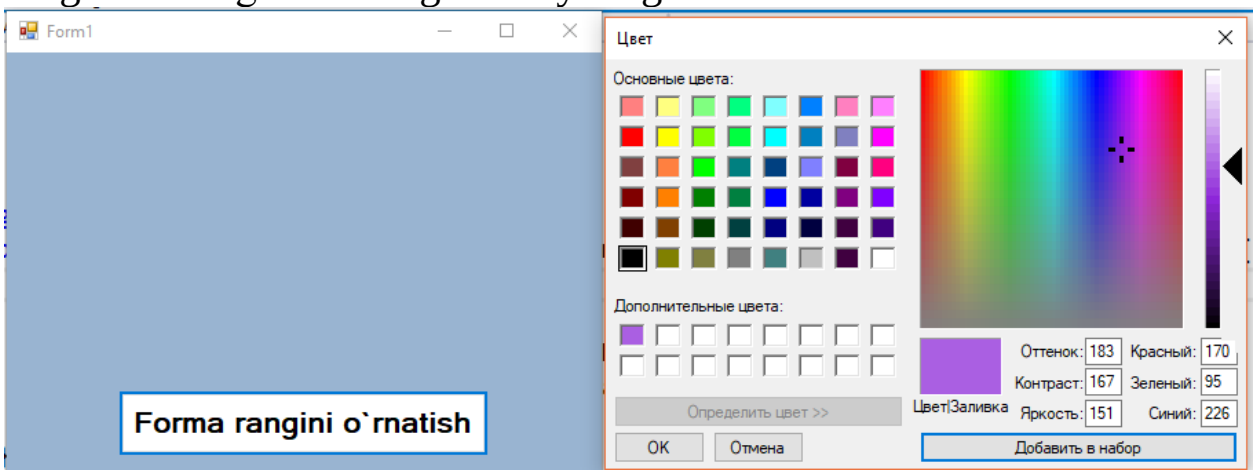
HelpRequest	Foydalanuvchi umumiy muloqot oynasidagi yordam tugmasini bosganda sodir bo‘ladi.

ColorDialog muloqot oynasidan foydalanish uchun formaga bir tugma o‘rnatamiz va uning Click hodisasi yordamida chaqiramiz. Tanlangan rang esa formaning va tugmaning fonini o‘zgartirsin.

Tugmaning Click hodisasi quyidagicha dastur franmentini o‘rnatamiz.

```
if(colorDialog1->ShowDialog() ==
::System::Windows::Forms::DialogResult::OK)
        Form1::BackColor = colorDialog1->Color;
        button1->BackColor = colorDialog1->Color;
```

Dastur fragmentida muloqot oynachi chaqirilganda va teskari aloqasi ok obyektini qaytarsa forma va tugmaning mos xususiyatlar ranglarini o‘zgartirish algoritmi yozilgan.



11.2-rasm. ColorDialog muloqot oynasidan foydalanish.

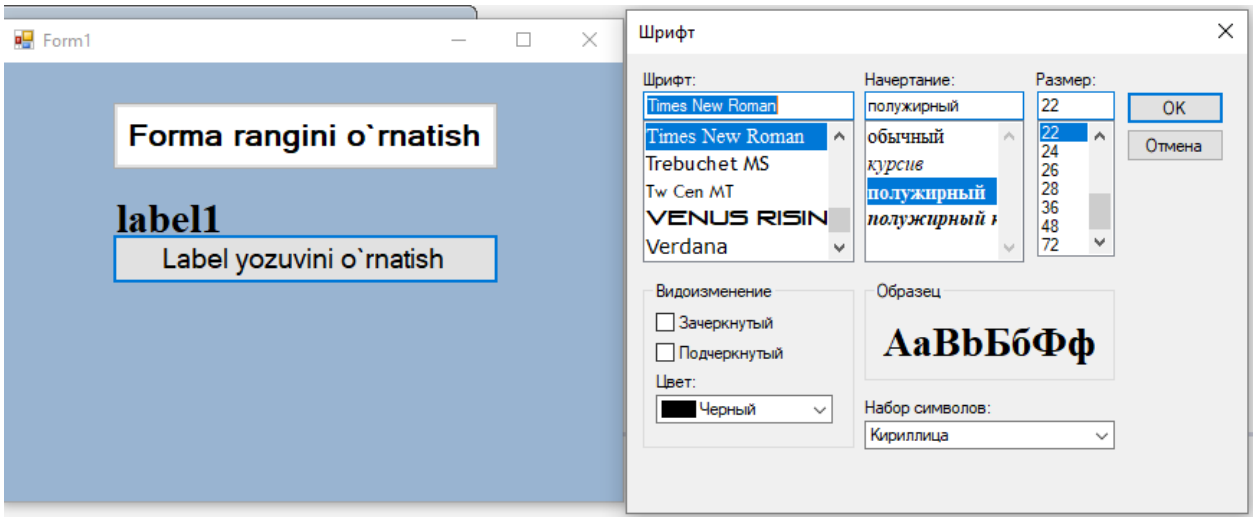
ColorDialog muloqot oynasining xususiyatlari, usullari va hodipsalarini masalaning ahamiyatiga qarab ishlatish mumkin.

2.FontDialog muloqot oynasi. Bu oyna – foydalanuvchilarga kerakli komponenta uchun shrift va uning xususiyatlarini o‘rnatish imkonini beruvchi muloqot oynasini ko‘rsatadi. Bu komponentdani formaga o‘rnatilganda hech qanday ko‘rinish hosil bo‘lmaydi, ammo formaning ichki tuzilmasiga qo‘yshiladi. Fomraning ishchi holatidagi formasining pastki qismida uning obyekti yaratiladi.

Bu muloqot oynasini ishlatish uchun formaga bir Label va button obyektlarini o‘rnatamiz. Tugma bosilganda Label obyektining matnini yozuv xususiyatlarini o‘rnatishni ko‘rib chiqamiz. Buning uchun tunmaning bosilganda xususiyatiga quyidagi dastur fragmentni yozish yetarli.

```
fontDialog1->ShowColor = true;
fontDialog1->Font = label1->Font;
fontDialog1->Color = label1 ->ForeColor;
Color color = label1->ForeColor;
System::Drawing::Font^ font = label1->Font;
System::Windows::Forms::DialogResult result =
fontDialog1->ShowDialog();
if(result ==
::System::Windows::Forms::DialogResult::OK) {
    label1->Font = fontDialog1->Font;
    label1->ForeColor = fontDialog1->Color;
    }
```

Dastuda dastlab Label obyektning rang va yozuvlarini saqlab olinadi, chunki yozuvni formatlash muloqot oynasi chaqirilganda joriy holatni olish uchun. Dasturda rang va yozuv qiymatlarini saqlash uchun o‘zgaruvchilarni aniqlash olib ham keltirilgan. Muloqot oynasining [OK] hodisasi bajarilganda yozuv va rangni o‘zgartirish ko‘rsatilgan.



11.3-rasm. FontDialog muloqot oynasidan foydalanish.

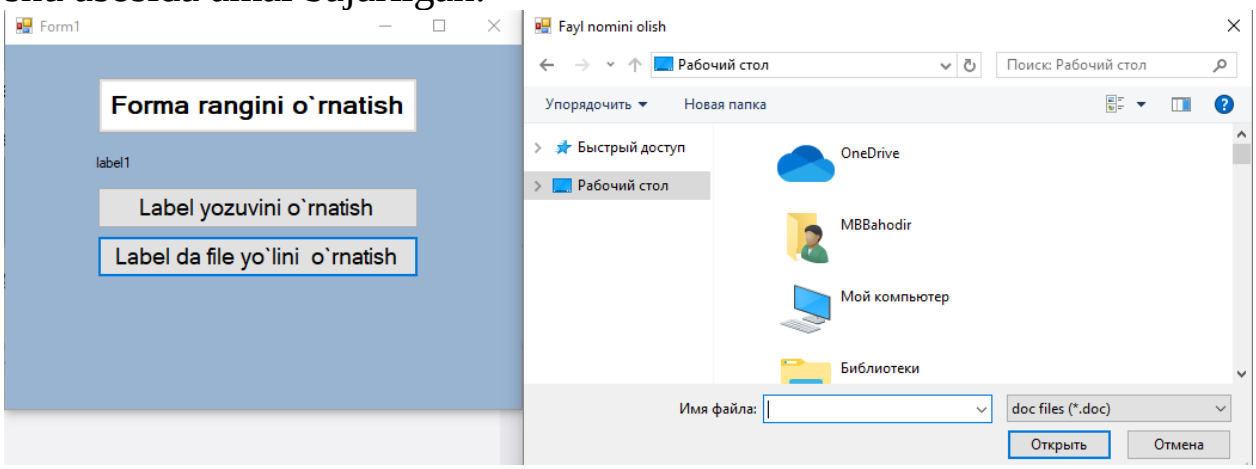
Bu sinfning xususiyailari, usullari va hodisalari mavjud. Ular amaliy vazifalarni bajarishda foydalanish mumkin va mustaqil o‘rganish lozim. Chunki bir vazifani amalga oshirish uchun turli xil algoritmlardan foydalanish mumkin.

3.OpenFileDialog muloqot oynasi. Bu oyna - foydalanuvchilar uchun faylni tanlash imkonini beradigan muloqot oynasini ko‘rsatadi. Bu muloqot oynasi ham yuqoridagidek foydalaniladi. Shuningdek, sinfning mos xususiyatlari, usullari va hodisalari mavjud. Farqli xususiyatlarni ko‘rsatish uchun bir misol olamiz. Unda tugma

bosilganda label matniga faylning to‘liq yo‘lini olish va o‘rnatish uchun quyidagi dastur fragmentini yoziladi.

```
openFileDialog1->InitialDirectory = "c:\\";
openFileDialog1->Filter = "txt files
(*.txt)|*.txt|doc files (*.doc)|*.doc,*.docx|All
files (*.*)|*.*";
openFileDialog1->FilterIndex = 2;
openFileDialog1->RestoreDirectory = true;
openFileDialog1->Title = "Fayl nomini olish";
if ( openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK ){
 if (openFileDialog1->OpenFile() != nullptr ){
    label1->Text = openFileDialog1->FileName;
    }
 }
```

Dasturning birinchi satri joriy katalog o‘rnatiladi. Shuningdek, joriy foydalanuvchining kerakli kataloglarini ham o‘rnatish qiymatlari bor. So‘ng, fayllarni tipi bo‘yicha filtrlash o‘rantiladi. Hozirda 3 ta format o‘rnatilgan. Keyingi satrda fayl kengaytmalaridan qaysi biri joriy bo‘lib chiqishini belgilash amalga oshirilgan. Ko‘rsatilgan katalogni faollashtirish bajarilgan va muloqot oynasining sarlavhasida kerakli matn joylashtirilgan. Muloqot oynaning teskari aloqasi tekshirilgan va shu asosida amal bajarilgan.



11.4-rasm. FontDialog muloqot oynasidan foydalanish.

Odatda bu muloqot oynasi ma’lum tizimsha fayllarni matnini joylashtirish uchun ishlatiladi. Buni qanday amalga oshirish mumkin. Buning uchun kichik bo‘lsa ham matn muharriri yaratish lozim va unga OTdan oddiy manli fayllarni yuklab olish mumkin. Fayl ichidagi ma’lumotlarni olish uchun quyidagicha dastur fragmenti yozish mumkin:

```
openFileDialog1->Filter = "text files
(*.txt)|*.txt|cpp files (*.cpp)|*.cpp|All files
(*.*)|*.*";
openFileDialog1->FilterIndex = 2;
openFileDialog1->RestoreDirectory = true;
openFileDialog1->Title = "Fayl nomini olish";
openFileDialog1->FileName = "";
if ( openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK){
    if ( (myStream = openFileDialog1->OpenFile())
!= nullptr ){
    System::IO::StreamReader ^ sr = gcnew
System::IO::StreamReader(openFileDialog1-
>FileName);
    richTextBox1->Text = sr->ReadToEnd();
        sr->Close();
    }
}
```

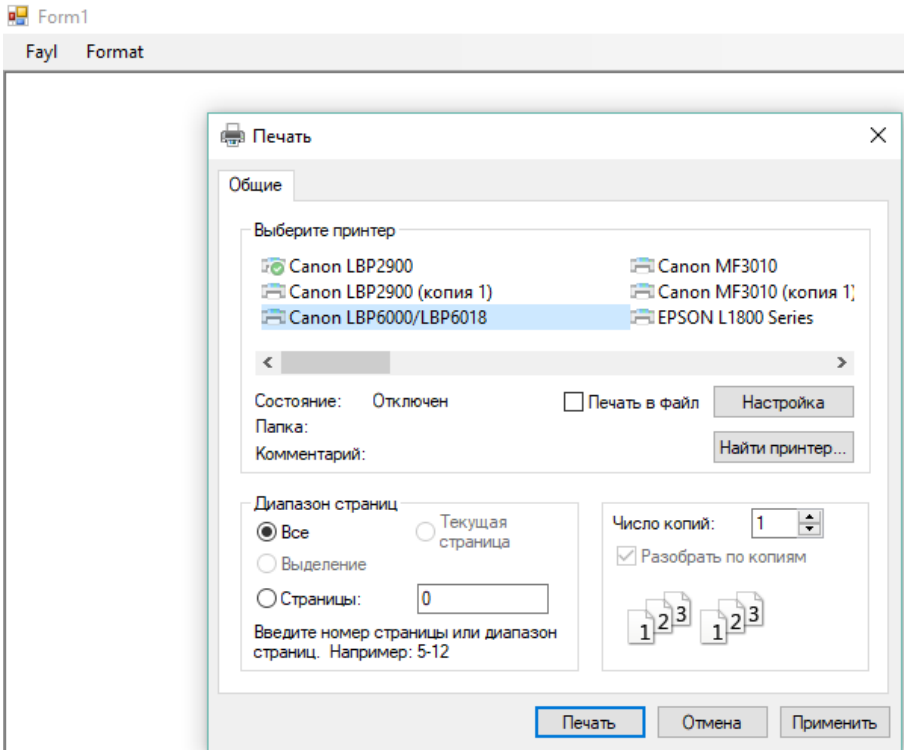
Bu amaliy dastur fragmentini amaliyotga sinab ko‘rish orqali tahlil qiling.

4.PrintDialog muloqot oynasi. Bu oyna - foydalanuvchilarga printerni tanlash va uning xususiyatlarini o‘rnatish imkonini beruvchi muloqot oynasini ko‘rsatadi. Oynaning juda ko‘p xususiyatlari, usullari va hodisalari bor. Bularni mustaqil ishlarni bajarish vaqtida amalga oshirish mumkin. Shuningdek, bu muloqot oynani ishlatish usullari ham juda ko‘p. Odatda dasturchining loyiha holatidan kelib chiqqan holda foydalaniladi.

```
     printDialog1->AllowSomePages = true;
        printDialog1->ShowHelp = true;
   if ( printDialog1 == nullptr )
   System::Windows::Forms::MessageBox::Show("pnull"
);
   System::Windows::Forms::DialogResult result =
printDialog1->ShowDialog();

System::Windows::Forms::MessageBox::Show(result.ToStr
ing());
```

```
   if ( result ==
System::Windows::Forms::DialogResult::OK )
   {
     // docToPrint->Print();
     System::Windows::Forms::MessageBox::Show("Chop
qilish boshlanadi");
   }
```

Dastur fragmentida ma’lumotlarni chop qilish uchun avval uni ma’lum bir chiquvchi oqimga yozish va oqimni esa, hujjat formatiga joylashtirish kerak. Hujjatni esa, sahifalarni sozlab docToPrint obyekt yaratilishi kerak. Ammo chop qillishning turli parametrlarini ishlatish uchun muloqot oynasini chiqarib beradi. Kerakli xususiyatlarni o‘rnatgandan so‘ng amalni bajarish mumkin.



11.5-rasm. PrintDialog muloqot oynasidan foydalanish.

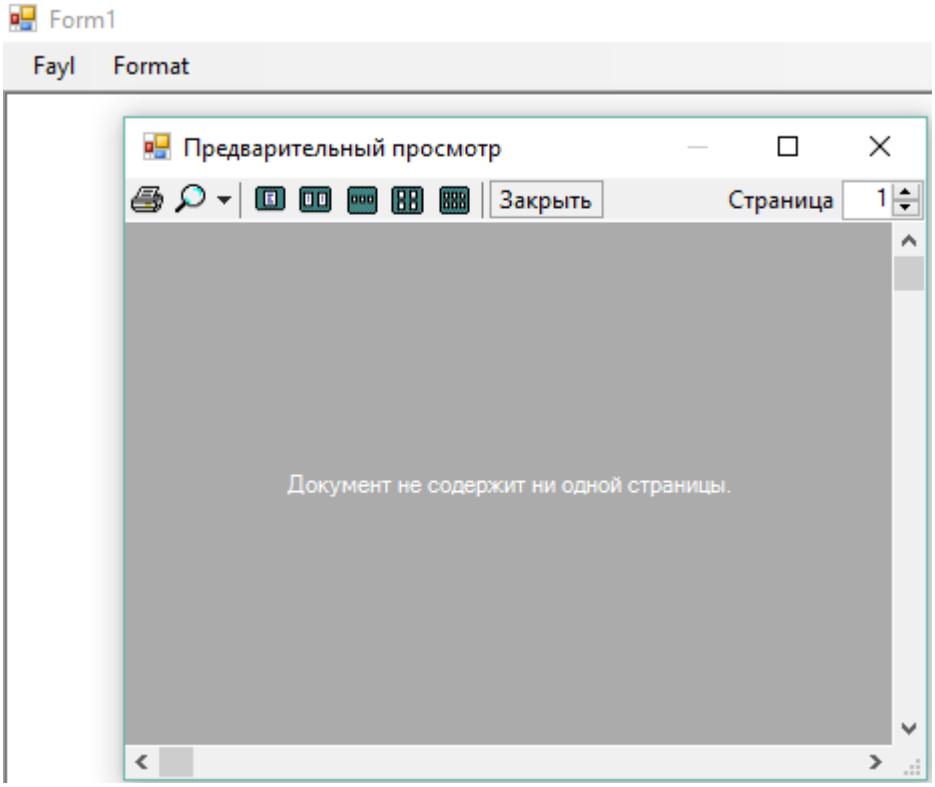
Dastur fragmentini bir loyiha joylashtirib, ishlatsangiz rasmdagidek ishlashi kerak.

5.PrintPreviewDialog muloqot oynasi. Bu oyna - foydalanuvchilar uchun chop qilishda PrintDocument boshqaruv elementining ko‘rinishining ko‘rsatish imkonini beradigan muloqot oynasini ko‘rsatadi.

```
printPreviewDialog1->MinimumSize =
System::Drawing::Size( 375, 250 );
            printPreviewDialog1->UseAntiAlias =
```

```
true;
                printPreviewDialog1->Document =
document;
                        printPreviewDialog1-
>ShowDialog();
```

Bu muloqot oynasi ham yuqoridagi muloqot oynasi kabi sozlashlarni bajarshgandan so‘ng foydalanish mumkin. Dastur fragmentida qarasangiz document obyektini yaratish lozim. Chop qilishninig obyektidan farq qilgan xolda oqimdagi ma’lumotni formatlash va uni chiqishini A4 shaklga keltrish orqali amlga oshiriladi.



11.6-rasm. PrintPreviewDialog muloqot oynasidan foydalanish.

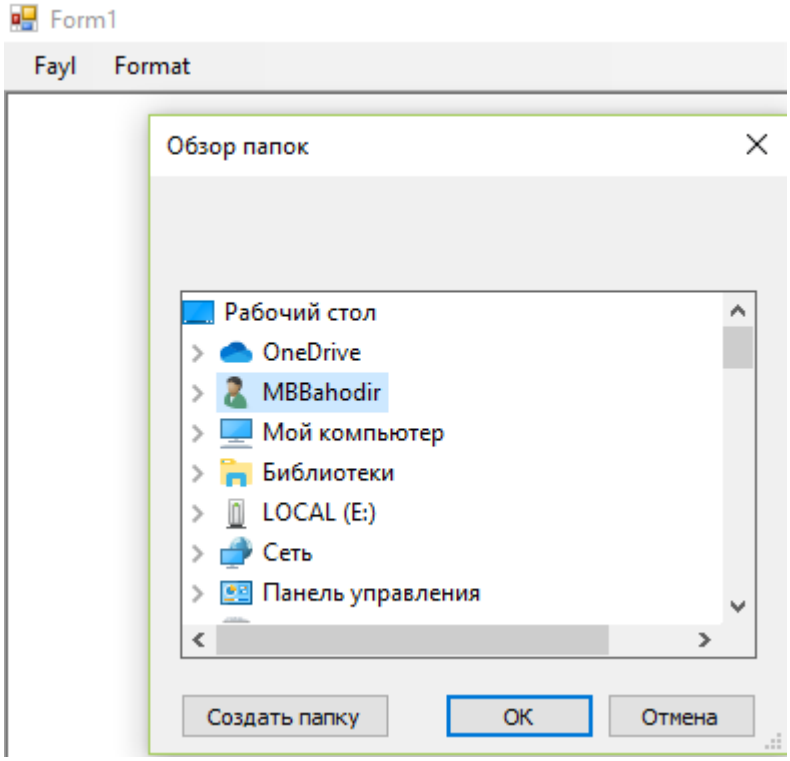
6.FolderBrowserDialog muloqot oynasi. Bu oyna - foydalanuvchilar uchun papkalar ko‘rish, yaratish va tanlash imkonini beradigan muloqot oynasini ko‘rsatadi. Bundan ma’lumotlarni ko‘chirishda, papkalarni taqqsolashda ishlatish mumkin. Muloqot oynasining xususiyatlari, usullari va hodisalar mavjud. Ularning ba‘zilarini quyidagi dastur fragmentiga keltirib o‘tamiz.

```
System::IO::Stream ^ myStream;
System::Windows::Forms::DialogResult result =
folderBrowserDialog1->ShowDialog();
 if ( result ==
System::Windows::Forms::DialogResult::OK ) {
    System::String^ folderName =
folderBrowserDialog1->SelectedPath;
```

```
    openFileDialog1->InitialDirectory = folderName;
    openFileDialog1->FileName =
String::Concat(folderName,"\\1.cpp");
     if ( (myStream = openFileDialog1->OpenFile())
!= nullptr ){
     System::IO::StreamReader ^ sr = gcnew
System::IO::StreamReader(openFileDialog1-
>FileName);
     richTextBox1->Text = sr->ReadToEnd();
    sr->Close();
     }
}
```

Dastur fragmentida papka uchun muloqot oynasi chaqirilgan va ko‘rsatilgan papkadan 1.cpp faylini yuklab kelgan. Bunday holat fayl menejerlar uchun papkadagi fayllarning ro‘yhatini ham olish mumkin.

Dastur fragmentida bir StreamReader oqim yaratilgan. Oqim faylning ma’lumotlarini o‘qish uchun yaratilgan. Oqimning konstruktori asosida sr oqim obyekti yaratiladi. Bu oqimning ReadToEnd funksiyasi orqali richTextBox ga ma’lumotlar joylashtiriladi. FolderBrowserDialog muloqot oynasining ko‘rinishi quyidagicha:



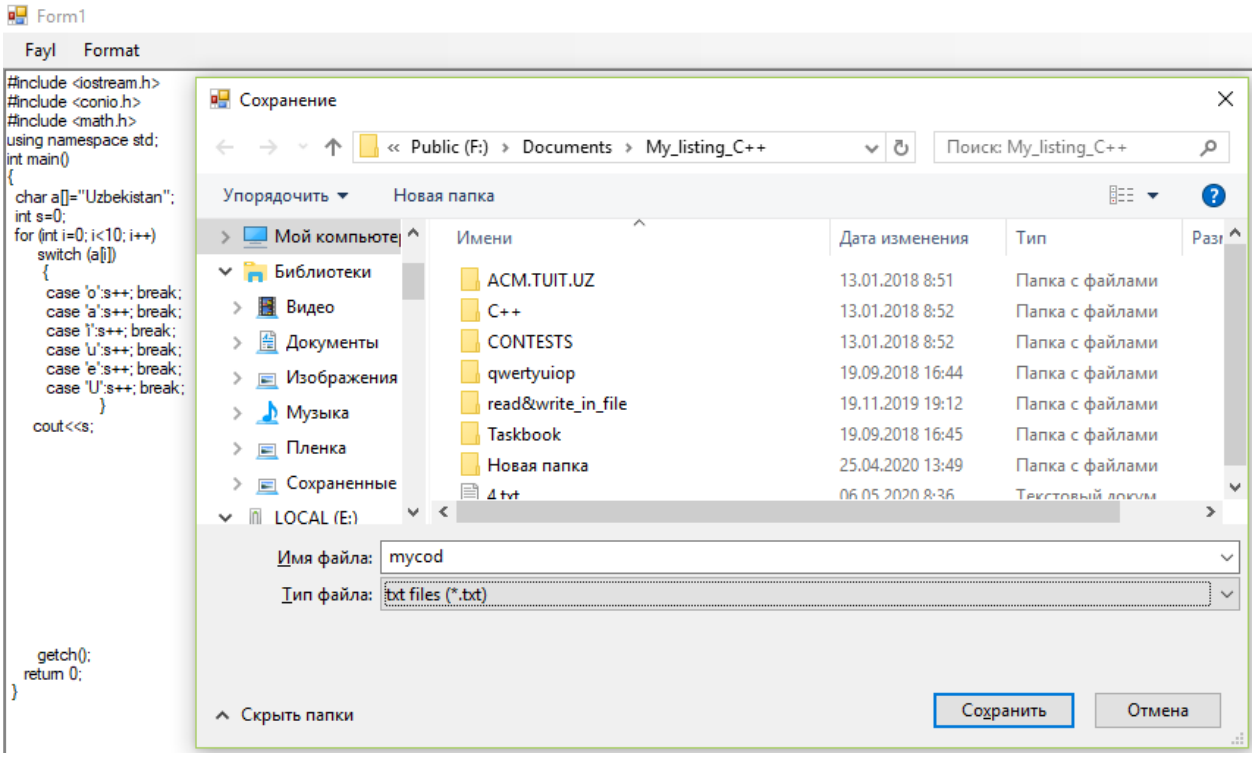
11.7-rasm. FolderBrowserDialog muloqot oynasidan foydalanish

7.SaveFileDialog muloqot oynasi. Bu oyna - foydalanuvchilar uchun faylni saqlash imkonini beradigan muloqot oynasini ko‘rsatadi. Bu ham asosan matnli va maxsus tuzilmalari ma’lumotlarini saqlash uchun ishlatiladi. Ma’lumot qanaqa tuzilmada yozilsa, shunday tuzilmada o‘qiladi.

```
System::IO::Stream^ myStream;
        MemoryStream^ userInput = gcnew
MemoryStream();

        richTextBox1->SaveFile( userInput,
RichTextBoxStreamType::PlainText );
        userInput->WriteByte( 32 );

     // saveFileDialog1->CreatePrompt = true;
        // saveFileDialog1->OverwritePrompt = true;
        // saveFileDialog1->FileName = "myText";
        // saveFileDialog1->DefaultExt = "txt";

      saveFileDialog1->Filter = "txt files
(*.txt)|*.txt";
      saveFileDialog1->FilterIndex = 2;
      saveFileDialog1->RestoreDirectory = true;
if ( saveFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK ){
          myStream = saveFileDialog1->OpenFile();
          userInput->Position = 0;
          userInput->WriteTo( myStream );
          myStream->Close();

      }
```

Dastur fragmentida ikki xil oqim yaratilgan. myStream - birinchisi an‘anaviy oqim va MemoryStream -xotirali oqim yaratilgan. Biri ma’lumotlarni olish va ikkinchisi ma’lumotlarni xotiraga yozish, berilgan nom bilan nomlashgan qaratilgan. Position bu poizitsiyaani boshlash, ammo olib tashlasa ham ishlaydi.

Bu muloqot oynasi nafaqat matnli ma’lumotlarni balki, foydlanuvchining xoxlagan ma’lumotni matn yoki binar kodlash orqali saqlash mumkin. Muloqot oynasining ko‘rinishi quyidagi rasmda keltirilgan.



11.8-rasm. **SaveFileDialog** muloqot oynasidan foydalanish

Ko‘rib chiqilgan muloqot oynalari barchasi OT bilan ishlashga mo‘ljallanganligi ko‘rinib, turibdi. OT qanday sozlangan bo‘lsa, bu muloqot oynalari ham shu rejimda ishlaydi. Oynalarga deyarli o‘zgartirish kiritish shart emas, xuddiki barchasi kutilgandek yaratilganga o‘xshaydi. Faqat eng katta muammosi lokalizatsiya qilish.

Muloqot oynalari boshqarish elementlari. Bunda interaktiv muloqt qilish oynalari tushiniladi. Ularni yaratish va boshqarish dasturchining xoxishiga qarab amalga oshiriladi. Visual C++ da MessageBox sinfi haqida yuqoridaga aytib o‘tgan edik. Unda 21 ta turli kombinatsiyali show funksiyasi borligini ham. Shuningdek bu funksiyalar va ularning parametrlari hamda parametrlarining qiymatlari to‘g‘risidagi ma’lumotlarga, muloqot oynalarini yaratishga va ishlov berishga e‘tiborni qaratamiz.

1.Show(String) funksiyasi. Bir argumentli funksiya bo‘lib, belgilangan matn bilan xabar ko‘rsatadigan muloqot oynasini yaratish uchun ishlatiladi. Uning kiruvchi parametri System::String tipida bo‘lib, System::Windows::Forms::DialogResult tipidagi qiymat qaytaradi.

```
if( MessageBox::Show("Bu oddiy sohow") ==
System::Windows::Forms::DialogResult::OK)
                    this->Close();
```

2.Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, String, String) funksiyasi. Bu ko‘p paramerli bo‘lib, belgilangan

parametlar asosida interaktiv muloqot oynasini yaratish uchun ishlatiladi. Parametlari quyidagi qiymatlari qabal qiladi.

IWin32Window Interface - System.Windows.Forms nomlar fazosi va System.Windows.Forms.dll kutubxonasidan foydalanib, Win32 HWND ni joriy qilish interfeysi beradi. IWin32Window sinf interfeysidan merosxo‘r oladi. System.Windows.Forms.Control va System.Windows.Forms.NativeWindow umumiy ruxsat sinflari interfeyslarini ishlatadi, qiymatlar sifatida ComVisibleAttribute, GuidAttribute, InterfaceTypeAttribute tiplarini ishlatadi.

MessageBoxButtons - Enum tipidagi parametrdir. System.Windows.Forms nomlar fazosi va System.Windows.Forms.dll kutubxonasidan foydalanadi. Muloqot oynalarida ko‘rsatilishi kerak bo‘lgan tugmalarni aniqlaydi. Qiymatlari public enum class MessageBoxButtons ta‘luqlidir.

11.6-jadval.MessageBoxButtons qabul qiluvchi qiymatlar

№	Qiymat nomi	vazifasi
1	AbortRetryIgnore	Muloqot oynasida Abort, Retry Ignore tugmalarini o‘rnatish
2	OK	Muloqot oynasida OK tugmasini o‘rnatish
3	OKCancel	Muloqot oynasida OK va Cancel tugmalarini o‘rnatish
4	RetryCancel	Muloqot oynasida Retry va Cancel tugmalarini o‘rnatish
5	YesNo	Muloqot oynasida Yes va No tugmalarini o‘rnatish
6	YesNoCancel	Muloqot oynasida Yes, No va Cancel tugmalarini o‘rnatish

Muloqot oynasiga MessageBoxButtons tugmalarini o‘rnatish dastur fragmenti:

```
if ((MessageBox::Show("Joriy oynani yopishni
xoxlaysizmi?",
                "Xabar", MessageBoxButtons::YesNo) ==
System::Windows::Forms::DialogResult::Yes)){
                    this->Close();
                }
```

MessageBoxIcon – Enum tipidagi obyekt bo‘lib, u ham yuqoridagidek, nomlar fazosi va kutubxona bilan ishlaydi. Muloqot oynasida ko‘rsatish uchun muloqot oynalarining turlarini belgilovchi konstantalarni aniqlaydi.

11.7-jadval. MessageBoxIcon qabul qiluvchi qiymatlar

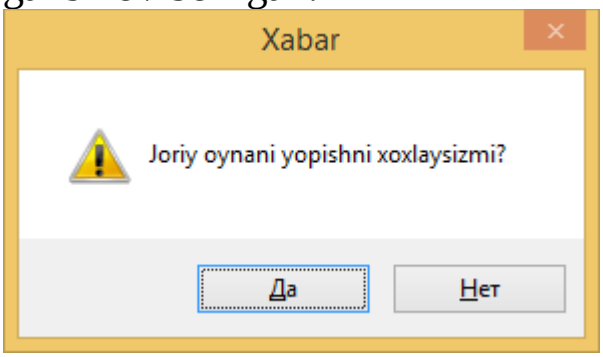
№	Qiymat nomi	belgisi	vazifasi
1	Asterisk		Qandaydir hodisa haqida faqat xabar beruvchi muloqot oynasi uchun ikonka
2	Error		Qandaydir hodisa haqida faqat xato amal bajarilganligi haqida ma’lumot beruvchi muloqot oynasi uchun ikonka
3	Exclamation		Qandaydir hodisa haqida faqat ogohlantirish haqida ma’lumot beruvchi muloqot oynasi uchun ikonka.
4	Hand		Qandaydir hodisa haqida faqat xato amal bajarilganligi haqida ma’lumot beruvchi muloqot oynasi uchun ikonka
5	Information		Qandaydir hodisa haqida faqat xabar beruvchi muloqot oynasi uchun ikonka
6	None		Ikonkasiz muloqot oynasi uchun
7	Question		Qandaydir hodisa haqida faqat savolga javob oluvchi muloqot oynasi uchun ikonka
8	Stop		Qandaydir hodisa haqida faqat xato amal bajarilganligi haqida ma’lumot beruvchi muloqot oynasi uchun ikonka
9	Warning		Qandaydir hodisa haqida faqat ogohlantirish haqida ma’lumot beruvchi muloqot oynasi uchun ikonka.

MessageBoxIcon ni ishlatish uchun joriy formani yopish uchun muloqot oynani ishlatish uchun dastur fragmentini keltiramiz. Buning uchun formaning FormClosing hodisasiga quyidagi dastur fragmentini yozamiz:

```
System::Void Form1_FormClosing(System::Object^
sender, System::Windows::Forms::FormClosingEventArgs^
e) {
                    if ((MessageBox::Show(
                        "Joriy oynani yopishni
xoxlaysizmi?",
                        "Xabar",
```

```
MessageBoxButtons::YesNo,
MessageBoxIcon::Exclamation) ==
System::Windows::Forms::DialogResult::No)){
                            e->Cancel = true;
                }
            }
```

Dastur fragmentida hodisaning to‘liq yozilishi keltirilgganligining sababi unda hodisaga ishlov berilgan.



11.9-rasm. MessageBoxIcon bilan muloqot oynasini yaratish.

Mazkur oynada yo‘q tugmasi bosilsa, forma yopilmaydi va aksincha ha tugmasi bosilsa, forma yopiladi.

MessageBoxDefaultButton - Enum tipidagi obyekt bo‘lib, u ham yuqoridagidek, nomlar fazosi va kutubxona bilan ishlaydi. Muloqot oynasida standart tugmalarni o‘rnatuvchi o‘zgarmaslarni belgilaydi.

11.7-jadval. MessageBoxDefaultButton qabul qiluvchi qiymatlar

№	Qiymat nomi	vazifasi
1	Button1	Muloqot oynasida starndart birinchi tugmani joriy tugma sifatida o‘rnatish
2	Button2	Muloqot oynasida starndart ikkinchi tugmani joriy tugma sifatida o‘rnatish
3	Button3	Muloqot oynasida starndart uchinchi tugmani joriy tugma sifatida o‘rnatish

Tugma bosilganda formani yopishni so‘rash muloqot oynasini yaratish dastur fragmentini keltiramiz.

```
String^ message = "Joriy oynani yopishni
xoxlaysizmi?";
String^ caption = "Xabar";
MessageBoxButtons buttons = MessageBoxButtons::YesNo;
System::Windows::Forms::DialogResult result;
System::Windows::Forms::DialogResult mayli =
```

```
System::Windows::Forms::DialogResult::Yes;

result = MessageBox::Show( this, message, caption,
buttons, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button2);
                  if ( result == mayli )
                  {
                      this->Close();
                  }
```

Odatda tugmalarning farqini aniqlash ancha murakkab, agar OT maska bo‘lsa, farqlash mumkin.

MessageBoxOptions - enum tipidagi obyekt bo‘lib, u ham yuqoridagidek, nomlar fazosi va kutubxona bilan ishlaydi. Muloqot oynasida standart xususiyatlarni o‘rnatish uchun foydalaniladi.

11.8-Jadval . MessageBoxOptions

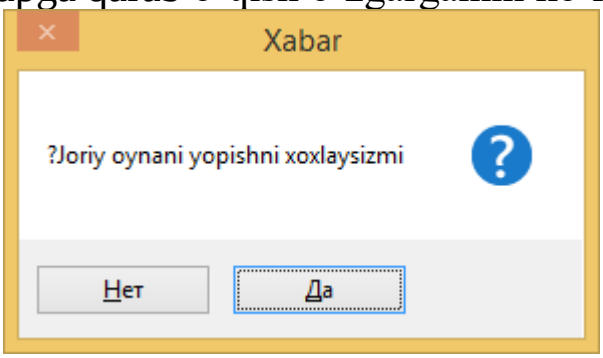
№	Qiymat nomi	vazifasi
1	DefaultDesktopOnly	Oddiy muloqot oynalariga o‘xshaydi ammo faol ishchi maydonda ko‘rsatiladi.
2	RightAlign	Muloqot onynasining ma’lumotini to‘g‘ri joylashtirish
3	RtlReading	Muloqot oynasida ma’lumotni o‘ngdan chapga qarab o‘qish uchun moslab joylashtirish.
4	ServiceNotification	Oddiy muloqot oynalariga o‘xshaydi ammo faol ishchi maydonda ko‘rsatiladi va javob bermasdan o‘tib ketish mumkin xususiyati mavjud

Tugma bosilganda formani yopishni so‘rash muloqot oynasini yaratish dastur fragmentini keltiramiz.

```
String^ message = "Joriy oynani yopishni
xoxlaysizmi?";
                  String^ caption = "Xabar";
                  MessageBoxButtons buttons =
MessageBoxButtons::YesNo;
                  System::Windows::Forms::DialogResult
result;
```

```
System::Windows::Forms::DialogResult mayli =
System::Windows::Forms::DialogResult::Yes;
                    result = MessageBox::Show( this,
message, caption, buttons,
MessageBoxIcon::Question,MessageBoxDefaultButton::But
ton1,MessageBoxOptions::RtlReading);
                   if ( result == mayli )
                   {
                       this->Close();
                   }
```

Dastur fragmentini ishlatib ko‘rib, natijasini tahlil qilish va nimalarni o‘ngdan chapga qarab o‘qish o‘zgarganini ko‘rish mumkin.



