O‘ZBEKISTON RESPUBLIKASI
OLIY VA O‘RTA MAXSUS TA’LIM VAZIRLIGI
MIRZO ULUG‘BEK NOMIDAGI
O‘ZBEKISTON MILLIY UNIVERSITETI

A. M. Polatov

ALGORITMLAR VA C++ TILIDA DASTURLASH ASOSLARI

(*O‘quv qo‘llanma*)

Toshkent
“Universitet”
2017

UDK
KBK

Polatov A.M. Algoritmlar va C++ tilida dasturlash asoslari. Toshkent. “Universitet” - 2017. 123 bet.
KBK

Annotatsiya. O‘quv qo‘llanmada “algoritm” tushunchasining, xossalari, turlari va taqdim etish usullari, C++ algoritmik tilining asosiy tushuncha va atamalari keltirilgan, ma’lumotlarni qayta ishlash bilan bog‘liq masalalar ko‘rib chiqilgan, hisoblash jarayonini tashkillashtirish aniq misollarda ko‘rsatilgan. Chiziqli, tarmoqlanuvchi va takrorlanuvchi jarayonlarni, shuningdek, massiv elementlarini qayta ishlash bilan bog‘liq algoritmlarni amalga oshirish uchun zarur dasturlash, hamda funksiyalar yaratish va ulardan foydalanish uslubiyatiga oid mavzular yoritilgan.

Аннотация. В учебном пособии на конкретных примерах приведены понятия, свойства, типы и способы представления алгоритмов, а также основные понятия и термины алгоритмического языка C++. Рассмотрены задачи связанные с обработкой данных и на конкретных примерах приведены способы организации вычислительных процессов. Приведены реализация линейных, алгоритмов ветвления и циклических процессов, реализация алгоритмов и программ, связанных с обработкой элементов массивов, а также создание и методика применения функций.

Annotation. In the tutorial on specific examples are given concepts, properties, types and methods of representing algorithms, as well as the basic concepts and terms of the algorithmic language C ++. The problems connected with data processing are considered and the ways of organization of computing processes are given on specific examples. The implementation of linear, branching algorithms and cyclic processes, the implementation of algorithms and programs related to the processing of array elements, as well as the creation and methods of application of functions are presented.

Tuzuvchi: professor v.b. Polatov Asxad Muxamedjanovich
Ma’sul muharrir: O‘zMU professori M.M.Aripov
Taqrizchilar: TATU kafedra mudiri Sh.A.Sadullaeva,
O‘zMU professori N.A.Ignatyev

O‘quv qo‘llanmani nashr etishga O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta’lim vazirligining 2017 yil 24- avgustdagi 603 sonli buyrug‘iga asosan ruxsat berilgan (ro‘yxatga olish raqami 603-157)

ISBN

MUNDARIJA

Dastur yaratish - anglamoq demakdir
Kristin Nyugard

KIRISH

“Dasturlash asoslari” fanining bosh maqsadi talabalarga qo‘yilgan tatbiqiy masalani anglash, echish algoritmini ishlab chiqish va dasturiy ta’minotini yaratish asoslarini o‘rgatishdir [1]. Shu maqsadda mazkur o‘quv qo‘llanmada masala echish matematik modellari, usullari va algoritmlar yaratish asoslari hamda kompyuterda masalalarni echish uchun C++ dasturlash tilining tayanch tushunchalari keltirilgan.

O‘quv qo‘llanmada kompyuter vositasida dasturlashga kirishning nazariy asosi bo‘lgan “algoritm” tushunchasiga alohida e’tibor qaratilgan. Mazkur qo’llfnmada algoritmlarni tavsiflash va keyinchalik kompyuter vositasida bajarish uchun zarur bo‘lgan bir qator matematik tushunchalar – chiziqli, tarmoqlanuvchi va takrorlanuvchi jarayonlar, yordamchi algoritm, massiv, indeks, rekursiya, funksiya va parametr kiritilgan bo‘lib, turli sohalarga oid masalalarni echish algoritmlari va dasturlariga oid mavzular yoritilgan.

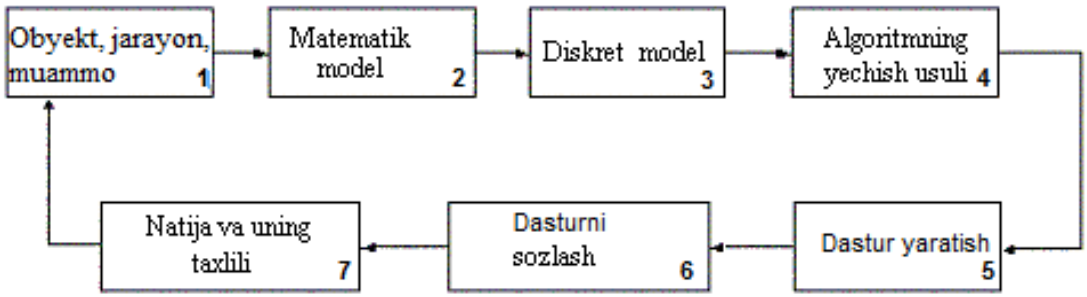
Dasturlash tili – tadqiq qilinadigan jarayonga mos keladigan matematik modeldagi munosabatni yechish usuli uchun tuzilgan algoritmni kompyuterda amalga oshirish vositasidir. Shu sababli o‘quv qo‘llanmada tatbiqiy masalalar yyechishning algoritmik asoslarini o‘rganish, kompyuterda berilganlar va buyruqlarni tasvirlash, shuningdek C++ tilida dasturlash asoslariga alohida e’tibor berilgan. Shu bilimlarga tayangan holda talaba kompyuter vositasida tatbiqiy masalalarning dasturiy ta’minotini yaratishda zarur bo‘lgan bilimlarga ega bo‘ladi.

Mazkur o‘quv qo‘llanma asosida ko‘plab matematik masalalarni kompyuter vositasida yechish algoritmlari, dasturlash usullari va tahlil qilish natijasida har bir bakalavr olgan bilim va ko‘nikmalarini ishlab chiqarishda, ilmiy-tadqiqot faoliyatida, shuningdek, talim tizimida samarali foydalanish imkoniyatiga ega boladi.

1-BOB. ALGORITMLAR

1.1. Amaliy masalalarni kompyuterda yechish jarayoni

Kundalik faoliyatimizda biz turli xil muammolar, vazifalarga duch kelamiz. Biz uchun ular talab yoki savol shaklida bo‘ladi. Masalan, «Kvadrat tenglamani yechish», «Dengizda nechta tomchi suv bor?», «Ikki karra ikki necha bo‘ladi?» kabi. Masalani yechish uchun kerakli amallarni bajarish, ma’lum bir ketma-ketlikda qator harakatlarni amalga oshirish darkor. Aynan shu harakatlar ketma-ketligini to‘liq tasavvur etishimiz va tasvirlab berishimiz kerak. Turli muammo, masala yoki jarayonlarni o‘rganishni kompyuter yordamida amalga oshirish uchun, birinchi navbatda, tadqiq qilinayotgan masala, jarayon - obyektning matematik ifodasi, ya’ni matematik modelini qurish kerak bo‘ladi. Matematik model real obyektning tasavvurimizdagi mavhum ko‘rinishi bo‘lib, u matematik belgilar va ba’zi-bir qonun-qoidalar bilan ifodalangan bo‘ladi. Qurilayotgan obyektning matematik modelini yaratish juda murakkab jarayon bo‘lib, o‘rganilayotgan obyektga bog‘liq ravishda turli soha mutaxassislarining ishtiroki talab etiladi. Umuman, biror masalani kompyuter yordamida yechishni quyidagi bosqichlarga ajratish mumkin (1.1.-rasm).



1.1-rasm. Hisoblash eksperimentining sxemasi

Misol sifatida, kosmik kemani Erdan Zuhro sayyorasiga eng qulay trayektoriya bo‘yicha uchirish masalasini hal qilish talab qilingan bo‘lsin.

Birinchi navbatda, qo‘yilgan masala turli soha mutaxassislari tomonidan atroflicha o‘rganilishi va bu jarayonni ifodalaydigan eng muhim bo‘lgan asosiy

parametrlarni aniqlash kerak bo‘ladi (1.1.- rasm, 1-blok). Masalan, fizik-astronom muhandis tomonidan masala qo‘yilishining o‘rinli ekanligi, ya’ni sayyoralar orasidagi masofa va atmosfera qatlamlarining ta’siri, yerning tortish kuchini engib o‘tish va kemaning og‘irligi, zarur bo‘lgan yoqilg‘ining optimal miqdori va kosmik kemani qurishda qanday materiallardan foydalanish zarurligi, inson sog‘lig‘iga ta’siri va sarflanadigan vaqt va yana turli tuman ta’sirlarni hisobga olgan holda shu masalaning matematik modelini tuzish zarur bo‘ladi. Zikr etilgan ta’sirlarni va fizika qonunlarini hisobga olgan holda bu masalani ifodalaydigan muayyan differensial yoki boshqa ko‘rinishdagi modellovchi tenglama hosil qilish mumkin bo‘ladi. Balki bu masalani bir nechta alohida masalalarga ajratib o‘rganish maqsadga muvofiqdir. Bu matematik modelni o‘rganish asosida mazkur masalani ijobiy hal qilish yoki hozirgi zamon sivilizatsiyasi bu masalani yechishga qodir emas degan xulosaga ham kelish mumkin. Ushbu fikrlar, yuqorida keltirilgan sxemaning 2-blokiga mos keladi.

Faraz qilaylik, matematik model ishlab chiqildi. Endi uni kompyuter vositasida yechish masalasi paydo bo‘ladi. Bizga ma’lumki, kompyuter faqat 0 va 1 diskret qiymatlar va ular ustida arifmetik va mantiqiy amallarni bajara oladi xolos. SHuning uchun matematik modelga mos diskret modelni qurish zaruriyati tug‘iladi (3-blok). Odatda, matematik modellarga mos keluvchi diskret modellar ko‘p noma’lumli murakkab chiziqsiz algebraik tenglamalar sistemasi (chekli ayirmali tenglamalar-sxemalar) ko‘rinishida bo‘ladi (4-blok). Endi, hosil bo‘lgan diskret modelni sonli yechish usuli – algoritmini yaratish zarur bo‘ladi. Algoritm esa tuziladigan dastur uchun asos bo‘ladi (5-blok). Odatda, tuzilgan dasturni ishchi holatga keltirish uchun uning xato va kamchiliklarini tuzatish – sozlash zarur bo‘ladi (6-blok). Olingan sonli natijalar hali dasturning to‘g‘ri ishlayotganligiga kafolatini bermaydi. SHuning uchun olingan natijalarni masalaning mohiyatidan kelib chiqqan holda tahlil qilish kerak bo‘ladi (7-blok). Agar olingan natija o‘rganilayotgan jarayonni ifodalay olmasa, masalani 1.1- rasmdagi sxema asosida qaytadan ko‘rib chiqish va zarur bo‘lgan joylarda o‘zgartirishlar kiritish kerak bo‘ladi. Bu jarayon to kutilgan ijobiy yoki salbiy natija olinguncha davom ettiriladi

va bu takrorlanuvchi jarayon hisoblash eksperimenti deb ataladi. Odatda, hisoblash eksperimenti deganda, soddaroq holda, model, algoritm va dastur uchligini (triadasini) tushunish mumkin (1.2-rasm).

1.2 – rasm. Hisoblash eksperimenti uchligi

1.2. “Algoritm” tushunchasi

Yuqorida qayd etganimizdek, qo‘yilgan biror masalani kompyuterda yechish uchun, avval uning matematik modelini, keyin esa yechish algoritmi va dasturini tuzish kerak bo‘ladi. Ushbu uchlikda algoritm bloki muhim ahamiyatga ega. Endi algoritm tushunchasining ta’rifi va xossalarini bayon qilamiz. Masala echimini cheklangan qadamlar natijasida hosil qiladigan, oldindan tayinlangan va aniq belgilangan qoidalar yoki buyruqlar ketma-ketligi *algoritm* deyiladi. Soddaroq qilib aytganda, algoritm bu - oldimizga qo‘yilgan masalani yechish uchun zarur bo‘lgan amallar ketma-ketligidir. Algoritm tuzish - bu dasturlashdir, algoritmni tuzuvchilar esa dasturchilardir.

Masalan, $ax^2+bx+c=0$ kvadrat tenglamani yechish uchun quyidagi amallar ketma-ketligi zarur bo‘ladi:

1. *a, b, c* koeffitsiyentlar berilgan bo‘lsin.
2. Berilgan *a, b, c* koeffitsiyentlar yordamida discriminant.

   $D=b^2-4ac$ hisoblanadi.
3. $D>0$ bo‘lsa $X_{1/2}=\left(-b\pm\sqrt{D}\right)/(2*a)$- hisoblanadi.
4. $D<0$ bo‘lsa, haqiqiy echimi yo‘q.

Misol sifatida berilgan *a, b, s* tomonlari bo‘yicha uchburchakning yuzasini Gerron formulasi bo‘yicha hisoblash masalasini ko‘rib chiqaylik.

1. *a, b, c* uchburchak tomonlari uzunliklari.
2. $r=(a+b+c)/2$ - perimetrning yarmi hisoblansin.
3. $T=p(r-a)(r-b)(r-c)$ – hisoblansin.
4. $S=\sqrt{T}$ hisoblansin.

Yuqoridagi misollardan ko‘rinib turibdiki, algoritmning har bir qadamida bajariladigan amallar tushunarli va aniq tarzda ifodalangan hamda chekli sondagi amallar bajarilgandan keyin aniq natija olish mumkin.

Zikr etilgan, tushinarlilik, aniqlilik, cheklilik va natijaviylik tushunchalari algoritmning asosiy xossalarini tashkil etadi. Bu tushunchalar keyingi paragraflarda alohida ko‘rib o‘tiladi.

“Algoritm” so‘zi va tushunchasi IX asrda yashab ijod etgan buyuk alloma Muhammad al-Xorazmiy nomi bilan uzviy bog‘liq. Algoritm so‘zi Al-Xorazmiy nomini Evropa olimlari tomonidan buzib talaffuz qilinishidan yuzaga kelgan. Al-Xorazmiy birinchi bo‘lib o‘nlik canoq sistemasining tamoyillarini va undagi to‘rtta amalni bajarish qoidalarini asoslab bergan.

1.3. Algoritmning asosiy xossalari

Algoritmning 5 ta asosiy xossasi bor.

1. ***Diskretlilik (Cheklilik).*** Bu xossaning mazmuni algoritmlarni doimo chekli qadamlardan iborat qilib bo‘laklash imkoniyati mavjudligida. Ya’ni uni chekli sondagi oddiy ko‘rsatmalar ketma-ketligi shaklida ifodalash mumkin. Agar kuzatilayotgan jarayonni chekli qadamlardan iborat qilib qo‘llay olmasak, uni algoritm deb bo‘lmaydi.

2. ***Tushunarlilik.*** Biz kundalik hayotimizda berilgan algoritmlar bilan ishlayotgan elektron soatlar, mashinalar, dastgohlar, kompyuterlar, turli avtomatik va mexanik qurilmalarni kuzatamiz.

Ijrochiga tavsiya etilayotgan ko‘rsatmalar uning uchun tushinarli mazmunda bo‘lishi shart, aks holda, ijrochi oddiygina amalni ham bajara olmaydi. Bundan tashqari, ijrochi har qanday amalni bajara olmasligi ham mumkin.

Har bir ijrochining bajarishi mumkin bo‘lgan ko‘rsatmalar yoki buyruqlar majmuasi mavjud, u ijrochining ko‘rsatmalar tizimi (sistemasi) deyiladi. Demak, ijrochi uchun berilayotgan har bir ko‘rsatma ijrochining ko‘rsatmalar tizimiga mansub bo‘lishi lozim.

Ko‘rsatmalarni ijrochining ko‘rsatmalar tizimiga tegishli bo‘ladigan qilib ifodalay olishimiz muhim ahamiyatga ega. Masalan, quyi sinfning a’lochi o‘quvchisi "son kvadratga oshirilsin" degan ko‘rsatmani tushunmasligi natijasida bajara olmaydi, lekin "son o‘zini o‘ziga ko‘paytirilsin" shaklidagi ko‘rsatmani bemalol bajaradi, chunki u ko‘rsatma mazmunidan ko‘paytirish amalini bajarish kerakligini anglaydi.

3. ***Aniqlik.*** Ijrochiga berilayotgan ko‘rsatmalar aniq va mazmunli bo‘lishi zarur. Chunki ko‘rsatmadagi noaniqliklar mo‘ljaldagi maqsadga erishishga olib kelmaydi. Inson uchun tushunarli bo‘lgan "3-4 marta silkitilsin", "5-10 daqiqa qizdirilsin", "1-2 qoshiq solinsin", "tenglamalardan biri echilsin" kabi noaniq ko‘rsatmalar kompyuterni qiyin ahvolga solib qo‘yadi.

Bundan tashqari, ko‘rsatmalarning qaysi ketma-ketlikda bajarilishi ham muhim ahamiyatga ega. Demak, ko‘rsatmalar aniq berilishi va faqat algoritmda ko‘rsatilgan tartibda bajarilishi shart ekan.

4. ***Ommaviylik.*** Har bir algoritm mazmuniga ko‘ra bir turdagi masalalarning barchasi uchun ham o‘rinli bo‘lishi kerak. Ya’ni masaladagi boshlang‘ich ma’lumotlar qanday bo‘lishidan qat’i nazar algoritm shu xildagi har qanday masalani yechishga yaroqli bo‘lishi kerak. Masalan, ikki oddiy kasrning umumiy maxrajini topish algoritmi, kasrlarni turlicha o‘zgartirib bersangiz ham ularning umumiy maxrajlarini aniqlab beraveradi. Yoki uchburchakning yuzini topish algoritmi, uchburchakning qanday bo‘lishidan qat’i nazar, uning yuzasini hisoblab beraveradi.

5. ***Natijaviylik.*** Har bir algoritm chekli sondagi qadamlardan so‘ng, albatta, natija berishi shart. Bajariladigan amallar ko‘p bo‘lsa ham baribir natijaga olib kelishi kerak. Chekli qadamdan so‘ng qo‘yilgan masalae chimga ega emasligini aniqlash ham natija hisoblanadi. Agar ko‘rilayotgan jarayon cheksiz davom etib natija bermasa, uni algoritm deb atay olmaymiz.

1.4. Algoritmni tasvirlash usullari

Yuqorida ko‘rilgan misollarda, odatda, biz masalani yechish algoritmini so‘zlar va matematik formulalar orqali ifodaladik. Lekin algoritm boshqa ko‘rinishlarda ham berilishi mumkin. Biz endi algoritmlarning eng ko‘p uchraydigan turlari bilan tanishamiz.

1. *Algoritmning so‘zlar orqali ifodalanishi.* Bu usulda ijrochi uchun beriladigan har bir ko‘rsatma jumlalar, so‘zlar orqali buyruq shaklida beriladi.

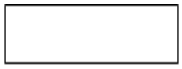
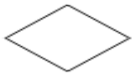
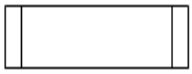
2. *Algoritmning formulalar bilan ifodalanish* usulidan matematika, fizika, kimyo kabi aniq fanlardagi formulalarni o‘rganishda foydalaniladi. Bu usul ba’zan analitik ifodalash deyiladi.

3. *Algoritmlarning maxsus geometrik shakllar yordamida ifodalanishida* masala yechish jarayoni aniq va ravon tasvirlanadi va bu ko‘rinish blok-sxema deyiladi.

4. *Algoritmning jadval ko‘rinishda berilishi*. Algoritmning bunday ifodasidan ham ko‘p foydalanamiz. Masalan, maktabda qo‘llanib kelinayotgan to‘rt xonali matematik jadvallar yoki turli xil lotereyalar jadvali. Funksiyalarning grafiklarini chizishda ham algoritmlarning qiymatlari jadvali ko‘rinishlaridan foydalanamiz. Bu kabi jadvallardan foydalanish algoritmlari sodda bo‘lgani tufayli ularni o‘zlashtirib olish oson.

Yuqorida ko‘rilgan algoritmlarni tasvirlash usullarining asosiy maqsadi, qo‘yilgan masalani yechish uchun zarur bo‘lgan amallar ketma-ketligining eng qulay holatini aniqlash va shu bilan inson tomonidan dastur yozishni yanada osonlashtirishdan iborat. Aslida, dastur ham algoritmning boshqa bir ko‘rinishi bo‘lib, u insonning kompyuter bilan muloqotini qulayroq amalga oshirish uchun mo‘ljallangan.

Blok-sxemalarni tuzishda foydalaniladigan asosiy sodda geometrik figuralar quyidagilardan iborat:

Figura shakli	Vazifasi
	oval, algoritmning boshlanishi yoki tugallanishini belgilaydi
	parallelogramm, ma'lumotlarni kiritish yoki chop etishni belgilaydi
	to‘g‘ri to‘rtburchak, amal bajarish jarayonini belgilaydi
	romb, shart bajarilishi tekshirilishini belgilaydi
	yordamchi algoritmga murojaatni belgilaydi
	oltiburchak, takrorlash operatorini ifodalashni belgilaydi
	strelka, amallar bajarilish ketma-ketligini aniqlaydi
=> (n)	so‘zlar orqali ifodalangan algoritmda *n* - chi amalga o‘tishni ko‘rsatadi

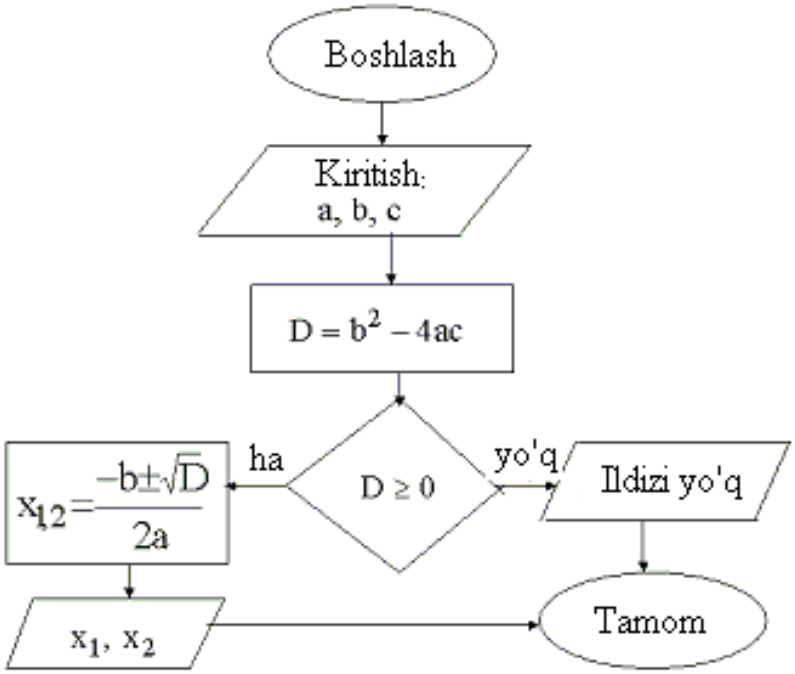
Blok-sxemalar bilan ishlashni yaxshi o‘zlashtirib olish zarur, chunki bu usul algoritmlarni ifodalashning eng qulay usullaridan biri bo‘lib, dastur tuzishni osonlashtiradi, dasturlash qobiliyatini mustahkamlaydi. Algoritmik tillarda blok - sxemaning asosiy strukturalariga maxsus operatorlar mos keladi.

Shuni aytish kerakni, blok-sxemalardagi yozuvlar odatdagi yozuvlardan katta farq qilmaydi.

Masalan, misol sifatida 1.2 punktda keltirilgan $ax^2+bx+c=0$ kvadrat tenglamaning haqiqiy echimlarini hisoblash uchun quyidagi amallar ketma-ketligi zarur bo‘ladi:

1) berilganlarni kiritish (*a, b, c*);
2) $D=b^2-4ac$ diskriminantni hisoblash;
3) *agar* $D>0$ bo‘lsa $X_{1/2} = \left(-b \pm \sqrt{D}\right)/(2*a)$;
4) *aks holda,* $D<0$ bo‘lsa haqiqiy echimi yo‘q.

Bu amallar ketma-ketligiga mos algoritm 1.3-rasmda blok-sxema ko‘rinishida keltirilgan.

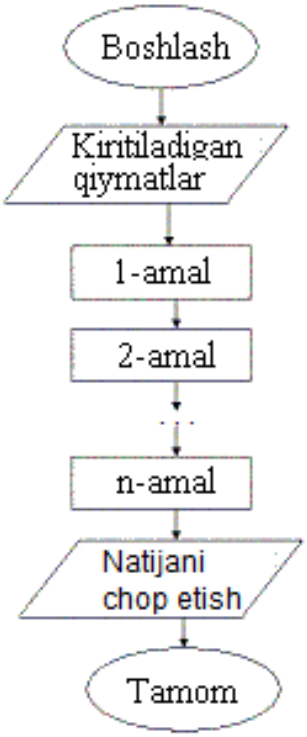


1.3-rasm. Kvadrat tenglamani yechish blok-sxemasi

1.5. Chiziqli algoritmlar

Har qanday murakkab algoritmni ham uch asosiy struktura yordamida tasvirlash mumkin. Bular ketma-ketlik, ayri va takrorlash strukturalaridir. Ushbu strukturalar asosida chiziqli, tarmoqlanuvchi va takrorlanuvchi hisoblash jarayonlarining algoritmlarini tuzish mumkin. Umuman olganda, algoritmlarni shartli ravishda quyidagi turlarga ajratish mumkin:

- *chiziqli* algoritmlar;
- *tarmoqlanuvchi* algoritmlar;
- *takrorlanuvchi* algoritmlar;
- *ichma-ich joylashgan takrorlanuvchi* algoritmlar;
- *rekurrent* algoritmlar;
- *takrorlanishlar soni oldindan no'malum* algoritmlar;
- *ketma-ket yaqinlashuvchi* algoritmlar.