11.10-rasm. MessageBoxOptions bilan muloqot oynasini yaratish.

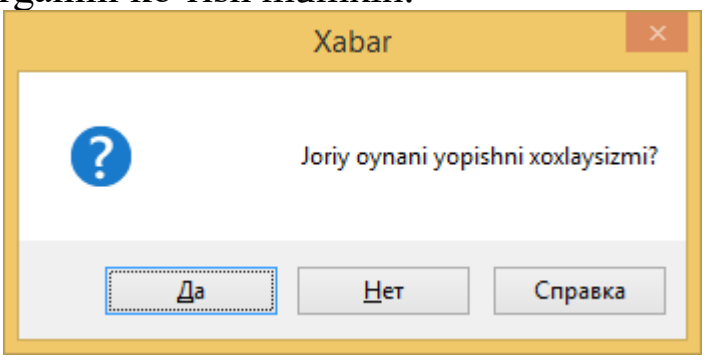
Barcha muloqot oynalari kabi yuqorida keltirilgan dastur fragmentlaridan ko‘rinib turibdiki, DialogResult tipini qaytaradi.

3. Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, String, String) funksiyasi. Bu funksiyaga yordam fayli va ichki indeks fayllarga ko‘rsatkich yaratadi.

```
String^ message = "Joriy oynani yopishni
xoxlaysizmi?";
String^ caption = "Xabar";
MessageBoxButtons buttons = MessageBoxButtons::YesNo;
System::Windows::Forms::DialogResult result;
System::Windows::Forms::DialogResult mayli =
System::Windows::Forms::DialogResult::Yes;

result = MessageBox::Show( this, message, caption,
buttons,
MessageBoxIcon::Question,MessageBoxDefaultButton::But
ton1,MessageBoxOptions::RightAlign,
"d:\tut.chm",HelpNavigator::KeywordIndex,"iv");
```

```
                if ( result == mayli )
                {
                    this->Close();
                }
```

Dastur fragmentini ishlatib ko‘rib, natijasini tahlil qilish va nimalarni o‘zgarganini ko‘rish mumkin.



11.11-rasm. Yordam tugmali muloqot oynasini yaratish.

Dastur fragmentida e‘tibor bilan qarasangiz, MessageBoxButtons::YesNo tshlatilgan, ammo, muloqot oynasida 3 ta tugma chiqqan, demak yordam tugmasining faol holatga kelganining ko‘rish mumkin.

Show usulining boshqa yangi parametrlarni qabul qilmaganligi uchun ularni yozilishi va dastur fragmentlaridan keltiramiz.

4.Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, String)

```
result = MessageBox::Show( this, message, caption,
buttons,
MessageBoxIcon::Question,MessageBoxDefaultButton::But
ton1,(MessageBoxOptions)0,
"d:\tut.chm",HelpNavigator::KeywordIndex,"iv");
```

5.Show(String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, String, HelpNavigator)

```
System::Windows::Forms::DialogResult r3 =
MessageBox::Show( message, caption,
MessageBoxButtons::OK, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1,
(MessageBoxOptions)0, "mspaint.chm",
HelpNavigator::Index );
```

6.Show(String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, String, String)

```
System::Windows::Forms::DialogResult r7 =
```

```
MessageBox::Show(message, caption,
MessageBoxButtons::OK, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1,
(MessageBoxOptions)0, "mspaint.chm",
"mspaint.chm::/paint_brush.htm" );
```

7.Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions)

```
 result = MessageBox::Show( this, message, caption,
buttons, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1,
MessageBoxOptions::RightAlign );
```

8.Show(String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, String)

```
System::Windows::Forms::DialogResult r1 =
MessageBox::Show( " message, caption,
MessageBoxButtons::OK, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1,
(MessageBoxOptions)0, "mspaint.chm" );
```

9.Show(String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, Boolean)

```
System::Windows::Forms::DialogResult r =
MessageBox::Show(message, caption,
MessageBoxButtons::OK, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1,
(MessageBoxOptions)0, true );
```

10.Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, String, HelpNavigator)

```
System::Windows::Forms::DialogResult r4 =
MessageBox::Show( this, message, caption,
MessageBoxButtons::OK, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1,
(MessageBoxOptions)0, "mspaint.chm",
HelpNavigator::Index );
```

11.Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton)

```
result = MessageBox::Show( this, message, caption,
buttons, MessageBoxIcon::Question,
```

```
MessageBoxDefaultButton::Button1,
MessageBoxOptions::RightAlign );
```

12.Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon)

```
result = MessageBox::Show( this, message, caption,
buttons, MessageBoxIcon::Question );
```

13.Show(String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton)

```
result = MessageBox::Show( this, message, caption,
buttons, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1 );
```

14.Show(IWin32Window, String, String, MessageBoxButtons)

```
result = MessageBox::Show( this, message, caption,
buttons);
```

15.Show(String, String, MessageBoxButtons, MessageBoxIcon)

```
 result = MessageBox::Show(message, caption,
buttons, MessageBoxIcon::Question);
```

16.Show(IWin32Window, String, String)

```
result = MessageBox::Show( this, message, caption);
```

17.Show(IWin32Window, String)

```
result = MessageBox::Show( this, message);
```

18.Show(String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions)

```
result = MessageBox::Show( this, message, caption,
buttons, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1,
MessageBoxOptions::RightAlign );
```

19.Show(IWin32Window, String, String, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions, String, HelpNavigator, Object)

```
System::Windows::Forms::DialogResult r6 =
MessageBox::Show(this, message, caption,
MessageBoxButtons::OK, MessageBoxIcon::Question,
MessageBoxDefaultButton::Button1,
(MessageBoxOptions)0, "mspaint.chm",
HelpNavigator::KeywordIndex, "ovals" );
```

Bu usullarni .NET 5 Preview 1, .NET Core 3.1 3.0 va .NET Framework
4.8, 4.7.2, 4.7.1, 4.7, 4.6.2, 4.6.1, 4.6, 4.5.2, 4.5.1, 4.5, 4.0, 3.5, 3.0, 2.0, 1.1 kabi varintlarda qo‘llab quvatlanadi.

Muloqot oynalarini yaratish. Foydalanuvchi tomonidan muloqot oynalarini yaratish oyna formasiga ishlov berish asosida amalga oshiriladi. Forma oynasiga muloqot oynasini o‘rnatish uchun quyidagi qadamlari bajarish lozim.

1-qadam. Menyudan foydlanib, [menu] → [project] → [addClass] → [CLR] → [Windows Form] buyruqlar ketma ketligi asosida yangi forma qo‘shiladi. Yoki, [menu] → [project] → [add new item] → [UI] → [Windows Form] ham bajarsa bo‘ladi.

2-qadam. Yaratilgan yangi forma oynasigi o‘tib, formaga quyidagicha ishlov beriladi. [Autosize] xususiyatning qiymatini [true] ga, [StartPosition] xususiyatining qiymatini [CenterParent]ga, [FormBorderStyle] xususiyati qiymatiga [none], [FixedDialog], [FixedToolWindows], [SizableToolWindows]larning birini o‘rnatish mumkin.

3-qadam. Forma oynasini o‘zinigz xolagandek loyihalashingiz mumkin. Masalan, bir Label, InputBox va bir Button joylashtiramiz, ularni ham kerakli xususiyatlari o‘rnatamiz.

4-qadam. Asosiy formaga o‘tib, unga #include "MyForm.h" sarlavha faylni qo‘shamiz. Bu muloqot oynasi uchun yaratilgan forma oynasi bilan ishlash uchun kerak.

5- Qadam. Asosiy formaga bir label va bir button joylashtiramiz. Tugmaning klik hodisasida muloqot oynasini chaqirish dastur fragmentini yozamiz.

```
                              // MyForm myForm;
                    // myForm.Show();
                    // myForm.ShowDialog();

                        MyForm^ myFormWith = gcnew
MyForm();
                     // myFormWith->Show();
                        myFormWith->ShowDialog();
```

Dastur fragmentida izohga olib qo‘yilagan yordami ham muloqot oynalarni yaratish va chaqirish mumkin. Ammo ularni farqlari mavjud. Shuning uchun ularni hammasini shu dastur fragmentida keltirdik. Birinchi qatorda formani MyForm myForm oddiy obyekt sifatida

yaratilgan. Uning birinchi Show() usuli hisoblanib, bu usul bilan muloqo oynasini chaqirish mantiqan xato, chuniki tizim oynani yaratadi va ustunlikni asosiy formaga beradi. Bu holda yaratilgan oyna bir lahzaga ko‘rinadi xolos. Ikkinchi usuli bu ShowDialog() usul yaxshi yondashuvlardan bo‘lib, muloqot formasiga o‘rnatilgan barcha xususiyatlarga rioya qiladi hamda foydalanish mumkin. Ikkinchi MyForm() konstruktordan foydalanib, gcnew operatori asosida yaratilgan, uning birinchi fuknsiya Show() ham oldingisiga o‘xshash bo‘lib, ammo ustunlikni foydalanuvchining o‘ziga qo‘yib beradi, ya‘ni foydalanuvchi muloqot oynaga javob bermasdan turib, asosiy forma oynasiga o‘tishi mumkin. Ikkinchisi esa bu ShowDialog()usul yaxshi yondashuvlardan bo‘lib, muloqot formasiga o‘rnatilgan barcha xususiyatlarga rioya qiladi hamda foydalanish mumkin.

6-qadam. Muloqot oynasini tugmasida quyidagicha algoritmni yozamiz. Muloqot oynasiga kiritilgan ma’lumotni asosiy formaga olib o‘tish uchun avval public: System::String^ email; kabi bir o‘zgaruvchi yaratib olamiz.

```
                                email = textBox1->Text;
                    this->Close();
```

7-qadam. Asosiy formaning tugmasining hodisasiga yozilgan dastur fragmenti davomidan quyidagini qo‘shib qshyamiz.

```
                                 label1->Text          =
myFormWith->email;
```

8-qadam. Loyihani ishlatib yaratilgan, yaratilgan muloqot oynani ishlashini ko‘rish mumkin.

Muloqot oynani yuqori darajada yaratish uchun unga yangi konstruktor yozish ham mumkin. Uni quyidagicha amalga oshiriladi.

```
public:  MyForm(System::String^ title)
        {
            InitializeComponent();
            _title = title;
        }
public: System::String^ _title;
//….
private: System::Void MyForm_Load(System::Object^
sender, System::EventArgs^  e) {

                this->Text = _title;
             }
```

```
// Aspsiy formada esa
                    MyForm^ myFormWith = gcnew
MyForm("Xabar");
                    myFormWith->ShowDialog();
                    label1->Text = myFormWith->email;
```

Bunday imkoniyat bilan foydalanuvchi uchun ixtiyoriy muloqot oynasini yaratish mumkin.

Muloqot oynalaridan foydalanish dastur foydalanuvchilariga ko‘plab qo‘layliklar yaratib beradi. Har bir dasturda bir asosiy oynaga kamida funksional imkoniyatiga qarab 4-5 ta muloqo oynalari bo‘ladi.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Axborot tizimlarida muloqot oynalarining qanday guruhlari bor.
2. Muloqot oynalariga qo‘yiladigan talablar bormi va bo‘lsa, nima uchun?
3. Muloqot oynasida teskari aloqa deganda nimani tushunasiz?
4. Muloqot oynada nimalar aniq ko‘rsatilishi kerak?
5. [Dialogs] tab oynasida joylashgan muloqot komponentalarini sanab bering?
6. ColorDialog komponentasining vazifasini ayting?
7. Qaysi komponenta foydalanuvchilar uchun papkalar ko‘rish, yaratish va tanlash imkonini beradigan muloqot oynasini ko‘rsatadi.
8. Interaktiv xabarlarni berish va aniq javoblarni olish Visual C++ da qanday sinfi mavjud.
9. MessageBox sinfnining show funksiyasi parametrlarini sanab bering?
10. MessageBoxOptions qanday obyekt tipi va nima vazifani bajaradi.
11. MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton obyektlarini birini qabul qiluvchi kengaytirilgan obyekt tipi nomini ayting.
12. Foydalanuvchi tomonidan yaratiladigan muloqot oynasi qanday yaratiladi.
13. Interaktiv muloqot oynalari kabi nimalarni yaratish va ularni boshqarish, kerakli ma’lumotlarni olish uchun ishlatiladi.

14. Qaysi sinfning nomlar fazosi System.Windows va kutubxonasi PresentationFramework.dll hisoblanadi.

15. **ColorDialog muloqot oynasining asosiy vazifasi va obyekti qanday yaratiladi.**

16. ColorDialog muloqot oynasidan foydalanish uchun qanday konstruktorni ishga tushirish lozim.

17. CanRaiseEvents xususiyatining vazifasini ayting?

18. Komponentaga ilova qilinadigan hodisalar ro‘yxatini oladigan xususiyat nomini ayting?

19. Muloqot oynasi ochilganda maxsus ranglarni yaratish uchun ishlatiladigan boshqaruv elementlari ko‘rinib turishini ko‘rsatuvchi qiymatni oladigan yoki o‘rnatadigan xususiyat nomini ayting?

20. Dispose() bu qanday usul?

21. Komponenta yoki uning konteyneri tomonidan taqdim etilgan xizmatni ifodalovchi obyektni qaytaradigan hodisani ayting?

22. Disposed hodisaning vazifasini ayting?

23. Foydalanuvchi umumiy muloqot oynasidagi yordam tugmasini bosganda sodir bo‘ladigan usul nomini ayting?

24. Foydalanuvchilarga kerakli komponenta uchun shrift va uning xususiyatlarini o‘rnatish imkonini beruvchi muloqot oynasini nomini ayting?

25. PrintPreviewDialog muloqot oynasi qaysi tabda joylashgan va vazifasini aniq aytib bering.

26. Dastur fragmentida bir StreamReader oqimi nima uchun yaratiladi.

27. Win32Window Interface nima uchun ishlatiladi.

28. Muloqot oynasida OK va Cancel tugmalarini o‘rnatish qayday amalga oshiriladi.

29. MessageBoxIcon muloqot oynalarida nima uchun ishlatiladi.

30. Qandaydir hodisa haqida faqat xato amal bajarilganligi haqida ma’lumot beruvchi muloqot oynasi uchun ikonka nomi va yozlishini tushuntirib bering.

31. Muloqot oynasida standart tugmalarni o‘rnatuvchi o‘zgarmaslarni belgilaydigan parametrni ayting?

32. RtlReading qaysi parametr uchun ishlatiladi va muloqot oynasida nima hodisa sodir bo‘ladi.

33. System::Windows::Forms::DialogResult r6 = MessageBox::Show(this, message, caption, MessageBoxButtons::OK, MessageBoxIcon::Question, MessageBoxDefaultButton::Button1,

(MessageBoxOptions)0, "mspaint.chm", HelpNavigator::KeywordIndex, "ovals") – dastur fragmentini tushuntirib bering

34. Asosiy formaga yangi forma qanday qo‘shiladi

35. Nima uchun [StartPosition] xususiyatining qiymatini [CenterParent]ga tenglashtiriladi.

36. ShowDialog() va Show() usullarining farqini tushuntirib bering.

37. Oddiy obyekt yaratish va konstruktor asosida obyektni yaratish farqini tushuntirib bering.

38. Muloqot oynalarni yaratishda foydalanuvchi yangi konstruktor yaratishi mumkinmi?

39. [menu] → [project] → [add new item] → [UI] → [Windows Form] ketma ketlik qanday amalni bajaradi.

40. Muloqot oynasidagi ma’lumotlarni qanday qilib asosiy formaga o‘tkazish mumkin.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASh HAMDA RIVOJLANTIRISh UChUN ASSISMENT TOPShIRIG‘I.

ASSISMENT TOPShIRIG‘I	
□	Muloqot oynalari bilan ishlashga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring. ☝□ Bu vazifalarni kichik loyiha sifatida bajariladi, kichik loyiha deganda kamida 100 ta vazifasi bo‘lgan dastur tushuniladi. Jamoa bilan ishlashga maqsadga muvofiq. Har bir vazifani jamoa bilan bajarish kerak, vazifalarni tarqatish uchun 1ta proyekt menejer (20%), 1ta dizayner (10%), 2ta funksiya va algoritmlarni yozadigan (45%), 1ta dasturchi (25%), foizda ularni ulushlari keltirilgan.
1. Oddiy matn muharriri dasturini tuzing. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing. _____ _____ _____ _____ _____ _____	

_____
_____
_____

2. Interaktiv muloqot oynalaridan foydlanib, ko‘p xonali sonlar uchun arifmetik savol javobga natijasini aytuvchi dastur yarating. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____

3. Intellektual turli ma‘lamotlarni kiritishni ta‘lab qiluvchi IQ testlardan iborat dastur tuzing. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____

4. C++ dasturlash muhitini yarating. C++ kompilyatorini va kutub‘onalarini tayyor tizimdan oling. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____

5. Barcha turdagi muloqot oynalarini o‘z ichiga olgan, interaktiv savol javob tizimini yarating va savollarni, javoblarini faylga saqlaydigan dastur tuzing. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____
_____

6. Barcha vazifalarni o‘rganganingiz bo‘yicha eslab ko‘ring, nimalarni o‘rgandingiz va bir takrorlang.
7. Barcha bilim va ko‘nikmalar asosida kichik loyiha tayyorlang.

3.4. Visual C++ning grafik imkoniyatlari.

📖 Visual C++ning grafik imkoniyatlari uchun Graphics sinfi xususiyatlari va usullari, Graphics sinfi usulari asosida tasvirlarni turli xilda chizish, matn joylashtirish, rasmlarni yaratish va saqlash, tahrirlash amallari, Chart komponenta xususiyati va hodisalari, turli tipli diagrammalarni hosil qilish uslublari, Funksiyalarni grafiklarini qurish va Visual Basic Power Packs komponentalari va ularni ishlatish, tasvirlarni harakatlantirish, oqimlar asosida annimatsiyalarni yaratish bo‘yicha nazariy va amaliy bilimlarni hamda asoslash uchun dasturlar tuzib ko‘rsatilgan. Bilimlarni mustahkamlash uchun 30 ta nazariy savol va amaliy ko‘nikma va malakalarni rivojlantrish uchun 5 assisment topshirig‘i berilgan. Bu topshiriqlarni bajarish mavzuni mustahkamlash uchun xizmat qiladi.

✎ *Kalit so‘zlar*. Grafika, piksel, nuqta, Graphics, geometrik shakllari, Clip, ClipBounds, CompositingMode, CompositingQuality, DpiX, DpiY, Display, Document, Inch, Millimeter, Pixel, Point, World, Pen, DrawArc, DrawBezier, DrawCurve, DrawEllipse, DrawImage, DrawLines, DrawPolygon, DrawRectangle, DrawString, FillPie, FillEllipse, Chart, ChartAreas, Series, ChartType, Visual Basic Power Packs, PrintForm, LineShape, OvalShape, RectangleShape, DataRepeater.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Grafika, geometriya, xarakatning paydo bo‘lishi, sinf va sinf obyekti, xususiyat, hodisa, forma, komponenta dasturlashga oid dastlabki tushunchalar hamda C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Visual C++ muhitida dasturlash, Visual C++ning grafik imkoniyatlari uchun Graphics sinfi xususiyatlari va usullari,

Graphics sinfi usulari asosida tasvirlarni turli xilda chizish, matn joylashtirish, rasmlarni yaratish va saqlash, tahrirlash amallari, Chart komponenta xususiyati va hodisalari, turli tipli diagrammalarni hosil qilish uslublari, funksiyalarni grafiklarini qurish va Visual Basic Power Packs komponentalari va ularni ishlatish, tasvirlarni harakatlantirish, oqimlar asosida annimatsiyalarni yaratishga talablar, o‘rnatish va komponentalar bilan bog‘lash usullari, shuningdek, Clip, ClipBounds, CompositingMode, CompositingQuality, DpiX, DpiY, Display, Document, Inch, Millimeter, Pixel, Point, World, Pen, DrawArc, DrawBezier, DrawCurve, DrawEllipse, DrawImage, DrawLines, DrawPolygon, DrawRectangle, DrawString, FillPie, FillEllipse, Chart, ChartAreas, Series, ChartType, Visual Basic Power Packs, PrintForm, LineShape, OvalShape, RectangleShape, DataRepeater usular va ba‘zi bir xususiyatlari, hodisalari o‘rganishingiz mumkin.

REJA

1. Visual C++ning grafik imkoniyatlari
2. Graphics sinfi usulari asosida tasvirlarni qurish
3. Chart komponentalari.
4. Funksiyalarning grafiklarini qurish.
5. Visual Basic Power Packs komponentalari va ularni ishlatish.

KIRISH

Visual C++ o‘zi grafik tuzilmali muhit bo‘lib hisoblanadi. Grafik deganda ixtiyoriy narsani kompyuterda piksellarda hosil qilish tushuniladi. Pikselda ikkita argumentlari bor, ya‘ni koorlinatalari A(x,u) ko‘rinishida. Har qanday IDE muhitlarda grafika bilan ishlash imkoni bor. Ular turlicha nomlanishi mumkin. Ammo, ularning ruchkasi (qalami) va mo‘yqalami bo‘ladi. GUI ko‘rinishda ishlash to‘liq grafika bilan bog‘liq bo‘lib, texnikaning grafik rejimda ishlashini ta‘minlaydi. Hozirda deyarli barcha foydalanuvchilar grafik imkoniyatlaridan foydalanib ishlaydilar.

Visual C++ning grafik imkoniyatlari. Grafik imkoniyatga ega bo‘lgan tizimlarda asosan, nuqta, chiziq, to‘rt burchak, aylana, ko‘p burchak kabi shakllarni qamrab oladi. GUI asosidagi barcha elementlarga diqqat bilan qarasangiz shu grafik tuzilmalardan iborat bo‘ladi.

Visual C++ning imkoniyatlarini ko‘rish uchun Graphics sinfiga murojaat qilamiz. Bu sinfning nomlar fazosi System.Drawing va kutubxonasi System.Drawing.Common.dll bo‘lib hisoblanadi.

Chizish uchun GDI+ modulni inkapsulyatsiya qiladi va bu sinfdan merosxo‘r olish mumkin emas. Bu sinf MarshalByRefObject, IDisposable, System::Drawing::IdeviceContextga asoslangan bo‘lib, Object → MarshalByRefObject → Graphics sinfining merosxo‘ri hisoblanadi.

Grafika sinfi ekranga tasvirlarni chizish usullarini o‘zi ichiga qamarab olgan. Grafika muayyan qurilmani kontekst bilan bog‘lash uchun ham xizmat qiladi.

Grafik obyekt yordamida ko‘p turli shakl va chiziqlar chizish mumkin. Chiziqlar va shakllar chizish uchun maxsus DrawGraphicalElement usullarni o‘rganish lozim. Bu usullar DrawLine, DrawArc, DrawClosedCurve, DrawPolygon va DrawRectangle o‘z ichiga oladi. Chiziqlar va shakllar chizish uchun qalam yordamida va shakllarni to‘ldirish uchun mo‘yqalam yordamida amalga oshiriladi.

Grafika sinfi imkoniyatlarini uning xususiyatlari va usulari orqali ko‘rsatib o‘tamiz. Shuning yangi paraemet kelsa, unga ham to‘xtalib o‘tiladi.

12.1-jadval.Grafika sinfi xususiyatlari.

Xususiyait nomi	vazifasi
Clip	Grafikaning chizilgan chegarasini cheklaydigan chegarani oladi yoki o‘rnatadi.
ClipBounds	Grafikaning kesish chegarasini chegaralovchi RectangleF tuzilishini oladi.
CompositingMode	Kompozit tasvirlar chizish holatini oladi yoki o‘rnatadi.
CompositingQuality	Grafikaga chizilgan kompozitsion tasvirlarning ko‘rsatish sifatini o‘rnatadi.
DpiX	Ushbu grafikaning gorizontal o‘lchamini oladi.
DpiY	Ushbu grafikaning vertikal o‘lchamini oladi.
InterpolationMode	Ushbu grafikalar bilan bog‘liq interpolyatsiya rejimini oladi yoki o‘rnatadi.
IsClipEmpty	Bu grafika kesish sohasini bo‘sh yoki yo‘qligini ko‘rsatib, bir qiymat oladi.
IsVisibleClipEmpty	Bu grafika aniq kesish sohasini bo‘sh yoki yo‘qligini ko‘rsatib, bir qiymat oladi.
PageScale	Bu grafika uchun sahifa moduli va birlik moduli o‘rtasida chegarasini sozlash.

PageUnit	Bu grafika sahifa koordinatalarini uchun ishlatiladigan o‘lchov birligi sozlash.
PixelOffsetMode	Bu grafika ko‘rsatish paytida Piksel ofset qanday ko‘rsatilgan qiymat sozlash.
RenderingOrigin	Bu grafika ko‘rsatish rejimini o‘rnatadi.
SmoothingMode	Grafikalar uchun ko‘rsatish sifatini oladi yoki o‘rnatadi.
TextContrast	Matn ko‘rsatish uchun gamma qiymatini belgilash.
TextRenderingHint	Bu grafika bilan bog‘liq matn uchun ko‘rsatish rejimini o‘rnatadi.
Transform	Grafikalar uchun geometrik o‘zgarishining nusxasini oladi yoki o‘rnatadi.
VisibleClipBounds	Grafikning aniq kesish sohasini tekslash uchun to‘rtburchak oladi.

Grafika sinfi usulari quyidagilardan iborat:

1. AddMetafileComment(Byte[]) - Rasm metafayliga izoh qo‘shadi. Bunda Byte[] belgili massiv bo‘lib, <System::Byte> ^ data tipida aniqlanadi va masalan, array<Byte>^metaCom = {(Byte)'T',(Byte)'e',(Byte)'s',(Byte)'t'}; kabi aniqlanishi mumkin.
2. BeginContainer() - Grafikaning hozirgi holati bilan grafik konteynerni saqlaydi va yangi grafik konteynerni ochadi va ishlatadi.

BeginContainer(Rectangle, Rectangle, GraphicsUnit) - Grafik joriy holati bilan bir grafik konteyner saqlaydi, belgilangan parametrli o‘zgartirish bilan yangi grafik konteynerdan foydalanadi va ochadi. Bunda Rectangle to‘rt burchak bo‘lib, Rectangle(0,0,200,200) kabi aniqlanadi. Birinchi to‘rt burchak konteyner uchun shkalani va ikkinchisi konteyner uchun soha o‘zgarishiri aniqlaydi. GraphicsUnit – konteyner uchun o‘lchov birligini aniqlash uchun ishlatiladi. Uning enum bo‘lib, quyidagi 12.2–jadvalga keltirilgan qiymatlarni oladi va GraphicsUnit::Pixel ko‘rinishda foydalaniladi.

12.2-jadval. GraphicsUnit qiymatlari.

Qiymat nomi	vazifasi
Display	Ko‘rsatish qurilmasining o‘lchov birligini belgilaydi. Odatda video displeylar uchun piksellar va printerlar uchun 1/100 dyuym.
Document	O‘lchov birligi sifatida hujjat birligini (1/300 dyuym) belgilaydi.

Inch	O‘lchov birligi sifatida dyumni bildiradi.
Millimeter	Millimetrni o‘lchov birligi sifatida belgilaydi.
Pixel	O‘lchov birligi sifatida qurilma pikselini belgilaydi.
Point	O‘lchov birligi sifatida printerning nuqtasini (1/72 dyuym) belgilaydi.
World	Koordinata tizimi birligini o‘lchov birligi sifatida belgilaydi.

BeginContainer(RectangleF, RectangleF, GraphicsUnit) - grafik joriy holati bilan bir grafik konteyner saqlaydi, belgilangan sohali o‘zgartirish bilan yangi grafik konteynerdan foydalanadi va ochadi. Bininchi ikkita parametlar tuzilma bo‘lib, ikki xil aniqlanishi mumkin: RectangleF(PointF(30.0F,40.0F),SizeF(50.0F,100.0F)) va RectangleF(float x, float y, float width, float height); GraphicsUnit – esa konteyner uchun o‘lchov birligini aniqlash uchun ishlatiladi.

3. Clear(Color) - butun chizma yuzasini tozalaydi va belgilangan fon rangi bilan to‘ldiradi. Color tuzilma bo‘lib uning qiymatlariga kengaytirish amali [::] bilan murojaat qiladilar.

4.CopyFromScreen(Int32, Int32, Int32, Int32, Size) - Pikselli to‘rtburchakka mos rangli ma’lumotlarni ekrandan grafikaning chizma yuzasiga bit-blokli uzatishni amalga oshiradi. CopyFromScreen(int sourceX, int sourceY, int destinationX, int destinationY, System::Drawing::Size blockRegionSize) – bunda x,u lar тщrtburchakning mos koordinatalari va Size sohaning o‘lchamidir. Ham tuzilma bo‘lib, Size(int width, int height) ko‘rinishda yaratiladi.

CopyFromScreen(Int32, Int32, Int32, Int32, Size, CopyPixelOperation) -Pikselli to‘rtburchakka mos rangli ma’lumotlarni ekrandan grafikaning chizma yuzasiga bit-blokli uzatishni amalga oshiradi. Buning paremetrlari yuqoridagidek bo‘lib, CopyPixelOperation dagi manba rangi yakuniy rangga olib keladigan maqsad rangi bilan birlashtirilganligini aniqlaydi va Blackness, CaptureBlt, DestinationInvert, MergeCopy, MergePaint, NoMirrorBitmap, NotSourceCopy, NotSourceErase, PatCopy, PatInvert, PatPaint, SourceAnd, SourceCopy, SourceErase, SourceInvert, SourcePaint, Whiteness kabi qiymatlarni qabul qiladi.

CopyFromScreen(Point, Point, Size) - Ekrandan grafikaning chizma yuzasiga pikselli to‘rtburchakka mos rangli ma’lumotlarni bit-blokli uzatishni amalga oshiradi. Point – bu nuqta yaratuvchi tuzilma bo‘lib, o‘zining 3ta Point(Int32), Point(Int32, Int32), Point(Size) konstruktorlari mavjud.

CopyFromScreen(Point, Point, Size, CopyPixelOperation) - ekrandan grafikaning chizma yuzasiga pikselli to‘rtburchakka mos rangli ma’lumotlarni bit-blokli uzatishni amalga oshiradi.

5. CreateObjRef(Type) - obyekt bilan muloqot qilish uchun ishlatiladigan proksi ishlab chiqarish uchun zarur bo‘lgan barcha tegishli ma’lumotlarni o‘z ichiga olgan obyekt yaratadi. Type - requestedType tipidagi abstrakt tip, array<Type^> { int::typeid, int::typeid }); kabi aniqlanadi.

6. Dispose() - Bu grafika tomonidan ishlatiladigan barcha resurslarni null qiladi.

7. DrawArc. Bu funksiya uch o‘lchovli sohani ifodalovchi yoyni chizadi. Uning quyidagi variantlari bor:

DrawArc(Pen, Int32, Int32, Int32, Int32, Int32, Int32) - `void DrawArc(System::Drawing::Pen ^ pen, int x, int y, int width, int height, int startAngle, int sweepAngle);`

DrawArc(Pen, Rectangle, Single, Single) - `void DrawArc(System::Drawing::Pen ^ pen, System::Drawing::Rectangle rect, float startAngle, float sweepAngle);`

DrawArc(Pen, RectangleF, Single, Single) - `void DrawArc(System::Drawing::Pen ^ pen, System::Drawing::RectangleF rect, float startAngle, float sweepAngle);`

DrawArc(Pen, Single, Single, Single, Single, Single, Single) - `void DrawArc(System::Drawing::Pen ^ pen, float x, float y, float width, float height, float startAngle, float sweepAngle);`

DrawArc funksiyasida Pen bu qalam bo‘lib, uning uchun Pen sinfi ishlatiladi. Pen ning 4 ta konstruktori mavjud bo‘lib, Pen(Brush), Pen(Brush, Single), Pen(Color), Pen(Color, Single) konstruktorlardan foydalaniladi, xususiyat va usullarilarni o‘zgartirish orqali o‘rnatish mumkin. Brush bu mo‘yqalam bo‘lib, Brush sinfi ishlatiladi. Brush ning Brush() konstruktori mavjud bo‘lib, xususiyat va usullarilarni o‘zgartirish orqali o‘rnatish mumkin. Funksiyaning qolgan parametrlari barchasi yuqorida keltirilgan parametrlar kabi amalga oshiriladi.

8. DrawBezier. Bu funksiya berilgan 4 ta nuqtali soha uchun splayin chizadi. Uning 3 ta turi bor:

DrawBezier(Pen, Point, Point, Point, Point) - `void DrawBezier(System::Drawing::Pen ^ pen,`

```
System::Drawing::Point pt1, System::Drawing::Point pt2, System::Drawing::Point pt3, System::Drawing::Point pt4);
```

DrawBezier(Pen, PointF, PointF, PointF, PointF) - `void DrawBezier(System::Drawing::Pen ^ pen, System::Drawing::PointF pt1, System::Drawing::PointF pt2, System::Drawing::PointF pt3, System::Drawing::PointF pt4);`

DrawBezier(Pen, Single, Single, Single, Single, Single, Single, Single, Single) - `void DrawBezier(System::Drawing::Pen ^ pen, float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4);`

9. DrawBeziers. Bu funksiya berilgan nuchtalar sohasi uchun splayin chizadi. Uning 2 ta turi bor:

DrawBeziers(Pen, Point[]) - `void DrawBeziers(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Point> ^ points);`

DrawBeziers(Pen, PointF[]) - `void DrawBeziers(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points);`

Funksiyada Point[] - ikki o‘lchovli tekislikda nuqta belgilaydigan x va y koordinatalar uchun tartibli juftlikni ifodalaydigan massivdir.

10. DrawClosedCurve. Bu funksiya yopiq egri chiziq chizishga mo‘ljallangan. Uning 4 ta turi bor:

DrawClosedCurve(Pen, Point[]) - `void DrawClosedCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Point> ^ points);`

DrawClosedCurve(Pen, Point[], Single, FillMode) - `void DrawClosedCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Point> ^ points, float tension, System::Drawing::Drawing2D::FillMode fillmode);`

DrawClosedCurve(Pen, PointF[]) - `void DrawClosedCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points);`

DrawClosedCurve(Pen, PointF[], Single, FillMode) - `void DrawClosedCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points, float`

```
tension, System::Drawing::Drawing2D::FillMode fillmode);
```

Bu funksiyada FillMode tipi bor, bu sohani bo‘yaash turini belgilaydi. FillMode ning 2 xil usuli bor, Alternate - Muqobil to‘ldirish va Winding o‘rab to‘ldirish rejimlarini belgilaydi.

11. DrawCurve. Bu funksiya berilgan parametrlar asosida spalyn (egri chiziq) chizadi. Uning 7 ta turi mavjud.

DrawCurve(Pen, Point[]) - `void DrawCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Point> ^ points);`

DrawCurve(Pen, Point[], Int32, Int32, Single) - `void DrawCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Point> ^ points, int offset, int numberOfSegments, float tension);`

DrawCurve(Pen, Point[], Single) - `void DrawCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Point> ^ points, float tension);`

DrawCurve(Pen, PointF[]) - `void DrawCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points);`

DrawCurve(Pen, PointF[], Int32, Int32) - `void DrawCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points, int offset, int numberOfSegments);`

DrawCurve(Pen, PointF[], Int32, Int32, Single) - `void DrawCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points, int offset, int numberOfSegments, float tension);`

DrawCurve(Pen, PointF[], Single) - `void DrawCurve(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points, float tension);`

Bu funksiyaning barcha parametrlari tasniflangan.

12. DrawEllipse. Bu funksiya berilgan parametrlar asosida ellipis chizadi. Uning 4 ta turi mavjud.

DrawEllipse(Pen, Int32, Int32, Int32, Int32) - `void DrawEllipse(System::Drawing::Pen ^ pen, int x, int y, int width, int height);`

DrawEllipse(Pen, Rectangle) - void
```
DrawEllipse(System::Drawing::Pen ^ pen,
System::Drawing::Rectangle rect);
```
DrawEllipse(Pen, RectangleF) - void
```
DrawEllipse(System::Drawing::Pen ^ pen,
System::Drawing::RectangleF rect);
```
DrawEllipse(Pen, Single, Single, Single, Single) - void
```
DrawEllipse(System::Drawing::Pen ^ pen, float x,
float y, float width, float height);
```
Bu funksiyaning barcha parametrlari tasniflangan.

13. DrawIcon. Bu funksiya berilgan parametrlar asosida belgilangan belgi bo‘yicha chizadi. Uning 3 ta turi mavjud.

DrawIcon(Icon, Int32, Int32) - void
```
DrawIcon(System::Drawing::Icon ^ icon, int x, int y);
```
DrawIcon(Icon, Rectangle) - void
```
DrawIcon(System::Drawing::Icon ^ icon,
System::Drawing::Rectangle targetRect);
```
DrawIconUnstretched(Icon, Rectangle) - void
```
DrawIconUnstretched(System::Drawing::Icon ^ icon,
System::Drawing::Rectangle targetRect);
```
Bu funksiyaning barcha parametrlari tasniflangan.