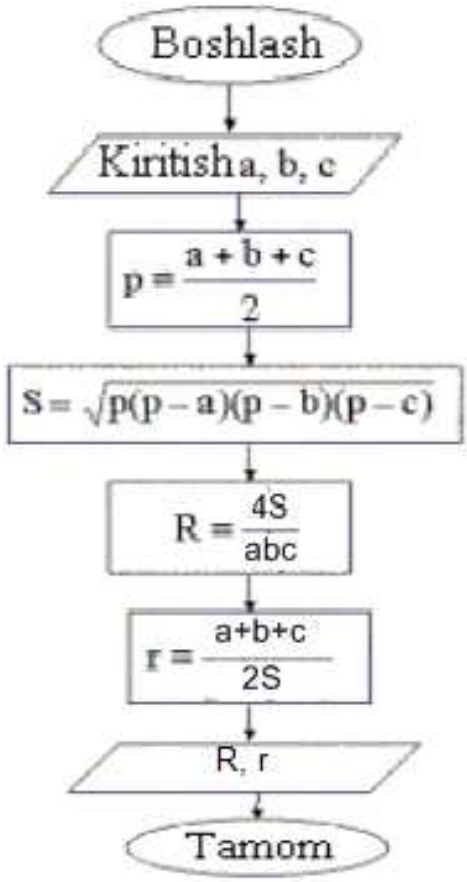


1.4-rasm. Chiziqli algoritmlar blok–sxemasining umumiy tuzilishi

Faqat ketma-ket bajariladigan amallardan tashkil topgan algoritmlarga - *chiziqli* algoritmlar deyiladi. Bunday algoritmni ifodalash uchun ketma-ketlik strukturasi ishlatiladi. Strukturada bajariladigan amal mos keluvchi shakl bilan ko‘rsatiladi. Chiziqli algoritmlar blok-sxemasining umumiy tuzilishi 1.4-rasmda keltirilgan.

1-*misol.* Uchburchak tomonlarining uzunligi bilan berilgan. Uchburchakka ichki *r* va tashqi *R* chizilgan aylanalar radiuslarini hisoblang.

Ichki chizilgan aylana radiusi $r = (a+b+c)/2S$, tashqi chizilgan aylana radiusi $R=\frac{4S}{abc}$ formulalar orqali hisoblanadi. Bu yerda *S* - uchburchakning yuzi, *a, b, c* – uchburchak tomonlarining uzunliklari. Masala echimining blok-sxemasi 1.5-rasmda keltirilgan.



1.5-rasm. Uchburchakka ichki va tashqi chizilgan aylanalar radiuslarini hisoblash blok-sxemasi

2-*misol*. Quyida keltirilgan munosabatni hisoblash algoritmini ko‘rib chiqaylik. Jarayon amallarni ketma-ket bajarilishidan iborat.

$$z = x^2 + \sqrt{|\sin(x+y)|},$$

bu yerda

$x = \cos(a - b)$, $y = \ln(a^2 - x^2)$, $a = 0.7$, $b = 2.1$.

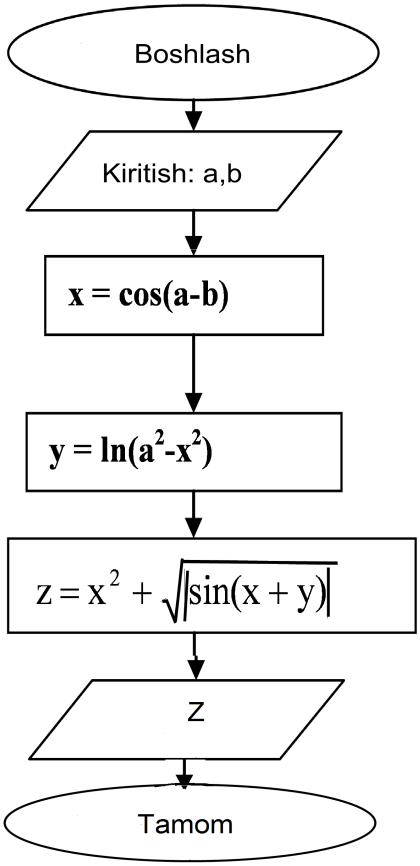
Bunda:

a, b - aniq qiymatga ega bo‘lgan boshlang‘ich ma’lumotlar;

x, y - oraliq ma’lumotlar;

z - natija.

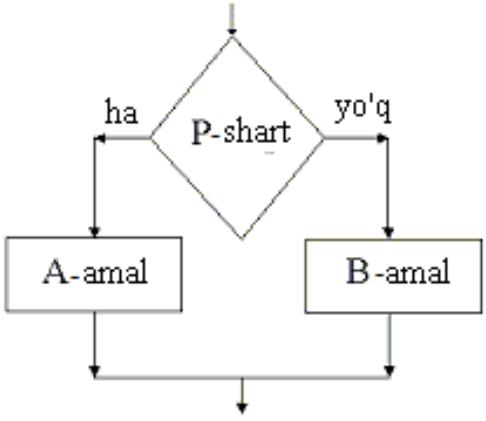
Masalani yechish jarayoni chiziqli hisoblanadi, chunki boshlang‘ich ma’lumotlar kiritilgach, munosabatlarning qiymati dasturda joylashgan tartibda hisoblanadi, ya’ni dastlab *x*, so‘ng - *y* qiymati va nihoyat *z* natija hisoblanadi. Mazkur jarayonning blok-sxemasi 1.6-rasmda keltirilgan.



1.6-rasm. Hisoblash blok-sxemasi

1.6. Tarmoqlanuvchi algoritmlar

Agar hisoblash jarayoni biror-bir berilgan shartning bajarilishiga qarab turli tarmoqlar bo‘yicha davom ettirilsa va hisoblash jarayonida har bir tarmoq faqat bir marta bajarilsa, bunday hisoblash jarayonlari tarmoqlanuvchi algoritmlar deyiladi. Tarmoqlanuvchi algoritmlarni tasvirlash uchun “ayri” tuzilmasi ishlatiladi. Tarmoqlanuvchi tuzilmasi berilgan shartning bajarilishiga qarab ko‘rsatilgan tarmoqdan faqat bittasining bajarilishi ta’minlanadi (1.7-rasm).



1.7-rasm. Tarmoqlanishning umumiy ko‘rinishi

Berilgan R-*shart* romb figurasi ichida tasvirlanadi. Agar *shart* bajarilsa, "ha" tarmoq bo‘yicha A-amal, aks holda (*shart* bajarilmasa) "yo‘q" tarmoq bo‘yicha V-amal bajariladi.

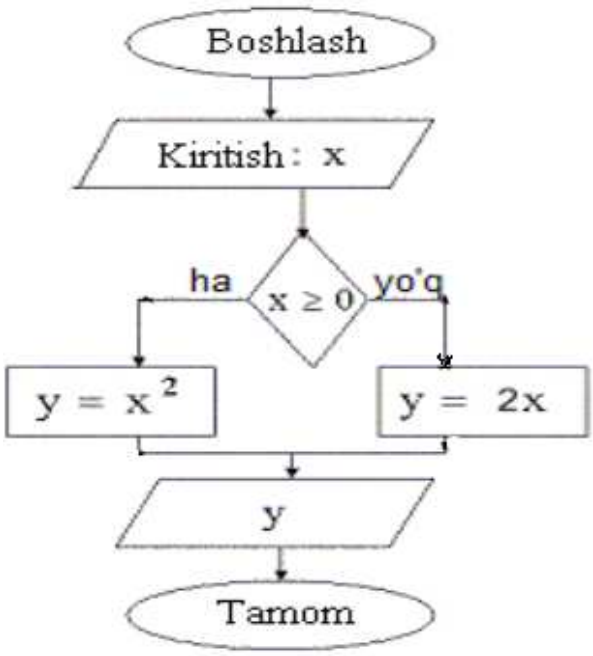
1-*misol*. Tarmoqlanuvchi algoritmga *misol* sifatida quyidagi sodda *masala* keltiriladi:

$$y = \begin{cases} x^2, & agar \quad x \geq 0 \\ 2x, & aks \quad holda. \end{cases}$$

Natijaviy qiymat *y* berilgan *x* ning qiytmatiga bog‘liq holda bo‘ladi: agar x≥0 shart *rost* bo‘lsa, tarmoq bo‘yicha $y = x^2$ munosabatning qiymati, aks holda, $y = 2*x$ munosabatning qiymati hisoblanadi. Bu masala bajarilishining so‘z bilan ifodalangan algoritmi quyidagicha:

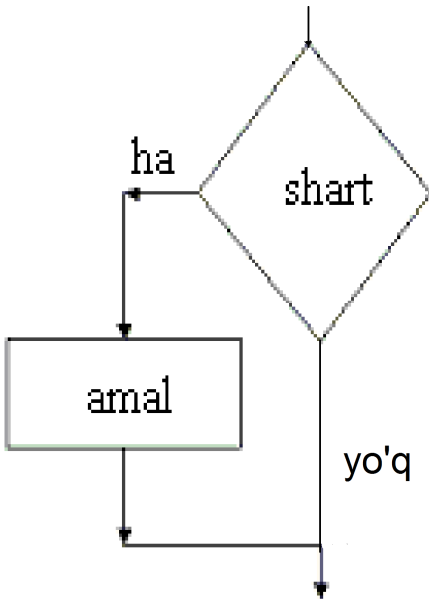
agar ($x \geq 0$) shart bajarilsa, u holda $u=x^2$, aks holda $u=2*x$.

Masala echimining blok-sxemasi 1.8-rasmda keltirilgan.



1.8-rasm. Interval ko‘rinishidagi funksiya qiymatini hisoblash blok-sxemasi

Ko‘pgina masalalarni yechishda, *shart* asosida tarmoqlanuvchi algoritmning ikki tarmog‘idan biri, ya’ni «rost» yoki «yolg‘on»ning bajarilishi etarli bo‘ladi. Bu holat tarmoqlanuvchi algoritmning xususiy holi sifatida qisqartirilgan strukturasi deb atash mumkin. Qisqartirilgan struktura blok-sxemasi quyidagi ko‘rinishga ega (1.9-rasm).



1.9-rasm. Qisqartirilgan strukturaning umumiy ko‘rinishi

2-*misol*. Berilgan *x, u, z* sonlari ichidan eng kattasini aniqlang. Ushbu masalaga mos matematik modelni quyidagicha tasvirlash mumkin:

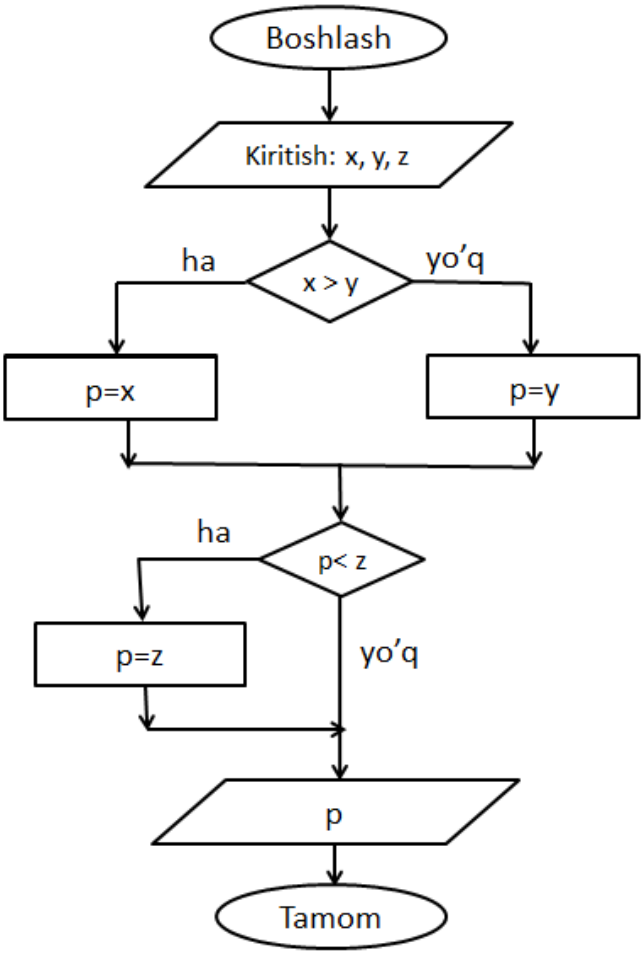
$$p = \max\{x, y, z\}.$$

Berilgan *x, y, z* sonlardan eng kattasi *p* deb belgilangan. So‘zlar orqali ifodalangan algoritm asosida masala echimini quydagicha tasvirlash mumkin:

1) kiritish (*x, y, z*);
2) agar ($x > u$) bo‘lsa, u holda $p = x$, *aks holda* $p = u$;
3) agar ($r < z$) bo‘lsa, u holda $p = z$;
4) muhrlash (*r*).

Keltirilgan algoritmga mos blok-sxema 1.10-rasmda tasvirlangan.

Bu algoritmda, avva, *x* va *y* o‘zaro solishtiriladi, katta qiymatligi ega r ga yuklanadi. So‘ngra *x* va *y* larning kattasi deb aniqlangan *r* va *z* o‘zaro solishtiriladi. Agar $r < z$ sharti bajarilsa, u holda eng katta qiymat $p = z$ deb olinadi, aks holda boshqarish navbatdagi amalga uzatiladi. Natijada *p* da uchta qiymatdan eng kattasi aniqlanadi.



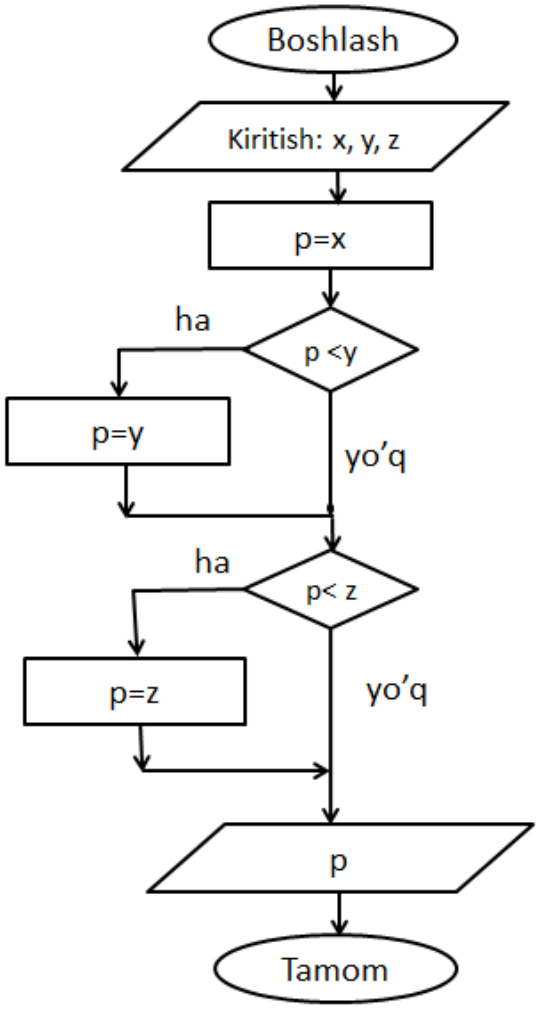
1.10-rasm. Berilgan *x, y, z* sonlar ichidan eng kattasini topish blok-sxemasi

Ushbu masalani yechish algoritmining yana bir usulini ko‘rib chiqamiz.

1) kiritish (*x, y, z*);
2) $p = x$;
3) agar ($p < y$) bo‘lsa, u holda $p = y$;
4) agar ($p < z$) bo‘lsa, u holda $p = z$;
5) muhrlash (*r*).

Bu algoritmga mos blok-sxema 1.11-rasmda tasvirlangan.

Bu usulga asosan, avvalo sonlar ichida birinchisi eng kattasi deb faraz qilinadi, ya’ni $p = x$. So‘ngra har bir qadamda navbatdagi son – *r* ning qiymati bilan solishtiriladi va shart bajarilsa, u eng kattasi deb qabul qilinadi. Bu algoritmning afzalligi shundaki, uning asosida uchta va undan ko‘p sonlar ichidan eng kattasini (kichigini) topishning qulay imkoniyati mavjud.



1.11-rasm. Hisoblash blok-sxemasi

3-*misol*. Quyidagi ifoda bilan berilgan munosabatni hisoblang [2, 52 b.].

$$Y = \begin{cases} b - x, & agar \quad x > 0, \\ x + a, & agar \quad x < 0, \\ a + b, & aks \ \ holda \ . \end{cases}$$

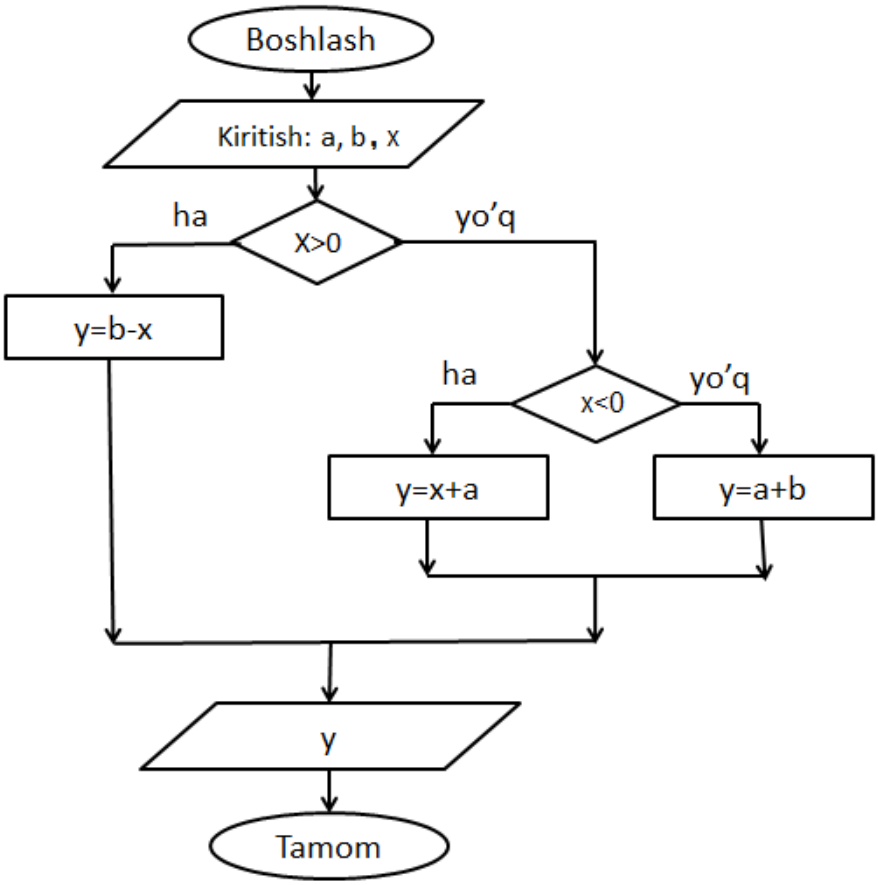
Bu misol natija x ning qiymatiga bog‘liq shart bilan berilgan va masala quyidagicha so‘zlar orqali ifodalangan algoritm asosida aniqlanadi:

agar $x > 0$ bo‘lsa, u holda $u = b - x$ bo‘ladi, aks holda;

agar $x < 0$ bo‘lsa, u holda $u = x + a$, aks holda $u = a + b$.

Avvalo, birinchi shart tekshiriladi va agar u bajarilsa, $y = b - x$ amal bajariladi, aks holda $Y = \begin{cases} x + a, & agar \quad x < 0, \\ a + b, & aks \ \ holda \ . \end{cases}$ munosabat hisoblanadi.

Bu fikrlar quyidagi blok-sxemada o‘z aksini topgan (1.12-rasm).



1.12-rasm. Hisoblash blok-sxemasi

1.7. Takrorlanuvchi algoritmlar

Agar biror masalani yechish uchun zarur bo‘lgan amallar ketma-ketligining ma’lum bir qismi biror parametrga bog‘liq holda ko‘p marta qayta bajarilsa, bunday jarayon takrorlanuvchi algoritm deyiladi. Takrorlanuvchi algoritmlarga misol sifatida odatda qatorlarning yig‘indisi yoki ko‘paytmasini hisoblash jarayonlarini qarash mumkin.

1-misol. Birdan n gacha bo‘lgan natural sonlarning yig‘indisini hisoblash algoritmini tuzaylik. Masalaning matematik modeli quyidagicha:

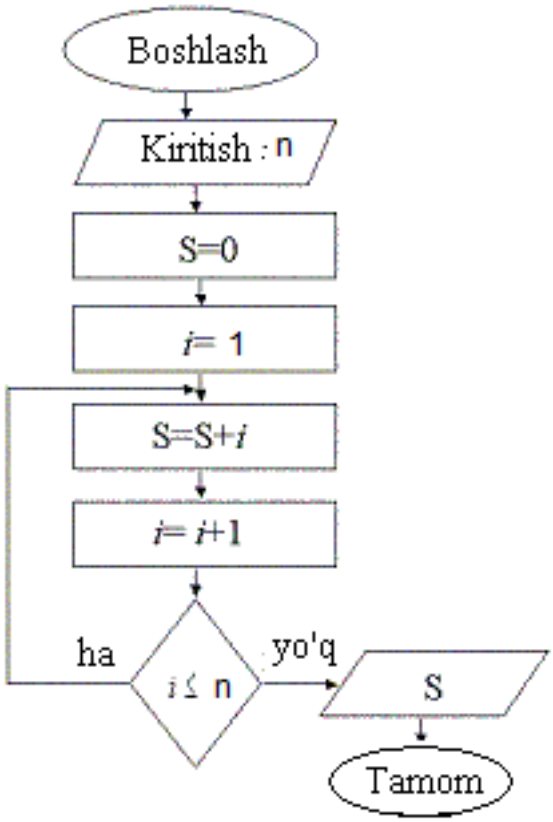
$$S = 1 + 2 + 3 + ... + n = \sum_{i=1}^{n} i$$

Bu yig‘indini hisoblash uchun, avvalo, natiga boshlangich qiymatini S=0 va indeksning boshlangich qiymatini $i = 1$ deb olamiz va joriy amallar $S = S + i$ va $i = i + 1$ hisoblanadi. Bu erda birinchi va ikkinchi qadamlar uchun yig‘indi

hisoblandi va keyingi qadamda *i* parametr yana bittaga orttiriladi va navbatdagi qiymat avvalgi yig‘indi *S* ga qo‘shiladi. Mazkur jarayon shu tartibda indeksning joriy qiymati $i \leq n$ sharti bajarilmaguncha davom ettiriladi va natijada, izlangan yig‘indiga ega bo‘lamiz. Ushbu fikrlarni quyidagi so‘zlar orqali ifodalangan algoritm bilan ifodalash mumkin:

1) *kiritish (n);*
2) *S=0 - natijaning boshlang‘ich qiymati;*
3) *i=1 - indeksning boshlang‘ich qiymati;*
4) *S=S+i - natijaning joriy qiymatini hisoblang;*
5) *i=i+1- indeksning joriy qiymatini hisoblang;*
6) *agar* $(i \leq n)$ *sharti tekshirilsin va u bajarilsa =>(4) ;*
7) *muhrlash (S).*

Bu jarayonga mos keladigan blok-sxemaning ko‘rinishi 1.13-rasmda tasvirlangan.



1.13-rasm. 1 dan n-gacha bo‘lgan sonlar yig‘indisini hisoblash blok-sxemasi

Yuqorida keltirilgan so‘zlar asosida ifodalangan algoritm va blok-sxemadan ko‘rinib turibdiki, amallar ketma-ketligining ma’lum qismi parametr i ga nisbatan n marta takrorlanadi.

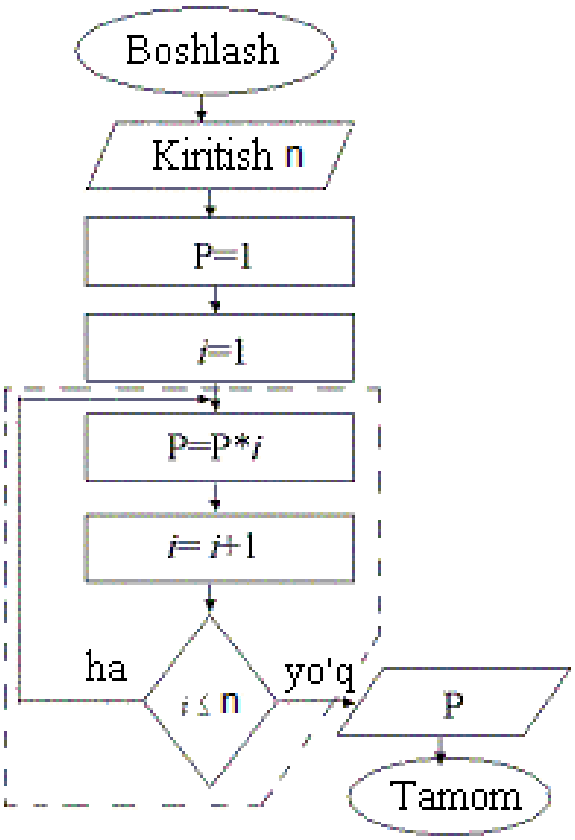
2-misol. Quyidagi ko‘paytmani hisoblash algoritmi va blok-sxemasini tuzaylik: $P = 1 \cdot 2 \cdot 3 \cdots n = n!$ (odatda, 1 dan n gacha bo‘lgan natural sonlarning ko‘paytmasi $n!$ ko‘rinishda belgilanadi va “en” faktorial deb ataladi: P=n!