14. DrawImage. Bu funksiya berilgan parametrlar asosida belgilangan tasvirni chizadi. Uning 30 ta turi mavjud.

DrawImage(Image, Int32, Int32)- void
```
DrawImage(System::Drawing::Image ^ image, int x, int
y);
```
DrawImage(Image, Int32, Int32, Int32, Int32) - void
```
DrawImage(System::Drawing::Image ^ image, int x, int
y, int width, int height);
```
DrawImage(Image, Int32, Int32, Rectangle, GraphicsUnit) - void
```
DrawImage(System::Drawing::Image ^ image, int x, int
y, System::Drawing::Rectangle srcRect,
System::Drawing::GraphicsUnit srcUnit);
```
DrawImage(Image, Point) - void
```
DrawImage(System::Drawing::Image ^ image,
System::Drawing::Point point);
```

DrawImage(Image, Point[]) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::Point> ^ destPoints);

DrawImage(Image, Point[], Rectangle, GraphicsUnit) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::Point> ^ destPoints, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit);

DrawImage(Image, Point[], Rectangle, GraphicsUnit, ImageAttributes) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::Point> ^ destPoints, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttr);

DrawImage(Image, Point[], Rectangle, GraphicsUnit, ImageAttributes, Graphics+DrawImageAbort) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::Point> ^ destPoints, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttr, System::Drawing::Graphics::DrawImageAbort ^ callback);

DrawImage(Image, Point[], Rectangle, GraphicsUnit, ImageAttributes, Graphics+DrawImageAbort, Int32) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::Point> ^ destPoints, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttr, System::Drawing::Graphics::DrawImageAbort ^ callback, int callbackData);

DrawImage(Image, PointF) - void DrawImage(System::Drawing::Image ^ image, System::Drawing::PointF point);

DrawImage(Image, PointF[]) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::PointF> ^ destPoints);

DrawImage(Image, PointF[], RectangleF, GraphicsUnit) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::PointF> ^ destPoints, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit srcUnit);

DrawImage(Image, PointF[], RectangleF, GraphicsUnit, ImageAttributes) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::PointF> ^ destPoints, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttr);

DrawImage(Image, PointF[], RectangleF, GraphicsUnit, ImageAttributes, Graphics+DrawImageAbort) - void DrawImage(System::Drawing::Image ^ image, cli::array <System::Drawing::PointF> ^ destPoints, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttr, System::Drawing::Graphics::DrawImageAbort ^ callback);

DrawImage(Image, PointF[], RectangleF, GraphicsUnit, ImageAttributes, Graphics+DrawImageAbort, Int32) - <System::Drawing::PointF> ^ destPoints, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttr, System::Drawing::Graphics::DrawImageAbort ^ callback, int callbackData);

DrawImage(Image, Rectangle) - void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle rect);

DrawImage(Image, Rectangle, Int32, Int32, Int32, Int32, GraphicsUnit) - void DrawImage(System::Drawing::Image ^

```
image, System::Drawing::Rectangle destRect, int srcX,
int srcY, int srcWidth, int srcHeight,
System::Drawing::GraphicsUnit srcUnit);
```

DrawImage(Image, Rectangle, Int32, Int32, Int32, Int32, GraphicsUnit, ImageAttributes) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle destRect, int srcX, int srcY, int srcWidth, int srcHeight, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttr);`

DrawImage(Image, Rectangle, Int32, Int32, Int32, Int32, GraphicsUnit, ImageAttributes, Graphics+DrawImageAbort) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle destRect, int srcX, int srcY, int srcWidth, int srcHeight, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttr, System::Drawing::Graphics::DrawImageAbort ^ callback);`

DrawImage(Image, Rectangle, Int32, Int32, Int32, Int32, GraphicsUnit, ImageAttributes, Graphics+DrawImageAbort, IntPtr) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle destRect, int srcX, int srcY, int srcWidth, int srcHeight, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttrs, System::Drawing::Graphics::DrawImageAbort ^ callback, IntPtr callbackData);`

DrawImage(Image, Rectangle, Rectangle, GraphicsUnit) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle destRect, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit);`

DrawImage(Image, Rectangle, Single, Single, Single, Single, GraphicsUnit) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle destRect, float`

```
srcX, float srcY, float srcWidth, float srcHeight,
System::Drawing::GraphicsUnit srcUnit);
```

DrawImage(Image, Rectangle, Single, Single, Single, Single, GraphicsUnit, ImageAttributes) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle destRect, float srcX, float srcY, float srcWidth, float srcHeight, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttrs);`

DrawImage(Image, Rectangle, Single, Single, Single, Single, GraphicsUnit, ImageAttributes, Graphics+DrawImageAbort) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle destRect, float srcX, float srcY, float srcWidth, float srcHeight, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttrs, System::Drawing::Graphics::DrawImageAbort ^ callback);`

DrawImage(Image, Rectangle, Single, Single, Single, Single, GraphicsUnit, ImageAttributes, Graphics+DrawImageAbort, IntPtr) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::Rectangle destRect, float srcX, float srcY, float srcWidth, float srcHeight, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Imaging::ImageAttributes ^ imageAttrs, System::Drawing::Graphics::DrawImageAbort ^ callback, IntPtr callbackData);`

DrawImage(Image, RectangleF) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::RectangleF rect);`

DrawImage(Image, RectangleF, RectangleF, GraphicsUnit) - `void DrawImage(System::Drawing::Image ^ image, System::Drawing::RectangleF destRect, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit srcUnit);`

DrawImage(Image, Single, Single) - `void DrawImage(System::Drawing::Image ^ image, float x, float y);`

DrawImage(Image, Single, Single, RectangleF, GraphicsUnit) - `void DrawImage(System::Drawing::Image ^ image, float x, float y, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit srcUnit);`

DrawImage(Image, Single, Single, Single, Single) - `void DrawImage(System::Drawing::Image ^ image, float x, float y, float width, float height);`

Bu funksiyaning parametrlari ko‘pchiligi tasvirlarga ishlov berish bilan bog‘liq. Shuning uchun o‘rnaish vaqtida zarurlar parametrlari keltirilgan. Shuningdek, bir xulosa shundan iboratki, men o‘zim bularning hammasini ishlatib ko‘rmaganman.

15. DrawImageUnscaled. Bu funksiya berilgan parametrlar asosida tasvirni belgilangan joyda asl kattaligidan foydalanib chizadi. Uning 5 ta turi mavjud.

DrawImageUnscaled(Image, Int32, Int32) - `void DrawImageUnscaled( System::Drawing::Image ^ image, int x, int y);`

DrawImageUnscaled(Image, Int32, Int32, Int32, Int32) - `void DrawImageUnscaled (System ::Drawing::Image ^ image, int x, int y, int width, int height);`

DrawImageUnscaled(Image, Point) - `void DrawImageUnscaled(System:: Drawing::Image ^ image, System::Drawing::Point point);`

DrawImageUnscaled(Image, Rectangle) - `void DrawImageUnscaled(System:: Drawing::Image ^ image, System::Drawing::Rectangle rect);`

DrawImageUnscaledAndClipped(Image, Rectangle) - `void DrawImageUnscaledAndClipped(System::Drawing::Image ^ image, System::Drawing::Rectangle rect);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

16. DrawLine. Bu funksiya berilgan parametrlar asosida chiziq chizadi. Uning 4 ta turi mavjud.

DrawLine(Pen, Int32, Int32, Int32, Int32) - `void DrawLine(System::Drawing::Pen ^ pen, int x1, int y1, int x2, int y2);`

DrawLine(Pen, Point, Point) - `void DrawLine(System::Drawing::Pen ^ pen, System::Drawing::Point pt1, System::Drawing::Point pt2);`

DrawLine(Pen, PointF, PointF) - `void DrawLine(System::Drawing::Pen ^ pen, System::Drawing::PointF pt1, System::Drawing::PointF pt2);`

DrawLine(Pen, Single, Single, Single, Single) - `void DrawLine(System::Drawing::Pen ^ pen, float x1, float y1, float x2, float y2);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

22. DrawLines. Bu funksiya berilgan parametrlar asosida ulanadigan chiziq chizadi. Uning 2 ta turi mavjud.

DrawLines(Pen, Point[]) - `void DrawLines(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Point> ^ points);`

DrawLines(Pen, PointF[]) - `void DrawLines(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

17. DrawPath. Bu funksiya berilgan parametrlar asosida bog‘langan chiziqlar va egri chiziqlar qatorini chizadi. Bu sinf merosxo‘r bo‘lishi mumkin emas.

DrawPath(Pen, GraphicsPath) - `void DrawPath(System::Drawing::Pen ^ pen, System::Drawing::Drawing2D::GraphicsPath ^ path);`

18. DrawPie. Bu funksiya berilgan parametrlar asosida Koordinata jufti, kengligi, balandligi va ikkita radial chiziq bilan belgilangan ellips bilan belgilangan nok shaklini chizadi. Uning 4 ta turi mavjud.

DrawPie(Pen, Int32, Int32, Int32, Int32, Int32, Int32) - `void DrawPie(System::Drawing::Pen ^ pen, int x, int y, int width, int height, int startAngle, int sweepAngle);`

DrawPie(Pen, Rectangle, Single, Single) - `void DrawPie(System::Drawing::Pen ^ pen, System::Drawing::Rectangle rect, float startAngle, float sweepAngle);`

DrawPie(Pen, RectangleF, Single, Single) - `void DrawPie(System::Drawing::Pen ^ pen, System::Drawing::RectangleF rect, float startAngle, float sweepAngle);`

DrawPie(Pen, Single, Single, Single, Single, Single, Single) - `void DrawPie(System::Drawing::Pen ^ pen, float x, float y, float width, float height, float startAngle, float sweepAngle);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

19. DrawPolygon. Bu funksiya berilgan parametrlar asosida ko‘p shaklini chizadi. Uning 4 ta turi mavjud.

DrawPolygon(Pen, Point[]) - `void DrawPolygon(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Point> ^ points);`

DrawPolygon(Pen, PointF[]) - `void DrawPolygon(System::Drawing::Pen ^ pen, cli::array <System::Drawing::PointF> ^ points);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

20. DrawRectangle. Bu funksiya berilgan parametrlar asosida to‘rtburchak shaklini chizadi. Uning 3 ta turi mavjud.

DrawRectangle(Pen, Int32, Int32, Int32, Int32) - `void DrawRectangle(System::Drawing::Pen ^ pen, int x, int y, int width, int height);`

DrawRectangle(Pen, Rectangle) - `void DrawRectangle(System::Drawing::Pen ^ pen, System::Drawing::Rectangle rect);`

DrawRectangle(Pen, Single, Single, Single, Single) - `void DrawRectangle(System::Drawing::Pen ^ pen, float x, float y, float width, float height);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

21. DrawRectangles. Bu funksiya berilgan parametrlar asosida ulanadigan to‘rtburchak shaklini chizadi. Uning 3 ta turi mavjud.

DrawRectangles(Pen, Rectangle[]) - `void DrawRectangles(System::Drawing::Pen ^ pen, cli::array <System::Drawing::Rectangle> ^ rects);`

DrawRectangles(Pen, RectangleF[]) - `void DrawRectangles(System::Drawing::Pen ^ pen, cli::array <System::Drawing::RectangleF> ^ rects);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

22. DrawString. Bu funksiya berilgan parametrlar asosida shrift obyektlar bilan belgilangan joyda belgilangan matn chizadi (joylashtiradi). Uning 6 ta turi mavjud.

DrawString(String, Font, Brush, PointF) - `void DrawString(System::String ^ s, System::Drawing::Font ^ font, System::Drawing::Brush ^ brush, System::Drawing::PointF point);`

DrawString(String, Font, Brush, PointF, StringFormat) - `void DrawString(System::String ^ s, System::Drawing::Font ^ font, System::Drawing::Brush ^ brush, System::Drawing::PointF point, System::Drawing::StringFormat ^ format);`

DrawString(String, Font, Brush, RectangleF) - `void DrawString(System::String ^ s, System::Drawing::Font ^ font, System::Drawing::Brush ^ brush, System::Drawing::RectangleF layoutRectangle);`

DrawString(String, Font, Brush, RectangleF, StringFormat) - `void DrawString(System::String ^ s, System::Drawing::Font ^ font, System::Drawing::Brush ^ brush, System::Drawing::RectangleF layoutRectangle, System::Drawing::StringFormat ^ format);`

DrawString(String, Font, Brush, Single, Single) - `void DrawString(System::String ^ s, System::Drawing::Font ^ font, System::Drawing::Brush ^ brush, float x, float y);`

DrawString(String, Font, Brush, Single, Single, StringFormat) - `void DrawString(System::String ^ s, System::Drawing::Font ^ font, System::Drawing::Brush ^ brush, float x, float y, System::Drawing::StringFormat ^ format);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

23. EndContainer. Bu funksiya berilgan parametrlar asosida joriy grafik konteynerni yopadi va BeginContainer() usuli uchun ko‘rsatkich orqali saqlangan bu grafik holatini chizadi. EndContainer(GraphicsContainer) - `void EndContainer(System::Drawing::Drawing2D::GraphicsContainer ^ container);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

24. EnumerateMetafile. Bu funksiya berilgan parametrlar asosida matnni metafaylga uzatadi va belgilangan joyda chizadi (joylashtiradi). Uning 36 ta turi mavjud.

EnumerateMetafile(Metafile, Point, Graphics+EnumerateMetafileProc) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Point destPoint, System::Drawing::Graphics::EnumerateMetafileProc ^ callback);

EnumerateMetafile(Metafile, Point, Graphics+EnumerateMetafileProc, IntPtr) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Point destPoint, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData);

EnumerateMetafile(Metafile, Point, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Point destPoint, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData, System::Drawing::Imaging::ImageAttributes ^ imageAttr);

EnumerateMetafile(Metafile, Point, Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Point destPoint, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData, System::Drawing::Imaging::ImageAttributes ^ imageAttr);

EnumerateMetafile(Metafile, Point, Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Point destPoint, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit,

```
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData);
```
EnumerateMetafile(Metafile, Point, Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) -
```
void EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, System::Drawing::Point destPoint,
System::Drawing::Rectangle srcRect,
System::Drawing::GraphicsUnit unit,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData,
System::Drawing::Imaging::ImageAttributes ^
imageAttr);
```
EnumerateMetafile(Metafile, Point[], Graphics+EnumerateMetafileProc) -
```
void EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::Point> ^
destPoints,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback);
```
EnumerateMetafile(Metafile, Point[], Graphics+EnumerateMetafileProc, IntPtr) -
```
void EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::Point> ^
destPoints,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData);
```
EnumerateMetafile(Metafile, Point[], Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) -
```
void EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::Point> ^
destPoints,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData,
System::Drawing::Imaging::ImageAttributes ^
imageAttr);
```
EnumerateMetafile(Metafile, Point[], Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc) -
```
void EnumerateMetafile(System::Drawing::Imaging::Metafile
```

^ metafile, cli::array <System::Drawing::Point> ^ destPoints, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Graphics::EnumerateMetafileProc ^ callback);

EnumerateMetafile(Metafile, Point[], Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, cli::array <System::Drawing::Point> ^ destPoints, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData);

EnumerateMetafile(Metafile, Point[], Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, cli::array <System::Drawing::Point> ^ destPoints, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit unit, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData, System::Drawing::Imaging::ImageAttributes ^ imageAttr);

EnumerateMetafile(Metafile, PointF, Graphics+EnumerateMetafileProc) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::PointF destPoint, System::Drawing::Graphics::EnumerateMetafileProc ^ callback);

EnumerateMetafile(Metafile, PointF, Graphics+EnumerateMetafileProc, IntPtr) - void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::PointF destPoint, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData);

EnumerateMetafile(Metafile, PointF, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - void EnumerateMetafile(System::Drawing::Imaging::Metafile

```
^ metafile, System::Drawing::PointF destPoint,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData,
System::Drawing::Imaging::ImageAttributes ^
imageAttr);
```

EnumerateMetafile(Metafile, PointF, RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, System::Drawing::PointF destPoint,
System::Drawing::RectangleF srcRect,
System::Drawing::GraphicsUnit srcUnit,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback);
```

EnumerateMetafile(Metafile, PointF, RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, System::Drawing::PointF destPoint,
System::Drawing::RectangleF srcRect,
System::Drawing::GraphicsUnit srcUnit,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData);
```

EnumerateMetafile(Metafile, PointF, RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, System::Drawing::PointF destPoint,
System::Drawing::RectangleF srcRect,
System::Drawing::GraphicsUnit unit,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData,
System::Drawing::Imaging::ImageAttributes ^
imageAttr);
```

EnumerateMetafile(Metafile, PointF[], Graphics+EnumerateMetafileProc) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::PointF> ^
destPoints,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback);
```

EnumerateMetafile(Metafile, PointF[], Graphics+EnumerateMetafileProc, IntPtr) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::PointF> ^
destPoints,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData);
```

EnumerateMetafile(Metafile, PointF[], Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::PointF> ^
destPoints,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData,
System::Drawing::Imaging::ImageAttributes ^
imageAttr);
```

EnumerateMetafile(Metafile, PointF[], RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::PointF> ^
destPoints, System::Drawing::RectangleF srcRect,
System::Drawing::GraphicsUnit srcUnit,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback);
```

EnumerateMetafile(Metafile, PointF[], RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::PointF> ^
destPoints, System::Drawing::RectangleF srcRect,
System::Drawing::GraphicsUnit srcUnit,
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData);
```

EnumerateMetafile(Metafile, PointF[], RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - void

```
EnumerateMetafile(System::Drawing::Imaging::Metafile
^ metafile, cli::array <System::Drawing::PointF> ^
destPoints, System::Drawing::RectangleF srcRect,
System::Drawing::GraphicsUnit unit,
```

```
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData,
System::Drawing::Imaging::ImageAttributes ^
imageAttr);
```

EnumerateMetafile(Metafile, Rectangle, Graphics+EnumerateMetafileProc) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Rectangle destRect, System::Drawing::Graphics::EnumerateMetafileProc ^ callback);`

EnumerateMetafile(Metafile, Rectangle, Graphics+EnumerateMetafileProc, IntPtr) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Rectangle destRect, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData);`

EnumerateMetafile(Metafile, Rectangle, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Rectangle destRect, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData, System::Drawing::Imaging::ImageAttributes ^ imageAttr);`

EnumerateMetafile(Metafile, Rectangle, Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Rectangle destRect, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Graphics::EnumerateMetafileProc ^ callback);`

EnumerateMetafile(Metafile, Rectangle, Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Rectangle destRect, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit srcUnit,`

```
System::Drawing::Graphics::EnumerateMetafileProc ^
callback, IntPtr callbackData);
```

EnumerateMetafile(Metafile, Rectangle, Rectangle, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::Rectangle destRect, System::Drawing::Rectangle srcRect, System::Drawing::GraphicsUnit unit, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData, System::Drawing::Imaging::ImageAttributes ^ imageAttr);`

EnumerateMetafile(Metafile, RectangleF, Graphics+EnumerateMetafileProc) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::RectangleF destRect, System::Drawing::Graphics::EnumerateMetafileProc ^ callback);`

EnumerateMetafile(Metafile, RectangleF, Graphics+EnumerateMetafileProc, IntPtr) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::RectangleF destRect, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData);`

EnumerateMetafile(Metafile, RectangleF, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::RectangleF destRect, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData, System::Drawing::Imaging::ImageAttributes ^ imageAttr);`

EnumerateMetafile(Metafile, RectangleF, RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::RectangleF destRect, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit srcUnit,`

```
System::Drawing::Graphics::EnumerateMetafileProc ^
callback);
```

EnumerateMetafile(Metafile, RectangleF, RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::RectangleF destRect, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit srcUnit, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData);`

EnumerateMetafile(Metafile, RectangleF, RectangleF, GraphicsUnit, Graphics+EnumerateMetafileProc, IntPtr, ImageAttributes) - `void EnumerateMetafile(System::Drawing::Imaging::Metafile ^ metafile, System::Drawing::RectangleF destRect, System::Drawing::RectangleF srcRect, System::Drawing::GraphicsUnit unit, System::Drawing::Graphics::EnumerateMetafileProc ^ callback, IntPtr callbackData, System::Drawing::Imaging::ImageAttributes ^ imageAttr);`

25. Equals. Belgilangan obyekt joriy obyektga teng yoki yo‘qligini aniqlaydi. Equals(Object) - Belgilangan obyekt joriy obyektga teng yoki yo‘qligini aniqlaydi.

26. ExcludeClip. Bu funksiya berilgan parametrlar asosida ko‘rsatilgan maydonni istisno qilish uchun ushbu grafikaning klip hududini yangilaydi. Uning 2 ta turi mavjud.

ExcludeClip(Rectangle) - `void ExcludeClip(System::Drawing::Rectangle rect);`

ExcludeClip(Region) - `void ExcludeClip(System::Drawing::Region ^ region);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

27. FillClosedCurve. Bu funksiya berilgan parametrlar asosida yopiq egri chiziqli splaynni bo‘yalgan holda chizadi. Uning 6 ta turi mavjud.

FillClosedCurve(Brush, Point[]) - `void FillClosedCurve(System::Drawing::Brush ^ brush, cli::array <System::Drawing::Point> ^ points);`

FillClosedCurve(Brush, Point[], FillMode) - `void FillClosedCurve(System::Drawing::Brush ^ brush, cli::array <System::Drawing::Point> ^ points, System::Drawing::Drawing2D::FillMode fillmode);`

FillClosedCurve(Brush, Point[], FillMode, Single) - `void FillClosedCurve(System::Drawing::Brush ^ brush, cli::array <System::Drawing::Point> ^ points, System::Drawing::Drawing2D::FillMode fillmode, float tension);`

FillClosedCurve(Brush, PointF[]) - `void FillClosedCurve(System::Drawing::Brush ^ brush, cli::array <System::Drawing::PointF> ^ points);`

FillClosedCurve(Brush, PointF[], FillMode) - `void FillClosedCurve(System::Drawing::Brush ^ brush, cli::array <System::Drawing::PointF> ^ points, System::Drawing::Drawing2D::FillMode fillmode);`

FillClosedCurve(Brush, PointF[], FillMode, Single) - `void FillClosedCurve(System::Drawing::Brush ^ brush, cli::array <System::Drawing::PointF> ^ points, System::Drawing::Drawing2D::FillMode fillmode, float tension);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

28. FillEllipse. Bu funksiya berilgan parametrlar asosida ellipisni bo‘yalgan holda chizadi. Uning 4 ta turi mavjud.

FillEllipse(Brush, Int32, Int32, Int32, Int32) - `void FillEllipse(System::Drawing::Brush ^ brush, int x, int y, int width, int height);`

FillEllipse(Brush, Rectangle) - `void FillEllipse(System::Drawing::Brush ^ brush, System::Drawing::Rectangle rect);`

FillEllipse(Brush, RectangleF) - `void FillEllipse(System::Drawing::Brush ^ brush, System::Drawing::RectangleF rect);`

FillEllipse(Brush, Single, Single, Single, Single) - `void FillEllipse(System::Drawing::Brush ^ brush, float x, float y, float width, float height);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

29. FillPath. Bu funksiya berilgan parametrlar asosida bog‘langan va egri chiziqlar bo‘yalgan holda chizadi.

FillPath(Brush, GraphicsPath) - `void FillPath(System::Drawing::Brush ^ brush, System::Drawing::Drawing2D::GraphicsPath ^ path);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

30. FillPie. Bu funksiya berilgan parametrlar asosida bir juft koordinatalar, kenglik, balandlik va ikkita radial chiziq bilan belgilangan ellipsni bo‘yalgan holda chizadi. Uning 3 ta turi mavjud.

FillPie(Brush, Int32, Int32, Int32, Int32, Int32, Int32) - `void FillPie(System::Drawing::Brush ^ brush, int x, int y, int width, int height, int startAngle, int sweepAngle);`

FillPie(Brush, Rectangle, Single, Single) - `void FillPie(System::Drawing::Brush ^ brush, System::Drawing::Rectangle rect, float startAngle, float sweepAngle);`

FillPie(Brush, Single, Single, Single, Single, Single, Single) - `void FillPie(System::Drawing::Brush ^ brush, float x, float y, float width, float height, float startAngle, float sweepAngle);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

31. FillPolygon. Bu funksiya berilgan parametrlar asosida maydonni bo‘yalgan holda chizadi. Uning 4 ta turi mavjud.

FillPolygon(Brush, Point[]) - `void FillPolygon(System::Drawing::Brush ^ brush, cli::array <System::Drawing::Point> ^ points, System::Drawing::Drawing2D::FillMode fillMode);`

FillPolygon(Brush, Point[], FillMode) - `void FillPolygon(System::Drawing::Brush ^ brush, cli::array <System::Drawing::PointF> ^ points);`

FillPolygon(Brush, PointF[]) - `void FillPolygon(System::Drawing::Brush ^ brush, cli::array <System::Drawing::PointF> ^ points, System::Drawing::Drawing2D::FillMode fillMode);`

FillPolygon(Brush, PointF[], FillMode) - `void FillRectangle(System::Drawing::Brush ^ brush, int x, int y, int width, int height);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

32. FillRectangle. Bu funksiya berilgan parametrlar asosida to‘rtburchakni bo‘yalgan holda chizadi. Uning 4 ta turi mavjud.

FillRectangle(Brush, Int32, Int32, Int32, Int32) - `void FillRectangle(System::Drawing::Brush ^ brush, System::Drawing::Rectangle rect);`

FillRectangle(Brush, Rectangle) - `void FillRectangle(System::Drawing::Brush ^ brush, System::Drawing::RectangleF rect);`

FillRectangle(Brush, RectangleF) - `void FillRectangle(System::Drawing::Brush ^ brush, float x, float y, float width, float height);`

FillRectangle(Brush, Single, Single, Single, Single) - `void FillRectangles(System::Drawing::Brush ^ brush, cli::array <System::Drawing::Rectangle> ^ rects);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

33. FillRectangles. Bu funksiya berilgan parametrlar asosida ulanadigan to‘rtburchakni bo‘yalgan holda chizadi. Uning 2 ta turi mavjud.

FillRectangles(Brush, Rectangle[]) - `void FillRectangles(System::Drawing ::Brush ^ brush, cli::array <System::Drawing::RectangleF> ^ rects);`

FillRectangles(Brush, RectangleF[]) - `void FillRegion(System::Drawing::Brush ^ brush, System::Drawing::Region ^ region);`

Bu funksiyaining barcha parametrlari tahlil qilingan.

34. FillRegion. Bu funksiya berilgan parametrlar asosida sohani bo‘yalgan holda chizadi.

FillRegion(Brush, Region) - `void FillRegion(System::Drawing::Brush ^ brush, System::Drawing::Region ^ region);`

Grafika sinfining 34 ta usullarini varinatlari asosida keltirdik. Bu usullarni va xususiyatlarni tasvirlarni qurish orqali ko‘rib chiqamiz.

Graphics sinfi usulari asosida tasvirlarni qurish. Tasvirlarni qurish uchun **PictureBox** komponentasidan foydalanamiz.

Chiziq chizish. Bunda komponentaning berilgan joyida, rangli chiziq chizishni ko‘ramiz. Buning uchun 1 ta komponenta, 1 ta tugma va 4 ta textBox komponentalarini oynaga qulay qilib joylashtiramiz.

1-qadam. Oynaning Form1_Load hodisasida quyidagi algoritmni joylashtiramiz.

```
this->Text = "DrawLine - chiziq chizish";
button1->Text = "CHIZISH";
```

2-qadam. Komponenta va textBox xususiyatlarini sozlash amallarni bajarish mumkin.

3-qadam. Chiziq chizish uchun nuqtalarga atab `int myPoint[4];` o‘zgaruvchisi olamiz.

4-qadam. Komponentaning Paint degan hodisasiga quyidagi algoritmni joylashtiramiz.

```
e->Graphics->DrawLine(System::Drawing::Pens::Red,
                  myPoint[1], myPoint[2], myPoint[3],
myPoint[4]);
```

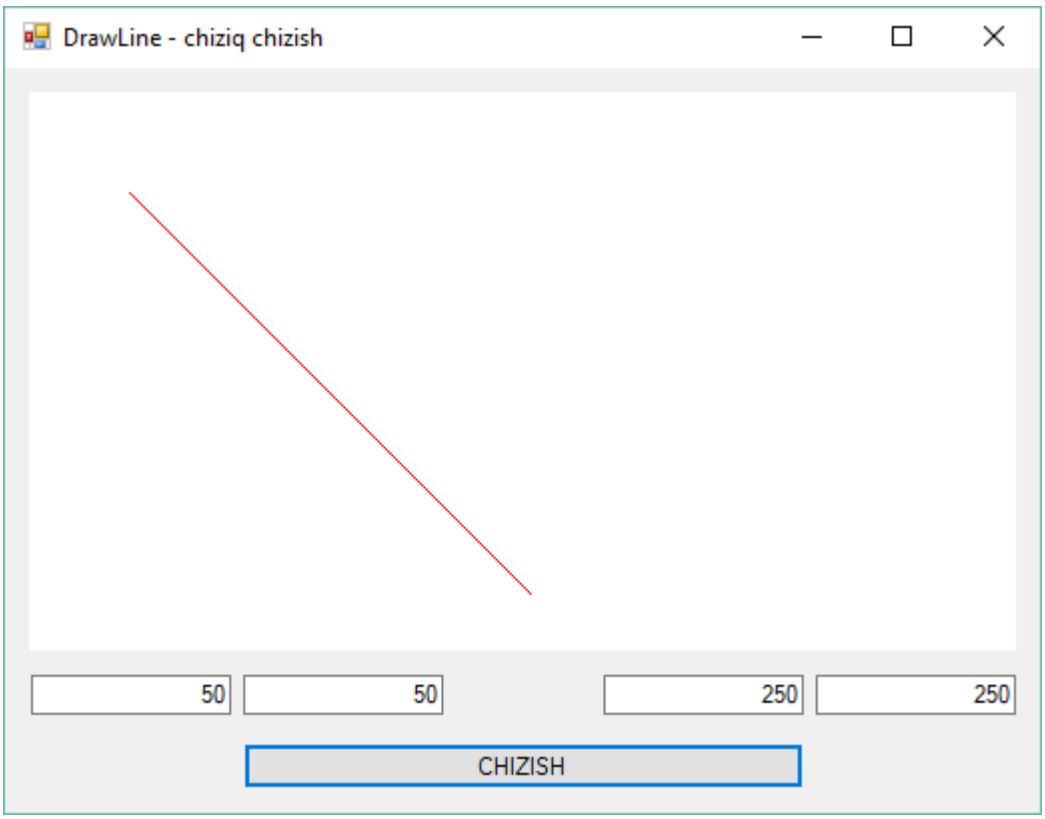
5-qadam. Tugmaning `button1_Click` hodisasida quyidagi algoritmni joylashtiramiz.

```
myPoint[1] = Convert::ToInt32(textBox1->Text);
myPoint[2] = Convert::ToInt32(textBox2->Text);
myPoint[3] = Convert::ToInt32(textBox3->Text);
myPoint[4] = Convert::ToInt32(textBox4->Text);
pictureBox1->Refresh();
```

Bunda matndan son tipiga o‘tish amalidan foydalanilgan va har bir tugma bosilganda komponentaning chizmasi o‘zgarib turadi.

Agar loyihani ishga tushursangiz, textBoxlarga kerakli sonlarni kiritib, chiziq chizish mumkin. Komponentaning o‘lchamidan katta chiziqlarni chizish mumkin ammo ko‘rinmaydi. Uni brshqarish lozim, ya‘ni kerakli kattalikdan oshib ketganda foydalanuvchiga muloqot oynasi bilan xabar berish mumkin, buni textBoxlarning fokusi o‘zgarganda amalga oshirish maqsadga muvofiq.

Chiziq chizishning natijasi 12.1-rasmda keltirilgan.



12.1-rasm. Chiziq chizish natijasi.

Uchburchak chizish. Bunda komponentaning berilgan joyida, rangli chiziqlar orqali uchbursak chizishni ko‘ramiz. Buning uchun 1 ta komponenta, 1 ta tugma va 6 ta textBox komponentalarini oynaga qulay qilib joylashtiramiz. Chuniki 3 ta chiziqning tutashtirsak, uchburchak hosil bo‘ladi.

1-qadam. Oynaning Form1_Load hodisasida quyidagi algoritmni joylashtiramiz.

```
this->Text = "DrawLine - uchburchak chizish";
button1->Text = "CHIZISH";
```

2-qadam. Komponenta va textBox xususiyatlarini sozlash amallarni bajarish mumkin.

3-qadam. Chiziq chizish uchun nuqtalarga atab `int myPoint[6];` o‘zgaruvchisi olamiz.

4-qadam. Komponentaning Paint degan hodisasiga quyidagi algoritmni joylashtiramiz.

```
e->Graphics->DrawLine(System::Drawing::Pens::Red,
                  myPoint[1], myPoint[2], myPoint[3],
myPoint[4]);
e->Graphics->DrawLine(System::Drawing::Pens::Black,
                  myPoint[3], myPoint[4], myPoint[5],
myPoint[6]);
e->Graphics->DrawLine(System::Drawing::Pens::Blue,
```

```
                    myPoint[1], myPoint[2], myPoint[5],
myPoint[6]);
```

5-qadam. Tugmaning button1_Click hodisasida quyidagi algoritmni joylashtiramiz.

```
myPoint[1] = Convert::ToInt32(textBox1->Text);
myPoint[2] = Convert::ToInt32(textBox2->Text);
myPoint[3] = Convert::ToInt32(textBox3->Text);
myPoint[4] = Convert::ToInt32(textBox4->Text);
myPoint[5] = Convert::ToInt32(textBox3->Text);
myPoint[6] = Convert::ToInt32(textBox4->Text);
pictureBox1->Refresh();
```

Agar loyihani ishga tushursangiz, textBoxlarga kerakli sonlarni kiritib, uchburchakni chizish mumkin.

Ellipis chizish. Bunda komponentaning berilgan joyida, rangli ellipis chizishni ko‘ramiz. Buning uchun 1 ta komponenta, 1 ta tugma va 4 ta textBox komponentalarini oynaga qulay qilib joylashtiramiz.

1-qadam. Oynaning Form1_Load hodisasida quyidagi algoritmni joylashtiramiz.

```
this->Text = "Ellipse - chizish";
button1->Text = "CHIZISH";
```

2-qadam. Komponenta va textBox xususiyatlarini sozlash amallarni bajarish mumkin.

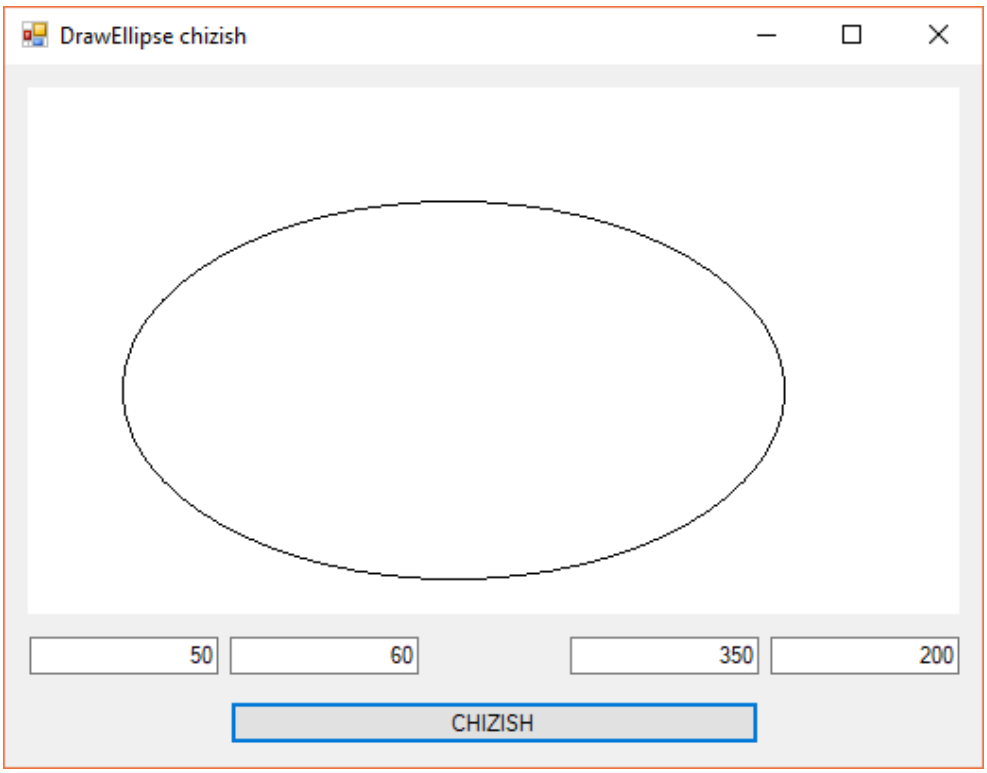
3-qadam. Chiziq chizish uchun nuqtalarga atab `int myPoint[4];` o‘zgaruvchisi olamiz.

4-qadam. Komponentaning Paint degan hodisasiga quyidagi algoritmni joylashtiramiz.

```
Pen ^ pen = gcnew Pen(Color::Black);
             e->Graphics->DrawEllipse(pen,
                 myPoint[1], myPoint[2], myPoint[3],
myPoint[4]);
```

5-qadam. Tugmaning button1_Click hodisasida quyidagi algoritmni joylashtiramiz.

```
myPoint[1] = Convert::ToInt32(textBox1->Text);
myPoint[2] = Convert::ToInt32(textBox2->Text);
myPoint[3] = Convert::ToInt32(textBox3->Text);
myPoint[4] = Convert::ToInt32(textBox4->Text);
pictureBox1->Refresh();
```



12.2-rasm. DrawEllipse chizish natijasi.

Yuqorida keltirilgan chizishlar orqali boshqa ixtiyoriy shakllarni ham chizish mumkin. Turli shakllarni chizib bo‘yashni ko‘ramiz.

Turli shakllarni bo‘yash. Bunda komponentaning berilgan joyida, bo‘yalgan shaklarni chizishni ko‘ramiz. Buning uchun **Brush** – mo‘yqalamdan foydalanib, 1 ta komponenta, 1 ta label va 1 ta Combox komponentalarini oynaga qulay qilib joylashtiramiz.

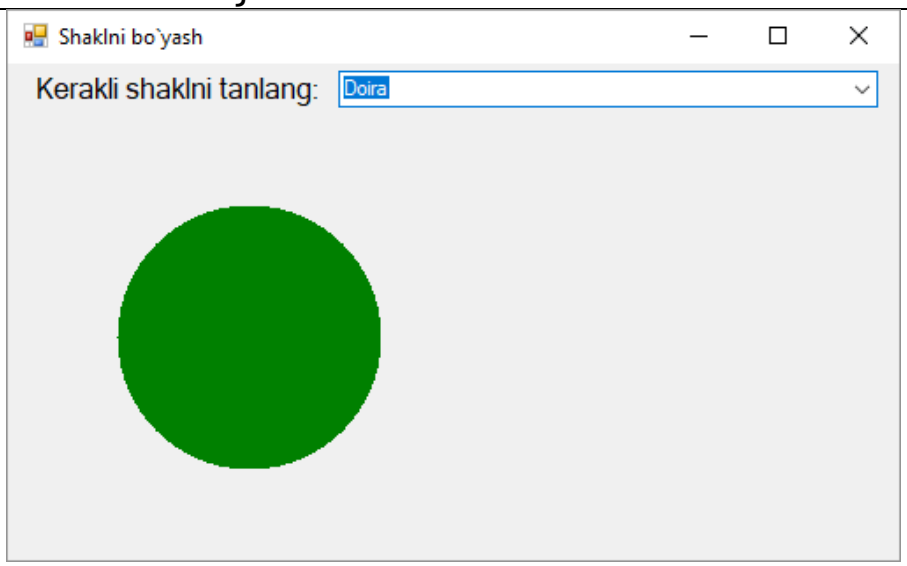
1-qadam. Oynaning Form1_Load hodisasida quyidagi algoritmni joylashtiramiz.

```
this->Text = "Shaklni bo‘yash";
                label1->Text = "Kerakli shaklni
tanlang:";
                comboBox1->Text = "Shakllar";
                comboBox1->Items-
>Add("To‘rtburchak");
                comboBox1->Items->Add("Ellipis");
                comboBox1->Items->Add("Doira");
                comboBox1->Items->Add("Qirqilgan
doira ");
```

2-qadam. Komponenta va ComboBox xususiyatlarini sozlash amallarni bajarish mumkin.

3-qadam. ComboBox komponentasining `comboBox1_SelectedIndexChanged` hodisasida quyidagi algoritmni joylashtiramiz.

```
Graphics ^grp = pictureBox1->CreateGraphics();
Brush ^ brsh = gcnew SolidBrush(Color::Green);
grp->Clear(SystemColors::Control);

switch (comboBox1->SelectedIndex){
     case 0:
        grp->FillRectangle(brsh,50,50,150,150);
break;
    case 1:
        grp->FillEllipse(brsh,50,50,300,150); break;
    case 2:
        grp->FillEllipse(brsh,50,50,150,150); break;
    case 3:
        grp->FillPie(brsh,50,50,150,150,150,100);
break;
    default:
              grp->Clear(SystemColors::Control);break;
                  }
```



12.3-rasm. Turli shakllarni bo‘yash natijasi.

Ko‘p burchaklarni chizish. Bunda komponentaning berilgan joyida, bo‘yalgan foydalanuvchi shaklarni chizishni ko‘ramiz. Buning uchun **Brush** – mo‘yqalamdan foydalanib, 1 ta komponenta, 1 ta button komponentasini oynaga qulay qilib joylashtiramiz.

1-qadam. Oynaning Form1_Load hodisasida quyidagi algoritmni joylashtiramiz.

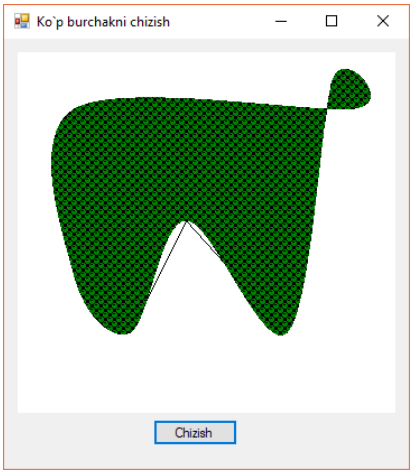
```
                    this->Text = "Ko‘p burchakni
chizish";
                    button1->Text = "Chizish";
```

2-qadam. Komponenta va button xususiyatlarini sozlash amallarni bajarish mumkin.

3-qadam. Button komponentasining `button1_Click` hodisasida quyidagi algoritmni joylashtiramiz.

```
Graphics^ graf = pictureBox1->CreateGraphics();
Pen^ gPen = gcnew Pen( Color::Black,1);
HatchBrush^ HBrush = gcnew HatchBrush(
HatchStyle::Sphere, Color::Green, Color::Black );

Point point1 = Point(50,50);
Point point2 = Point(50,200);
Point point3 = Point(100,250);
Point point4 = Point(150,150);
Point point5 = Point(240,250);
Point point6 = Point(280,25);
Point point7 = Point(300,50);
 array<Point> ^Points =
{point1,point2,point3,point4,point5,point6,point7};

graf->DrawPolygon( gPen, Points );
graf->FillClosedCurve( HBrush, Points);
```

Bu algoritmdagi HatchBrush sinfi ishlashi uchun using namespace System::Drawing::Drawing2D; nomlar fazosini qo‘shib qo‘yish kerak. Algoritmda 7 ta nuqtalarni birlashtirish orqali shakd chiziladi.



12.4-rasm. Turli shakllarni bo‘yash natijasi.

Hodisalar orqali shakl chizish. Hamma "Paint" kabi dasturlarni biladi. Uning eng ajoyib xususiyatlaridan biri sichqoncha bilan chiziqlar chizishdir. Buni amalga oshirish uchun komponentning sichqoncha bilan ishlash "**MousDown**", ""**MausUp**" i "**MouseMove**" hodisalarijan foydalanish mumkin. Dasturning algoritmi g‘oyasi quyidagicha: foydalanuvchi sichqonchaning chap tugmasini bosganda kursor orqasiga juda ko‘p kichik kvadratlar chizila boshlaydi. Bu kvadratlar hajmi kodda ko‘rsatilgan. Bundan tashqari, "tugma" tugmachasini rasmga tushiradigan shaklga ko‘chirishingiz kerak. Bunda komponentaning berilgan joyida, sichqoncha hodisalari orqali shaklarni chizishni ko‘ramiz. Buning uchun **Brush** – mo‘yqalamdan foydalanib, 1 ta komponenta, 1 ta button komponentasini oynaga qulay qilib joylashtiramiz.

1-qadam. Oynaning Form1_Load hodisasida quyidagi algoritmni joylashtiramiz.

```
button1->Text = "Tozalash";
this->Text = "Shakl chizish";
```

2-qadam. Komponenta va Button xususiyatlarini sozlash amallarni bajarish mumkin.

3-qadam. Button komponentasining `button1_Click` hodisasida quyidagi algoritmni joylashtiramiz. Bu algoritm komponentaning tozalash uchun ishlatiladi.

```
  Graphics ^ grp = pictureBox1->CreateGraphics();
  grp->Clear(SystemColors::Window);
```

4-qadam. Komponentaning sichqoncha hodisalarini qayta ishlash uchun mantiqiy bir o‘zgaruvchi olinadi, uning qiymati [0] bo‘lsin. Bool `Drow = false;`

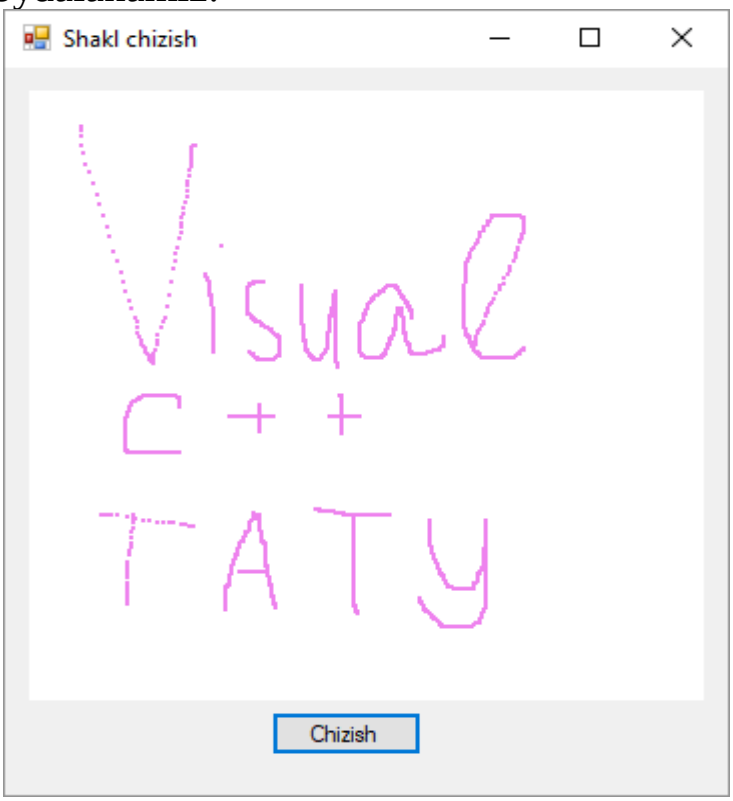
5-qadam. Sichqoncha komponentaning ustiga kelganda mantiqiy o‘zgaruvchining qiymati 1 ga o‘zgaradi. Chunki chizishni boshlash uchun. Buni amalga oshirishsh uchun sichqonchaning pictureBox1_MouseDown hodisasiga Drow = true; ni yozib qo‘yamiz.

6-qadam. Sichqoncha komponentaning ustidan ketganda mantiqiy o‘zgaruvchining qiymati 0 ga o‘zgaradi. Chunki chizishni tugatish uchun. Buni amalga oshirishsh uchun sichqonchaning pictureBox1_MouseUp hodisasiga Drow = false; ni yozib qo‘yamiz.

7-qadam. Sichqoncha komponentaning ustida kelganda chizishi uchun quyidagi algoritmni kiritamiz.

```
 Graphics^ graf = pictureBox1->CreateGraphics();
    if (Drow == true){
    graf->FillEllipse(Brushes::Violet, e->X, e->Y,
3,3); // mo‘y qalam qalinligi
}
```

Algoritmda shakllarni hosil qilish uchun juda kichik ellipischalardan foydalanamiz.



12.5-rasm. Sichqoncha bilan turli shakllarni chizish.

Rasmlarni o‘zgartirish. Buning uchun fayllarni yuklash **openFiledialog** va fayllarni saqlash **saveFiledialog** komponentalaridan foydalanamiz. Buning uchun **Brush** – mo‘yqalamdan foydalanib, 1 ta komponenta, 3 ta button komponentasini oynaga qulay qilib joylashtiramiz. Button larning birinchisi, rasmni tahrirlashni tozalashni

amalga oshiradi, ikkinchisi, tahrirlangan rasmni saqlash uchun, uchinchisi, rasmni tahrirlash uchun yuklashni amalga oshiradi. `using namespace System::Drawing::Drawing2D; ni qo‘shishb qo‘yish kerak.`

1-qalam. Rasmlarni o‘zgartirish uchun ishlatiladigan o‘zgaruvchilarni e‘lon qilamiz.

```
        private: Bitmap ^ bmp_for_draw;
    private: Point start_point;
    private: bool Draw;
    public: Pen^ pen_for_draw;
    private: String ^ full_name_of_image;
```

2-qadam. Dastur oynasining `Form1_Load` hodisasiga dastlabki sozlamalarni o‘rnatamiz.

```
this->Text = "Rasmni tahrirlash";
  button1->Text = "Tozalash";
  button2->Text= "Rasmni saqlash";
  button3->Text = "Rasmni yuklash";
  pen_for_draw = gcnew Pen(Color::Black, 4);
  pen_for_draw->StartCap =
System::Drawing::Drawing2D::LineCap::Round;
  pen_for_draw->EndCap                                    =
System::Drawing::Drawing2D::LineCap::Round;
```

Bunda komponentalarni sozlash va bir qalamni yaratib olish, uning StartCap, EndCap xususiyatlarini o‘rnatish amalga oshiriladi.

3-qadam. Komponentdagi rasmni tozalash uchun button1_Click hodisaga algoritmni yozamiz.

```
Graphics^ grp = pictureBox1->CreateGraphics();
grp->Clear(SystemColors::Window);
```

4-qadam. Sichqoncha komponentaning ustida kelganda chizishi uchun pictureBox1_MouseDown hodisasiga quyidagi algoritmni kiritamiz

```
if (e->Button ==
System::Windows::Forms::MouseButtons::Left){
                    Draw = true;
                    start_point = e->Location;
                }
```

5-qadam. Sichqoncha komponentaning ustida ketganda chizishni to‘xtatish uchun pictureBox1_MouseUp hodisasiga quyidagi algoritmni kiritamiz

```
    if (e->Button ==
System::Windows::Forms::MouseButtons::Left){
                        Draw = false;
                        }
```

6-qadam. Komponentaga kerakli rasmni yuklash uchun button3_Click hodisasiga quyidagi algoritmni yoziladi

```
OpenFileDialog ^ open_dialog = gcnew
OpenFileDialog();
open_dialog->Filter = "Image
Files(*.BMP;*.JPG;*.GIF;*.PNG)|*.BMP;*.JPG;*.GIF;*.PN
G|All files (*.*)|*.*";
if (open_dialog->ShowDialog() ==
System::Windows::Forms::DialogResult::OK){
    try{
        full_name_of_image = open_dialog->FileName;
           bmp_for_draw = gcnew Bitmap(open_dialog-
>FileName);
        //pictureBox1->Size = bmp_for_draw->Size;
        pictureBox1->SizeMode =
PictureBoxSizeMode::StretchImage;
        pictureBox1->Image = bmp_for_draw;
        pictureBox1->Invalidate();
    } catch(Exception^ e) {
    System::Windows::Forms::DialogResult rezult =
MessageBox::Show("Tanlangan faylni ochib bo'lmaydi "+
e->ToString(),"Diqqat", MessageBoxButtons::OK,
MessageBoxIcon::Error);
    }}
```

Bundagi asosi nuqson shundan iboratki, agar rasmning o‘lchamlari komponentaning o‘lchamlari bilan bir xil bo‘lsa, yaxshi chizadi, bir xil bo‘lmasa, foydalanuvchi ekranining proporsiyasini olib chizadi.

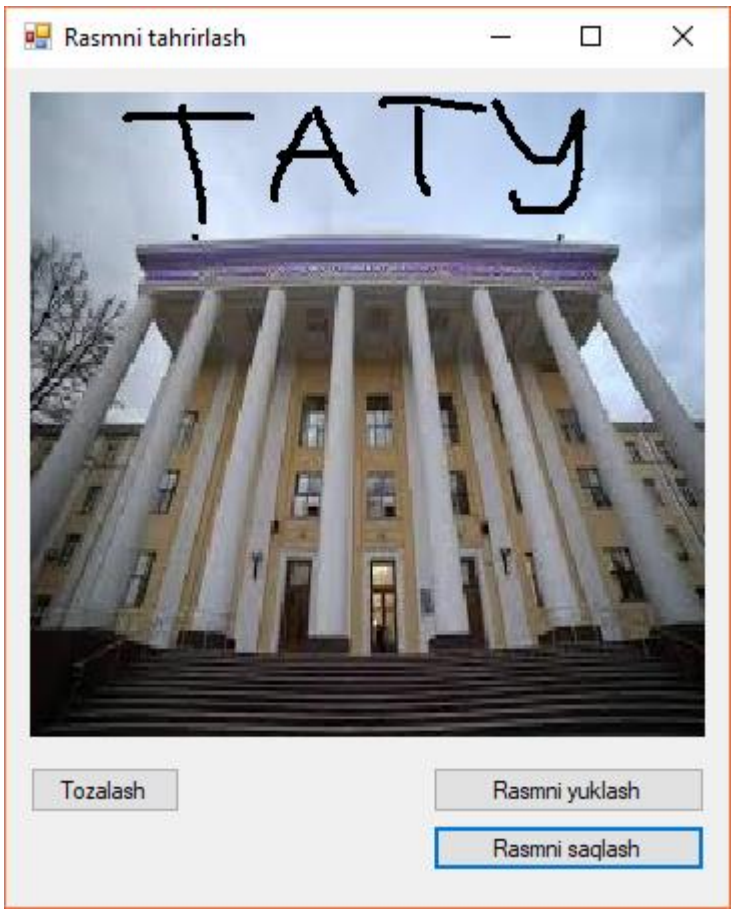
7-qadam. Komponentadagi rasmni saqlash uchun button2_Click hodisasiga quyidagi algoritmni yoziladi

```
if (pictureBox1->Image != nullptr){
String ^format = full_name_of_image-
>Substring(full_name_of_image->Length - 4, 4);
SaveFileDialog ^savedialog = gcnew SaveFileDialog();
savedialog->Title = "Rasmni saqlash ...";
savedialog->OverwritePrompt = true;
savedialog->CheckPathExists = true;
savedialog->Filter = "Image Files(*.BMP)|*.BMP|Image
Files(*.JPG)|*.JPG|Image Files(*.GIF)|*.GIF|Image
Files(*.PNG)|*.PNG|All files (*.*)|*.*";
savedialog->ShowHelp = true;
// If selected, save
if (savedialog->ShowDialog() ==
System::Windows::Forms::DialogResult::OK){
    try{
bmp_for_draw->Save(savedialog->FileName,
System::Drawing::Imaging::ImageFormat::Jpeg);
} catch(Exception ^ e) {
System::Windows::Forms::MessageBox::Show("Impossible
to save image"+e->ToString(), "FATAL ERROR",
MessageBoxButtons::OK, MessageBoxIcon::Error);
} } }
```

8-qadam. Sichqoncha komponentaning ustida kelganda chizishi uchun quyidagi algoritmni kiritamiz.

```
if (e->Button ==
System::Windows::Forms::MouseButtons::Left){
Graphics ^graf = Graphics::FromImage(pictureBox1-
>Image);
graf->DrawLine(pen_for_draw, start_point, e-
>Location);
start_point = e->Location;
pictureBox1->Invalidate();
}
```

Dasturni ishlatganda quyidagi oyna chiqadi.



12.6-rasm. Rasmlarni tahrirlash.

Matnlarni tasvir kabi joylashtirish. Bunda komponentaning belgilangan joyida, aniq o‘lchamli va rangli matnni joylashtirishni ko‘ramiz. Buning uchun komponentaga 3 ta textbox va 1ta botton komponentlarini qulay qilib joylashtiramiz. Birinchisi matn yozish uchun, qolgan ikkitasi matnni joylashtirish nuqtasi uchun.

1-qadam. Textbox ni rejimini multiline rejimiga va scrolBars xususityatiga Vertical qiymati o‘rnatiladi.

2-qadam. Oynaning Form1_Load hodisasida quyidagi algoritmni joylashtiramiz.

```
  Font = gcnew System::Drawing::Font("Times new
Roman", 12, FontStyle::Bold);
  button1->Text = "Chizish";
  this->Text = "Matn chizish";
```

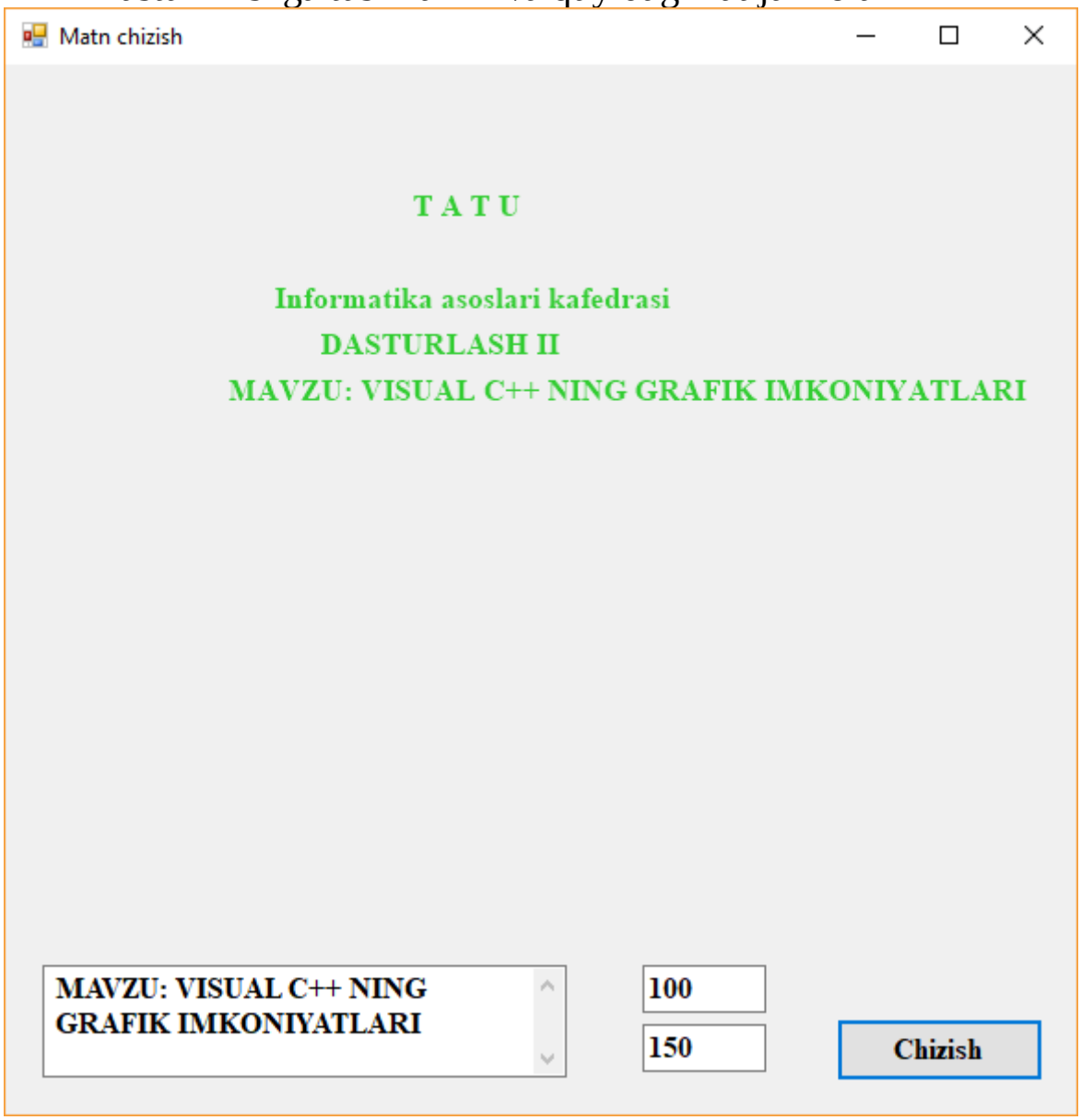
3-qadam. Komponenta va button xususiyatlarini sozlash amallarni bajarish mumkin.

4-qadam. Button komponentasining `button1_Click` hodisasida quyidagi algoritmni joylashtiramiz. Bu algoritm komponentaning tozalash uchun ishlatiladi.

```
  String^ Text = String::Format("{0}", textBox1-
```

```
>Text);
  Brush^ brsh = gcnew SolidBrush(Color::LimeGreen);
  Graphics^ G = pictureBox1->CreateGraphics();
  G->TextRenderingHint =
System::Drawing::Text::TextRenderingHint::AntiAlias;
  int xP = Convert::ToInt16(textBox2->Text);
  int yP = Convert::ToInt32(textBox3->Text);
  G->DrawString(Text, Font, brsh, xP, yP); //
joylashtirish manzili
```

Dasturni ishga tushiramiz va quyidagi natijani olamiz



12.7-rasm. Matnlarni joylashtirish.

Matnli rasmlarni yaratish. Matnli rasmlarni yaratish uchun 1 ta komponenta, 4 ta label, 4 ta button, 4 ta textbox larni qulay qilib joylashtiriladi.

1-qadam. 2 ta Textbox ni rejimini multiline rejimiga va scrolBars xususityatiga Vertical qiymati o‘rnatiladi.

2-qadam. Oynaning Form1_Load hodisasida quyidagi algoritmni joylashtiramiz.

```
                this->label1->Text = "| Yuqoridagi
matn";
                label2->Text = "|";
                label3->Text = "| Pastdagi matn";
                label4->Text = "|";
                button1->Text = "Tozalsh";
                button2->Text = "Chizish";
                button3->Text = "Saqlash";
                button4->Text = "Rasm yuklash";
```

3-qadam. Komponenta va button xususiyatlarini sozlash amallarni bajarish mumkin.

4-qadam. TextBox3 va TextBox4 ni mos ravishda TextBox1 va TextBox2 lar orqasiga tashlaymiz.

5-qadam. Formaning formBorderStyle xususiyatiga SizableToolWindow ni o‘rnatamiz. Formani o‘lchamlarini o‘zgartirmaslik uchun, buni boshqacha ham amalga oshirish mumkin.

6-qadam. TextBox3 va TextBox4 ni shrift o‘lchamlarini 12, TextBox1 va TextBox2 lar boshqa kattaroq o‘lchamni o‘rnatamiz. Chunki dastur ishga tushgada asosiy ko‘rinadigann matnlar ko‘rinmaydigan matnlarga ko‘chiriladi va undan komponentaga ko‘chiriladi.

7-qadam. Zarurriy o‘zgaruvchilarni aniqlashmiz.

```
            private: Bitmap^ bmp_for_draw;
        private: String^ full_name_of_image;
```

8-qadam. Oynaning Form1_Load hodisasi davomidan quyidagi algoritmni joylashtiramiz.

```
       Text = "Matnni rasmga joylashtirish";
       Font = gcnew System::Drawing::Font("Times new
Roman", 32, FontStyle::Bold);
          this->pictureBox1->SizeMode                =
PictureBoxSizeMode::StretchImage;
```

9-qadam. Komponentaga kerakli rasmni yuklash uchun button4_Click hodisasiga quyidagi algoritmni yoziladi

```
OpenFileDialog^ open_dialog = gcnew OpenFileDialog();
open_dialog->Filter = "Image
```

```
Files(*.BMP;*.JPG;*.GIF;*.PNG)|*.BMP;*.JPG;*.GIF;*.PN
G|All files (*.*)|*.*";
if (open_dialog->ShowDialog() ==
System::Windows::Forms::DialogResult::OK)        {
    try{
              full_name_of_image = open_dialog-
>FileName;
        bmp_for_draw = gcnew Bitmap(open_dialog-
>FileName);
           pictureBox1->Image = bmp_for_draw;
        pictureBox1->Invalidate();
        } catch(Exception^ e) {
        System::Windows::Forms::DialogResult rezult =
MessageBox::Show("Fayl xato tanlandi"+e-
>ToString(),"Warning",MessageBoxButtons::OK,
MessageBoxIcon::Error);
           } }
```

10-qadam. Komponentadagi rasmni saqlash uchun button3_Click hodisasiga quyidagi algoritmni yoziladi

```
if (pictureBox1->Image != nullptr){
  String^ format = full_name_of_image-
>Substring(full_name_of_image->Length - 4, 4);
  SaveFileDialog^ savedialog = gcnew
SaveFileDialog();
  savedialog->OverwritePrompt = true; // Agar shu
nomli fayl bo‘lsa
  savedialog->CheckPathExists = true; // Agar
noto‘g‘ri nom kiritilsa
  savedialog->ShowHelp = true;
  savedialog->Filter = "Image
Files(*.BMP)|*.BMP|Image Files(*.JPG)|*.JPG|Image
Files(*.GIF)|*.GIF|Image Files(*.PNG)|*.PNG|All files
(*.*)|*.*";
  if (savedialog->ShowDialog() ==
System::Windows::Forms::DialogResult::OK){
   try {
       Bitmap^ MBB = gcnew Bitmap(pictureBox1-
>Image);
```

```
       MBB->Save(savedialog->FileName,
System::Drawing::Imaging::ImageFormat::Jpeg);
      } catch(Exception^ e){
      MessageBox::Show("Rasmni saqlashda xatolik",
"FATAL ERROR",  MessageBoxButtons::OK,
MessageBoxIcon::Error);
      }  }  }
```

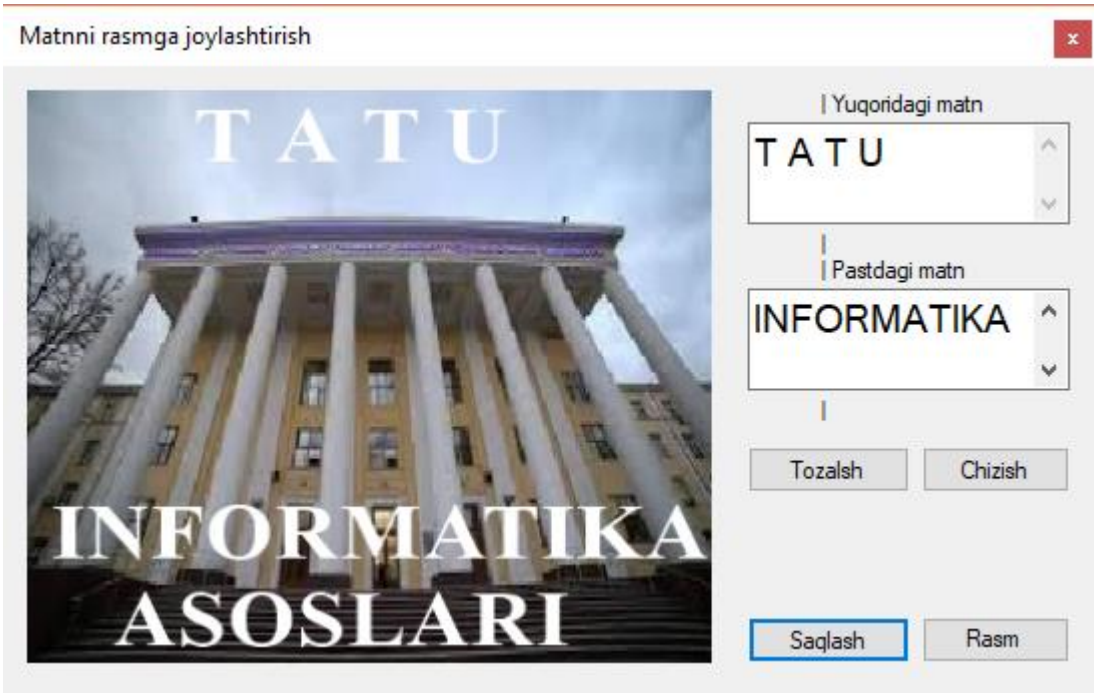
11-qadam. Komponentadagi rasmni tozalash uchun button1_Click hodisasiga quyidagi algoritmni yoziladi

```
Bitmap^ BM = gcnew Bitmap(pictureBox1->Image);
Graphics^ C =
System::Drawing::Graphics::FromImage(BM);
//Graphics C = CreateGraphics();
C->Clear(pictureBox1->BackColor);
pictureBox1->Image = BM;
```

12-qadam. Komponentaga matnlarni joylashtirish uchun uchun button2_Click hodisasiga quyidagi algoritmni yoziladi

```
System::Drawing::Font^ FontTemp  = gcnew
System::Drawing::Font("Times new Roman", 32,
FontStyle::Bold);
 textBox4->Text = textBox1->Text;
textBox3->Text = textBox2->Text;
SaveFileDialog^ savedialog = gcnew SaveFileDialog();
String^ Text = String::Format("{0}", textBox4->Text);
String^ Txt = String::Format("{0}", textBox3->Text);
Brush^ brsh = gcnew SolidBrush(Color::White);
Bitmap^ MBB = gcnew Bitmap(pictureBox1->Image);
Graphics^ G =
System::Drawing::Graphics::FromImage(MBB);
Graphics^ Q = G;
G->TextRenderingHint =
System::Drawing::Text::TextRenderingHint::AntiAlias;
G->DrawString(Text, FontTemp, brsh, 75, 2);
Q->TextRenderingHint =
System::Drawing::Text::TextRenderingHint::AntiAlias;
Q->DrawString(Txt, FontTemp, brsh, 5, 225);
pictureBox1->Image = MBB;
```

Dasturni ishga tushiramiz va quyidagi natijani olamiz



12.8-rasm. Matnlarni rasmga joylashtirish.

Graphics sinfi usulari asosida tasvirlarni qurishning turli usullari qarab chiqdik, bu amallar yordamida rasmlani tahrirlash, ishlov berish, matnlarni joylashtirish kabi ixtiyoriy amallarni bajarish mumkin.

Chart komponenta xususiyati va hodisalari. Bu komponenta Data tab bo‘limiga joylashgan bo‘lib, asosan ma’lumotlarni infografiklarni yaratish uchun ishlatiladi. Infografik uchun ma’lumotlar to‘plami kerak.

Bu komponentaning xusussiyati va hodisalari boshqa komponentalarniki kabi bo‘lib, xuddi o‘shalar kabi bo‘limlarga bo‘lingan. Ularning maxsuslarini keltirib o‘tamiz.

1. BorderSkin xususiyatlar gruppasi bo‘lib, unda komponentaning yangi niqobga solish mumkin. Niqob deganda, uning yangi ko‘rinishi inobatga olingan. Rang (color), rasm (image), stil (style), kengligi (width) kabi xususiyatlari mavjud. Bu xususiyatlarni o‘rnatish muammo keltirib chiqarmaydi. Oldingi o‘rganganlaringizda bunday xususiyatlardan foydalangansiz. Shuningdek asosiy niqob bu SkinStyle bo‘lib, komponentaning asosiy ko‘rinishini o‘zgartrish uchun xizmat qiladi. Uning mos qiymatlar ro‘yxati mavjud, shundan keraklisini tanlabolish mumkin.
2. Palette (palitra) – xususiyati yordamida komponentaning infografikani ko‘rsatadigan shaklini tanlash mumkin. Uning mos qiymatlar ro‘yxati mavjud.
3. PaletteCustomColors – bunda ham komponentaning infografikani ko‘rsatadigan shaklini tanlash mumkin. Ammo foydalanuvchi o‘zining

rangi tanlashimi mumkin. Bunda maxsus muloqot oynasi aosida palitraga turli ranglarni qo‘shish mumkin.

4. Annotations – bunda komponentaga izohlarni yozish mumkin. Buning o‘zining xususiyatlari maxsus muloqot oynasi yordamida o‘rnatiladi.

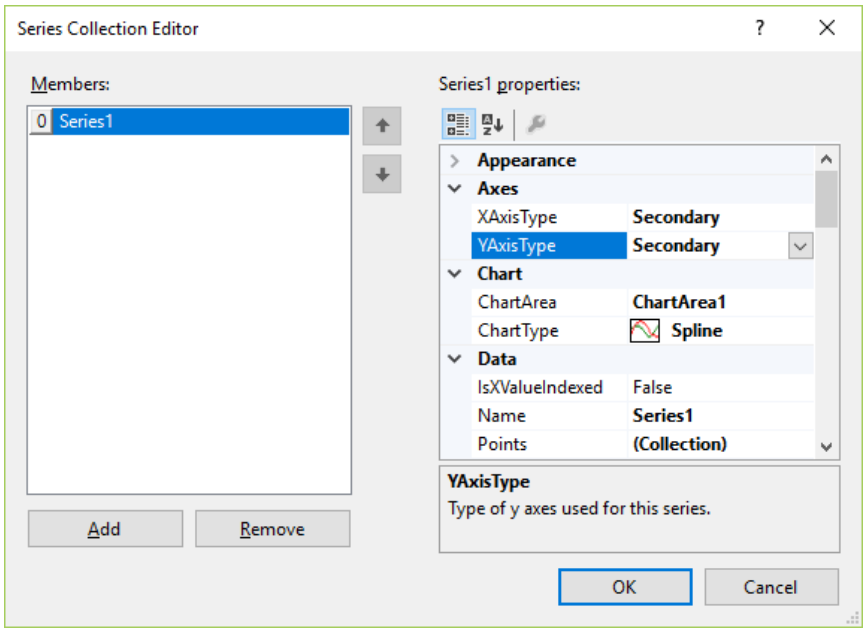
5. ChartAreas – bu xususiyat orqali komponentaga bir nechata chart infografika joylashtirish mumkin. Buning ham o‘zi mos xususiyatlarini mos muloqot oynasi bilan o‘rnatish lozim.

6. Legends – infografikaga keltirilgan qiymatlarining joylashish maydoni. Buni ham maxsus muloqot oynasi asosida tahrir qilish mumkin

7. Series – bu xususiyat asosiy bo‘lib, infografikaning qiymatlarini belgilovchi, har bir qiymat tegishliligini bildiradi. Buning uchun maxsus mulovot oynasi mavjud. Bunga to‘liqroq to‘xtalib o‘tamiz.

8. Titles – komponentalarga joylashtirilgan infografikalarga sarlavha qo‘yish uchun ishlatiladi. Uning maxsus muloqot oynasi orqali ishlov berish orqali o‘rnatish mumkin.

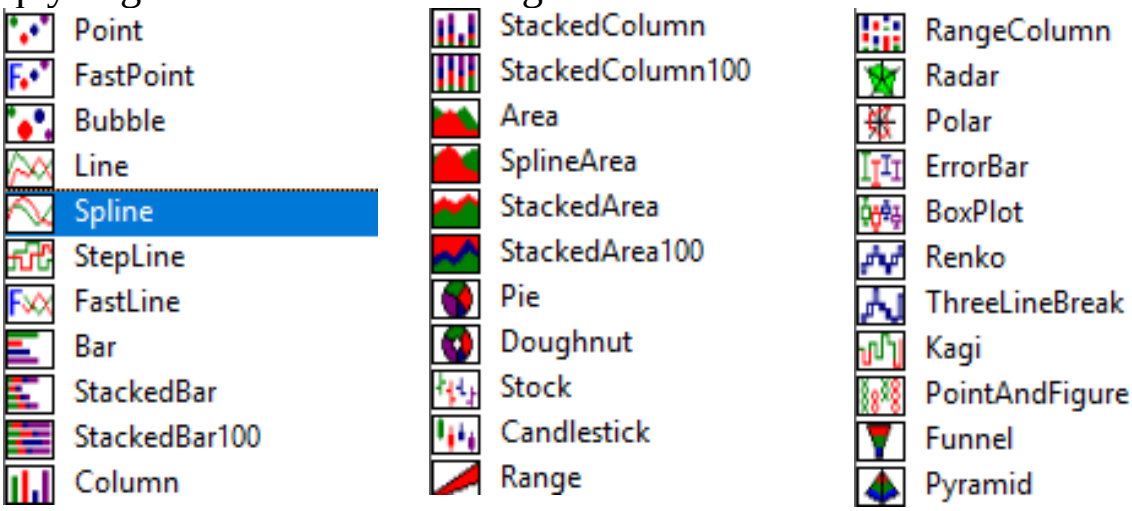
Series – bu komponentaga qiymatlarni qo‘shish va uni tasvirlash uchun xizmat qiladi. Komponentaning ichiga joylashtirilgan sohani tahrirlash uchun ishlatiladi. Unga bosganda quyidagi muloqot oynasi chiqadi.