(jarayonning matematik modeli: $P = \prod_{i=1}^{n} i$) [2, 57-58 b.].

Ko‘paytmani hosil qilish algoritmi ham yig‘indini hosil qilish algoritmiga o‘xshash, faqat ko‘paytmani hosil qilish uchun, avvalo, $i=1$ da $P=1$ deb olinadi, so‘ngra $i = i + 1$ da $P = P * i$ munosabatlar hisoblanadi. Keyingi qadamda i parametrning qiymati yana bittaga orttiriladi va navbatdagi qiymat avvalgi hosil bo‘lgan ko‘paytma - P ga ko‘paytiriladi. Bu jarayon shu tartibda to $i \leq n$ sharti bajarilmaguncha davom ettiriladi va natijaviy ko‘paytmaning qiymatiga ega bo‘lamiz. Quyidagi so‘zlar orqali ifodalangan algoritmda bu fikrlar o‘z aksini topgan:

1) kiritish *(n);*
2) $P=1$ - natijaning boshlang‘ich qiymati;
3) $i=1$ - indeksning boshlang‘ich qiymati;
4) $P=P*i$ - natijaning joriy qiymatini hisoblash;
5) $i=i+1$ - indeksning joriy qiymatini hisoblash;
6) *agar (i <= n)* shart bajarilsa, u holda => (4);
7) muhrlash *(P).*

Bu algoritmga mos blok-sxema 1.14-rasmda keltirilgan.



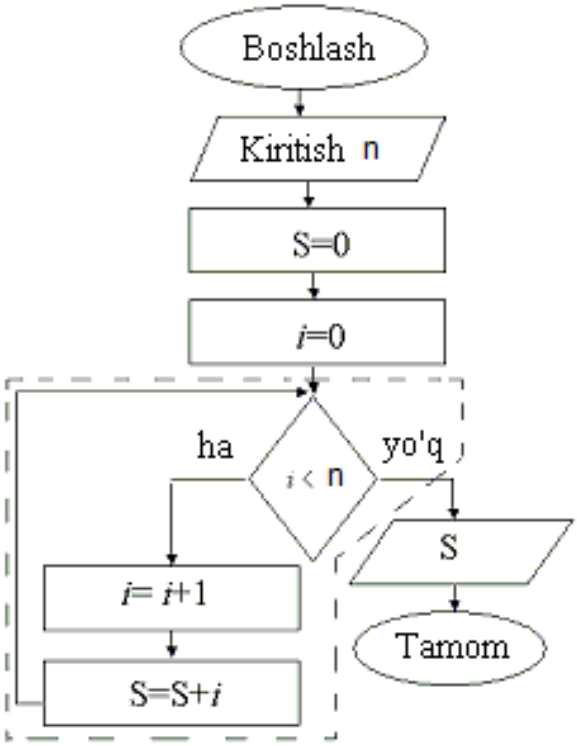
1.14-rasm. 1 dan n gacha bo‘lgan sonlar ko‘paytmasini hisoblash blok-sxemasi

Yuqorida ko‘rilgan yig‘indi va ko‘paytmalarning blok-sxemalaridagi takrorlanuvchi qismlariga (punktir chiziqlar ichiga olingan) 1.14 rasmdagi sharti keyin berilgan takrorlanuvchi struktura mos kelishini ko‘rish mumkin.

3-misol. Yuqoridagi blok-sxemalarda shartni oldin tekshiriladigan holatda chizish mumkin edi. Masalan, $S = \sum_{i=1}^{n} i$ yig‘indini xisoblash algoritmi tadqiqi keltiriladi. Bu masalani yechishda algoritmning takrorlanuvchi qismiga quyidagi sharti oldin berilgan takrorlanuvchi strukturaning mos kelishini ko‘rish mumkin.

1) kiritish (n);
2) S=0;
3) i = 0;
4) agar (i > n) =>(8);
5) i = i + 1;
6) S = S + i ;
7) shartsiz o‘tish=>(4);
8) muhrlash (S).

Bu algoritmga mos blok-sxema 1.15- rasmda keltirilgan.



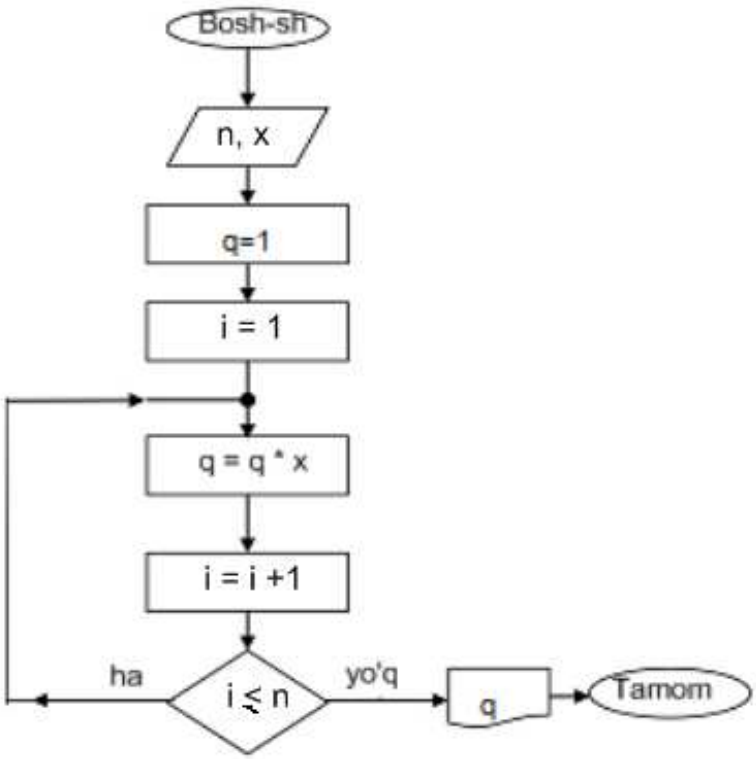
1.15-rasm. 1 dan *n* gacha bo‘lgan sonlar yig‘indisini hisoblash blok-sxemasi

4-*misol.* Haqiqiy x sonining n chi darajasini hisoblash masalasi ko‘riladi. Uning matematik modeli: $q = x^n$ ko‘rinishga ega.

Takrorlanuvchi jarayonni tashkil etish quyidagidan farqli, yuqoridagilar bilan bir xil:

- ko‘paytirish jarayoni uchun boshlang‘ich qiymat berilishi: `q = 1` ;
- joriy natijani hisoblash: `q = q *` x ifoda bo‘yicha amalga oshiriladi.

Shunday qilib, x ning n chi darajasini hisoblash uchun takrorlanuvchi jarayonni tashkil etish blok-sxemasi 1.16-rasmda keltirilgan.



1.16-rasm. Hisoblash blok-sxemasi

5-*misol*. Quyidagi munosabatni hisoblash kerak bo‘lsin [2, 55-56 b.]:

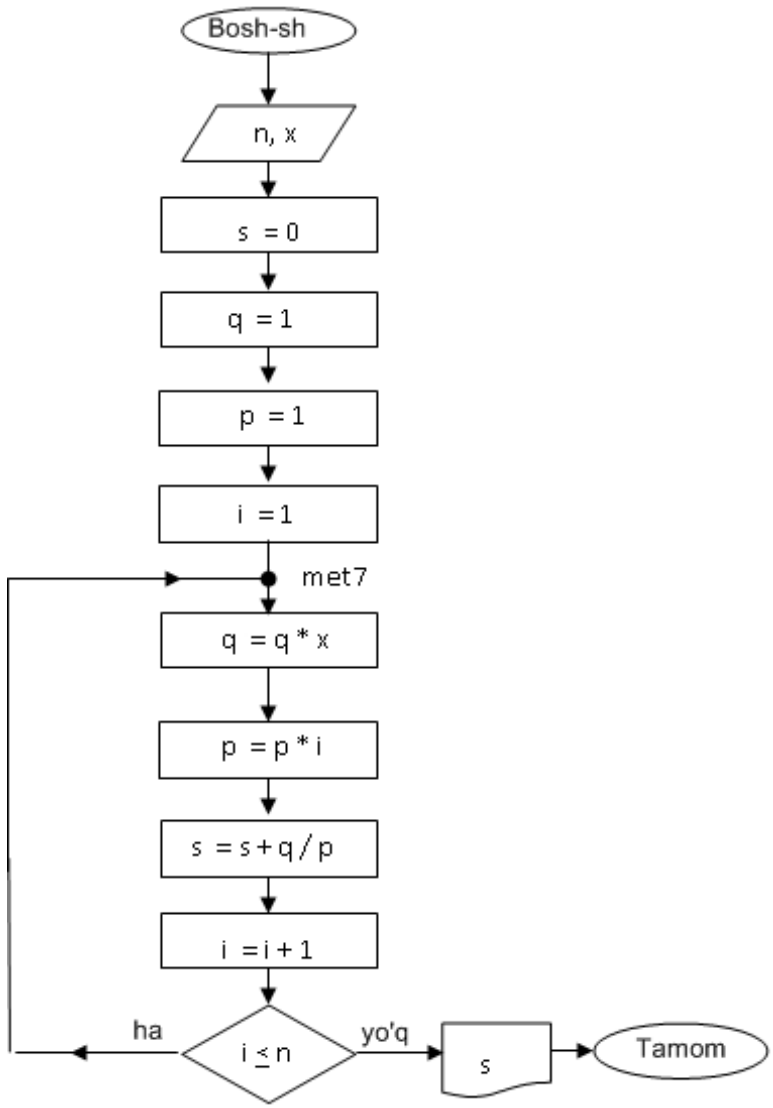
$$S = \sum_{i=1}^{n} \frac{x^{i}}{i!} .$$

Munosabatni ochib, quyidagi ko‘rinishda yozish mumkin:

$s = x^1/1! + x^2/2! + \cdots + x^n/n!$.

Masalani yechish algoritmida boshlang‘ich qiymat sifatida s=0 ni olamiz, chunki ifodada yig‘indi belgisi mavjud. Yig‘indi belgisi ostidagi munosabat kasr sonni anglatadi: suratda - x^i, mahrajda - $i!$. Ularning har biri uchun boshlang‘ich va joriy munosabatlar shakllantiriladi:

	surat	mahraj	natija
boshlang‘ich munosabat	q = 1	p = 1	s=0
joriy munosabat	q = q * x	p = p * i	s = s + q / p



1.17-rasm. Hisoblash blok-sxemasi

Bu jarayonni shakllantirish uchun ***i*** indeks-parametri ishlatiladi.

Indeks-parametrni boshqarish amallari quyidagicha:

a) $i = 1$ – parametrning boshlang‘ich qiymati,

b) $i = i + 1$ – parametrning orttirmasi (orttirma h=1),

c) $i \leq n$ – jarayon yakunlanish sharti.

Bunga muvofiq, masalani yechish blok-sxemasi quyidagi 1.17-rasmdagi ko‘rinishga ega bo‘ladi.

6-*misol*. A={a_i} (i=1, 2, …, n) massiv elementlarining yig‘indisini hisoblash jarayonini aks ettiradigan algoritm yarating.

Masalaning matematik modeli quyidagidan iborat: $S = \sum_{i=1}^{n} a_i$.

Yig‘indini hisoblash uchun S o‘zgaruvchidan foydalanamiz va uning boshlang‘ich qiymati deb $S = 0$ olinadi. So‘ngra indeksning $i = 1$ qiymatidan boshlab, uning $i = i + 1$ orttirmasi bilan to (i <= n) shart bajarilguncha $S = S + a_i$ munosabat qiymati ketma-ket hisoblanadi.

Quyidagi algoritmda jarayon amallari bajarilishi ketma-ketligi keltiriladi:

1) kiritish (n, a_i);
2) $S = 0$,
3) $i = 1$,
4) $S = S + a_i$,
5) $i = i + 1$,
6) agar (i <= n) shart bajarilsa, u holda => (4),
7) muhrlash (S) .