12.9-rasm. Tahrirlash oynasi.

Bu oynda yangi qiymatlar maydoni qo‘shish uchun [Add] tugmasi va uni o‘chirish uchun [Remove] tugmasi ishlatiladi. Infografikaning asosiy xususiyatlarini boshqarish uchun o‘ng tomondagi xususiyatlar panelidan foydalanish mumkin. Unda infografikani taxhrirlash uchun

zarur bo‘lgan barcha xususiyatlar bor. Ulardan biri bu ChartType bo‘lib, infografikaning turlarini belgilash uchun xizmat qiladi. Uning turlari quyidagi 12.10-rasmda keltirilgan.



12.10-rasm. Infografikaning turlari

Shuningdek, infografikaning ma’lumotlari, yozuvlari, qiymatlari, maydoni, qiymat turlari, chegaralari bilan ishlash xususiyatlari xam mavjud. Odatda bu xususiyatlarni dasturlash orqali dastur fragmentlarida o‘rnatish va foydalanish dasturchiga qulay hisoblanadi. Ammo vizual dusturlashning imkoniyatidan foydalanish uchun buni ham ishlatishni o‘rganish lozim.

Funksiyalarni grafiklarini qurish. Infografika komponentasiga mos ravishda funksiyalarni grafikgini chizish usullari ko‘rib chiqamiz.

Matematik funksiyalarni grafiklarini chizish uchun avval shu funksiyalarni bir sinfda yaratib olamiz.

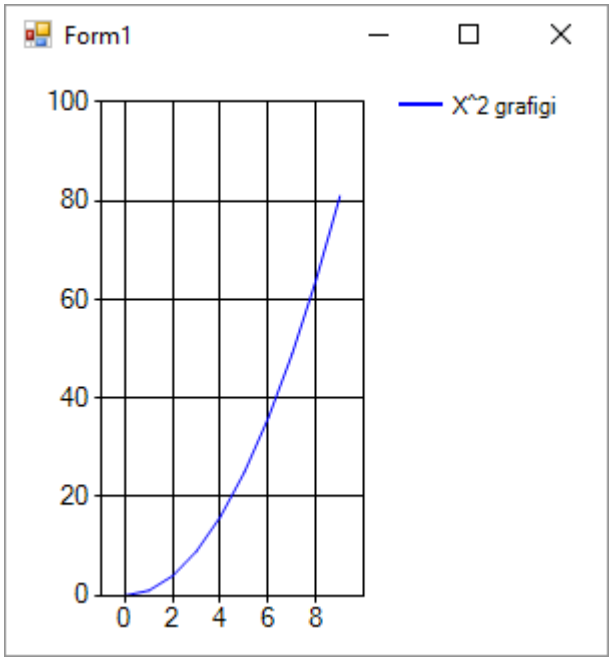
```
private value class MyFunction
    {
    private:
        double _value;
    public:
        double getValuePow(double x) {
            return Math::Pow(x, 2);
        }
        double  getValueX4(double x){
            return x*x*x*x;
        }
        double  getValuekxa(double x, int k, int a){
            return k*x+a;
        }
     };
```

Bu ko‘rinishda matematikaning barcha funksiyalarini yaratib olish yoki to‘g‘ridan to‘g‘ri foydalanish mumkin.

Chart komponentasiga grafikni chizish uchun Form1_Load hodisasiga quyidagicha algoritmni yozamiz.

```
chart1->Series->Clear();
    Series^ series1 = gcnew Series(L"X^2 grafigi");
    // rangni tanlash
    series1->Color = Color::Blue;
    series1->IsVisibleInLegend = true;
    series1->IsXValueIndexed = true;
    // infografikani turini tanlash
    series1->ChartType = SeriesChartType::Line;
    // qiymatlar qatlamini qo‘shish
    chart1->Series->Add(series1);
    // qiymatlarni
    MyFunction^ func = gcnew MyFunction();
    for (double i = 0; i < 10; i++) {
        series1->Points->AddXY(i, func-
>getValuePow(i));
    }
```

Agar dasturni ishlatsak, bir x^2 funksiyaning grafigini chizish imkoniyatini beradi.

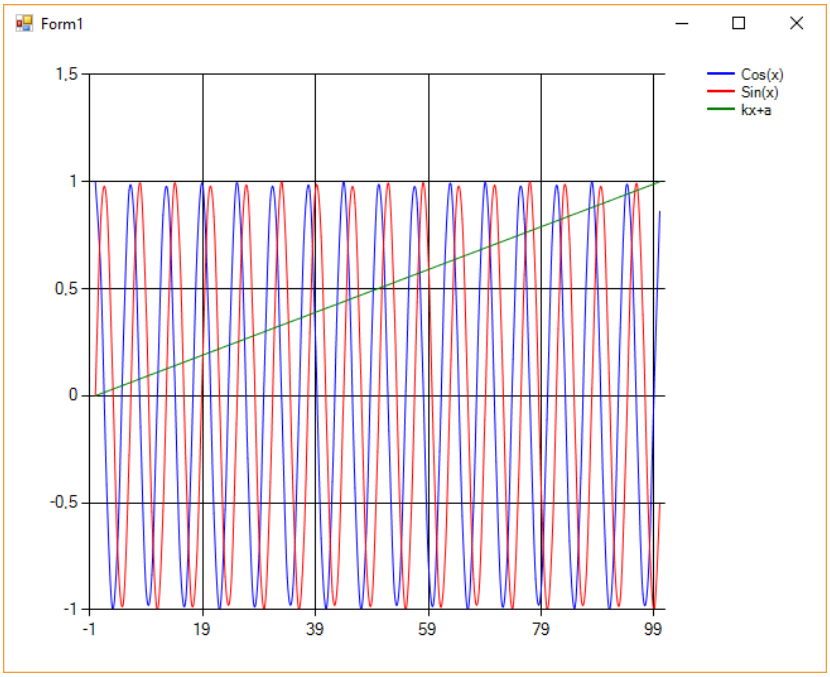


12.11-rasm. x^2 funksiyaning grafigi.

Bir vaqtning o‘zida bir nechta funksiyaning grafiklarini chizish uchun yuqorida aniqlangan sinfdan foydalanib, Form1_Load hodisasiga quyidagicha algoritmni yozamiz.

```
chart1->Series->Clear();
    Series^ series1 = gcnew Series(L"Cos(x)");
    Series^ series2 = gcnew Series(L"Sin(x)");
    Series^ series3 = gcnew Series(L"kx+a");
    // rangni tanlash
    series1->Color = Color::Blue;
    series2->Color = Color::Red;
    series3->Color = Color::Green;
    // infografikani turini tanlash
    series1->ChartType = SeriesChartType::Spline;
    series2->ChartType = SeriesChartType::Spline;
    series3->ChartType = SeriesChartType::Spline;
    // qiymatlar qatlamini qo‘shish
    chart1->Series->Add(series1);
    chart1->Series->Add(series2);
    chart1->Series->Add(series3);
    // qiymatlarni
    MyFunction^ func = gcnew MyFunction();
    for (double i = 0; i <= 100; i++) {
        series3->Points->AddXY(i, func-
>getValuekxa(i*0.01,1,0));
        series1->Points->AddXY(i, Math::Cos(i));
        series2->Points->AddXY(i, Math::Sin(i));
    }
```

Agar dasturni ishlatsak, bir funksiyalarning grafigini chizish imkoniyatini beradi.

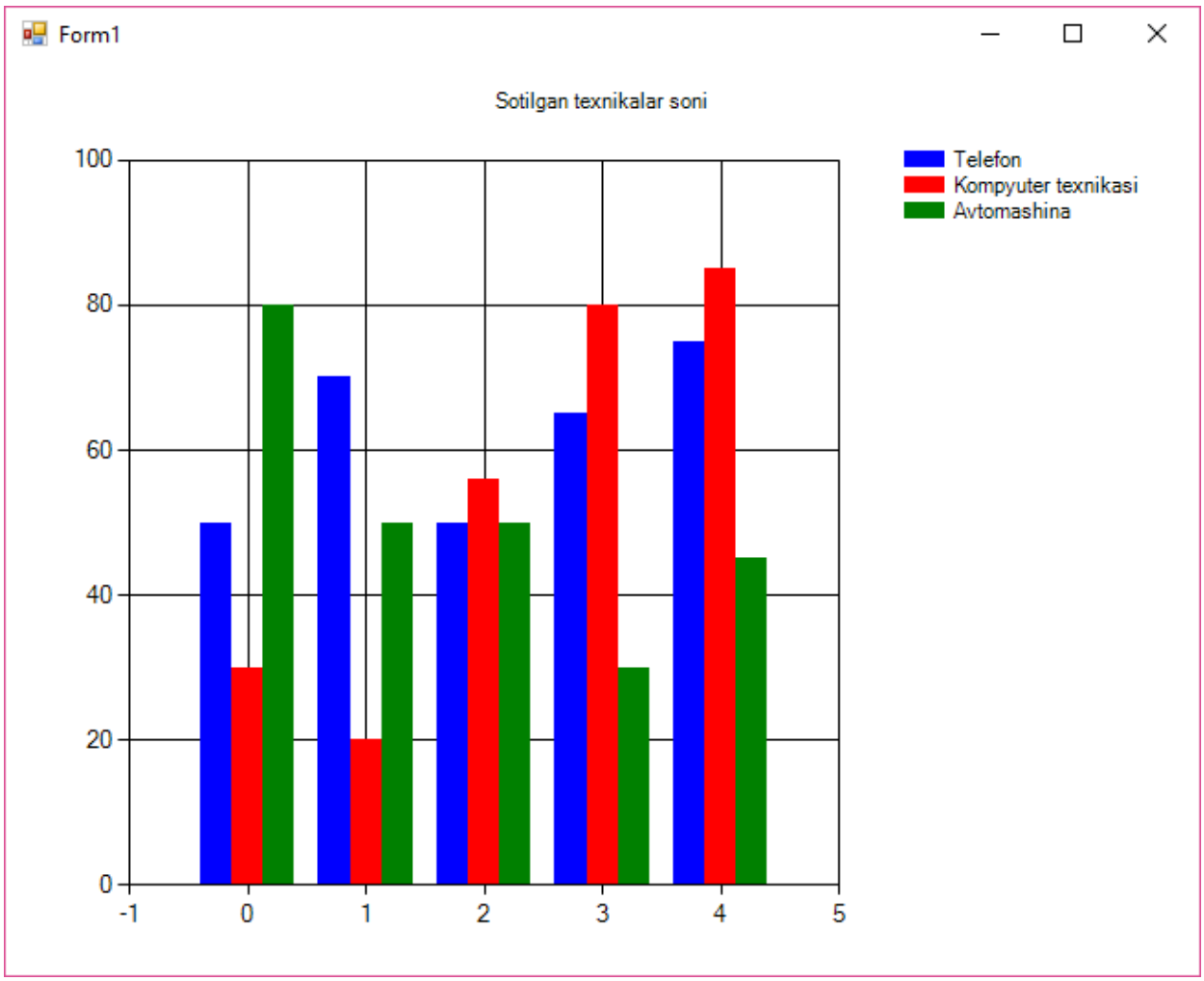


12.12-rasm. Funksiyalarning grafigi.

Gistagramma grafiklarini chizish uchun massivlardan yoki ixtiyoriy to‘plamlardan foydalanish mumkin. Buning uchun 3 ta massiv olamiz, Form1_Load hodisasiga quyidagicha algoritmni yozamiz

```
chart1->Series->Clear();
    Title^ title = gcnew Title("Sotilgan texnikalar
soni ");
    chart1->Titles->Add(title);
    Series^ series1 = gcnew Series(L"Telefon");
    Series^ series2 = gcnew Series(L"Kompyuter
texnikasi");
    Series^ series3 = gcnew Series(L"Avtomashina");
    // rangni tanlash
    series1->Color = Color::Blue;
    series2->Color = Color::Red;
    series3->Color = Color::Green;
    // infografikani turini tanlash
    series1->ChartType = SeriesChartType::Column;
    series2->ChartType = SeriesChartType::Column;
    series3->ChartType = SeriesChartType::Column;
    // qiymatlar qatlamini qo‘shish
    chart1->Series->Add(series1);
    chart1->Series->Add(series2);
    chart1->Series->Add(series3);
    // qiymatlarni
    array<int>^ arr1 = {30,20,56,80,85};
    array<int>^ arr2 = {50,70,50,65,75};
     array<int>^ arr3 = {80,50,50,30,45};
for (int i = 0; i < arr1->Length; i++) {
        series3->Points->AddXY(i, arr3[i]);
        series1->Points->AddXY(i, arr2->GetValue(i));
        series2->Points->AddXY(i, arr1->GetValue(i));
    }
```

Agar dasturni ishlatsak, Gistagramma grafigini chizish imkoniyatini beradi.



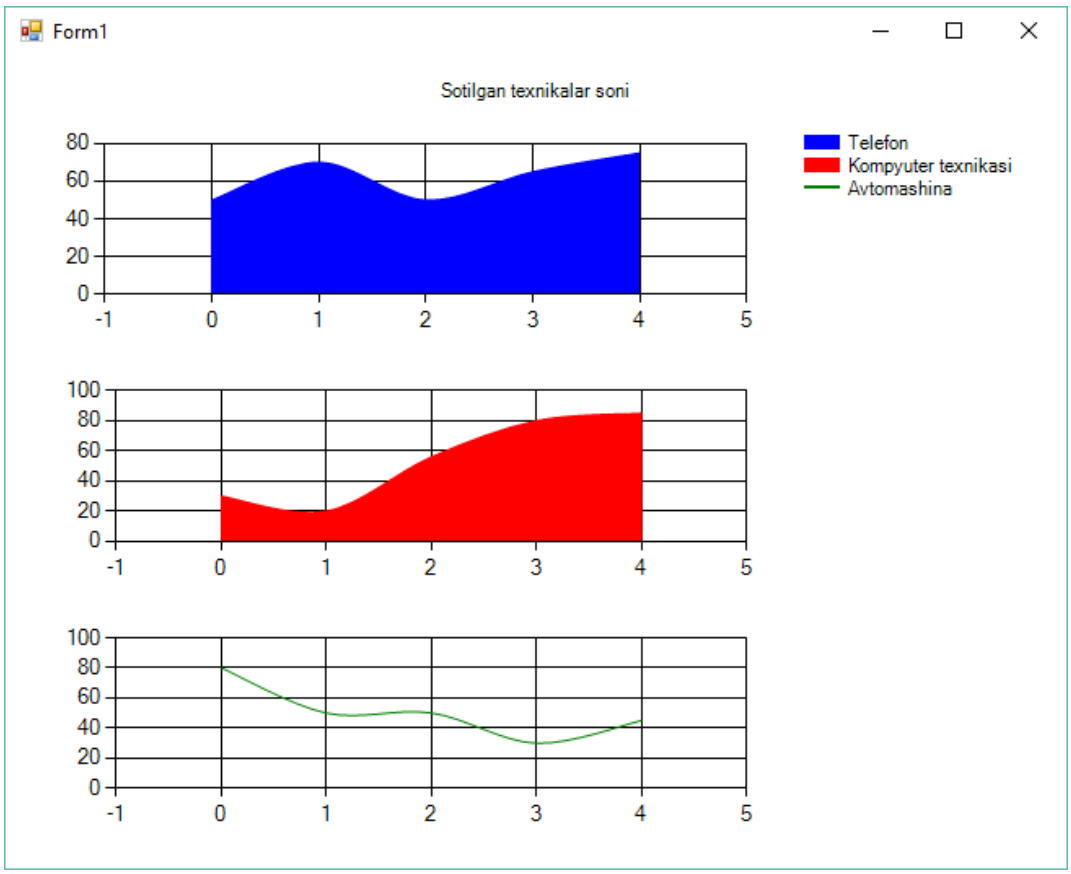
12.13-rasm. Gistagramma grafiklarini chizish

Grafiklarini alohida sohasha chizish uchun massivlardan yoki ixtiyoriy to‘plamlardan foydalanish mumkin. Buning uchun 3 ta massiv olamiz, Form1_Load hodisasiga quyidagicha algoritmni yozamiz

```
chart1->Series->Clear();
    ChartArea^ chartArea1 = gcnew ChartArea();
    ChartArea^ chartArea2 = gcnew ChartArea();
    ChartArea^ chartArea3 = gcnew ChartArea();
    chartArea1->Name = "1";
    chartArea2->Name = "2";
    chartArea3->Name = "3";
    this->chart1->ChartAreas->Add(chartArea1);
    this->chart1->ChartAreas->Add(chartArea2);
    this->chart1->ChartAreas->Add(chartArea3);
    Title^ title = gcnew Title("Sotilgan texnikalar
soni ");
    chart1->Titles->Add(title);
    Series^ series1 = gcnew Series(L"Telefon");
```

```
    Series^ series2 = gcnew Series(L"Kompyuter
texnikasi");
    Series^ series3 = gcnew Series(L"Avtomashina");
    // rangni tanlash
    series1->Color = Color::Blue;
    series2->Color = Color::Red;
    series3->Color = Color::Green;
    // infografikani turini tanlash
    series1->ChartType = SeriesChartType::SplineArea;
    series2->ChartType =
SeriesChartType::SplineRange;
    series3->ChartType = SeriesChartType::Spline;
    series1->ChartArea = chartArea1->Name;
    series2->ChartArea = chartArea2->Name;
    series3->ChartArea = chartArea3->Name;
    // qiymatlar qatlamini qo‘shish
    chart1->Series->Add(series1);
    chart1->Series->Add(series2);
    chart1->Series->Add(series3);
    // qiymatlarni
    array<int>^ arr1 = {30,20,56,80,85};
    array<int>^ arr2 = {50,70,50,65,75};
    array<int>^ arr3 = {80,50,50,30,45};
    for (int i = 0; i < arr1->Length; i++) {
        series3->Points->AddXY( i, arr3[i]);
        series1->Points->AddXY( i, arr2-
>GetValue(i));
        series2->Points->AddXY( i, arr1-
>GetValue(i));
    }
```

Dastur fragmentini tahlil qilish orqali yangi infografika sohalarni qo‘shish va ularni qiymatlar o‘plami bilan bog‘lashni ko‘rishinig mumkin. Agar dasturni ishlatsak, Gistagramma grafigini chizish imkoniyatini beradi.



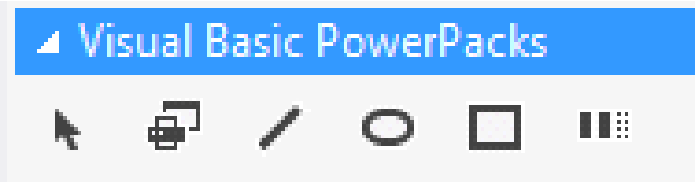
12.14-rasm. Bitta Chart ga bir nechta sohani joylashtirish.

Chart komponentasi yordamida foydlanuvchining ixtiyoriy ma’lumotlarini infografikasini yaratish mumkin. Bu juda katta imkoniyatli komponenta hisoblanadi. Buni hammasini o‘rganish uchun ko‘proq amaliyot qilish lozim. Yuqorida uning ba‘zi imkoniyatlarini keltirgan holda komponentaning xususiyatlari va hodisalari bo‘yicha nazariy va amaliy ma’lumotlarni keltirdik.

Visual Basic Power Packs komponentalari va ularni ishlatish. Bu komponentalar guruhi alohida bo‘lib, ularni Microsoft Visual Basic Power Packs deb atashadi. Buni yangi varintlarini ham internetdan olish va o‘rnatish mumkin.

Asosiy vazifasi formani bezash uchun ishlatiladi, shuning uchun barcha komponentalarga bor bo‘lgan hodisalar mavjud.

Bu Microsoft.VisualBasic PowerPacks nomlar fazosida joylashgan bo‘lib, Visual Basic Power Packs elementlar uchun sinflari mavjud. Visual Basic Power to‘plamlar elementlari qo‘shimcha Windows formalari elementlari uchun mo‘ljallangan. Ular dastlab bepul plagin-smaylik sifatida ishlatilgan va endi Visual Studio tarkibiga kiritilgan.



12.15-rasm. Visual Basic Power Packs komponentalari.

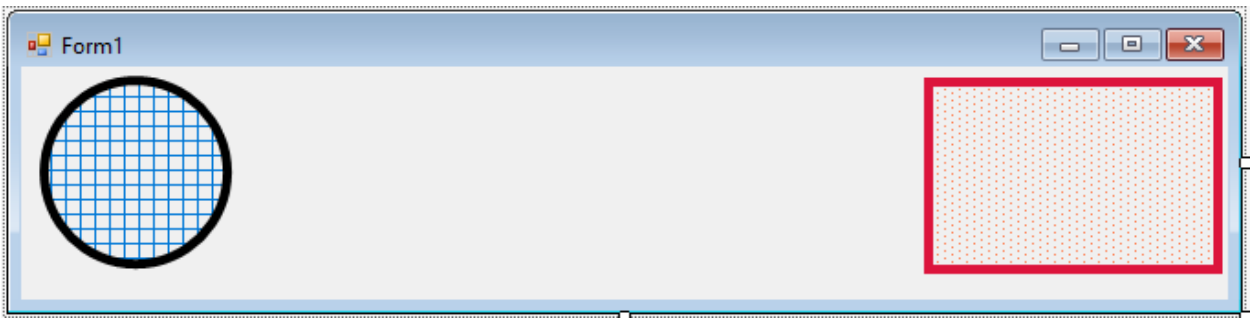
12.3-jadval. Visual Basic Power Packs komponentalari

Rasmi	Komponenta nomi	vazifasi
	PrintForm	Formani chop qilishga ruxsat berishni boshqaradi
	LineShape	Gorozantal, vertikal, diognal chiziqlarni boshqarishni ta‘minlaydi
	OvalShape	Oval ko‘rinishdagi obyektlarni boshqarishni ta‘minlaydi
	RectangleShape	To‘rtburchak ko‘rinishdagi obyektlarni boshqarishni ta‘minlaydi
	DataRepeater	Talab asosida formatlangan ma’lumotlarni ko‘rsatadi

OvalShape komponentasining ba‘zi xususiyatlari va hodisalari

1. BackColor – komponentaning orqa rangini o‘rnatadi va qiymatlari ranglar to‘plami.
2. BackStyle - komponentaning orqa stilini o‘rnatadi va qiymatlari 2 ta statik berilgan.
3. BorderColor - komponentaning chegara rangini o‘rnatadi va qiymatlari ranglar to‘plami.
4. BorderStyle - komponentaning chegara stilini o‘rnatadi va qiymatlari 6 ta statik berilgan.
5. BorderWight - komponentaning chegara rang qalinligini o‘rnatadi va qiymatlari sonlar.
6. FillColor - komponentaning aktiv bo‘lgandagi rangni o‘rnatadi va qiymatlari ranglar.
7. Location – komponentaning joylashuv o‘rnini o‘rnatish..
8. SelectionColor - komponentaning tanlangan bo‘lgandagi rangni o‘rnatadi va qiymatlari ranglar.

Bir loyiha yarating va unda 12.16-rasm kabi formani tayyorlang.



12.16-rasm. Loyiha ko‘rinishi.

Yaratilgan ovalShape1 ni forma bo‘yicha xarakatlanishni masalasini ko‘ramiz. Sizning formadagi 2 obyektni yaratishda ularning xususiyatlari bilan tanishib olgansiz.

Buning uchun loyihaga bir Timer1 obyektidan joylashtiramiz. Tez bajarilishi uchun uning interval xususiyatiga 1 qiymatni kiritamiz. Timer1 obyektining ustiga sichqonchani ikki marta bosib, timer1_Tick hodisasiga quyidagicha algoritm kiritamiz.

```
if(ovalpos <= (this->Size.Width - ovalShape1-
>Size.Width)){
  ovalpos += 5;
  ovalShape1->Location = Point(ovalpos,ovalShape1-
>Location.Y);
     }
 }
```

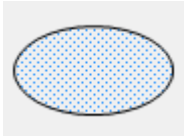
Bunda ovalpos qiymati 0ga teng bo‘lgan butun son tipidagi o‘zgaruvchi. Location – bu obyektning joylashinini belgilaydi va juft qiymatga ega Point qiymatni qabul qiladi.

Shuningdek, Form1_Load hodisasida quyidagi algoritmni yozamiz.

```
ovalpos = 0;
timer1->Start();
```

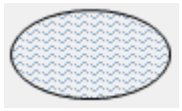
Agar dasturni ishga tushirsangiz ovalShape1 obyektining harakatini ko‘rasiz.

Shakllarning FillColor, FillGradientColor, FillGradientStyle, FillStyle xususiyatlaridan foydalanib, turli xil tugmalarni ham yaratish mumkin.



FillColor	MenuHighlight
FillGradientColor	Maroon
FillGradientStyle	ForwardDiagonal
FillStyle	Percent20

FillColor	Control
FillGradientColor	Tomato
FillGradientStyle	Horizontal
FillStyle	Solid



FillColor	ActiveCaption
FillGradientColor	White
FillGradientStyle	Horizontal
FillStyle	Wave
FillColor	ActiveCaption
FillGradientColor	OrangeRed
FillGradientStyle	Horizontal
FillStyle	SolidDiamond

Berilgan button tugmani bezash masalasini qaraymiz. Buning uchun formaga bitta tugma joylashtiramiz va uning xususiyatlarini quyidagicha aniqlaymiz.

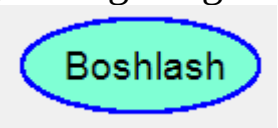
```
using namespace System::Drawing::Drawing2D;
using namespace System::Drawing::Text;
// ...
this->button1->Location = System::Drawing::Point(115,
438);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(121, 49);
this->button1->TabIndex = 0;
this->button1->Text = L"button1";
this->button1->UseVisualStyleBackColor = true;
```

Tugmaning button1_Paint hodisasiga o‘tib, quyidagi algoritimni yozamiz.

```
Pen^ pen = gcnew Pen(Color::Blue,10);
Brush^ brush = gcnew SolidBrush(Color::FromKnownColor
(KnownColor::Control));
SolidBrush ^ brushinside = gcnew SolidBrush(Color::
Aquamarine);
Graphics ^gr = e->Graphics;
gr->FillRectangle(brush, 0, 0, button1-
>Width,button1->Height);
gr->FillEllipse(brushinside, 0, 0, button1->Width,
button1->Height);
gr->DrawEllipse(pen, 0, 0, button1->Width, button1-
>Height);
GraphicsPath^ path = gcnew GraphicsPath();
button1->Region = gcnew
System::Drawing::Region(path);
String^ text = "Boshlash";
System::Drawing::Font^ drawfont = gcnew
```

```
System::Drawing::Font("Arial", 14);
SolidBrush^ exbrush = gcnew SolidBrush(Color::Black);
RectangleF rect = RectangleF(button1->Width/2-
40,button1->Height/2-10,button1->Width, button1-
>Height);
gr->DrawString(text, drawfont, exbrush,rect);
```

Bunda chizish uchun qalam - Pen , mo‘yqalam - Brush, to‘liq bo‘yash uchun mo‘yqalam - SolidBrush yaratib olamiz. Kerakli shakllarni chizish obyektinini tugmaning o‘lchamiga moslab tayyorlaymiz. Tugmaning ustiga matn joylagtirish uchun yana bir obyekt yaratamiz, unga matn, matn rangi, formati va egallash sohasini ko‘rsatib, DrawString bilan tugmaning ustiga chizamiz.



12.17-rasm. Tugmani bezashga misol.

Yangi yaratilgan tugmachalar orqali shakllarni oqim orqali boshqarish masalasini qaraymiz. Buning uchun Loyiha formasiga, 1 ta PictureBox, 2 ta button tugmani yuqoridagidek qilib joylashtiramiz. PictureBox ning BackColor xususiyatiga oq rangni o‘rnatimiz. Shaklning xarakatlantirish uchun uning tayanch nuchtasini ko‘chirib, oldingisi oq bilan bo‘yaladi va keyingisi hosil qilinadi, shunday qilib iteratsiya davom etaveradi.

Ikkita berilgan tugmalarni yuqoridagidek qilib, moslab olamiz.

Oqimni boshqarish uchun avval bir funksiya yaratib olamiz. Bu funksiya chizilgan shaklni PictureBox bo‘ylab harakatlanish va oldin keltirilgan iteratsiyani amalga oshirish uchun kerak. Funksiyaning algoritmi quyidagicha:

```
public: void Go(){
    int R = 30;
//   button2->BeginInvoke(gcnew
InvokeDelegateSetEnabled(this,
&Form1::setEnabledToButton), false);
    Bitmap ^ image = gcnew Bitmap(pictureBox1->Width,
pictureBox1->Height);
    Graphics^ g = Graphics::FromImage(image);
    for (int x = 0; x <= pictureBox1->Width - 2 * R;
x += 2){
                g->Clear(Color::White);
```

```
                g->FillEllipse(Brushes::Aqua, x, 0, 2
* R, 2 * R);
                pictureBox1->BeginInvoke(gcnew
InvokeDelegateSetImage(this,
&Form1::setImageToPictureBox), image);
                Thread::Sleep(10);
            }
    for (int y = 0; y <= pictureBox1->Height - 2 * R;
y += 2){
                g->Clear(Color::White);
                g->FillEllipse(Brushes::Aqua,
pictureBox1->Width - 2 * R, y, 2 * R, 2 * R);
                pictureBox1->BeginInvoke(gcnew
InvokeDelegateSetImage(this,
&Form1::setImageToPictureBox), image);
                Thread::Sleep(10);
            }
    for (int x = pictureBox1->Width - 2 * R; x >= 0;
x -= 2){
                g->Clear(Color::White);
                g->FillEllipse(Brushes::Aqua, x,
pictureBox1->Height - 2 * R, 2 * R, 2 * R);
                pictureBox1->BeginInvoke(gcnew
InvokeDelegateSetImage(this,
&Form1::setImageToPictureBox), image);
                Thread::Sleep(10);
            }
    for (int y = pictureBox1->Height - 2 * R; y >= 0;
y -= 2)         {
                g->Clear(Color::White);
                g->FillEllipse(Brushes::Aqua, 0, y, 2
* R, 2 * R);
                pictureBox1->BeginInvoke(gcnew
InvokeDelegateSetImage(this,
&Form1::setImageToPictureBox), image);
                Thread::Sleep(10);
            }
// button2->BeginInvoke(gcnew
```

```
InvokeDelegateSetEnabled(this,
&Form1::setEnabledToButton), true);
        }
```

Bu funksiyani ishlatish uchun qiymatni to‘g‘ridan to‘g‘ri berib bo‘lmaydi shuning uchun bir deligant (funksiya ko‘rsatkich saqlovchi) va qiymatni funksiya yaratamiz.

```
 delegate void InvokeDelegateSetImage(Bitmap^ image);
 public: void setImageToPictureBox(Bitmap^ image){
                                  pictureBox1->Image =
image;
                              }
```

Formaning ochiq joyiga oqim yaratib olamiz.

```
    using namespace System::Threading;
// ...
       private: Thread^ myThread;
    private: Thread^ myThreadOne;
```

Birichi tugma bosilganda mazkur oqimlarni ishga tushurish uchun `button1_Click` hodisasiga quyidagicha algoritm yoziladi.

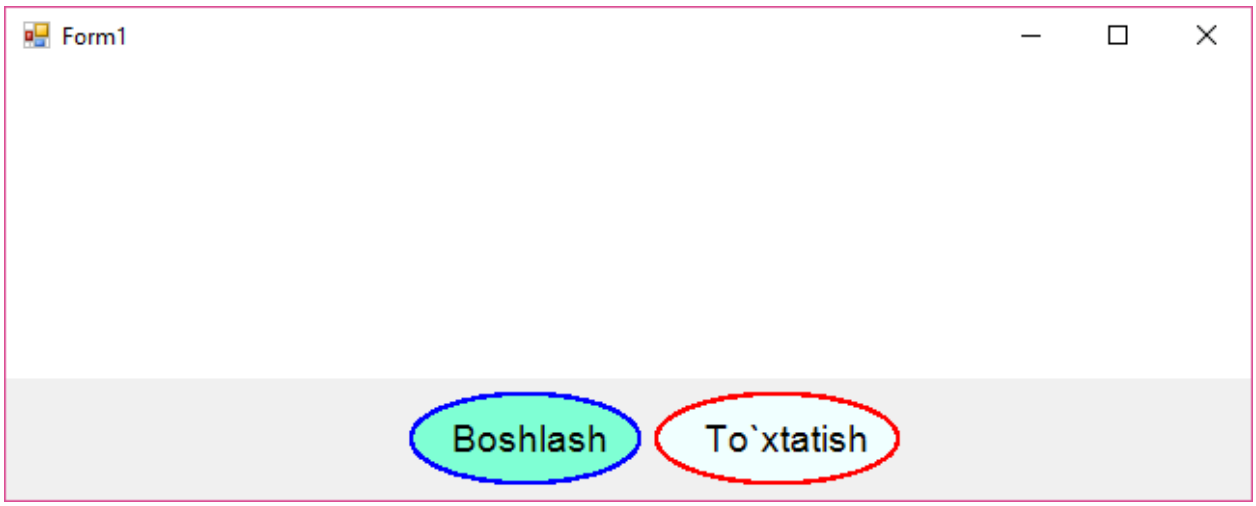
```
  myThread = gcnew Thread(gcnew
ThreadStart(this,&Form1::Go));
  myThread->Start();
  myThreadOne = gcnew Thread(gcnew
ThreadStart(this,&Form1::GoOne));
  myThreadOne->Start();
```

Oqimni saqlab tuurish uchun yaratilgan funksiyada Thread::Sleep(10); dan foydalanilgan.

Oqim ishga tushgan paytdan ishlaydi, ammo forma yopilganda uning ishini to‘xtatish lozim.Shuning uchun oqimni to‘xtashish uchun button2_Click va Form1_FormClosing hodisalariga quyidagi algoritmni kiritamiz.

```
try{
  myThread->Abort();
  myThreadOne->Abort();
     }
     catch(...){}
```

Loyihani ishga tushirsangiz, quyidagi 12.18- rasm hosil bo‘ladi.



12.18- rasm. Oqimlar asosida xarakatni boshqarish

Dasturlash boshlash tugmasini bosganda ikkita paralel bo‘lgan shakllarining harakatini ko‘rasiz. To‘xtash bosilganda ularni to‘xtashini ko‘rasiz. Xarakat davom etatgan vaqtda ham formani yopiq ishni tugallash mumkin.

Umuman olgada grafika bilan ishlash uchun geometriyani yaxshi bilish talab qilinadi. Visual C++ da grafika sinfi, uning usullari va turli shakllarni chizish usullari, Chart va Shape sinflarining xususiyatlari va usullarini кщrib chiqdik. Bularni щziga xos ma’lumotlarda ishlatishni tavsiya qilamiz.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Chiziqlar va shakllar chizish uchun maxsus qanday usullarni o‘rganish lozim
2. DrawLine, DrawArc, DrawClosedCurve, DrawPolygon va DrawRectangle o‘z ichiga oladigan sinfni ayting.
3. Grafika sinfi xususiyatlari sanab bering.
4. grafika sahifa koordinatalarini uchun ishlatiladigan o‘lchov birligi sozlash xususiyatini ayting.
5. GraphicsUnit qiymatlari sanab bering.
6. - butun chizma yuzasini tozalaydi va belgilangan fon rangi bilan to‘ldiradigan usulning nomini ayting.
7. Bu funksiya berilgan parametrlar asosida tasvirni belgilangan joyda asl kattaligidan foydalanib chizadi. Funksiya nomini ayting.
8. Chiziq chizishni qanday amalga oshiriladi.
9. Uchburchak chizishni qanday amalga oshiriladi
10. Ellipis chizishni qanday amalga oshiriladi

11. Turli shakllarni bo‘yashni qanday amalga oshiriladi
12. Ko‘p burchaklarni chizishni qanday amalga oshiriladi
13. Hodisalar orqali shakl chizishni qanday amalga oshiriladi
14. Rasmlarni o‘zgartirishni qanday amalga oshiriladi
15. Matnlarni tasvir kabi joylashtirishni qanday amalga oshiriladi
16. Matnli rasmlarni yaratishni qanday amalga oshiriladi
17. Chart komponenta xususiyati va hodisalari haqida gapirib bering.
18. BorderSkin nima.
19. Qaysi xususiyat yordamida komponentaning infografikani ko‘rsatadigan shaklini tanlash mumkin
20. Qaysi xususiyat asosiy bo‘lib, infografikaning qiymatlarini belgilovchi, har bir qiymat tegishliligini bildiradi
21. ChartType infografikaning turlarini belgilash uchun xizmat qiladi. Uning qiymatlarini bilasizmi.
22. Bir infografikada nechtagacha gistogramma joylashtirish mumkin.
23. Chart komponentasi yordamida foydlanuvchining ixtiyoriy ma’lumotlarini infografikasini yaratish mumkinmi
24. Nimalar dastlab bepul plagin-smaylik sifatida ishlatilgan va endi Visual Studio tarkibiga kiritilgan
25. Visual Basic Power Packs komponentalarini sanab bering.
26. Visual Basic Power Packs komponentalarining asosiy vazifalari nimadan iborat.
27. Komponentaning chegara rang qalinligini o‘rnatadi va qiymatlari sonlardan iborat bo‘lgan xususiyat nomini ayting.
28. Shakllarning FillColor, FillGradientColor, FillGradientStyle, FillStyle xususiyatlaridan foydalanib, nima qilish mumkin.
29. Tugmaning ko‘rinishini o‘zgartirish uchun nima amallar bajarish kerak.
30. Rasmlarni harakatlantirishda oqimlardan qanday foydalanamiz.

AMALIY KO‘NIKMA VA MALAKALARNI ANIQLASh HAMDA RIVOJLANTIRISh UChUN ASSISMENT TOPShIRIG‘I.

ASSISMENT TOPShIRIG‘I	
	Grafika bilan ishlashga oid berilgan quyidagi dastur bo‘yicha berilgan topshiriqlar kerakli fragmentlari asosida bajaring.

	☝ Bu vazifalarni kichik loyiha sifatida bajariladi, kichik loyiha deganda kamida 100 ta vazifasi bo‘lgan dastur tushuniladi.

1. Oddiy rasm muharriri dasturini tuzing. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____
_____

2. Interaktiv muloqot oynalaridan foydlanib, diagrammalarni yaratuvchi dastur yarating. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____

3. Intellektual turli ma‘lamotlarni kiritishni talab qiluvchi funksiyalar grafigini yaratish dastur tuzing. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____

4. Tugmalarni bezash orqali kichik kalkulyator dasturini tuzing. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____

_____
_____
_____
_____
_____

5. Barcha turdagi muloqot oynalarini o‘z ichiga olgan, interaktiv savol javob tizimini yarating va javoblarga yasama harakatlanish vositalaridan foydalanish dastur tuzing. Loyiha bo‘yicha qisqacha asosiy fragmentlarni yozing.

_____
_____
_____
_____
_____
_____
_____
_____
_____

6. Barcha vazifalarni o‘rganganingiz bo‘yicha eslab ko‘ring, nimalarni o‘rgandingiz va bir takrorlang.
7. Barcha bilim va ko‘nikmalar asosida kichik loyiha tayyorlang.

3.5. OLE, MFC texnologiyalari. Ko‘p oynali muhitlar bilan ishlash.

📖 Visual C++ning operatsion tizim, amaliy dasturlari bilan o‘zaro bog‘lanish uchun MFC va OLE texnologiyalarining imkoniyati va variantlari, ularning o‘zaro hamkorligi, ko‘p oynali dasturlarni yaratish bo‘yicha nazariy bilimlar keltirilgan.. Bilimlarni mustahkamlash uchun 15 ta nazariy savol berilgan.

✎ *Kalit so‘zlar.* Operatsion tizim, amaliy dastur, integratsiya, o‘zaro bog‘lanish, MFC, OLE, SDI, MDI, ko‘p oynali dastur..

☑ ***Bilish shart bo‘lgan tushunchalar.*** Operatsin tizim va amaliy dasturlarda ishlash, sinf va sinf obyekti, xususiyat, hodisa, forma, komponenta dasturlashga oid dastlabki tushunchalar hamda C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Visual C++ muhitida dasturlash, Visual C++ning MFC texnologiyasi imkoniyatlari va variantlari, OLE texnologiyasi imkoniyatlari va variantlari, ularning o‘zaro bog‘lash uchun loyiha yaratish, ko‘p oynali dasturlarni yaratish uchun SDI, MDI

konstruktorlari usulari va ba‘zi bir xususiyatlari, hodisalari o‘rganishingiz mumkin.

Reja

1.MFC texnologiyasi.

2.OLE texnologiyasi.

3.Visual C++ muhitida ko‘p oynali muhitni yaratish.

MFC texnologiyaci. Microsoft Foundation Class paketi (MFC) Microsoft tomonidan ishlab chiqilgan bo‘lib, C++ kutubxonalari yordamida Microsoft Windows uchun GUI dasturlar ishlab chiqishni engillashtirish uchun mo‘ljallangan. Unda kutubxonalarning va sinflarining soni juda ko‘p.

MFC texnologiyasi

MFC kutubxonasi, uning asosiy raqobatchisi bo‘lgan Borland VCL kabi, GUI bilan ishlashni osonlashtiradi dastur asoslarini yaratish, ya‘ni ma’lum bir tartib asosida avtomatik ravishda yaratiladi.

Foydalanuvchi interfeysi va butunlay uning amaliy va texnik xizmat ko‘rsatish uchun muntazam harakat va hodisalar bajariladi(masalan, oyna amallari, elementlar va dastur o‘zgaruvchilar ichki obyektlar o‘rtasida ma’lumotlarni yuborish, va hokazo.). Dastur asoslarini ishlab chiqqandan so‘ng, dasturchi faqat kodni maxsus harakat va hodisalar talab qilinadigan joylarga kiritishi kerak. Bunday Framework yaxshi belgilangan tuzilishga ega bo‘lishi kerak, shuning uchun visual C++ uni yaratish va o‘zgartirish uchun yordamchi komponentalarni beradi.

Bundan tashqari, MFC Windows API vazifalari turli obyektga yo‘naltirilgan qatlamlarini beradi, ular bilan ishlash uchun bir oz osondir. Bu qatlam ko‘p ajralmas obyektlarni ifodalaydi (Windows, vidjet, fayllar, instrumentlar, sinflar, komponentalar va boshqalar.). Sinflar sifatida joriy hodisalarni yopish va xotira ajratish/bo‘shatish kabi muntazam amallarni bajaradi.

Dastur doirasiga kod qo‘shish. MFC Framework uchun dastur kodi kiritish uchun ikki yo‘l bor. Birinchisi kutubxonadan meros mexanizmidan foydalanadi: asosiy dastur doirasida tuzilmalar meros qilib olingan sinflar sifatida namoyon bo‘ladi. Bu sinflar dasturda ma’lum nuqtalarda chaqiriladigan ko‘pgina virtual funksiyalarni ta‘minlaydi. Bu funksiyalarni yanada belgilab (ko‘p hollarda bazaviy sinf vazifasini chaqirish kerak), dasturchi o‘z kodida shu nuqtalarda bajarishni qo‘shishi mumkin.

Ikkinchi usul oyna hodisalarini qo‘shish uchun ishlatiladi. Yordamchi instrument Windows — message maps (inglizcha: message map) bilan bog‘liq sinf qoliblari ichida [message ID — pointer to handler] juftlarini o‘z ichiga olgan maxsus massivlarni yaratadi. Qo‘shish yoki o‘rnatish yordamchi instrument asosida tegishli xabar orqali o‘zgarishlar qiladi.

MFC ning yaratilish tarixi. MFC ning birinchi versiyasi 1992-yilda Microsoft kompaniyasining 16-bitli C/C++ kompilyatorining ettinchi versiyasi bilan birga chiqarilgan. API funksiyalari yordamida ilovalarni ishlab chiqayotganlar uchun MFC paketi juda katta o‘sishni vujudga keltirdi.

1. Dasturlash jarayonini boshqarish. MFC ning e‘tiborli xususiyatlaridan biri "AFX" prefiksi bo‘lib, ko‘plab funksiyalar, makroslar nomlarida va "STDAFX" standart sarlavhali faylining nomi bilan ishlatiladi. Rivojlanishning dastlabki bosqichida, keyinchalik MFC nomi "dastur" deb nomlangan. AFX doirasida rengaytmalari Microsoft Foundation sinflar nomini o‘zgartirish uchun qaror (MFC) kodi AFX murojaatlarni o‘zgartirish uchun juda kech qilingan.

2. C++ kompilyatori bilan ishlash uchun Borland tomonidan ishlab chiqilgan Object Windows Library (OWL) shu davrda joriy etilgan raqobatchi mahsulot edi. Oxir oqibat Borland rivojlanishdan to‘xtadi va MFC bilan ishlash uchun kutubxona, foydalanish uchun qisqa muddatli litsenziya sotib oldi. Lekin uning mahsulotlari MFC uchun to‘liq qo‘llab-quvvatlash amalga oshirolmadi.

3. MFC Microsoft markazida Microsoft.Net Framework tanlangan. Biroq, shunga qaramay, MFC hali ishlab chiquvchilar orasida mashhur.

Birinchilardan bo‘lib, Rossiyaning bcgsoft kompaniyasidan sotib olingan. MFC yangi dizaynli interfeysi qo‘llab-quvvatlash va takomillashtirilgan interfeys bilan bog‘liq bir necha boshqa o‘zgartirishlarni kiritgan va nazorat qilmoqda. Visual Studio 2008 tizimidan ba‘zi bir komponentalarni olib, bu sinflar asosida MFC ajralmas qismi qilib yaratildi.

13.1-jadval. MFC mahsulotlari va varintlari.

Mahsulot nomi	**MFC varianti**	**Joriy qilingan sana**
Microsoft C/C++ 7.0	MFC 1.0	1992

Visual C++ 1.0	MFC 2.0 (arxitekturasi yangilangan)	
Visual C++ 1.5	MFC 2.5 (ODBC i drag and-drop ni qo‘llab quvvtlovchi texnologiya joriy qilingan)	
Visual C++ 1.52c	MFC 2.5 (MS Windows 3.x uchun yaratilgan oxirgi versiya)	
Visual C++ 2.0	MFC 3.0 (ko‘p masalali va Unicode ni qo‘llab quvvatlash joriy qilindi)	
Visual C++ 2.1	MFC 3.1	
Visual C++ 2.2	MFC 3.2	
Visual C++ 4.0	MFC 4.0 (mfc40.dll - Windows 95 tarkibiga kiritilgan)	1995 y. avgust
Visual C++ 4.1	MFC 4.1	
Visual C++ 4.2	MFC 4.2 (mfc42.dll - Windows 98 tarkibiga kiritilgan)	1998 y. mart
eMbedded Visual C++ 3.0	MFC 4.2 (mfc42.dll)	
Visual C++ 5.0	MFC 4.21 (mfc42.dll), MFC 4.2. ning yangilanganligi	
Visual C++ 6.0	MFC 6.0 (mfc42.dll)	1998
eMbedded Visual C++ 4.0	MFC 6.0 (mfcce400.dll)	
Visual C++ .NET 2002 (Visual C++ 7.0)	MFC 7.0 (mfc70.dll), .NET 1.0	2002 y. fevral
Visual C++ .NET 2003 (Visual C++ 7.1)	MFC 7.1 (mfc71.dll), .NET 1.1	2003 y. aprel
Visual C++ 2005[2] (Visual C++ 8.0)	MFC 8.0 (mfc80.dll), .NET 2.0	2005 y. mart
Visual C++ 2008[2] (Visual C++ 9.0)	MFC 9.0.21022 (mfc90.dll), .NET 3.5	2007 noyabr
Visual C++ 2008[2] with	MFC 9.0.30411 (mfc90.dll)	2008 y. aprel

Feature Pack		
Visual C++ 2008[2] SP1	MFC 9.0.30729 (mfc90.dll)	2008 y. avgust
Visual C++ 2008[2] Security Update (KB971092)	MFC 9.0.30729.4148 (mfc90.dll)	2009 y. iyul
Visual C++ 2010	MFC 10.0.30319.1 (mfc100.dll), .NET 4.0	2010 y. aprel
Visual C++ 2010 SP1	MFC 10.0.40219.1 (mfc100.dll), .NET 4.0	2011 y. mart
Visual C++ 2010 + MS11-025	MFC 10.0.30319.415 (mfc100.dll), .NET 4.0	2011 y. aprel
Visual C++ 2012 (Visual C++ 11.0)	MFC 11.0.50727.1 (mfc110.dll), .NET 4.5	2012 y. iyul
Visual C++ 2012 Update 1 (Visual C++ 11.0)	MFC 11.0.51106.1 (mfc110.dll), .NET 4.5	2012y. noyabr
Visual C++ 2012 Update 3 (Visual C++ 11.0)	MFC 11.0.60610.1 (mfc110.dll), .NET 4.5	2012y. dekabr
Visual C++ 2013 (Visual C++ 12.0)	MFC 12.0.21005.1 (mfc120.dll), .NET 4.5.1	2013
Visual C++ 2013 Update 2 (Visual C++ 12.0)	MFC 12.0.30501.0 (mfc120.dll), .NET 4.5.1	2014
Visual C++ 2015 (Visual C++ 14.0)	MFC 14.0.23026.0 (mfc140.dll), .NET 4.6	2015
Visual C++ 2015 Update 1 (Visual C++ 14.0)	MFC 14.0.23506.0 (mfc140.dll), .NET 4.6.1	2015 y. noyabr
Visual C++ 2015 Update 2 (Visual C++ 14.0)	MFC 14.0.23918.0 (mfc140.dll)	2016 y. mart
Visual C++ 2015	MFC 14.0.24210.0 (mfc140.dll)	2016y.

Update 3 (Visual C++ 14.0)		iyun
Visual C++ 2015 Update 3 + KB3165756	MFC 14.0.24212.0 (mfc140.dll)	2016 y. avgust
Visual C++ 2017 (Visual C++ 15.0)	MFC 14.10.25008.0 (mfc140.dll), .NET 4.6.2	2017 y. mart
Hozirgi vaqtda ham variantlari yangilanib borilmoqda.		

OLE texnologiyasi. OLE (Object Linking and Embedding) - Microsoft tomonidan obyektlarni boshqa bir hujjat va obektlarga integratsiya qilish uchun ishlab chiqilgan texnologiya. 1996 yilda Microsoft kompaniyasi ActiveX texnologiyasini qayta nomlagan.

OLE - bir qismini bir dasturdan boshqasiga o‘tkazish va natijalarni qaytarib olish imkonini beradi. Masalan, shaxsiy kompyuterga o‘rnatilgan amaliy tizimi ba‘zi matnni qayta ishlash uchun matn muharririga yoki biror tasvirni OLE texnologiyasi yordamida tasvir muharririga yuborishi mumkin.

OLEdan foydalanishning asosiy afzalligi (fayl hajmini kamaytirishdan boshqa) shundaki, u asosiy faylni, dastur kiritadigan funksiyalarning fayl kutubxonasini yaratish imkonini beradi. Bu fayl qayta ishlab, so‘ng manba hujjatga qaytariladi manba dasturi, ma’lumotlarni qo‘shishi mumkin.

OLE murakkab hujjatlarni qayta ishlash uchun ishlatiladi, drag-and-Drop interfeysi orqali turli bog‘liq bo‘lmagan tizimlar o‘rtasida ma’lumotlarni uzatish uchun foydalanish mumkin va clipboard operatsiyalarini amalga oshirish uchun. HTML-sahifalarda (gipermatnli markup tilida) yoki matn tahriridandan foydalanadigan boshqa fayllarda (masalan, XML va SGML) tasvirlar, tovush, video va animatsiya uzatiladigan web-sahifalarda (masalan, Web TV) multimediya kontenti bilan ishlashda amalga oshirish g‘oyasi keng qo‘llaniladi. Biroq OLE texnologiyasi "katta mijoz" arxitekturasidan, ya‘ni ortiqcha hisoblash resurslariga ega bo‘lgan tarmoq kompyuteridan foydalanadi. Bu amalga oshirishda fayl turi yoki dastur mijoz mashina mavjud bo‘lishi kerak. Masalan, OLE Microsoft Excel jadvallaridan foydalansa, Excel foydalanuvchi mashinasiga o‘rnatilishi kerak.

OLE 1.0 Microsoft Windows operatsion tizimining oldingi versiyalarida ishlatiladigan DDE (dinamik ma’lumotlar almashinuvi) texnologiyasi asosida 1990-yilda chiqarilgan. DDE texnologiyasi ikkita

ishlaydigan dastur o‘rtasida ma’lumotlarni uzatish soni va usullari bilan cheklangan bo‘lsa-da, OLE ikkita hujjat orasidagi faol ulanishlar ustida ishlay oldi va hatto bir turdagi hujjatni boshqa turdagi hujjatga joylashtirdi.

OLE serverlar va mijozlar virtual funksiya jadvallar yordamida tizimi kutubxonalar bilan o‘zaro bog‘lanadi. Bu jadvallarda tizim kutubxonasi server yoki mijoz bilan o‘zaro aloqa qilish uchun foydalanishi mumkin bo‘lgan funksiyalarga mos ko’rsatkichlar mavjud. Kutubxona OLESRV.DLL (server) va OLECLI.DLL (mijoz haqida) dastlab operatsion tizimi tomonidan taqdim etiladigan WM_DDE_YeXECUTE xabar yordamida bir-biri bilan o‘zaro moslashuvi kerak.

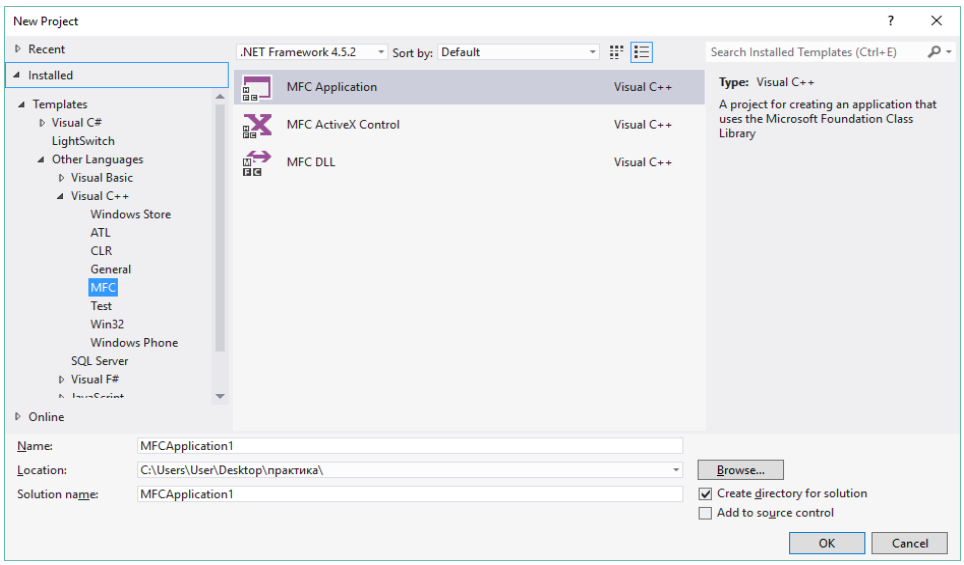
OLE 1.1 keyinchalik dasturiy komponentlar bilan ishlash uchun COM (butlovchi obyekt modeli) arxitekturasiga aylandi. Keyinchalik, DCOM sifatida ma’lum bo‘ldi.

Buferga OLE obyekti joylashtirilganda Windows formatlarida (bitmap yoki metafile kabi) saqlanadi hamda o‘z formatida saqlanadi. U formatni qo‘llab-quvvatlash OLE dasturi clipboardga ko‘chirilgan boshqa hujjatning bir qismini joylash va foydalanuvchi hujjatda uni saqlash imkonini beradi.

Keyingi evolyusion qadam OLE 2.0 edi, bu avvalgi versiya bilan bir xil maqsad va vazifalarni saqlab qolgan holda OLE 2.0 VTBL yordamida COM arxitekturasiga qo‘shilgan bo‘lgan. Yangi xususiyatlari Drag-and-Drop avtomatlashtirish, IN-joy faollashtirish va tuzilganlarni saqlash texnologiyalarini o‘z ichiga oladi.

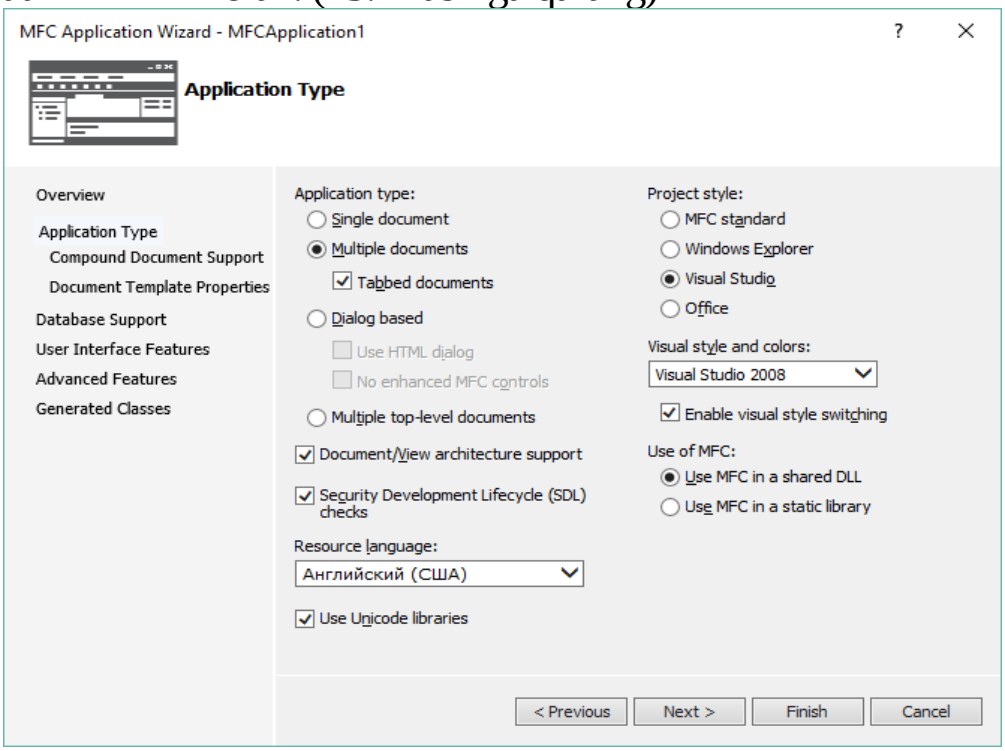
Visual C++dasturlarida OLE konteyneridan foydalanish.

Visual C++ OLE moslamalarni foydalanish misolini ta‘riflaydi deb atalmish OLE konteynerb yordamida ilovalar MFC platformasida amalga oshiriladi. OLE obyektlar eng tez-tez murakkab hujjat qismlari sifatida ishlatiladi. Hujjatlarga Microsoft Office ilovalari (Excel, Word, PowerPoint va boshqalar) yordamida yaratilgan fayllar kiradi. Biroq, Microsoft Office hujjatlarining o‘zi boshqa ilovalarda OLE obyektlari sifatida ishlatilishi mumkin. Misol sifatida, Visual Studio yangi MFC dastur loyihasini yaratib ko‘ring (13.1-rasmga qarang).



13.1-rasm. MFC application yaratish.

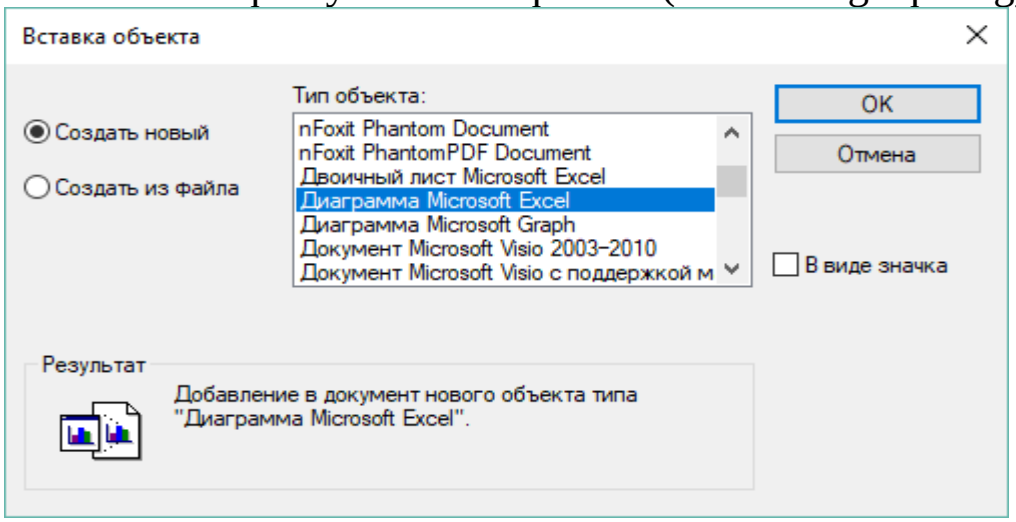
Keyingi qadam muloqot oynasida, kelajakda qo‘llash asoslarini tanlash. Joriy dastur "hujjat-ko‘rinish" arxitekturasidan foydalanib yaratilishi kerak. Ilova interfeysi bitta hujjat yoki ko‘p hujjat bo‘lishi mumkin. Shundan so‘ng "keyingi" tugmasini bosib va kompozit hujjatlarni konteyner sifatida qo‘llab-quvvatlash variantini tanlash lozim. Bundan tashqari, faol hujjat konteyner xususiyati o‘rnatish kerak. (13.2-rasmga qarang)



13.2-rasm. Paramertlarni sozlash.

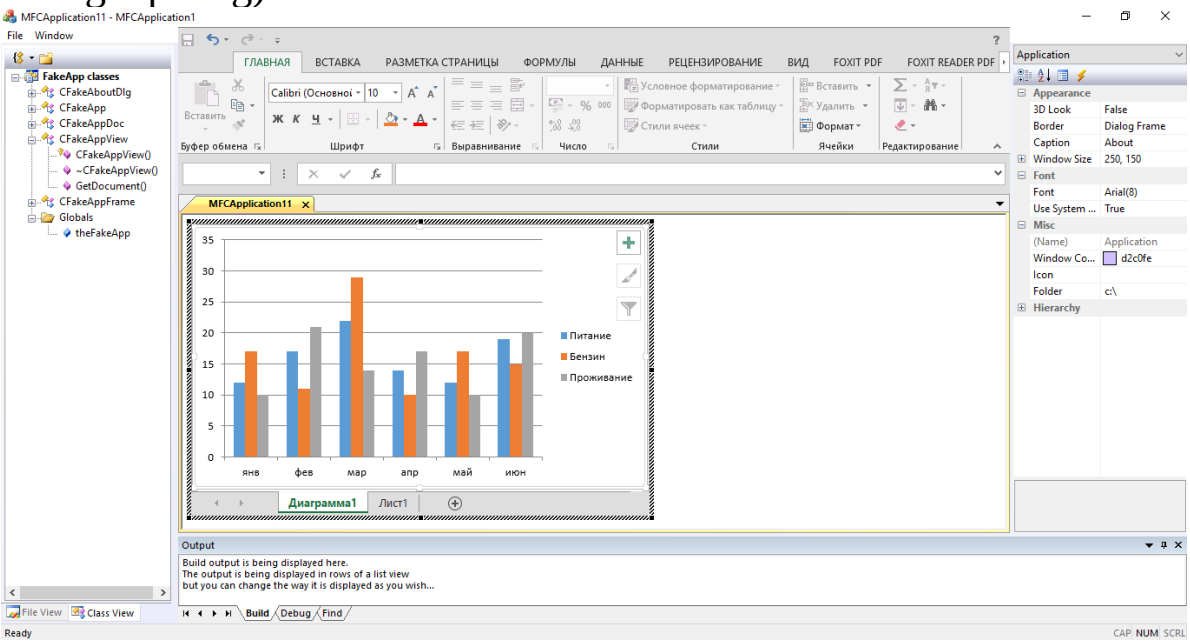
Shundan so‘ng "Finish" tugmasini bosish kerak yoki keyingi sahifalardan o‘tib, zarur hollarda loyiha parametrlaringizni ko‘rsatishingiz mumkin. Natijada siz kompilyatsiya uchun ishlatish mumkin, yangi loyiha hisoblanadi. Dastur oynasida Bosh menyu bandini

tanlash orqali " Edit Insert new Object...". orqali OLE obyektlarining ro‘yxati bilan muloqot oynasi hosil qilinadi (13.3-rasmga qarang).



13.3-rasm. OLE asosida bog‘lanish.

Misol sifatida Microsoft Excel grafik obyektini tanlasangiz, dastur oynasi obyekt kiritilgandan so‘ng shunday ko‘rinishga ega bo‘ladi (13.4-rasmga qarang).



13.4-rasm. Microsoft Excel bilan bog‘lanish.

Bu OLE obyekt faol, uni o‘zgartirish mumkin: grafik uslubi o‘zgartirish, ma’lumotlarga o‘zgarishlar qilish, va konteyner ichida nuqta formatini o‘rnatish ham mumkin.

OLE bir konteyner serialization xususiyatiga ega, bir fayl uchun ma’lumotlarni saqlash va keyin fayl uni qaytarib o‘qish mumkin. Ushbu obyektni faylga saqlash uchun "File-Save as" ni tanlab, dasturning asosiy menyusida va muloqot oynasidagi fayl nomini ko‘rsatiladi.

OLE konteyneridan foydalanib, mavjud Microsoft Office hujjat fayllarini, masalan, Excel elektron jadvallarini yoki PowerPoint

taqdimotlarini, shuningdek Word hujjatlarini ham ko‘rishingiz va tahrirlashingiz mumkin. Buning uchun yaratilgan dasturni ishga tushiring va "Edit / Insert new Object" ni tanlang. Bu orqali barcha imkoniyatlarni amalga oshirish mumkin.

Visual C++ muhitida ko‘p oynali muhitni yaratish. Visual C++ muhitida ko‘p oynali dasturlarni yaratish uchun SDI va MDI ilovalar konstruktorlari mavjud. Hozirda MDI ilova konstruktori orqali kichik ko‘p oynali dastur yaratamiz. Bunda quyidagi qadamlardan foydalinadi.

1. Yangi forma yaratiladi.
2. Formaning xususiyatlari o‘zgartiriladi.

IsMdiContainer: True
StartPosition: CenterScreen
Text: **Notice**

3. Yangi forma qo‘shish uchun, asosiy menyusida, click Project -> Add Windows Form...
4. Yagona hujjatga nom o‘rnatish
5. Add tugmasini bosing
6. Asboblarning umumiy elementlari bo‘limidan RichTextBox ni tanlang
7. Xususiyatlar oynasida quyidagi xususiyatlarni o‘zgartiring:

(Name): **rtbNotice**
Dock: Fill
Modifiers: Public

8. Birinchi shaklni ko‘rsating
9. Formada o‘ting va yangi ikki marta sichqonchani bosing
10. Faylning yuqori qismida yagona hujjat header faylini tahrirlang:

```
#include "SingleDocument.h"
```

11. Kodning pastga tushing va quyidagicha hodisani amalga oshirish kodini yozing.

```
System::Void
mnuFilenew_Click(System::Object^  sender,
System::EventArgs^  e)
{
    SingleDocument ^ document = gcnew
SingleDocument;
    document->Text = "Untitled";
```

```
    document->MdiParent = this;
    document->Show();

    mnuFileClose->Enabled = true;
}
```

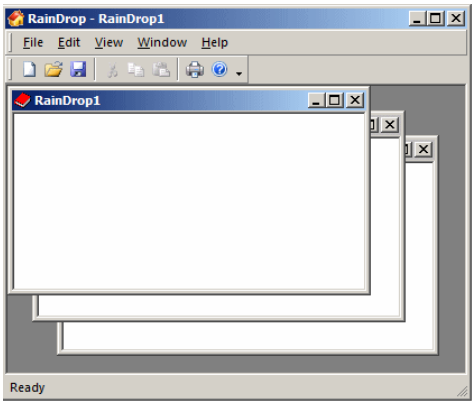
12. Formaga qaytiladi.
13. Asboblar panelidan yangi bir tugmani o‘rnating.
14. Xususiyatlar oynasining hodisalariga o‘tib, tugmani hodisalarini bosing. click maydonini bosing va o‘ng tomonida, mnufilenew_click tanlang
15. Formada o‘ting va yangi ikki marta sichqonchani bosing
16. Quyidagicha tadbir amalga oshiring:

```
System::Void
mnuFileOpen_Click(System::Object^
sender, System::EventArgs^  e)
{
    if( dlgFileOpen->ShowDialog() ==
System::Windows::Forms::DialogResult::OK
)
    {
        for each( String ^ strFile in
dlgFileOpen->FileNames )
        {
            SingleDocument ^ document =
gcnew SingleDocument;

            document->rtbNotice-
>LoadFile(strFile);

            document->MdiParent = this;
            document->Show();
        }
    }
}
```

17. Formaga qaytiladi.
18. Asboblar panelidan yangi bir tugmani o‘rnating.
19. Xususiyatlar oynasining hodisalariga o‘tib, tugmani hodisalarini bosing. click maydonini bosing va o‘ng tomonida, mnufilenew_click tanlang

20. mnufileopen_click tanlang
21. Formaga qayting.

Natijada yaratilgan loyihani ishga tushirsangiz quyidagicha, dastur hosil qilinadi.



13.5-rasm. Ko‘p oynali dastur.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. MFC nima, u qanday imkoniyat.
2. MFC va API ning bog‘liq xususiyatlarini bilasizmi?
3. MFC ning eng asosiy xususityai nima.
4. Visual Studio 2008 tizimidan qanday komponentalarni olib, qaysi sinflar asosida MFC ajralmas qismi qilib yaratildi.
5. MFC 9.0.21022 (mfc90.dll), .NET 3.5 ning imkoniyatlarini sanab bering.
6. MFC 14.10.25008.0 (mfc140.dll), .NET 4.6.2 ning imkoniyatlarini sanab bering.
7. OLE texnologiyasining imkoniyatini sanab bering.
8. OLE murakkab hujjatlarni qayta ishlash uchun ishlatiladi, qaysi interfeysi orqali turli bog‘liq bo‘lmagan tizimlar o‘rtasida ma’lumotlarni uzatish uchun foydalanish mumkin.
9. Qaysi texnologiya va uning metodi Microsoft Windows operatsion tizimining oldingi versiyalarida ishlatiladigan DDE (dinamik ma’lumotlar almashinuvi) texnologiyasi asosida 1990-yilda chiqarilgan
10. OLE 1.0 va OLE 2.2 texnologiyalarini farqini tushuntirib bering.
11. MFC va OLE texnologiyalari bog‘liqmi va nima uchun.
12. Ko‘p oynali muhitlarning avfzalligi nimada?
13. Forma xususiyatlarini o‘zgartirish nima uchun kerak.
14. Dock xususiyatini nima uchun Fill qiymatga o‘zgartiriladi.
15. mnufileopen_click hodisasining vazifasini tushuntirib bering.

3.6.Kichik loyihalarni yaratish

📖 *Visual C++ning loyiha yaratish usullari va uskunalar panelidagi elementlardan foydalanish asosida kichik loyihalarni yaratish talablari va usullari, loyihada foydalaniladigan algoritmlarni ifodalash va loyihalash asosida tahlil qilishni, hamda hisobotlarni yaratish usullari va elementlari bo‘yicha nazariy bilimlar keltirilgan. Bilimlarni mustahkamlash uchun 15 ta nazariy savol berilgan.*

✎ ***Kalit so‘zlar****. Operatsion tizim, amaliy dastur, kichik loyiha, axborot tizimi, IDEF, DFD, Hayot sikli, algoritm, EPS metodologiyasi, BRMI metodologiyaisi, hisobot, hisobot paneli.*

☑ ***Bilish shart bo‘lgan tushunchalar.*** *Visual C++ning loyiha yaratish usullari va uskunalar panelidagi elementlardan foydalanish, ishlash, sinf va sinf obyekti, xususiyat, hodisa, forma, komponenta dasturlashga oid dastlabki tushunchalar hamda C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.*

👍 ***Bilib olasiz****. Kichik loyihalarni loyihalashtirish usullari, hayot davri, loyihadagi algoritmlarning ifodalashning uchullari va talablari, Visual C++ muhitida dasturlash, Visual C++ning hisobotlarni yaratish konstruktorlari usulari va ba‘zi bir xususiyatlari, hodisalari o‘rganishingiz mumkin.*

REJA

1.Visual C++ muhitida turli sohaga oid masalalarni yechish uchun kichik loyihalarni loyihalash usullari.

2.Kichik loyihalar algorimlarini yozish usullari.

3.Visual C++ muhitida hisobot shaklini tayyorlash va chop etish elementlari.

KIRISH

Visual C++ muhitida turli sohaga oid masalalarni yechish uchun kichik loyihalarni loyihalash usullari asosan axborot tizimlarni loyihalashtirish usullari nazarda tutiladi. Shuning uchun axborot tizimining obyektlari va usullari, algoritmlariga, qo‘llanish sohasi va chegaralariga qarab loyihani katta, kichik loyihalar deyish mumkin. Shuning uchun kichik loyiha deganda funksional imkoniyatlari sanoqli bo‘lgan axborot tizimlari tushuniladi.

Axborot tizimlarining loyihalashtirish uchun axborot tizimining hayot davrini yaxshi bilish va tushunish kerak. Chunki axborot tizimini loyhalashda uchun hayot davri muhim ahamiyatga ega.

Axborot tizimining hayot davri. Hayot davri (HD) tushunchasi axborot tizimlarini loyihalash metodologiyasining asosiy tushunchalaridan biridir. Axborot tizimining hayotiy dari - bu axborot tizimini yaratish fikridan boshlanib, u butunlay foydalnishdan chiqqan paytda tugaydigan uzluksiz jarayondir. Hayot davrining tuzilishini belgilovchi asosiy standart GOST ISO/IEC 12207-02 hisoblanadi. Standartga koʻra, hayot davri strukturasi uchta jarayon guruhiga asoslangan:

1. **Asosiy** (buyurtma, ishlab chiqish, etkazib berish, sinov, texnik xizmat koʻrsatish);

2. **Yordamchi** (asosiy jarayonlarning bajarilishini taʻminlash uchun):

- Hujjatlar uchun - barcha manfaatdor shaxslar tomonidan zarur boʻlgan hujjatlarni ishlab chiqish, tahrirlash, tarqatish va saqlash boʻyicha ishlar;
- konfiguratsiyani boshqarish (configuration management) quyidagilarni oʻz ichiga oladi: tizimdagi dasturiy obyektlarning holatini aniqlash va oʻrnatish; obyektlarning oʻzgarishi va chiqarilishini boshqarish; obyektlarning toʻliqligi, mosligi va toʻgʻriligini taʻminlash; obyektlarni saqlash, tashish va etkazib berishni boshqarish;
- sifatni taʻminlash – yaratilayotgan tizim va amalga oshirilayotgan hayot davri jarayonlarining belgilangan talablar va tasdiqlangan rejalarga javob berishini taʻminlash ishlari;
- tekshirish - loyiha amalga oshirilayotganda yaratilgan muvaqqat natijalarning belgilangan talablarga javob berishini tekshirish uchun tegishli shaxs (buyurtmachi, etkazib beruvchi yoki mustaqil tomon) ning ishi. Shartnoma, jarayon, talablar, loyiha tizimi, qurilish tizimi va hujjatlarni tekshirish;
- sertifikatlash - talablarning va yakuniy mahsulotning tizimning funksional maqsadiga toʻliq mosligini tekshirish uchun tegishli shaxsning ishi;
- qoʻshma tahlil – har qanday ish (tizim) holatini yoki natijalarini baholash uchun ish);
- audit - mustaqil (loyihaga nisbatan) ekspertlarning korxona faoliyatining qabul qilingan talablar, rejalar va shartnoma shartlariga muvofiqligini aniqlash ishlari;
- muammoni hal qilish - loyihani amalga oshirish vaqtida topilgan muammolarni tahlil qilish va tuzatish ishlari;

3. **Tashkiliy**:

- loyiha boshqaruvi - rejalashtirish va jarayonlarni boshqarish, shu jumladan monitoring, tekshirish va hisobot bilan tugallangan ishlarni baholash;

-loyiha infratuzilmasini yaratish - boshqa har qanday jarayon uchun zarur bo‘lgan infratuzilmani tashkil etish va ta‘minlash uchun ishlaydi. Infrastrukturada texnik va dasturiy vositalar, vositalar, usullar, standartlar va tizimni ishlab chiqish, ishlatish yoki xizmat ko‘rsatish shartlari bo‘lishi mumkin;

- yaxshilash - hayot davri jarayonlarini baholash, kuzatish va yaxshilash uchun ishlash;

- o‘quv-kadrlar tayyorlashni rejalashtirish va o‘tkazish, shu jumladan o‘quv materiallarini ishlab chiqish. Shu bilan birga, xodimlar nafaqat tizimni boshqaradigan oxirgi foydalanuvchilarga, balki tizim ishlab chiquvchilariga ham tegishlidir. Masalan, ishlab chiquvchilar tashkilotda qabul qilingan texnologiyalar va dasturiy vositalar bo‘yicha o‘qitilishi, hatto oxirgi foydalanuvchilarni tizim bilan ishlashni to‘g‘ri amalga oshirish va o‘qitishga o‘rgatilishi kerak. Qanchalik paradoksal bo‘lmasin, lekin o‘qitishning to‘g‘ri usul va metodlarini o‘rgatish ham kerak.