7-misol. Massiv elementlari o‘rta qiymatini hisoblash. Masalaning matematik modeli : $Ð = \frac{1}{n}\sum_{i=1}^{n} a_i$. Yuqoridagi masaladan farqi – elementlar yig‘indisini elementlar soniga bo‘lish amali bilan algoritm to‘ldiriladi, ya’ni:

1) kiritish (n, a_i);
2) S = 0;
3) $i = 1$;
4) S = S + a_i ;
5) $i = i + 1$;
6) agar (i <= n) shart bajarilsa, u xolda => (4);
7) P =S / n ;
8) muhrlash (P) .

8-*misol*. Massiv elementlari qiymatlarining ko‘paytmasini hisoblash algoritmini tuzing. Masalaning matematik modeli quyidagidan iborat: $P = \prod_{i=1}^{n} a_i$.

Hisoblash jarayoni yuqoridagiga o‘xshash bo‘ladi, faqat ko‘paytmaning

boshlang‘ich qiymati R = 1 va joriy amal R = R * a_i bo‘ladi. Bu jarayonning so‘zlar orqali ifodalangan algoritmi quyidagicha:

1) kiritish (n, a_i);
2) R = 1;
3) i = 1;
4) R = R * a_i ;
5) $i = i + 1$;
6) agar (i <= n) shart bajarilsa, u holda => (4)
7) muhrlash (R) .

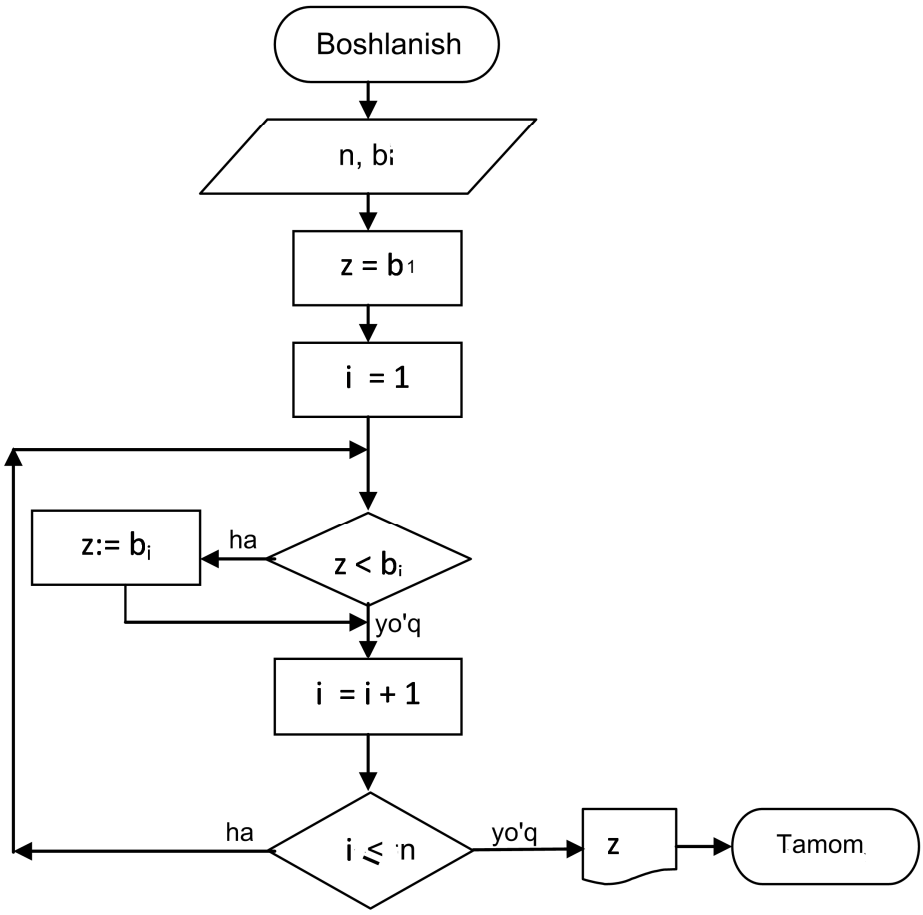
9-*misol*. B={b_i} massiv elementlari maksimal (eng katta) qiymatini aniqlash bilan bog‘liq masala ko‘riladi.

Mazkur masalaning matematik modeli quyidagi ko‘rinishga ega:

$$z = \max_{1 \le i \le m} b_i \quad m = 8.$$

Maksimal elementni aniqlash uchun quyidagi tadbirni amalga oshirish zarur. Avval, massivning birinchi elementi maksimal qiymatga ega deb taxmin qilinadi. So‘ngra taxmin qilingan maksimal element navbatdagi elementlar bilan navbatma-navbat solishtiriladigan takrorlash jarayoni tashkil etiladi. Agar massivning navbatdagi elementi maksimal deb belgilangan elementdan katta bo‘lsa, u holda joriy element maksimal deb belgilanadi. Takrorlashning yakunida o‘zgaruvchining qiymati massivning maksimal elementiga mos keladi.

Massivning maksimal elementini aniqlash algoritmi blok-sxemasi ko‘rinishi 1.18-rasmda keltirilgan.



1.18-rasm. Hisoblash blok-sxemasi

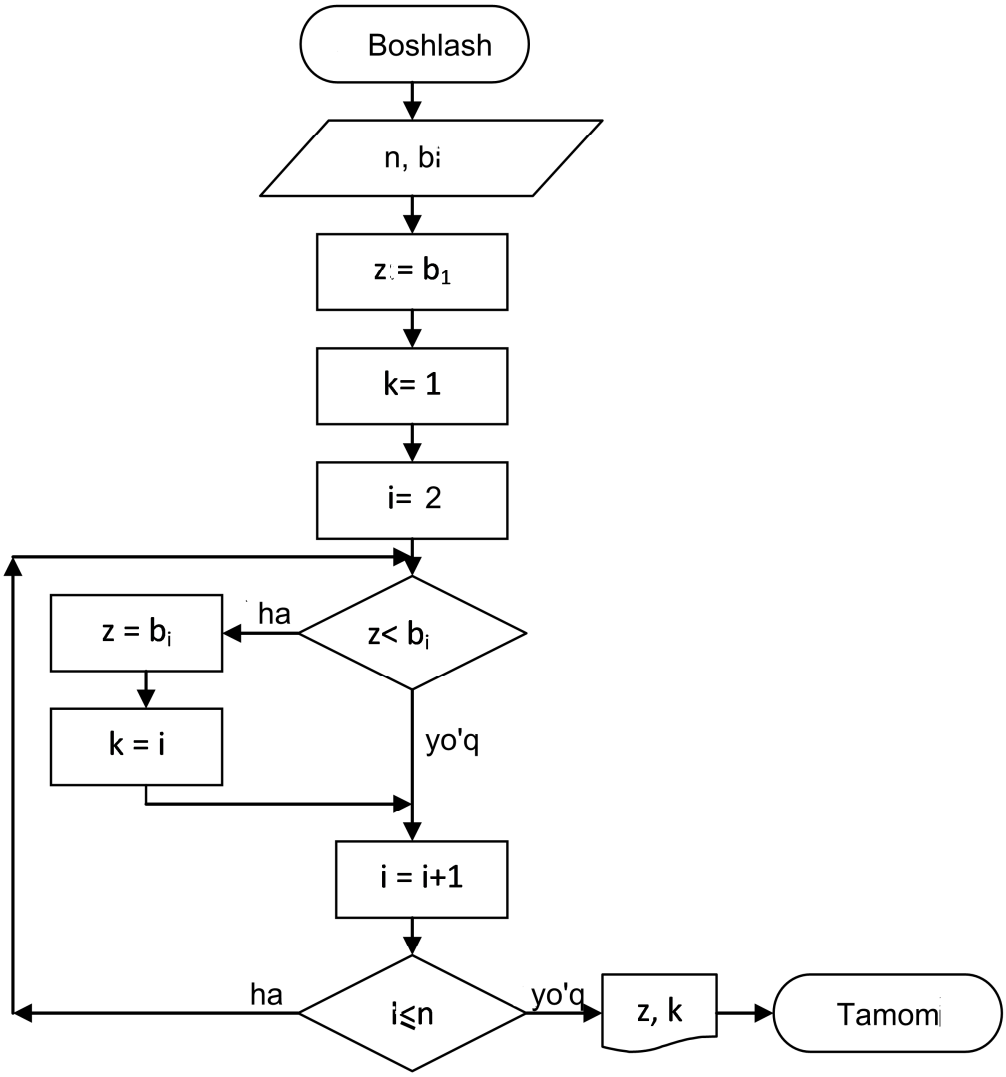
Minimal elementni aniqlash uchun shart ifodasida “**<**” (kichik) belgisini “**>**” (katta) belgiga o‘zgartirishning o‘zi kifoya.

10-misol. Massivning maksimal elementi indeksini, ya’ni u joylashgan o‘rnini aniqlash uchun yuqorida keltirilgan algoritmga boshlang‘ich va joriy elementining indeksini belgilaydigan o‘zgaruvchi qo‘shishning o‘zi kifoya:

1) $k = 1$ (birinchi element maksimal deb taxmin qilanadi);

2) $k = i$ (agar joriy i – chi element taxmin qilingan maksimumdan katta bo‘lsa, u qiymati bo‘yicha barcha elementlardan eng kattasi bo‘ladi).

Qo‘shimchalarni hisobga olgan holda blok-sxema 1.19-rasmda keltirilgan.



1.19-rasm. Hisoblash blok-sxemasi

Algoritmning so‘zlar orqali ifodalangan usulidan foydalanib, amallar ketma-ketligini keltiramiz:

1) kiritish (n, b_i)
2) z = b_1;
3) k = 1;
4) i = 2;
5) agar (z < b_i) shart bajarilsa, u holda { z= b_i; k=i; }
6) $i = i + 1$;
7) agar ($i \leq n$) shart bajarilsa, u holda => (5)
8) muhrlash (S, k).

11-misol. Vektorni vektorga skalyar ko‘paytmasi: s = A*V ni hisoblash masalasi ko‘riladi (vektorlar skalyar ko‘paytmasi).

Bu yerda: A={ a_i }, B={ b_i }, i =1,2, ..., n, c – skalyar.

Jarayonning matematik modeli (hisoblash formulasi) :

$$s = \sum_{i=1}^{n} a_i * b_i = a_1 \times b_1 + a_2 \times b_2 + \cdots + a_n \times b_n \quad ,$$

Bu munosabatni hisoblash - vektorlarning mos elementlari ko‘paytmalari yig‘ishdan iborat.

Algoritmning so‘zlar orqali ifodalangan usulidan foydalanib, amallar ketma-ketligi keltiriladi:

1) kiritish (n, a_i , b_i)
2) S = 0;
3) i = 1;
4) S = S +$a_i * b_i$;
5) i = i + 1;
6) agar (i <= n) shart bajarilsa, u holda => (4);
7) muxrlash (S).

12-misol. A={a_i} (i=1, 2, …, n) massiv elementlari qiymatlari yig‘indisidan eng katta elementi qiymatini ayrish jarayonini akslantiradigan algoritm yarating.
Masalaning matematik modeli quyidagidan iborat:

$$R = \sum_{i=1}^{n} a_i - \max_{1 \le i \le n} a_i .$$

Bu murakkab matematik model uchta nisbatan sodda munosabatlar ketma-ketligi bilan almashtiriladi (dekompozitsiya amali):

$$1)\ S = \sum_{i=1}^{n} a_i \ , \qquad 2)\ P = \max_{1 \le i \le n} a_i , \qquad 3)\ R = S - P.$$

Asosiy algoritmda amallar bajarilishi ketma-ketlikligi keltiriladi, ya’ni:

1) kiritish (n, m, a_i);

2) S = 0;

3) $i = 1$;

4) $S = S + a_i$;

5) $i = i + 1$;

6) agar (i <= n) shart bajarilsa, u holda => (4);

7) $P = a_1$;

8) $i = 2$;

9) agar ($P < a_i$) shart bajarilsa, u holda $P = a_i$;

10) $i = i + 1$;

11) agar (i <= n) shart bajarilsa, u holda => (9) ;

12) R = S – P ;

13) muhrlash (R) .

Yuqoridagi keltirilgan masalani yechish algoritmini ixchamlashtirish mumkin:

1) kiritish (n, m, a_i);

2) $S = a_1$;

3) $P = a_1$;

4) $i = 2$;

5) $S = S + a_i$;

6) agar ($P < a_i$) shart bajarilsa, u holda $P = a_i$;

7) $i = i + 1$;

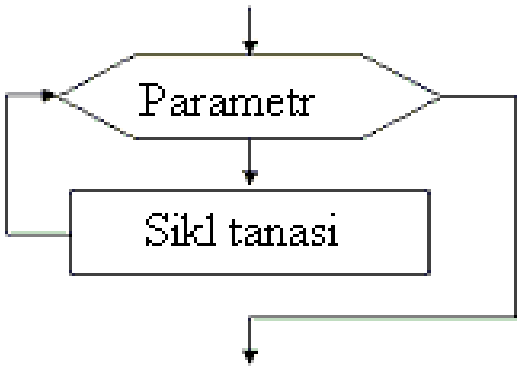
8) agar (i <= n) shart bajarilsa, u holda => (5) ;

9) R = S – P ;

10) muhrlash (R) .