AT yaratish uchun asosiy mezon quyidagilarni o‘z ichiga oladi:

- tizimning funksional va tashkiliy tuzilmalari;
- texnik va dasturiy vositalar majmuasining tarkibi va tuzilishi;
- ishlatiladigan uskunalar;
- axborotni qayta ishlash texnologiyalari;
- axborot bazasini yuritish tarkibi, tuzilishi va texnologiyasi;
- kirish va chiqish formalari;
- ma’lumotlarni qayta ishlash algoritmlari.

Axborot tizimni ishlab chiqishdan oldin mijoz va ishlab chiquvchi tizimda qanday funksionallik o‘rnatilishini va tizim ichida qanday funksional o‘zaro munosabatlar tashkil etilishini aniq tushunishi kerak.

Funksional modelni ishlab chiqishda (funksional talablarni belgilashda) ko‘pgina muammolar paydo bo‘lishi mumkin:

- mijoz axborot tizimi oldiga qanday vazifalar qo‘yilganini aniq ifoda eta olmaydi. Ko‘pincha mijoz ham talab nima va uni shakllantirish qanday bilmaydi;

- buyurtmachi vakillari (turli darajadagi rahbarlar, texnologiya mutaxassislari, oddiy foydalanuvchilar) kelajak tizimi haqida o‘z qarashlariga ega va ko‘pincha ularning tizimga bo‘lgan talablari o‘zaro

farq qiladi. Ushbu holat, ayniqsa, ishlab chiqilayotgan tizim bir nechta avtomatlashtirish obyektlarida amalga oshirilganda odatiy holdir;

- mijoz koʻpincha zamonaviy hisoblash tizimlarining imkoniyatlarini bilmaydi va avtomatlashtirish jarayonini qoʻlda bajariladigan elementar faoliyatni kompyuterlarga oddiy uzatish deb hisoblashga intiladi. Shu bilan birga, ular yangi texnologiyalar kelishi bilan tashkilot ichidagi biznes-jarayonlarni optimallashtirish haqida oʻylamaydilar;

- mijoz baʻzi funksiyalarni virtual mashinalar bajarishi mumkinligiga ishonmaydi.

Funksional model qurish, bu muammolarni eng avval hal qilish kerak. Bu jarayon batafsil koʻrib chiqamiz.

Uni ishlab chiqishda ishni mavjud tashkil etishning modeli avvalo ish tavsiflari, farmoyishlari, hisobotlari, meʻyoriy hujjatlari va boshqalar asosida quriladi. Modelni tahlil qilish zaif tomonlar qaerda, yangi jarayonlarning afzalliklari qanday boʻlishini va korxona (kompaniya, boʻlim) ning mavjud tashkilotiga qanday chuqur oʻzgarishlar kiritilishini tushunishga imkon berishi lozim. Faoliyatni samarasiz tashkil etish belgilari boʻlishi mumkin:

- foydasiz, nazoratdan tashqari va duplikativ ishlar;
- natijasiz ishlar;
- samarasiz hujjat oqimi (toʻgʻri hujjat oʻz vaqtida toʻgʻri joyda emas) va boshqalar.

Modelda topilgan kamchiliklar modelini yaratishda tuzatilib, korxonaning yangi tashkiloti modeli taklif qilinishi ham mumkin. Bu model muammoni hal qilishning muqobil yoʻllarini tahlil qilish va eng yaxshisini tanlash uchun ishlatiladi.

SADT metodologiyasi (Structured Analysis and Design Technique - strukturali tahlil va loyihalash texnikasi) - tizimning funksional modelini qurish uchun moʻljallangan usullar, qoidalar va protseduralar majmui.

Bu metodikani ishlab chiqishni XX asrning 60-yillari oʻrtalarida Duglas Ross (Aqsh) boshlab bergan. Keyin, Softech, Inc tahlilchilar tomonidan SADT yaxshilandi va muammolar keng koʻlamli hal qilish uchun ishlatiladi. Telefon tarmogʻi dasturiy taʻminot, diagnostika, uzoq muddatli va strategik rejalashtirish, kompyuter yordam ishlab chiqarish va loyihalash, kompyuter tizimi konfiguratsiya, kadrlar tayyorlash, va moliyaviy va logistika boshqarish SADT samarali ilovalar yaratila boshlandi. Keng sohalar SADT metodikasining koʻp qirraliligi va

qudratini ko‘rsatadi. Amerika qo‘shma Shtatlari Mudofaa vazirligining integratsiyalashgan kompyuter quvvat ishlab chiqarish (ICAM) dasturi SADTning foydaliligini tan oldi. Bu 1981-yilda uning bir qismini "IDEF0" (Integrated Computer Aided Manufacturing, ICAM) deb nomlab, dasturiy ta‘minotni ishlab chiqish bo‘yicha Federal standart sifatida chop etishga olib keldi. Bu nom ostida SADTdan harbiy va sanoat tashkilotlarida minglab mutaxassislar foydalanishgan. Milliy standartlar va texnologiyalar institutida IDEF0 standartining so‘nggi versiyasi 1993-yilning dekabrida chiqarildi.

Shuni ta‘kidlash kerakki, IDEF0 Rossiya va Respublikamiz Gosstandard tomonidan foydalanish uchun tavsiya etildi va milliy davlat idoralarida faol qo‘llaniladi.

Ushbu metodologiya axborot tizimining funksional jihatini tavsiflashda ma’lumotlar oqimiga yo‘naltirilgan usullar (DFD) bilan raqobatlashadi. Aksincha, IDEF0 imkoniyatlari ko‘yidagilardan beradi:

- har qanday tizimni tavsiflash, nafaqat axborot tizimlari (DFD dasturiy ta‘minotni tasvirlash uchun mo‘ljallangan);
- unga qo‘yiladigan yakuniy talablarni belgilashdan oldin tizim va uning tashqi muhit tavsifini yaratish. Boshqa so‘zlar bilan aytganda, bu metodologiyasi yordamida, siz asta-sekin qurish va uning bajarilishini tasavvur qilish qiyin bo‘lsa ham, tizim tahlil qilish mumkin.

Shunday qilib, IDEF0 keng doiradagi tizimlarni yaratishning dastlabki bosqichlarida foydalanish mumkin. Shu bilan birga mavjud tizimlarning vazifalarini tahlil qilish va ularni takomillashtirish yechimlarini ishlab chiqish uchun foydalanish mumkin.

IDEF0 metodikasi grafik jarayonni tasvirlash tiliga asoslangan. IDEF0 notatsiyadagi model iyerarxik tartibli va o‘zaro bog‘langan diagrammalar to‘plamidir. Har bir diagramma tizim tavsifining birligi bo‘lib, alohida varaqda joylashgan.

Model (AS-IS, TO-BE yoki SHOULD-BE) diagrammalar 4 turni o‘z ichiga olishi mumkin

- kontekst diagrammasi;
- parchalangan(dekompozitsiya qilingan) diagramsi;
- tugun daraxt diagrammalari;
- faqat EHM uchun grafiklar (FEO).

Diagrammalar daraxt tuzilishining yuqorisi bo‘lmish kontekst diagramma (top-level diagramma) tizimning maqsadi (asosiy vazifasi) va uning tashqi muhit bilan o‘zaro ta‘sirini ko‘rsatadi. Har bir model faqat bitta kontekst grafikka ega bo‘lishi mumkin. Asosiy funksiyani

tavsiflagandan so‘ng funksional dekompozitsiya amalga oshiriladi, ya‘ni asosiy funksiyani tashkil etuvchi funksiyalar aniqlanadi.

Bundan tashqari, o‘rganilayotgan tizimning kerakli darajada detallashtirilgunga qadar funksiyalar subfunksiyalarga bo‘linadi va hokazo. Sistemaning har bir shunday fragmentini tasvirlovchi diagrammalar dekompozitsion diagrammalar deyiladi. Har bir dekompozitsiya seansidan so‘ng ekspert tekshiruv seanslari o‘tkaziladi – domen ekspertlari real jarayonlar yaratilgan diagrammlarga mos kelishini ko‘rsatadilar. Topilgan har qanday nomuvofiqliklar bartaraf etiladi, so‘ngra jarayonlarni yanada batafsil davom etiradi.

Tugun daraxti diagrammasida funksiyalar (ish o‘rinlari) ning iyerarxik munosabatlari ko‘rsatilgan, lekin ular o‘rtasidagi munosabatlar emas. Ulardan bir nechtasi bo‘lishi mumkin, chunki daraxt har qanday chuqurlikka va har qanday tugundan qurilishi mumkin.

EHM diagrammalari tizimda yuz beradigan jarayonlarning muqobil ko‘rinishini ko‘rsatish uchun modelning alohida qismlarini ko‘rsatish uchun mo‘ljallangan (masalan, tashkilot boshqaruvi nuqtai nazaridan).

Kichik loyihalar algorimlarini yozish usullari. Tizim vazifalarini belgilab va axborot asoslarini ishlab chiqqandan so‘ng, keyingi qadam tizimida axborot oqimi harakati loyihalashtirish iborat. Tizimning modeli kerakli funksiyalarga qanday erishish va uni ta‘minlash uchun qanday ma’lumotlar ishlatilishini ko‘rsatadi. Shunday qilib, model bevosita tizimning funksional va axborot modellariga asoslanadi.

Bu modellar odatda IDEF0 va DFD dekompozitsiya diagrammalarining oxirgi darajalarida ko‘rsatilgan funksiyalar (jarayonlar) uchun quriladi. DFD uchun diagramma yoki diagramma ko‘rinishidagi modeli elementar jarayonlarni minispektlar ko‘rinishida tasvirlashdan ko‘ra yaxshiroq yechimdir.

Bunday vaziyatlarda funksional imkoniyatni tasvirlashning eng eng taniqli va mashhur usullari va uslubiyati bor:

- algoritmlarning blok diagrammalari;
- Jadvalning EPC ;
- BPMN metodologiyasi.

Algoritmlar va BPMN diagrammalarining oqimlari an‘anaviy tizimlarni tizimli yondashuv tamoyillariga asoslangan holda modellashtirish uchun eng mos keladi va funksiyalarni elementar bosqichlarga (protseduralar, operatorlar, harakatlar va hokazolarga) aniq

bo‘linishi bilan tavsiflanadi.) ma’lum ketma-ketlikda (algoritm bo‘yicha) bajariladi. EPC diagrammalar va BPMN diagrammalar bir harakat yoki ularning majmui ijrosi tizimida sodir bo‘lgan voqealar bog‘liq bo‘lgan voqeaga yo‘naltirilgan tizimlarini modellashtirish uchun yaxshi vosita hisoblanadi. Bundan tashqari, bu usul va uslubiyatlardan nafaqat xulq-atvorli modellashtirish uchun, balki funksional modellashtirish uchun ham foydalanish mumkin. Ularda ma’lum amallarni (mantiqiy simvollarni) bajarish shartlarini ko‘rsatishga imkon beruvchi elementlarning mavjudligi tizim funksiyalarining mantiqi va ketma-ketligini yaxshiroq tushunishga imkon beradi.

Algoritmlarni tasvirlashda uzoq vaqt davomida flowcharts (Basic Flowchart) muvaffaqiyatli ishlatilgan. Algoritmlarning blok diagrammalarini qurish GOST 19.701-90 (ISO 5807-85) dasturiy hujjatlarning yagona tizimi bilan tartibga solinadi. Dasturlar, ma’lumotlar va tizimlar uchun algoritmlarning sxemalari mavjud. Ushbu davlat standarti "ISO 5807-85" xalqaro standarti asosida tashkil etilgan. Axborotni qayta ishlash – ma’lumotlar, dastur va tizim oquvchilari, dastur tarmoq jadvallari va tizim resurslari jadvallari uchun hujjat simvollari va konvensiyalari moslashtirilgan.

GOST 19.701-90 ga ko‘ra, diagramma-bu muammoni hal qilishning ta‘rifi, tahlili yoki usulining grafik tasviri. Tizimning statik va dinamik jihatlarini ko‘rsatish uchun diagrammalardan foydalanishingiz mumkin. Davlat standartida berilgan simvollardan quyidagi turdagi sxemalarda foydalanish mumkin:

- ma’lumotlar sxemalari - ma’lumotlarni qayta ishlash ketma-ketligini va ularning tashuvchilarini aniqlaydi;
- dastur diagrammalari - dasturdagi amallar ketma-ketligini ko‘rsatish (aslida bu an‘anaviy ma‘noda algoritmlarning asosiy xususiyati);
- tizimning ishlash sxemasida tizimdagi amallar va ma’lumotlar oqimlari ko‘rsatiladi;
- dasturning o‘zaro ishlash diagrammalari – dasturlar (modullar) ning aktivlashtirish yo‘lini ko‘rsatish va ularning tegishli ma’lumotlar bilan o‘zaro hamkorligi uchun;
- tizim resurs diagrammalari - ma’lumotlar bloklari va qayta ishlash bloklari konfiguratsiyasini ko‘rsatish.

Yuqorida keltirilgan turdagi sxemalardan ko‘rinib turibdiki, ulardan nafaqat xulq-atvor obyektini modellashtirish uchun, balki

funksional, axborot va komponentli loyihalash vazifalari uchun ham foydalanish mumkin.

Tizimning funksional modelini qurishda strukturaviy yondashuvning asosiy tamoyillari - dekompozitsiya va iyerarxik algoritm tamoyillaridan foydalaniladi. Funksional modeli - bu turli darajadagi detallarga ega bo‘lgan o‘zaro bog‘langan diagrammalar majmui bo‘lib, har bir yangi detal darajasi bilan tizim yanada to‘liqlashadi.

Diagrammalar grafik belgilashning quyidagi elementlarini o‘z ichiga olishi mumkin:

- ma’lumot simvollari - ma’lumotlarning mavjudligini, axborotlarning turini yoki ma’lumotlarni kiritish va chiqarish usulini ko‘rsating;

- jarayon simvollari - ma’lumotlar ustida bajariladigan amallar;

- chiziq simvollari - jarayonlar va/yoki saqlash vositalari orasidagi ma’lumotlar oqimlarini, shuningdek jarayonlar orasidagi nazorat oqimlarini ko‘rsatadi;

-maxsus belgilar - diagrammalarni yozish va o‘qishni osonlashtirish uchun ishlatiladi.

Semantik mazmunga ko‘ra bo‘linishdan tashqari har bir turkum simvollari (maxsus kategoriyalardan tashqari) asosiy va o‘ziga xos simvollarga bo‘linadi. Asosiy belgi jarayon yoki ma’lumotlar tashuvchining aniq turi noma’lum bo‘lganda yoki haqiqiy ma’lumotlar tashuvchisi (jarayoni) ni ta‘riflashga hojat bo‘lmaganda qo‘llaniladi. Jarayon yoki ma’lumotlar tashuvchining aniq turi ma’lum bo‘lganda va diagrammada ko‘rsatilishi kerak bo‘lganda o‘ziga xos simvoldan foydalaniladi.

Hodisali jarayon zanjiri (EPC) - biznes-jarayonlarni modellashtirish, tahlil qilish va qayta tashkil etish (funksional modellashtirish) uchun ishlatiladigan diagramma turi. Shu bilan birga EPC diagrammalaridan funksiyalarni amalga oshirishda tizimning alohida qismlari funksionallikni modellashtirishda foydalanish va an‘anaviy oqimlarni almashtirish uchun xizmat qilish mumkin.

EPS metodi Avgust-Vilgelm Sheer tomonidan 1990-yillar boshida integrallashgan axborot tizimlari arxitekturasi metodikasiga oid ishining bir qismi sifatida ishlab chiqilgan.

EPC notatsiyasida jarayon (funksiya) diagrammasi hodisa va funksiyalarning tartibli birikmasidir. Har bir funksiya uchun unga hamroh bo‘lgan dastlabki va oxirgi hodisalar, ishtirokchilar, ijrochilar,

moddiy va axborot oqimlari aniqlanishi hamda quyi darajalarga bo‘linishi mumkin.

DFD kabi, EPC metodologiyasi turli notatsiyalarni (elementlarning sintaksisi va semantikasini) qo‘llab talqin qilish bilan rivojlanmoqda. EPC (ARIS, Microsoft Visio, Business Studio, Bflow) yordamida biznes-jarayonlarni modellashtirish qobiliyatini amalga oshirgan metodika muallifi ham, dasturiy ta‘minot ishlab chiqaruvchilari ham bunga qo‘l urishgan. Blok diagrammasiga o‘xshashlik bo‘yicha simvollar (elementlar) grafik yozuvlarini maqsadga muvofiq guruhlash mumkin. Quyidagi jadvalda adabiyot va dasturiy ta‘minotda eng ko‘p uchraydigan EPC lari va ularning muqobil tasvirlari mavjud.

EPC yordamida dekompozitsiya jarayonlarni modellashtirish va iyerarxik algoritmlar jarayonini modellashtirishning klassik tamoyillariga amal qiladi. Dekompozitsiya, alohida diagrammalarda ko‘rsatish bilan, IDEF0 diagrammalarda oldindan belgilangan jarayonlarda ishlashga o‘xshash funksiyalar uchun bajariladi.

1. EPC jarayonining diagrammasi kamida bir boshlash hodisasi bilan boshlash kerak va kamida bir tugash hodisasi bilan yakunlandi.

2. Jarayon rivojlanib borgan sari hodisalar va funksiyalar almashinib turishi kerak.

3. Diagramma vazifalari tavsiya soni ko‘pi bilan 20 ta bo‘lishi mumkin. Agar grafik funksiyalari soni 20 dan sezilarli darajada oshsa, yuqori darajadagi jarayonlar noto‘g‘ri tavsiflangan va modelni sozlash kerak bo‘ladi.

4. Hodisalar va funksiyalar jarayonning borishini aks ettiruvchi qat‘iy bitta kiruvchi va bitta chiquvchi aloqani (nazorat oqimini) o‘z ichiga olishi kerak.

5. Grafik bir tegishsiz elementlar bo‘lmasligi kerak.

6. Yuqori modul diagrammasida funksiya atrofidagi hodisalar va mantiqiy operatorlar funksiya dekompozitsiya diagrammasida boshlang‘ich / natijaviy hodisalar va operatorlar bo‘lishi kerak.

7. Har bir birlashma kamida ikkita kiruvchi ulanishga va faqat bitta chiquvchi, ayirmalar - faqat bitta kiruvchi aloqa va kamida ikkita chiquvchi bo‘lishi kerak. Operatorlar bir vaqtning o‘zida bir nechta kiruvchi va chiquvchi bog‘lanishlarga ega bo‘la olmaydi.

8. Mantiqiy operatorlar faqat funksiyalarni yoki faqat hodisalarni birlashtirishi yoki birlashtirmasligi mumkin. Bir vaqtning o‘zida funksiya va hodisani birlashtira olmaydi.

Visual C++ muhitida hisobot shaklini tayyorlash va chop etish elementlari. Buning uchun ma’lumotlar bazasini bilish talab qilinadi. Quyidagicha masala berilgan bo‘lsin.

Masala. Universitet ma’lumotlar bazasi berilgan bo‘lsin. Unda talaba va semestr nomli jadvallar bo‘lsin.

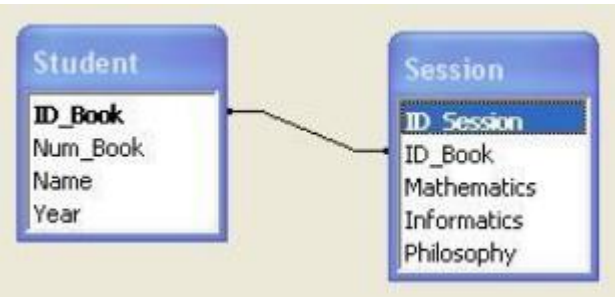
14.1-jadval.Talaba jadvalining tuzilishi

Maydon nomi	tipi	Izoh
ID_Book	Butun son (int)	Birlamchi kalit maydon
Num_Book	matnli (10 belgi)	Talabaning reyting daftari raqami
Name	matnli (30 belgi)	Talabaning ismi sharifi
Year	Butun son (int)	Tug‘ilgan yili

14.2-jadval. Semestr jadvalining tuzilishi

Maydon nomi	tipi	Izoh
ID_Session	Butun son (int)	Birlamchi kalit maydon
ID_Book	Butun son (int)	reyting daftari raqami
Mathematics	Butun son (int)	Matematika bahosi
Informatics	Butun son (int)	Informatika bahosi
Philosophy	Butun son (int)	Falsafa bahosi

Ushbu jadvallarning o‘zaro munosabati quyidagi bog‘lanish orqali ifodalangan bo‘lsin.



14.1-rasm. Jadvallarning relyatsion bog‘lanishi.

Berilgan ma’lumotlar asosida hisobotlardan foydalanishni tashkil qilib beruvchi dasturni ishlab chiqishni ko‘ramiz. Quyidagi ma’lumotlarni o‘z ichiga olgan hisobot yaratish lozim:

- Reyting daftarini
- Talabaning ismi – sharifi
- Matematika bahosi
- Informatika bahosi

- Falsafa bahosi
- Talabaning o‘rtacha bahosi

Joriy hisobotni yaratishda ma’lumotlar Microsoft Access DBMS tomonidan hosil qilingan so‘rov orqali olinadi. So‘rov Query1 deb ataladi. Hisobot alohida shaklda ko‘rsatilishi kerak.

Buning uchun quyidagi amallar bajariladi:

1. Microsoft Visual Studio tizimi orqali Windows Forms dastur shabloni yordamida loyiha yaratiladi. Microsoft Visual Studio ishga tushirilgandan so‘ng, yangi loyiha yaratish kerak. Windows Forms dastur shabloni yordamida yangi loyihani yaratish va saqlashning batafsilroq aytib o‘tilgan.

2. Ma’lumotlar bazasini yaratish yoki tayyor ma’lumotlar bazasi faylini yuklab olish lozim. Ma’lumotlar bazasi uchun Microsoft Access ma’lumotlar bazasini boshqarish tizimi yordamida yaratiladi. Ma’lumotlar bazasida ikkita tegishli jadval talaba va sesmestr, shuningdek, hisobotni yaratish uchun ishlatiladigan Query1 so‘rovi yaratib olish kerak.

3. Ma’lumotlar bazasi faylining loyihaga ulanishi. Ma’lumotlar bazasi faylini loyihaga ulashdan oldin ushbu faylni loyiha fayllari saqlanadigan papkaga saqlashni tavsiya etamiz. Foydalanish uchun “Education.mdb” ma’lumotlar bazasi, uni standart usulda loyihaga ulash kerak. Loyihaga ma’lumotlar bazasi faylini ulash uchun "Add Connection" dan foydalaniladi. Bu amal kontekst menyudan buyruq ro‘yxatida chaqiriladi.

4. Natijada “ma’lumotlar manbasini tanlash” oynasida ma’lumotlar manbai (Data Source) ni tanlash uchun - Microsoft Access ma’lumotlar bazasi fayli ulash kerak. Keyingi oynada "ma’lumotlar bazasi fayl nomi" dagi “ulanish Qo‘shish” ma’lumotlar bazasi fayliga yo‘lni ko‘rsatiladi (“Browse” tugmasi orqali).

5. Asosiy loyiha oynasini ishlab chiqish. Topshiriq shartiga ko‘ra hisobot alohida shaklda ko‘rsatilishi lozim. Shuning uchun loyihaning asosiy shakli 14.2-rasmda ko‘rsatilgandek shaklga ega bo‘ladi.

14.2-rasm.

Agar rasmda ko‘rib turganingizdek, shakl button1 nomidagi faqat bitta tugma turini o‘z ichiga oladi. "Show Report" tugmasini bosgandan so‘ng hisobot boshqacha shaklda ko‘rsatiladi. Quyidagi xususiyatlarini sozlash kerak:

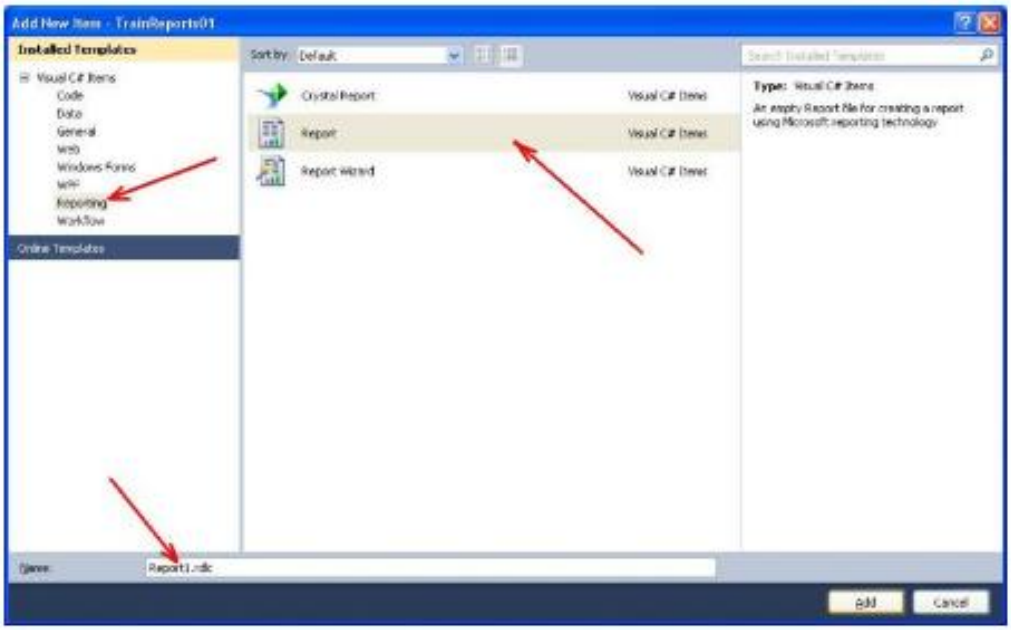
Boshqarish elementi button1 ning Text = «Show Button»

Boshqarish elementi Form1 ning Text = «Report Viewer»

6. Hisobot faylining loyihaga ulanishi. Microsoft Visual Studio dasturida har bir hisobotda «*.rdlc» kengaytmasiga ega fayl mavjud. Bu fayl hisobotda hosil qilingan ma’lumotlarni o‘z ichiga oladi. Microsoft Visual Studio dasturida fayl yaratish uchun loyiha menyusidan Add new Item buyrug‘ini chaqiriladi (14.3-rasm).

14.3-rasm. Loyiha menyusidan Add new Item amali

7. Natijada, «Add new Item…» hisobot shablonini tanlash kerak bo‘lgan oyna ochiladi (14.4-rasm) va hisobot fayli uchun nom belgilanadi.

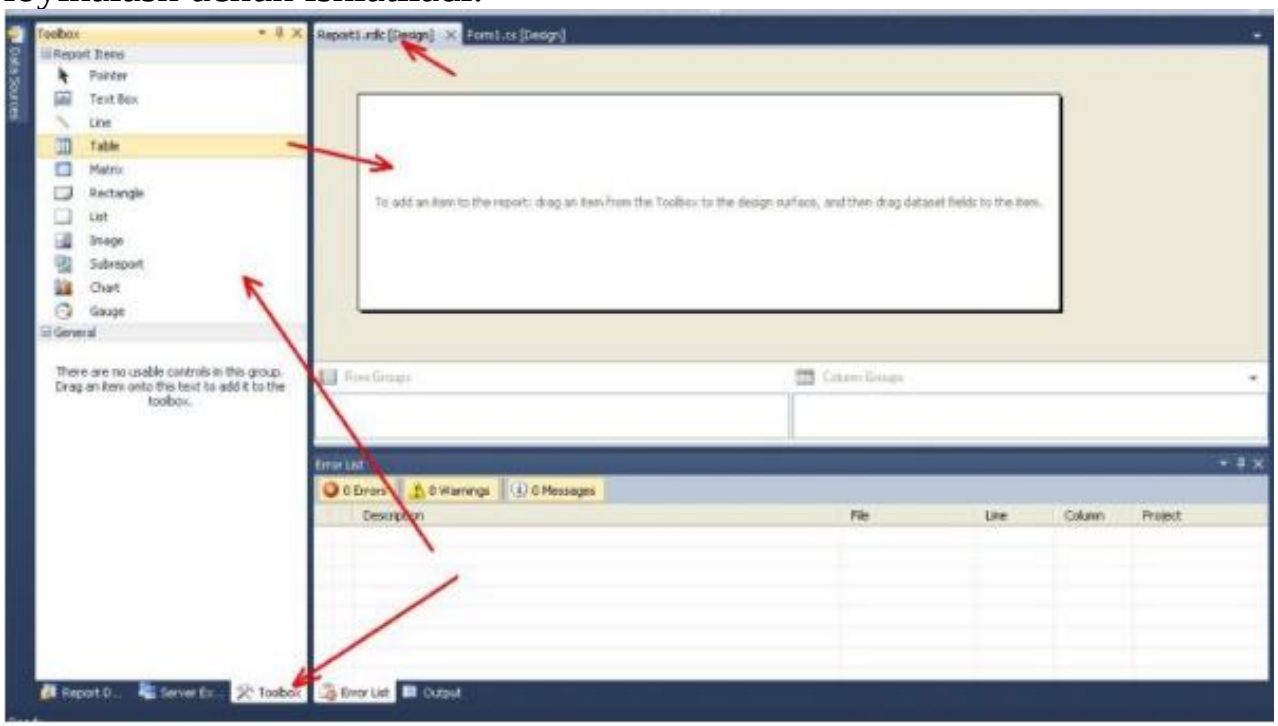


14.4-Rasm. Hisobot shablonini tanlash va hisobotni o‘rnatish

Tanlovni tasdiqlagandan so‘ng Microsoft Visual Studio oynasida hisobotning asosiy oynasi ko‘rsatiladi.

Hisobot oynasida hisobotni loyihalashtirish lozim. Buning uchun quyidagi lozim.

1. ToolBox asboblar paneli. ToolBox paneli - hisobot elementlarini loyihalash uchun ishlatiladi.



14.5-rasm. ToolBox panel va hisobot loyihalashtirish vositalari.

2. Hisobot yaratish uchun - Microsoft Visual Studio hisobotini ishlab chiqish uchun bir qator taklif etiladigan standart shablonlar bor.

Bu boshqaruvlardan turli hisobotlarning ko‘rinishini (taqdimotini) loyihalashtirish uchun foydalanishingiz mumkin.

Quyidagi elementlar taklif etiladi:

TextBox - Hisoblashlar natijasida olingan teglar, maydonlar yoki qiymatlarni ko‘rsatadi;

Line - nuqta yoki undan ko‘p qalinlikka ega bo‘lgan chiziq chizadi;

Table (jadval) - nazorat ustunlar va qatorlar o‘zgarmaydigan ma’lumotlarni ko‘rsatadi;

Matrix - o‘zgaruvchan raqamli, ustunlar va qatorlar o‘zgaruvchan raqami bor ma’lumotlarni jamlash ko‘rsatiladi;

Rectangle - hisobot elementlari uchun shablon sifatida to‘rtburchak chizadi;

List - har bir guruh yoki ma’lumotlar qatori uchun takrorlanadigan hisobot bandlari to‘plamini ko‘rsatadi;

Image - tasvirni bitmap sifatida ko‘rsatadi (masalan, logotip yoki fotosurat);

Subreport - hisobot chegaralari doirasida bog‘liq sub-hisobot ko‘rsatadi;

Chart - Ma’lumotlarni har xil turdagi diagrammalar ko‘rinishida ko‘rsatadi;

Gauge - Maydon yoki ifodaning qiymatini chiziqli yoki radial qarama-qarshilik ko‘rinishida ko‘rsatadi.

Har bir nazoratdan foydalanishning o‘ziga xos xususiyatlariga kirmasdan, muammoni hal qilish uchun jadval nazoratidan foydalanamiz. Bu nazorat ustunlar son va qatorlar bir o‘zgarmaydigan qator bor ma’lumotlarni ko‘rsatadi.

Bu elementlarning foydalanish xususiyatlari tizimda bir xil amalga oshiriladi.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Visual C++ muhitida turli sohaga oid masalalarni yechish uchun kichik loyihalarni loyihalash usullari asosan qanday tizimlarni loyihalashtirish usullari nazarda tutiladi?
2. Axborot tizimlarining loyihalashtirish uchun axborot tizimining qanday davrini yaxshi bilish va tushunish kerak?
3. Hayot davrining tuzilishini belgilovchi asosiy standart nomini ayting.

4. Hayot davri strukturasi nechta jarayon guruhiga asoslangan?
5. Mustaqil (loyihaga nisbatan) ekspertlarning korxona faoliyatining qabul qilingan talablar, rejalar va shartnoma shartlariga muvofiqligini aniqlash ishlari nima deb yuritiladi.
6. Strukturali tahlil va loyihalash texnikasiga asoslangan metodologiya nomini ayting.
7. Kichik loyihalar algorimlarini yozish usullari nechta?
8. BPMN diagrammalarning vazifasi nimadan iborat?
9. Algoritmlarning blok diagrammalarini qurish qanday GOST dasturiy hujjatlarning yagona tizimi bilan tartibga solinadi?
10. Ma’lumotlar sxemalari nima uchun kerak?
11. Hisobot shaklini tayyorlash va chop etish uchun nimani bilish kerak va nima uchun?
12. Jadvallarning relyatsion bog‘lanishi deganda nimani tushunasiz?
13. Yangi hisobot loyihasini yaratish uchun qanday amallar bajariladi?
14. Hisobot konchtruktorining elementlarini sanab bering.
15. Hisobotda jadvalga asoslangan ma’lumotlarni chiqarish uchun qaysi elementdag foydalanish maqsadga muvofiq?

3.7.Foydalanuvchi interfeysini loyihalash.

📖 Foydalanuvchi interfeysini loyihalash usullari, talablari, ma’lumotlarni eksport qilish vositalari va komponentalari, .NET texnologiyasining hisobotni yaratish konstruktori, testlash va instruksiya yozish usullari, sinovchi xodim xususiyatlari bo‘yicha nazariy bilimlar keltirilgan. Bilimlarni mustahkamlash uchun 15 ta nazariy savol berilgan.

🖋 *Kalit so‘zlar.* Interfeys, UI, WUI, amaliy dastur, ma’lumotlarni kiritish, PUI, integratsiya, o‘zaro bog‘lanish, xisobot, testlash.

☑ ***Bilish shart bo‘lgan tushunchalar.*** Operatsin tizim va amaliy dasturlarda ishlash, sinf va sinf obyekti, xususiyat, hodisa, forma, komponenta dasturlashga oid dastlabki tushunchalar hamda C++ tilini qo‘llab quvvatlovchi muhitda ishlashni bilish lozim.

👍 ***Bilib olasiz***. Visual C++ muhitida tuzilgan dasturlarning foydalanuvchi interfeysini loyihalash usullari, talablari, ma’lumotlarni eksport qilish vositalari va komponentalari, .NET texnologiyasining hisobotni yaratish konstruktori, testlash va instruksiya yozish usullari, sinovchi xodimga qo‘yiladigan talablarni o‘rganishingiz mumkin.

REJA

1.Foydalanuvchi interfeysini loyihalash usullari.
2.Ma’lumotlarni eksport qilish vositalari va komponentalari.
3.Testlash va instruksiya yozish usullari.

Inson-mashina interfeysi (IMI) keng tushunchadir. Inson-operator va ular boshqaradigan mashinalar bilan o‘zaro aloqasi ta‘minlashning muhandislik yechimlarni o‘z ichiga oladi. Inson - mashina interfeysi tizimlarini yaratishda yaqindan ergonomika tushunchalari bilan bog‘liq. IMI loyihalashtirish quyidagilarni o‘z ichiga oladi:

- ish joyini yaratish: ishchi maydon va boshqaruv paneli, qurilmalar va boshqaruvlarni joylashtirish, ish joyi va ehtimol ranglar uyg‘unligi.
- operatorning barcha boshqaruv bilan o‘zaro aloqasini ta‘minlash hisoblanadi: ularning mavjudligi va zarur harakatlari, kirish samaradorligi va tezligi, mustahkamligi, nazorat harakatlarining (izchilligi), joylashgan o‘rni , displeylar va ulardagi teglar hajmi (bularning barchasi mavjudligi).

Inson - kompyuter o‘zaro hamkorligi ilmiy ko‘p yo‘nalishli konteksti kompyuter grafika, muhandislik psixologiyasi, ergonomika, tashkilot nazariyasi, kognitiv fan, informatika va boshqalarni o‘z ichiga oladi.

Foydalanuvchi interfeysi-bu foydalanuvchi kompyuter bilan o‘zaro aloqa qilish imkonini beruvchi dasturiy va apparat vositalar majmuasidir. Bu o‘zaro muloqotlar dialoglarga asoslanadi. Bu holda dialog deganda inson va kompyuter o‘rtasida Real vaqtda amalga oshiriladigan va muayyan vazifani birgalikda hal etishga qaratilgan axborot almashuvi tushuniladi. Axborot almashish va harakatlarni muvofiqlashtirish har bir dialog foydalanuvchi va kompyuterni fizik ulaydigan alohida i/o jarayonlardan iborat. Xabarlarni uzatish orqali axborot almashiladi va nazorat signallari. Xabar-dialog almashinuvida ishtirok etuvchi axborot qismidir. Ular quyidagilardir:

- odam tomonidan kiritish vositalari yordamida hosil qilinadigan kirish xabarlari: klaviatura, manipulyatorlar, sichqoncha kabi va boshqalar;
- kompyuter tomonidan matnlar, audio signallar va/yoki tasvirlar ko‘rinishida hosil qilinadigan va foydalanuvchiga monitor ekranida yoki boshqa chiqish qurilmalarida ko‘rsatiladigan chiqish xabarlari.

Asosan foydalanuvchi quyidagi turdagi xabarlarni hosil qiladi: axborot so‘rovi, yordamchi so‘rovi, operatsiya yoki funksiya so‘rovi, ma’lumot kiritish yoki o‘zgartirish, shablon maydonini tanlash va

boshqalar. Bunga javoban, u qabul qiladi: maslahatlar yoki yordam, javob talab qilmaydigan axborot xabarlari, harakatni talab qiladigan so‘rovlar, javob kerak bo‘lgan xato xabarlari va shablon formatidagi o‘zgarishlar.

Dasturlashga protsessual va obyektga yo‘naltirilgan yondashuvlar bilan taqqoslaganda, interfeysni ishlab chiqishga protsessual yo‘naltirilgan va obyektga yo‘naltirilgan yondashuvlar mavjud. Protseduraga yo‘naltirilgan interfeyslar "protsedura" va "operatsiya" tushunchalariga asoslangan foydalanuvchi o‘zaro aloqasining an‘anaviy modelidan foydalanadi. Bu model doirasida dasturiy ta‘minot foydalanuvchiga tegishli ma’lumotlarni va uning natijasini foydalanuvchi aniqlaydigan ma’lum amallarni bajarish imkoniyatini beradi.

Obyekt yo‘naltirilgan interfeyslarni foydalanuvchi o‘zaro bir oz boshqacha modelini foydalanib, manipulyatsiya domen obyektlarini qaratishi mumkin.

Zamonaviy realliklardan farqli o‘laroq, birinchi kompyuterlar grafik foydalanuvchi interfeyslari juda zaif bo‘lgan. Shuning uchun, eng boshida, odamlar faqat buyruq qatordan foydalanishingan (CLI yoki buyruq qatorni interfeysi), unda buyruqlar so‘rovlar yordamida belgilangan edi. Keyinchalik, bu TUI interfeyslarni ishlab, endi operatsion tizimlari o‘rnatish jarayonida ishlatiladi. Kompyuterlarning mavjudligi foydalanuvchilarga qulay interfeysni ishlab chiqish zarurligiga olib keladi.

Grafik foydalanuvchi interfeysi-bu kompyuterning tobora ortib borayotgan ishlashi bilan birga mustahkam o‘rnatilgan interfeys turi. Yaqin kelajakda, odamlar nutq yordamida kompyuter bilan o‘zaro imkonini beradi foydalanuvchi audio interfeyslarni (VUI yoki ovozli foydalanuvchi interfeysi) bo‘lishi mumkin.

Turli kompyuter o‘yinlari tabiiy foydalanuvchi interfeysi (NUI yoki tabiiy foydalanuvchi interfeysi) yaratib beradi. Uning tizimi insonning harakatlarini tahlil qiladi va ularni o‘yindagi harakatlarga aylantiradi. Perseptual foydalanuvchi interfeysi (PUI) va aqlli-kompyuter interfeysi (BCI yoki brain-kompyuter interfeysi) hozirda ishlab chiqilmoqda. eng so‘nggi rivojlanish odamlarga kompyuterlarni fikri kuchi bilan boshqarish qobiliyatini ta‘minlashga qaratilgan.

Web-ilovalarning foydalanuvchi interfeysi ham katta auditoriyadan kelib chiqqan holda o‘ziga xos xususiyatlarga ega. Notiqlik haqida, notiqlik-ritorika haqida shunday fan mavjud. Fikrlash insoniyat bor

ekan, bu mavzu bo‘yicha birinchi ishlar Empedokllarga (taxminan miloddan avvalgi v asr) taalluqlidir. Ritorikadan bilamizki, hozirgi odamlar sonining ortishi bilan tomoshabinlarning intellektual darajasini kamaytirish qonuni mavjud. Auditoriya qanchalik katta bo‘lsa, nutq shunchalik sodda bo‘lishi, iboralar shunchalik qisqa bo‘lishi va nutqdagi pauzalar shunchalik muhim ahamiyat kasb etadi. Boshqa tomondan, ijtimoiy tarmoqlarning rivojlanishi bizga muloqot mutlaqo og‘zaki emasligini ko‘rsatadi va foydalanuvchi interfeysi orqali ham muloqot qilishingiz mumkin. Yuqoridagilardan kelib chiqqan holda, veb-dastur interfeysi sodda bo‘lishi kerakligi mantiqan kelib chiqadi. Iloji boricha oddiy. Aytaylik, ma’lum bir korxona ichida ishlash uchun mo‘ljallangan dastur uchun interfeys ishlab chiqyapsiz. Shu bilan birga, siz har doim tizimning kelajakdagi foydalanuvchilari uchun o‘quv kurslarini tashkil qilishingiz, ekranda nima borligini va ishlab chiquvchi sifatida nima demoqchi ekanligingizni aytishingiz mumkin.

FI dizayni ko‘plab yondashuvlarni o‘z ichiga oladi: kaskadli, ichki, tashqi, iterativ va boshqalar. Yondashuvlarning har biri ichida bir qator usullar ham mavjud. Yuqori darajadagi yondashuvdan qat‘iy nazar, yuqori darajadagi va past darajadagi o‘zaro aloqalarni loyihalash o‘ziga xos usullarni talab qiladi. Obyektga yo‘naltirilgan yondashuv tasvirlangan, garchi ko‘plab dizayn bosqichlari uchun an‘anaviy rivojlanish usullari bilan almashtirilishi mumkin. Ushbu yondashuv yordamida olingan dizayn natijalari FIga yo‘naltirilgan ilovalarni hujjatlashtirish, modellashtirish, prototiplash, baholash va amalga oshirish uchun foydalidir.

Nostandart yondashuv. Yuqori darajadagi loyiha tizimi bilan FI va foydalanuvchi o‘zaro standart xatti haqida o‘ylash mumkin, lekin tafsilotlar qo‘shiladi, deb, u standarti erishish uchun yanada qiyin bo‘ladi — va bir vaqtning o‘zida izchil-qo‘llash turli qismlariga ajratiladi. Tafsilotlar odatda kompromisslar va kutilmagan hodisalar bilan bohliqdir .

Foydali qoida. Bu tasodifan yoki loyihaga muvofiq sodir bo‘lsa, nostandart xatti harakatlari qo‘llash kerak.

Yangilangan grafik. Dastlabki grafik imkoniyatlar va vizuallashtirish masalalari dastlabki loyihalash bosqichlarida ishlab chiqiladi, shuning uchun batafsil loyihalash bu grafik imkoniyatlarni aniqlashni talab qiladi (ba‘zan grafikadan foydalanish dastlabki loyihalash tushunchalaridan tashqariga chiqadi). Bundan tashqari, bu bosqichda ekranlar va xabarlarning yakuniy batafsil tartibi talab qilinadi.

Javob vaqti. Og‘ir tahlil yoki simulyatsiya orqali, muhim va eng tez-tez ishlatiladigan dasturiy ta‘minot joylari uchun javob vaqt smetasini olish kerak. Oxirgi foydalanuvchi tomonidan amalga oshirilgan har bir funksiya uchun ehtimoliy javob vaqtiga asoslanib, dasturiy ta‘minot uchun mos teskari aloqa mexanizmi ishlab chiqiladi.

Interfeys yaratishning asosiy tamoyillari

1. Tabiiylik (intuitiv). Tizim bilan ishlash vazifasini hal qilish jarayonini boshqarish uchun kerakli direktivalarni (interfeys elementlarini) topishda foydalanuvchiga qiyinchilik tug‘dirmasligi kerak.

2. Mustahkamlik. Agar foydalanuvchi tizim bilan ishlayotgan paytda tizimning biror qismi bilan ishlash uchun ayrim metodlardan foydalangan bo‘lsa, u holda metodlar tizimning boshqa qismida bir xil bo‘lishi kerak. Shuningdek, interfeys orqali tizim bilan ishlash o‘rnatilgan, doimiy, qulay me‘yorlarga (masalan, Enter klavishidan foydalanish) mos kelishi kerak.

3. Ortiqchalilik. Bu foydalanuvchi faoliyat yoki tizimini boshqarish uchun faqat minimal ma’lumot kiritish kerak degan ma‘noni anglatadi. Masalan, foydalanuvchi ahamiyatsiz sonlarni kiritmasligi kerak (00010 o‘rniga 10). Xuddi shunday, foydalanuvchi oldindan kiritilgan yoki avtomatik ravishda tizimdan olinishi mumkin bo‘lgan ma’lumotlarni kiritishni talab qila olmaydi. Axborot kiritish jarayonini kamaytirish uchun iloji boricha standart qiymatlardan foydalanish tavsiya etiladi.

4. Yordam tizimiga to‘g‘ridan-to‘g‘ri kirish. Ish vaqtida tizim foydalanuvchiga kerakli ko‘rsatmalar berishi kerak bo‘ladi. Yordam tizimi uchta asosiy jihatga javob beradi: taqdim etilgan buyruqlar sifati va miqdori; xato xabarlarining tabiati va tizimning nima qilayotganini tasdiqlash. Xato xabarlari foydalanuvchi uchun foydali va tushunarli bo‘lishi kerak. Shu bilan birga, bu savollarga foydalanuvchilar uchun imkon qadar muloyim va tushunarli bo‘lgan tilda javob berishi kerak.

5. Moslashuvchanlik. Tizim interfeysi turli darajadagi tayyorgarlikka ega foydalanuvchilarga xizmat qilishi kerak. Tajribasiz foydalanuvchilar uchun interfeys iyerarxik menyu strukturasi va tajribali foydalanuvchilar uchun buyruqlar, kalit kombinatsiyalar va parametrlar sifatida tashkil etilishi mumkin.

Sayt moslashuvchanligi - foydalanuvchilar osonlik bilan saytida manipulyatsiyadan ortiqcha raqami bilan bezovta holda zarur ma’lumotlarni topish imkonini beradi va sayt materiallar tashkil etish

hisoblanadi. Maxsus tadqiqotlarga koʻra, oʻrtacha bir foydalanuvchi saytda oʻttiz etti soniya sarflaydi va bu vaqt ichida ular kamdan-kam sahifaning oxiriga oʻtadi. Shu munosabat bilan, sayt samimiy ma’lumotlar chiqishi tashkil etish sohasida zamonaviy standartlarga javob bermasa, agar, bunday sayt targʻib qilish, barcha saʻyharakatlari kerakli maqsadga erishish mumkin emas, chunki foydalanuvchi saytida zarur ma’lumotlarni topib boʻlmasa, ular shunchaki koʻproq samimiy boshqa resursga oʻtadi. Saytning eng samarali moslashuvchanligi taʻminlash Internetda axborot taqdim etish xususiyatlarini bilishni talab qiladi. Foydalanuvchilar, birinchi navbatda, ularning koʻz oldida moddiy koʻrish muhimdir. Shuning uchun interfeysning maksimal soddaligi va qulayligi internetda ma’lumotlarni tashkil qilishning asosiy tamoyilidir. Saytda oddiy va samimiy interfeysni yaratish uchun mavjudlikning asosiy tamoyillarini amalga oshirishga imkon beruvchi texnikalardan foydalanishin kerak. Butun sayt uchun navigatsiya umumiy boʻlishi kerak, sayt logo yuqori chap burchagida joylashtirilgan boʻlishi kerak va logo bosish bosh sahifa olib kelishi kerak va veb-saytida header va altbilgide aloqa ma’lumotlarni joylashtirilgan kerak. Saytda malakali qidiruvni tashkil etish saytning mavjudligini taʻminlashning muhim elementi hisoblanadi. Qidiruv tugmasi, logotipdan tashqari, sayt uchun ikkinchi eng muhim element hisoblanadi va u har qanday saytda foydalanuvchi uchun birinchi yordamchidir. Foydalanuvchi birinchi 2 soniya ichida bu tugmani topsa kamida, ular xavfsiz his va sayt tomon keskinlik ularning darajasi kamayadi. Moslashuchanlik sinov oʻtkazish paytida taʻqib qilinishi kerak va uning ma’lum bosqichlari bor. Sinov oʻtkazishning mohiyati quyidagicha:

1. Muammolarni aniqlash;
2. Gipotezani hosil qilish;
3. Test uchun oʻlchov aniqlash;
4. Belgilar va test ssenariysini aniqlash;
5. Tanlang respondents;
6. Anketalarni toʻldirish;
7. Kirish brifingini oʻtkazish:
8. Mavjudligi test oʻtkazish;
9. Intervyu respondents;
10. Natijalarni tahlil qilish;
11. Sayt dizayni uchun talablarni aniqlash.

Umuman olganda FIni ishlab chiqish soha va foydalanuvchilarning imkoniyatlari va xoxishlarini inobatga olishi kerak.

Ma’lumotlarni eksport qilish vositalari va komponentalari. Zamonaviy dasturlar foydalanuvchini qiziqtiradigan ma’lumotlarga kirishning ko‘p usullarini ta‘minlaydi. Ma’lumotlar bilan ishlash uchun ixtisoslashgan dastur ilovalari ishlab chiqilgan. Microsoft Office kabi umumiy ofis vositalari bilan integrallashgan ma’lumotlarni chiqarish imkoniyatlarini o‘z ichiga oladi. Foydalanuvchi hozirda ma’lumotlar bazasidan kerak ma’lumotlarni chiqarib eksport qilishi mumkin. Biroq, turli xil ma’lumotlar bilan ishlashning eng keng tarqalgan va tanish usuli an‘anaviy hisobot shaklidir. Bir tomondan, ko‘p foydalanuvchilar, ayniqsa, hali hisobot standart qog‘oz shaklidan voz kechishga kuch topa olmaydi. Boshqa tomondan, hisobot, asosan, turli xil xususiyatlarga ega bo‘lgan ma’lumotlarni birlashtiruvchi vositadir.

Hisobot berish funksiyalarini ta‘minlash talabi ko‘pincha zamonaviy dasturiy mahsulotlarni ishlab chiqishda paydo bo‘ladi. Databeacon Toolkit sifatida dastur plagin-smaylik va hisobot qo‘llab-quvvatlash vositalari va komponentalari mavjud.

Shunga o‘xshash vositalar Microsoft Visual Studio. net (VS.NET)dir. Microsoft Visual Studio. net (CRVS) uchun Crystal Reports deb nomlangan, biznes obyektlarini ishlab chiqish va litsenziyalash sxemasi bo‘yicha etkazib berishga yщnaltirilgan. Yana bir vositasi, Microsoft hisobot xizmatlar (MSRS), Microsoft SQL Server DBMS asoslangan platforma hisoblanadi.

Crystal Reports for Microsoft Visual Studio .NET (Hisobotlarni qurish instrumenti). Crystal Decisions mahsulotlarini Microsoft development tools bilan bog‘lash tarixi 1993-yilga borib taqaladi. O‘shanda Crystal Decisions hisobotlarning tarkibiy qismlari avval Visual Basic 3.0 da taqdim etilgan edi. 1996 Crystal hisobotlar dasturi Visual Studio development tools paketining bir qismiga kiritildi va 2002 da uning bir qismi bo‘ldi VS.NET va. net Framework, rasmiy nomi CRVS qabul qilindi. Crystal Reports komponentlarining paydo bo‘lishi va ularni o‘zaro rivojlangan varinatlari mavjud. Masalan, VS.NET 2003VS.NET 2005.

CRVS komponentalarining asosiy maqsadi dasturning hisobot funksiyalarini ishlab chiqishni qo‘llab-quvvatlashdir. Bu maxsus hisobot shablon talablariga muvofiq ishlab chiqilgan bo‘ladi. Hisobot sahifasida ma’lumotlarni joylashtirishni belgilovchi rpt kengaytmasidan foydalanib, shablonda qaysi ma’lumotlar tuzilmalari, hisobot ma’lumotlari uchun manba bo‘lib xizmat qilishi ko‘rsatilgan bo‘lishi kerak. Keyinchalik shablon manbadan ma’lumotlarni chiqarib, yakuniy

hisobotning tuzilishini shakllantirish uchun foydalanadigan CRVS komponentlarini kiritishga o‘tiladi. Hisobotni ko‘rsatish uchun ishlab chiqilayotgan dasturga ko‘shilgan maxsus funksiyalardan foydalanib web va Windows ilovalarini ham qo‘llab-quvvatlanadi.

Aslida, VS.NETda ekspert jarayoni kodi bir necha satrlarni chaqirib kamayadi, lekin " hisobot dizayner " yordamida amalga oshiriladi. Shunday qilib, hisobot shablonini yaratish uchun Microsoft. net platformasida dasturlashingiz shart emas (MS.NET). xohlasangiz, uni o‘zgartirish imkonini beruvchi, dastur hisobot shablonni o‘z ichiga olishi mumkin. Bu juda qulay, chunki bu holda, har doim ishlab chiqilgan dasturni qo‘llab-quvvatlash uchun amalga oshirish mumkin.

Yuqorida aytib o‘tilganidek, CRVS komponentlari vebga yo‘naltirilgan va muntazam Windows ilovalarini ishlab chiqish uchun ishlatilishi mumkin. Ma’lumotlar bilan ishlash nuqtai nazaridan Crystal hisobotlar komponentlari ikki rejimda ishlash - surish modeli (Push model) va tortish modelida (Pull model) ishlatilishi mumkin.

Surish modeli - oraliq ma’lumotlar tuzilmasi bilan ishlash modeli. Shu bilan birga hisobotlar uchun ma’lumotlar odatiy takrorlanish ko‘rinishida hosil qilinadi .NET Framework - ma’lumotlar bazasi deb ataladigan ma’lumotlar tuzilmasidir. Dastur hisobotlarni qurish uchun kiritish ma’lumotlarni tayyorlash barcha mantiq amallarga bog‘liq. Bu qulay, chunki ma’lumotlar faqat ma’lumotlar manbaiga bevosita kirish yo‘li bilan emas, balki mutlaqo biron-bir tarzda hosil qilinishi mumkin. Biroq, agar hisobot yetarlicha katta bo‘lsa, juda noqulay tuzilmani to‘ldirishingiz kerak.

Tortish modeli - talab bo‘yicha ma’lumotlarni so‘rash, ya‘ni ma’lumotlar bazasidan olish. Bu holda, Crystal Reports komponentlariga o‘tgan hisobot shablon manbadan ma’lumotlarni olish bilan bog‘liq ishlarni amalga oshiradi. Ushbu yondashuv ko‘p sahifali hisobotlarni namoyish qilish uchun juda samarali, chunki ko‘rsatilgan sahifalar uchun faqat ma’lumotlar manbadan olinadi.

Hisobotni ishlab chiqarish va aks ettirishdan tashqari, CRVS komponentlari olingan ma’lumotlarni PDF, HTML, XLS, RTF va DOC formatlariga eksport qilishni qo‘llab-quvvatlaydi. Bu holda, dastur fayl tizimi katalogda bevosita zarur hisobot fayllarni yaratish mumkin. Bu hisobot fayllarini yaratishning ommaviy rejimi uchun juda qulay, foydalanuvchi keyinchalik natijalarni manfaatdor tomonlarga uzatish uchun ularni oldindan ko‘rmasdan hisobotlar guruhini yaratish kerak.

CRVS ning afzalliklari va kamchiliklarini qisqacha eslatib o‘tamiz. Shubhasiz afzalligi. net Framework va ham yaqin integratsiya bo‘ladi VS.NET, qaysi ilova tuzuvchi rivojlanish vositalari bilan hisobot yaratish imkonini beradi. CRVS komponentlariga asoslangan yechimlar, shuningdek, bu xususiyat hisobotlar va Microsoft .NET. arxitekturasida joylashganligi sababli, murakkab va interaktiv hisobotlarni amalga oshirish qiyin emas, chunki ularni interaktiv nazorat elementlari kabi "kublar" CRVSda mavjud.

Ushbu mahsulot Microsoft texnologiyalari oldingi avlod (u Win32 va C++yordamida ishlab chiqilgan) bo‘yicha butlovchi yadroda yashirin amalga oshirishga mo‘ljallangan, uning ishlab chiqaruvchi hisoblanadi. Dinamik kutubxonalar ko‘rsatkichlarni o‘rnatish va dasturiy juda oddiy eksport hisobotlarni yordamida amalga oshiriladi. Shu bilan birga, yadro ishlaydigan bunday platforma .net va an‘anaviy xotira algoritm dinamik yig‘ish, raqobat kabi nostandart infratuzilma muammolari olib kelishi mumkin. Bu muammoni hal qilish yo‘lida harakat qilib, ishlab chiqaruvchi 1.1 versiyasi qismi sifatida erkin foydalanishga berilgan va .net platforma bosqichma-bosqich o‘tishni boshlagan VS.NET 2003). Bu o‘zgartirish Crystal hisobotlar komponentlarini foydalanadi va dastur bilan faqat. net ma’lumotlarni ekspert qilib beruvchi tomonidan tarqatish jarayonini soddalashtirish kerak. CRVSning platformaga to‘liq o‘tishi kutilmoqda.

CRVS komponentlarining arxitekturasi yakuniy hisobot ishlab chiqarish jarayonlarini ko‘p darajali taqsimlashning ssenariysini tashkil etishga imkon berishiga qaramasdan, ushbu mahsulot butun hisobotni boshqarish siklini tashkil qilish uchun mos emas. MSRS platformasi bu muammoni hal qilish uchun mo‘ljallangan va tayyor ekspert fayllarni hosil qilishga qaratilgan.

Testlash va instruksiya yozish usullari. Dasturiy ta‘minot testi-nuqsonlarni aniqlash va mahsulot sifatini yaxshilash maqsadida dasturiy ta‘minot va tegishli hujjatlarni tahlil qilish jarayonidir.

Dasturiy ta‘minot ishlab chiqish o‘n yillar davomida, turli xil yo‘llar bilan sinov va sifat kafolati orqali juda yaqinlashib qilindi. Uning bir necha asosiy "sinov davrlari" bor.

O‘tgan asrning 50-60-yillarida sinov jarayoni keng rasmiylashtirilgan, bevosita dasturiy ta‘minot ishlab chiqish jarayonidan ajralgan va "matematiklashtirilgan" edi. Aslida, test ko‘proq disk raskadrovka dasturlari kabi bo‘lgan. "Mukammal testing4"-deb atalmish

bir tushuncha bor ediki, barcha mumkin bo‘lgan kiritish ma’lumotlar bilan kodni ijrosi barcha mumkin bo‘lgan yo‘llarini tekshirish amalga oshirilardi. Kiritish ma’lumotlar soni juda katta, chunki, mukammal sinov mumkin emas, va bu yondashuv bilan u hujjatlardagi muammolarni topish qiyin bo‘lgan.

70 yillar sinov ikki asosiy g‘oyalar asosida bo‘lgan: birinchi muayyan belgilangan sharoitlarda (ijobiy testing) dastur faoliyatini isbotlash jarayoni sifatida qabul qilindi va keyin-qat‘iy qarama-qarshi: dastur muayyan belgilangan sharoitlarda (salbiy testing) inoperabl ekanligini isbotlash jarayoni sifatida qaraladi. Nafaqat bu ichki ziddiyat vaqt davomida g‘oyib bo‘lmaydi, lekin bugungi kunda ko‘p mualliflar juda to‘g‘ri sinov ikki to‘ldiruvchi maqsad bo‘lgan deb hisoblaydi. U aniqlangan muammolarga ko‘z yumishga ruxsat bermaydi.

80-yillarda, dasturiy ta‘minot ishlab chiqish sinov joyda asosiy o‘zgarish yuz berdi: loyiha yaratish yakuniy bosqichlarida biri, test butun rivojlanish sikli davomida tatbiq etildi (dasturiy lifecycle). Bundan tashqari, oldindan va ularning yuzaga keladigan vaziyatlarni oldini olish uchun emas, balki faqat ko‘p hollarda ruxsat berilgan "dasturiy ta‘minot ishlab chiqish modellari" iterativ ortib dasturiy ta‘minot ishlab chiqish modeli tavsifini ko‘rish maqsadga muvofiq.

90-yillarda, testlash jarayoni butun dasturiy ta‘minot ishlab chiqish sikli o‘z ichiga oladi va mavjud sinov hollarda va sinov muhitlar qo‘llab-quvvatlash, rejalashtirish, loyihalash, yaratish va sinov ishlarni amalga oshirish jarayonlarini ta‘sir "sifat kafolati" deb nomlangan yanada keng qamrovli jarayoni kabi sinov o‘tish bor edi. Sinov sifat jihatidan yangi darajaga etdi, bu tabiiy ravishda uslubiyotning yanada rivojlanishiga, sinov jarayonini boshqarish uchun juda kuchli vositalar va hozirgi avlodlariga juda o‘xshash bo‘lgan sinovlarni avtomatlashtirish vositalarini paydo bo‘lishiga olib keldi.

Testlashning boshida har qanday mutaxassis (tester ham bundan mustasno emas) ijrochi va talaba. Sinov holatlari va nuqson hisobotlari qanday ekanligini yaxshi tushunish, talablarni o‘qiy olish, bir nechta vositalardan foydalanish va jamoada yaxshi ishtirok etish kifoya. Asta-sekin sinaluvchi loyihani ishlab chiqishning barcha bosqichlariga kirish boshlanadi, bu esa o‘z navbatida ularni yanada to‘laroq anglatadi, ulardan nafaqat faol foydalana boshlaydi, balki loyiha hujjatlarini ishlab chiqib, yanada mas‘uliyatli qarorlar qabul qila boshlaydi.

Testlashning asosiy maqsadini majoziy ma‘noda ifodalasangiz, shunday bo‘ladi: loyiha hozirgi vaqtda nimaga muhtojligini, loyiha bu

zarurni o‘z o‘lchovida qabul qiladimi, yo‘qmi, qanday qilib vaziyatni yaxshiroq o‘zgartirish kerakligini tushuntirib berish. Loyiha menejerining maqsadiga o‘xshaydi, to‘g‘rimi? Sogi-hidori. hu Muayyan rivojlanish darajasidan boshlab, IT mutaxassislari, faqat texnik ko‘nikmalar to‘plamlarida va bu ko‘nikmalarni qo‘llashning asosiy sohasida farq qiladi.

Sinovchi sifatida ishlashni muvaffaqiyatli boshlash uchun qanday texnik ko‘nikmalar kerak?

1. Chet tillar ko‘nikmalari. Ha, bu texnik bo‘lmagan mahorat. Buni aksioma deb hisoblashingiz mumkin.

2. .Haqiqiy rivojlangan foydalanuvchi darajasida ishonchli kompyuter qobiliyatlari va bu sohada doimo rivojlanish kuzatib borish.

3. Dasturlashtirish. Bu tartibda birinchi navbatda har qanday IT shaxs va dasturiy ta‘minot tester hayotini osonlashtiradi. Dasturlash bilimisiz test qilish mumkinmi? Ha, mumkin. Bu, albatta, yaxshi amalga oshirilishi mumkin.

4. Ma’lumotlar bazalari va SQL tili. Bu yerda testlovchi ham tor mutaxassislar darajasida malakaga ega bo‘lishi talab etilmaydi, lekin eng ko‘p tarqalgan DBMlar bilan ishlashning minimal ko‘nikmalari va oddiy so‘rovlarni yozish qobiliyati majburiy deb hisoblanishi mumkin.

5. Tarmoqlar va operatsion tizimlar qanday ishlashi haqida tushuncha. Hech bo‘lmasa, muammoni tashxislash va iloji bo‘lsa, uni o‘zingiz hal qilish imkonini beruvchi minimal darajada.

6. Veb-ilovalar va mobil ilovalar tamoyillari haqida tushuncha. Shu kunlarda deyarli hamma narsa bunday ilovalar shaklida yoziladi va tegishli texnologiyalarni tushunish samarali sinov uchun majburiy bo‘ladi.

Xulosa sifatida testlovchiga tezda ajoyib mutaxassis bo‘lish imkonini beruvchi shaxsiy sifatlarni ham qayd etamiz:

1) mas‘uliyat va ishlashning ortishi;

2) yaxshi muloqot qobiliyatlari, o‘z fikrlarini aniq, tez, aniq ifodalash qobiliyati;

3) sabr-toqat, qat‘iyat, tafsilotga e‘tibor, kuzatish;

4) yaxshi mavhum va analitik fikrlash;

5) nostandart tajribalar o‘rnatish qobiliyati, tadqiqot uchun imkoniyat.

Bu fazilatlarga teng egalik qiladigan odamni topish qiyin, ammo o‘z-o‘zini rivojlantirish uchun ma’lum bir mos yozuvlar nuqtasiga ega bo‘lish har doim foydalidir.

NAZARIY BILIMLARNI TEKSHIRISH UCHUN SAVOLLAR.

1. Inson va mashinaning o‘zaro muloqotini ta‘minlash nima deb aytiladi.
2. Inson - kompyuter o‘zaro hamkorligi ilmiy ko‘p yo‘nalishlarini sanab bering.
3. Foydalanuvchi interfeysi nima?
4. Real vaqtda amalga oshiriladigan va muayyan vazifani birgalikda hal etishga qaratilgan axborot almashuvi nima deb aytiladi.
5. Kiritish vositalari yordamida hosil qilinadigan kirish xabarlarining texnik vositalarini sanib bering.
6. Interfeys yaratishning asosiy tamoyillari sanab bering.
7. Ma’lumotlarni eksport qilish vositalari va komponentalari deganda nima nazarda tutiladi.
8. Turli xil xususiyatlarga ega bo‘lgan ma’lumotlarni birlashtiruvchi vosita – bu nima.
9. Crystal Reports for Microsoft Visual Studio .NET tizimi haqida nimalarni bilasiz.
10. Surish modelining vazifasi
11. Tortish modelining vazifasi nimadan iborat.
12. Testlash deganda nima tushuniladi.
13. Testlashning rivojlanish bosqichlarini sanab bering.
14. Ikki yo‘nalishli testlash jarayoni qaysi bosqichga tegishli.
15. Zamonaviy testlashning qulayliklari va kamchiliklari.
16. Sinovchi xodim bo‘lishning xususiyatlari sanab bering?

Foydalanilgan adabiyotlar

1. Nazirov Sh.A., Qobulov R.V., Bobojanov M.R., Raxmanov Q.S. C va C++ tili. “Voris- nashriyot” MCHJ, Toshkent 2013, 488 b.
2. Horstmann, Cay S. C++ for everyone/Cay S. Horstmann. Printed in the United States of America - 2nd ed. 2010. – P. 562.
3. Horton I.-Beginning Visual C++ 2012/ I.Horton. Published simultaneously in Canada. –2012. –P. 988.
4. Алгоритмы. Справочник с примерами на C, C++, Java и Python. Джордж Хайнемаи, Гэри Поллис, Стэнли Селков. Москва «Альфа-Книга», 2017. –С254.
5. ASP.NET MVC Framework с примерами на C# для профессионалов. Стивен Сандерсон. Москва 2010. –С. 562.
6. Самоучитель Microsoft Visual Studio C++ и MFC. Сидорина Т.Л. Санкт-Петербург. «БХВ-Петербург» 2009. –С. 600.
7. Си Шарп: Создание приложения для Windows. В.В. Лабор. Минск, Харвест, 2003.
8. Полный справочник по C++. Герберт Шилдт. Москва, 2006.
9. Современные технологии разработки программ, взаимодействующих с базами данных. С.А. Минеев, Ю.Е. Чуманкин. Нежний Новгород, 2018.
10. C/C++. Программирование на языке высокого уровня. Т.А.Павловская. Санкт-Петербург, 2003.
11. Самоучитель C++. Герберт Шилдт. Санкт-Петербург. 2003.
12. C/C++ и MS Visual C++ 2010 для начинающих. Санкт-Петербург. 2011.
13. Программирование под Windows в среде Visual C++ 2005. М.В. Свиркин, А.С. Чуркин. Санкт-Петербург. 2008.
14. Простая графическая задача на Microsoft Visual C++ с использованием библиотеки MFC. Е.П. Дербакова. Москва 2015.
15. Программирование в Visual C++ с использованием библиотеки MFC. В.В. Васильчиков. Ярославль 2006.
16. MS Visual C++ 2010 в среде .NET. В.В. Зиборов. Санкт-Петербург, 2012
17. Herbert Schildt, Java the complete reference ninth edition, oracle press, 2014.
18. Алексеев А.П. Информатика. 2001. М., СОЛОН, 2001, 364 с.
19. Арипов М.М. Internet ва электрон почта асослари. –Т.: Университет, 2000. -132 б.

20. Страуструп. Б. Язык программирования С++. Специальное издание.-М.:ООО «Бином-Пресс», 2006.-1104 с.
21. Вирт Н. Алгоритмы + структуры данных = программа.-М.:Мир,1985.-405с.
22. Говорухин В., Цибулин В. Компьютер в математическом исследовании. Maple, Matlab, Latex и др. Учебный курс. Питер. 2001. – 624 с.
23. Информатика. Базовой курс. Учебник для Вузов., Санк-Петербург, 2001. под редакцией С.В.Симоновича.
24. Павловская Т.С. Щупак Ю.С. С/С++. Структурное программирование. Практикум.-СПб.: Питер,2002-240с
25. Павловская Т.С. Щупак Ю.С. С++. Объектно-ориентированное программирование. Практикум.- СПб.: Питер,2005-265с
26. Подбельский В.В. Язык СИ++.- М.; Финансы и статистика-2003 562с.
27. Романчик В.С., Люлькин А.Е. Программирование в С++ BUILDER. Учебное пособие. Мн.: БГУ, 2007. –126 с.
28. Юлдашев У.Ю., Боқиев Р.Р., Зокирова Ф.М. Информатика. – Т.: Ғ.Ғулом номидаги нашриёт-матбаа ижодий уйи, 2002. - 237 б.
29. Abduqodirov A.A., Hayitov A.G'., Shodiyev R.R. Axborot texnologiyalari //Akademik litsey va kasb-hunar kollejlari uchun darslik. -T.:O‘qituvchi, 2003. –152 b.
30. Axmedov A.B., Tayloqov N.I. Informatika. Akademik litsey va kasb hunar kollejlari uchun darslik.-T.:O‘zbekiston, 2001. 2 – nashri. – 272 b.
31. Mo‘minov B.B. Microsoft Excel bo‘yicha amaliy mashg‘ulotlar va ularni bajarish tartibi. Uslubiy qo‘llanma. – Buxoro: Ziyo Rizograf, 2008. – 72 b.
32. Mo‘minov B.B. Microsoft Excel dasturi bo‘yicha laboratoriya ishlar to‘plami. Uslubiy qo‘llanma.–Buxoro:Ziyo Rizograf, 2008. –92 b.
33. Mo‘minov B.B. Pedagogik dasturiy ta‘minot yaratish texnologiyasi. Monografiya. –Buxoro, Buxoro nariyoti, 2010. -168 b.
34. Xaldjigitov A.A., Madraximov Sh.F., Adamboev U.E .Informatika va programmalsh. O‘quv qo‘llanma. O‘zMU, 2005 yil, 145 bet.
35. http://aut.researchgateway.ac.nz/index.jsp Auckland University of Technology digital library

36. http://dastur.uz Kompyuter Dasturlari va Kompyuterda Dasturlashga oid Forum, Habar va Yangiliklar
37. http://google.com Google qidiruv tizimi
38. http://informatika.sch880.ru Основы логики
39. http://lex.uz/ O‘zbekiston Respublikasi qonun hujjatlari ma’lumotlari milliy bazasi
40. http://ru.wikipedia.org/wiki Википедия — свободная энциклопедия
41. http://uza.uz O‘zbekiton milliy axborot agentligi
42. http://winedt.com winedt rasmiy portali
43. http://www.aci.uz O‘zbekiston aloqa va axborotlashtirish agentligi
44. http://www.cplusplus.com cplusplus.com
45. http://www.edu.uz O‘zbekiston Respublikasi Oliy va O‘rta maxsus ta‘lim vazirligi
46. http://www.exponenta.ru Образовательный математический сайт Exponenta.ru
47. http://www.intuit.ru Интернет-Университет Информационных Технологий
48. http://ziyonet.uz Oliy va O‘rta maxsus ta‘lim vazirligi huzuridagi axborot ta‘lim portali

MUNDARIJA

MO‘MINOV Bahodir Boltaeyich
e-mail: Mbbahodir@gmail.com

D A S T U R L A S H I I

D A R S L I K

O‘zbek tilida

Toshkent – «NIHOL PRINT» OK – 2021

Muharrir:	A.Tog‘ayev
Tex. muharrir:	F.Tog‘ayeva
Musavvir:	B.Esanov
Musahhiha:	O.Muxammadiyeva
Kompyuterda sahifalovchi:	G.Tog‘ayeva

9323

№ 7439-765f-47f1-7ea1-a683-4648-1314.
Bosishga ruxsat etildi: . Bichimi 60x841 /16.
Shartli bosma tabog‘i 38,75. Nashr bosma tabog‘i 38,5.
Adadi 100. Buyurtma № 5.

«Nihol print» Ok da chop etildi.
Toshkent sh., M. Ashrafiy ko‘chasi, 99/101.