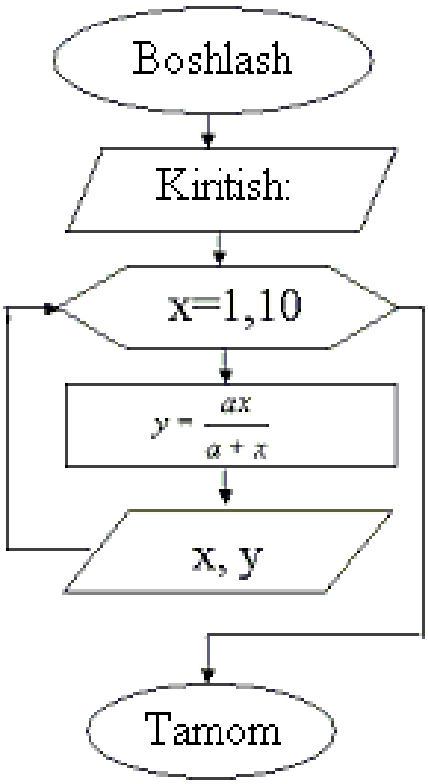
Algoritmda yig‘indi va maksimal qiymat aniqlash jarayonida boshlang‘ich indeks qiymatini tenglashtiriladi ($S = a_1$ va $P = a_1$) va jarayon massivning 2 chi elementini qayta ishlashdan boshlandi. Ya’ni bir takrorlash jarayonida ikkita: massiv element qiymatlari yig‘indisini hisoblash va maksimal qiymatni aniqlash amalga oshiriladi.

Blok-sxemalarning takrorlanuvchi qismlarini quyidagi parametrli takrorlash strukturasi ko‘rinishida ham ifodalash mumkin (1.20-rasm).



1.20-rasm. Parametrli takrorlash operatorining umumiy ko‘rinishi

13-*misol.* Parametrli takrorlash operatoriga masala sifatida berilgan *x=1,2,3,.....10* qiymatlarda $y = \frac{ax}{a+x}$ funksiyasining qiymatini hisoblash blok-sxemasiga keltiriladi (1.21-rasm).



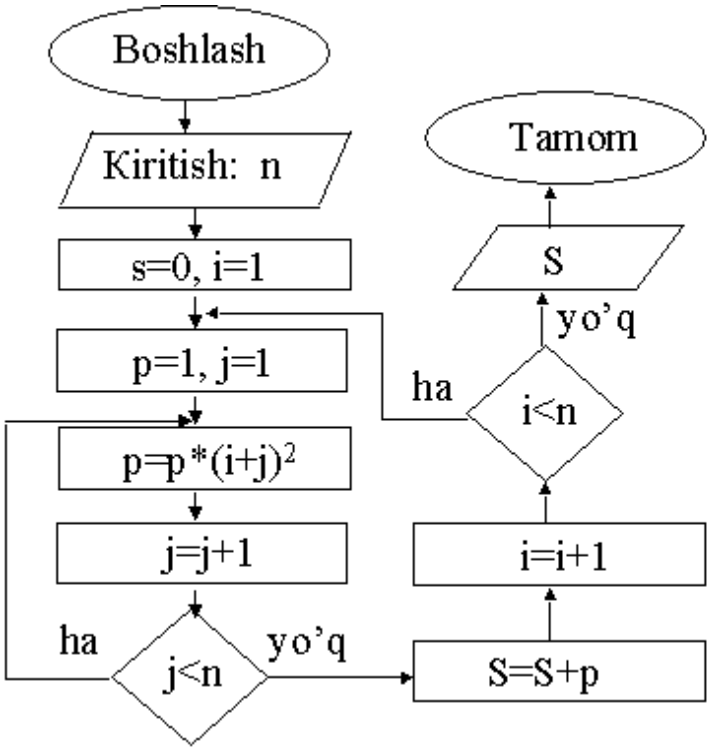
1.21-rasm. Parametrli takrorlash operatoriga doir blok-sxema

1.8. Ichma-ich joylashgan takrorlanuvchi jarayonlar

Ba'zan takrorlanuvchi algoritmlar bir nechta parametrga bog'liq bo'ladi. Odatda bunday algoritmlar ichma-ich joylashgan jarayonlar deb ataladi.

1-*misol*. Munosabatni hisoblang: $S = \sum_{i=1}^{n} \prod_{j=1}^{n} (i + j)^2$.

Yig'indi hisoblash uchun, i indeksning har bir qiymatida j indeks bo'yicha ko'paytmani hisoblab, avval yig'indi ustiga ketma-ket qo'shib borish kerak bo'ladi. Bu jarayon quyidagi ichma-ich joylashgan jarayonga doir blok–sxemada aks ettirilgan (1.22-rasm). Bu yerda indeks i dan tashqi takrorlash yig'indi uchun, j-dan esa-ichki takrorlash - ko'paytmani hosil qilish uchun foydalanilgan.



1.22-rasm. Ichma-ich joylashgan algoritmga doir blok-sxema

Shu bilan birga, keltirilgan murakkab munosabatni ikki nisbatan sodda munosabatlar ketma-ketligi bilan almashtirish (dekompozitsiya amali) maqsadga muvofiq:

$$1) \quad P_i = \prod_{j=1}^{n} (i + j)^2, i = 1,2,..., n; \quad 2) \quad S = \sum_{i=1}^{n} P_i .$$

2-*misol*. $B = b[i]$ ($i=1,2,\ldots,n$) massiv elementlarini o'sish (kamayish)

tartibida joylashtirish algoritmi va dasturini yaratish uchun yuqorida keltirilgan massiv elementlarining minimal (maksimal) qiymatli elementi va uning indeksini aniqlash algoritmidan foydalaniladi va quyidagi amallar ketma-ketligi bajariladi (bunda algoritmning so‘zlar orqali ifodalangan usulidan foydalaniladi) [2, 16-18 b.]:

1) kiritish (b_i , n);
2) i=1;
3) massivning i chidan to n chi elementlari orasidagi eng kichik (katta) element - z va uning indeksi - k aniqlanadi;
4) “uch likobcha” usuli asosida i-chi va minimal (maksimal) qiymatli elementlar joyma-joy almashtiriladi: $c=b[i]$; $b[i]= z$; $b[k]=c$, bunda c - yordamchi o‘zgaruvchi;
5) $i=i+1$;
6) agar $i<n$ bo‘lsa, u holda => (2).

Yuqoridagi algoritmning amallar ketma-ketligini to‘laligicha keltiramiz:

1) kiritish (n, b_i)
2) $i = 1$;
3) $z = b_i$;
4) $k = i$;
5) $j = i + 1$;
6) agar ($z < b_j$) shart bajarilsa, u holda {$z= b_j$; $k=j$;}
7) $j = j + 1$;
8) agar ($j <= n$) shart bajarilsa, u holda =>(6)
9) $c = b_i$;
10) $b_i = z$;
11) $b_k = c$;
12) $i = i+1$;
13) agar ($j <= n-1$) shart bajarilsa, u holda =>(3)
14) muhrlash (b_i).

Natijada, B = $\{b_i\}$– massiv elementlari o‘sish (kamayish) tartibida qayta joylashtiriladi.

3-*misol*. A=$\{a_{ij}\}$ matritsaning satr elementlari ko‘paytmalarining yig‘indisini hisoblash algoritmi tuzish talab etilsin. Bu masalaning matematik modeli quyidagicha ko‘rinishga ega:

$$S = \sum_{i=1}^{n} \prod_{j=1}^{m} a_{ij}.$$

Xususiy holda, agar n=3 va m=4 bo‘lsa, u holda $\{a_{ij}\}$ matritsaning ko‘rinishi quyidagicha bo‘ladi: $$A = \begin{pmatrix} a_{11}a_{12}a_{13}a_{14} \\ a_{21}a_{22}a_{23}a_{24} \\ a_{31}a_{32}a_{33}a_{44} \end{pmatrix}$$

Demak, masalaning echimi S= $(a_{11}*a_{12}*a_{13}*a_{14})+(a_{21}*a_{22}*a_{23}*a_{24})+ (a_{31}*a_{32}*a_{33}*a_{34})$ bo‘ladi.

Tashqi (satrlar) bo‘yicha takrorlash jarayonini – *i* indeks bilan, (*i*=1,2,3), ichki (ustunlar) bo‘yicha j - indeks bilan (j=1,2,3,4) belgilanadi. Tashqi indeks *i* bo‘yicha yig‘indi bajariladi, demak, uning boshlang‘ich qiymati S=0 deb olinadi. Tashqi indeksning har bir qiymatida ichki indeksning barcha qiymatlari bajariladi. Endi, ichki takrorlash jarayonida satr elementlarining ko‘paytmasi bajarilishi kerak bo‘ladi. Ko‘paytmaning boshlang‘ich qiymati uchun yordamchi R=1 o‘zgaruvchi ishlatiladi, va joriy amal P = P * a_{ij} ifoda yordamida satr elementlarining ko‘paytmasi hisoblanadi. Tashqi takrorlash jarayonining joriy amali S=S+R dan iborat. Shunday qilib, masalani yechish algoritmini so‘zlar orqali ifodalangan usulidan foydalanilsa, quyidagi ko‘rinishga ega:

1) kiritish (n, m, a_{ij});
2) S = 0;
3) i = 1;
4) P = 1;
5) j =1;
6) P = P * a_{ij};

7) j = j + 1;

8) agar (j <= m) bo‘lsa, u holda => (5);

9) S = S + P;

10) i = i + 1;

11) agar (i <= n) bo‘lsa, u holda => (4);

12) muhrlash (S).

Yuqoridagi keltirilgan munosabat ancha murakkab ma’noga va ko‘rinishga ega. SHu sababli masalani yechish uchun dekompozitsiya usulidan foydalanib, berilgan murakkab masalani ikki sodda masala ketma-ketligi ko‘rinishda tasvirlash mumkin, ya’ni

$$1)\ p_i = \prod_{i=1}^{m} a_{ij}, \quad i = 1,2,...,n; \quad 2)\ S = \sum_{i=1}^{n} p_i\ .$$

Bunda, avval mos satr elementlarning yig‘indisi $\{r_i\}$ oraliq massivga jamlanib, so‘ngra uning elementlari yig‘indisi S da hisoblanadi.

4-*misol*. Matritsani va vektorga ko‘paytmasini – C=A*B ni hisoblash masalasini ko‘riladi. Natija vektorning har bir elementi matritsa satr elementlarining vektorga skalyar ko‘paytmasidan iborat.

Bu yerda: $A=\{a_{ij}\}$, $b=\{b_j\}$, $C=\{c_i\}$, $1 \le i \le m$, $1 \le j \le n$.

Matematik modeli: $$c_i = \sum_{j=1}^{m} a_{ij} b_j, \quad i = 1, 2,...,n\ .$$

Bu masalani yechish algoritmi quyidagi amallardan iborat:

1) kiritish (n, m, a_{ij}, b_j);

2) *i* = 1;

3) P = 0;

4) j = 1;

5) P = P + a_{ij}* b_j ;

6) j = j + 1;

7) agar (j <= m) bo‘lsa, u holda => (5);

8) C_i = P;

9) $i = i + 1$;

10) agar (i <= n) bo‘lsa, u holda => (3);

11) muhrlash (C_i).

5-*misol*. Matritsani va matritsaga ko‘paytmasini – C=A*B hisoblash masalasi ko‘riladi. Bu yerda:

$A=\{a_{ik}\}$, $B=\{b_{kj}\}$, $C=\{c_{ij}\}$, $1 \le i \le n$, $1 \le j \le m$, $1 \le k \le l$.

Hisoblash formulasi:

$$c_{ij} = \sum_{k=1}^{l} a_{ik} * b_{kj},$$

Natijaviy $C=\{c_{ij}\}$ matritsasi har bir elementi a_{ij} matritsaning i chi satr elementlarini b_{kj} matritsa j-chi ustun elementlariga skalyar ko‘paytmasidan iborat.

Shuni hisobga olib, matritsa va matritsaga ko‘paytirish algoritmi quyidagicha tasvirlanadi:

1) kiritish (n, m, a_{ij}, b_{kj});
2) $i = 1$;
3) j = 1;
4) S=0;
5) k = 1;
6) $S = S + a_{ik} * b_{kj}$;
7) k = k + 1;
8) agar (k <= l) bo‘lsa, u holda => (6);
9) $C_{ij} = S$;
10) j = j + 1
11) agar (j <= m) bo‘lsa, u holda => (4);
12) $i = i + 1$;
13) agar (i <= n) bo‘lsa, u holda => (3);
14) muhrlash (C_{ij}).

6-*misol*. $A=\{a_{ij}\}$ matritsaning “egar” nuqtasini aniqlang. Matritsaning “egar” nuqtasi deganda bir vaqtda i - chi sart elementlari ichida eng katta va j chi

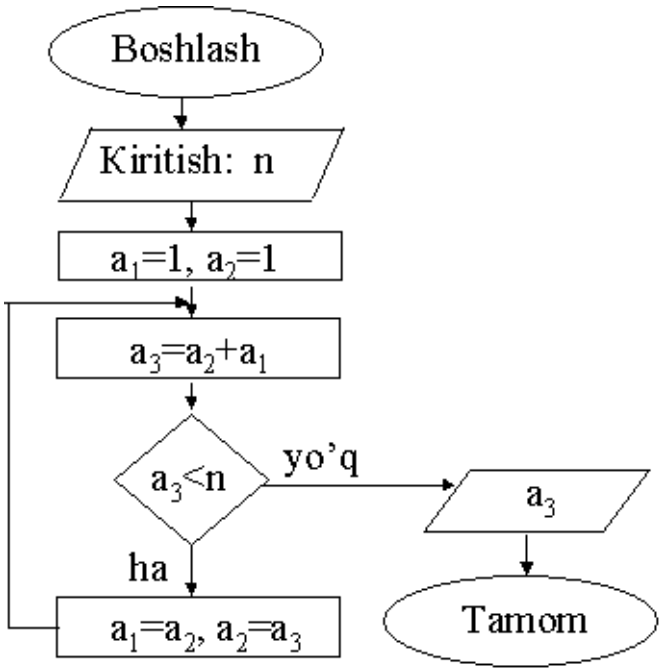
ustun elementlari ichida eng kichik bo‘lgan $a_{i\ j}$ elementidir. Agar matritsa elementlari har xil qiymatli bo‘lsa, u holda “egar” nuqtasi yagona bo‘ladi yoki mavjud emas. Demak, masalaning yechish algoritm, avvalo, tashqi takror jarayonida har bir *i*-satr bo‘yicha eng katta elementi joylashgan ustun indeksini aniqlab, shu ustun elementlar ichida eng kichik elementining indeksi $k = i$ ga tengligini tekshirishdan iborat bo‘ladi. Agar bu shart hech qaysi satrda bajarilmasa,-demak bu matritsada “egar” nuqta mavjud emas. Shu usulga moc algoritmni keltiramiz:

1) kiritish (n, m, $a_{i\,j}$)
2) p1=false;
3) *i=1;*
4) t=0;
5) p=$a_{i\,1}$;
6) k=1
7) *j=2;*
8) agar p < $a_{i\,j}$ bo‘lsa, u holda { p = $a_{i\,j}$; k = j };
9) j=j+1;
10) agar j <= m bo‘lsa, u holda => (8);
11) i=i+1;
12) agar i <= n bo‘lsa, u holda => (4);
13) *l*=1;
14) agar p < $a_{l\,k}$ bo‘lsa, u holda t=t+1;
15) agar (t = n) bo‘lsa, u holda {p1=true; muhrlash (*i*, k, p)}.
16) *l*=*l*+1;
17) agar (*l* <= n) bo‘lsa, u holda => (14);
18) agar (p1 = false) u holda muhrlash (egar nuqta yoq);

1.9. Rekursiyaga oid algoritmlar

Hisoblash jarayonida ba'zi bir algoritmlarning o'ziga qayta murojaat qilishiga to'g'ri keladi. O'ziga–o'zi murojaat qiladigan algoritmlarga rekurrent algoritmlar yoki rekursiya deb ataladi.

1-*misol*. Bunday algoritmga *misol* sifatida Fibonachchi sonlarini keltirish mumkin [2, 59 b.]. Ma'lumki, Fibonachchi sonlari quyidagicha aniqlangan: $a_0=a_1=1$, $a_i=a_{i-1}+a_{i-2}$, $i=2,3,4,\ldots$. Bu rekurrent ifoda algoritmiga mos keluvchi blok-sxema 1.23-rasmda keltirilgan. Eslatib o'tamiz, bu erda formuladagi *i*-indeksga hojat yo'q, lekin agar Fibonachchi sonining nomerini ham aniqlash zarur bo'lsa, birorta qo'shimcha parametr kiritish kerak bo'ladi.



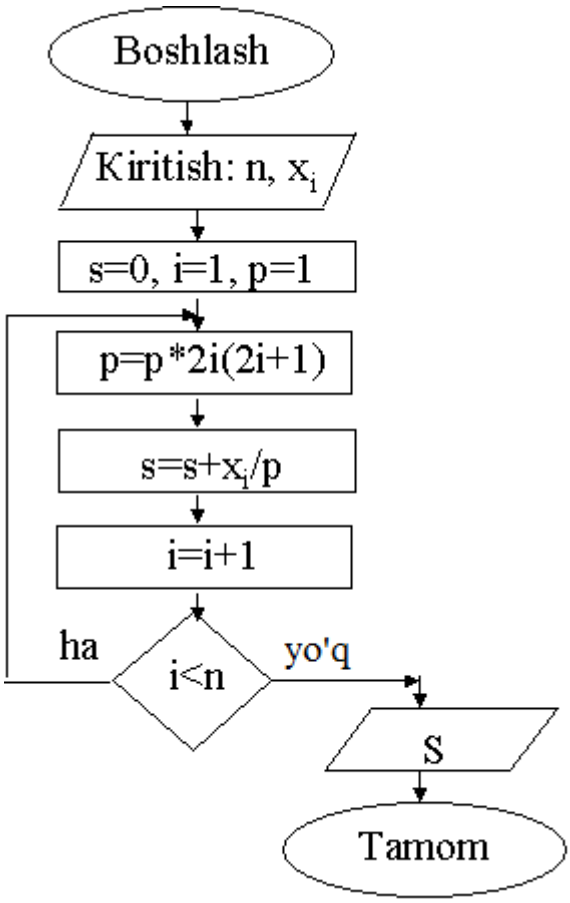
1.23-rasm. Fibonachchi sonlarining *n*- hadini hisoblash blok-sxemasi.

2-*misol*. $S=\sum_{i=1}^{n}\frac{x_i}{(2i+i)!}$ munosabatni hisoblang.

Bunda *i* indeksning har bir qiymatida faktorial va yig'indini hisoblash taqozo etiladi. Shuning uchun avval faktorialni hisoblashni alohida ko'rib chiqamiz. Quyidagi rekkurent ifoda faktorialni kam amal sarflab qulay usulda hisoblash imkonini beradi:

(1) $R=1$; (2) $R=R*2i*(2i+1)$.

Haqiqatan ham, *i=1*da *3!* ni, *i=2* da *R=3!*4*5=5!* ni va hokazo tarzda *(2i+1)!* ni yuqoridagi rekkurent formula yordamida hisoblash mumkin bo‘ladi. Bu misolga mos keluvchi blok-sxemasi 1.24-rasmda keltirilgan.



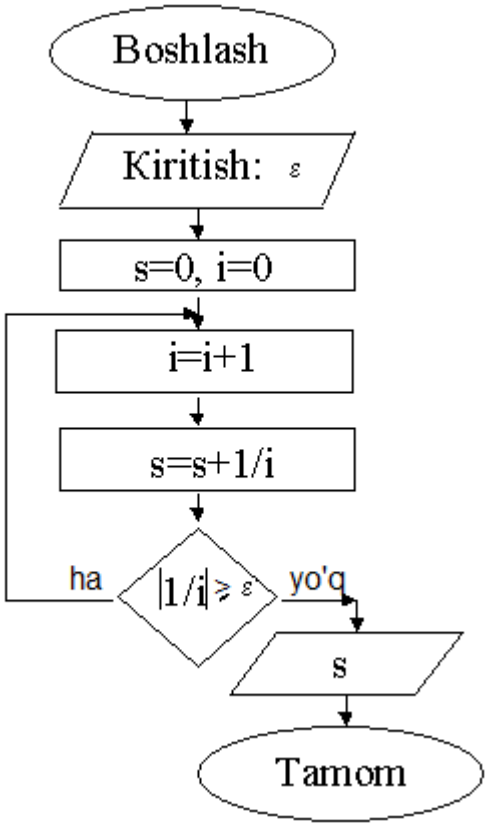
1.24-rasm. Rekurrent algoritmga doir blok-sxema

1.10. Soni noma’lum bo‘lgan takrorlash algoritmlari

Amaliyotda shunday masalalar uchraydiki, ularda takrorlanishlar soni oldindan berilmagan - noma’lum bo‘ladi. Ammo bu jarayonni tugatish uchun biror bir qo‘shimcha shart berilgan bo‘ladi.

1-*misol*. Quyidagi $S = 1 + \frac{1}{2} + \frac{1}{3} + ... = \sum_{i=1}^{\infty} \frac{1}{i}$ qatorda nechta had bilan chegaralanish berilmagan. Lekin qator hadlari yig‘indasini cheksiz kichik son $\varepsilon > 0$

aniqlikda hisoblash zarur bo‘ladi. Buning uchun $\left|\frac{1}{i}\right| < \varepsilon$ shartni olish mumkin. Shunday qilib, takrorlash jarayonining yakunlanishi shart bajarilguncha takrorlanadi (1.25-rasm).

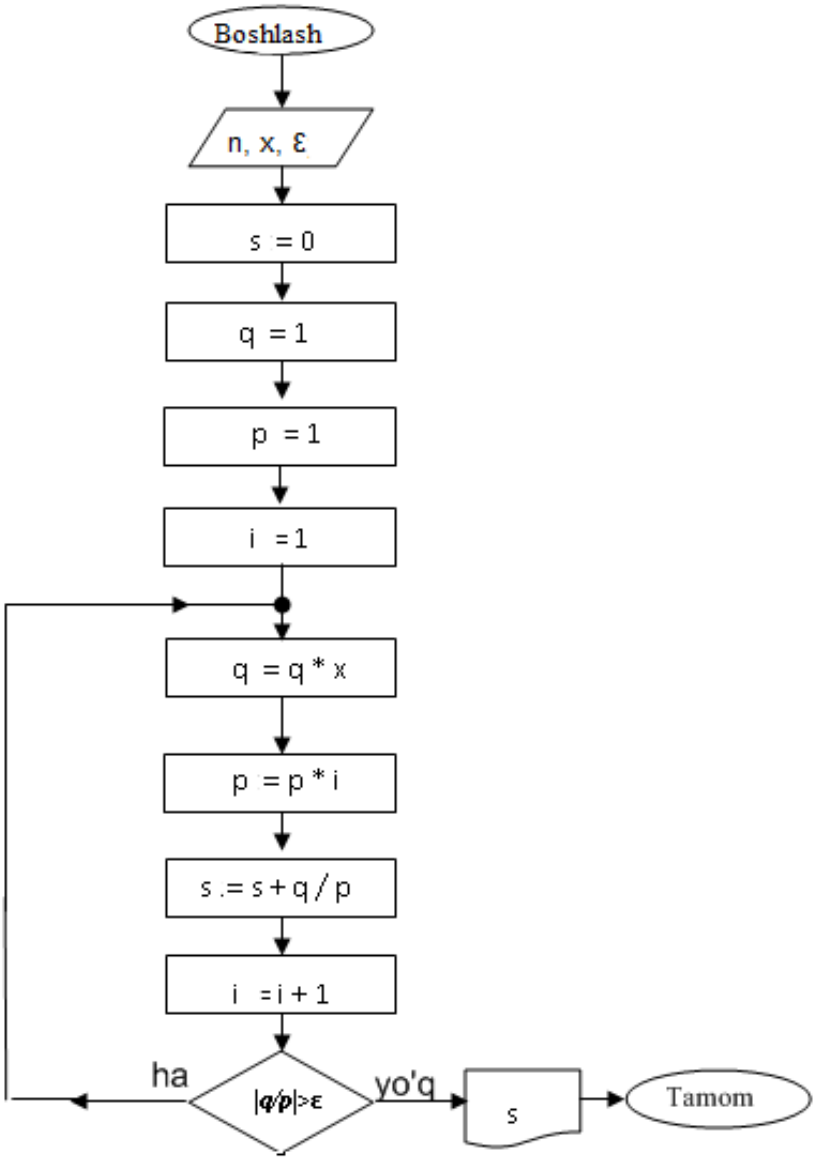


1.25-rasm. Takrorlanishlar soni oldindan noma’lum bo‘lgan algoritmlarga doir blok-sxema

2-*misol*. Musbat kichik son ε>0 aniqligida quyidagi munosabatni hisoblang: $s = x^1/1! + x^2/2! + \ldots + x^i/i! + \ldots$

Misolda cheksiz qatorning *i*-chi hadining absolyut qiymati ε>0 qiymatidan kichik bo‘lmaguncha yig‘indi davom ettirilishi kerak, ya’ni shart $|x^i / i!| > \varepsilon$ munosabat ko‘rinishida beriladi.

Misolni yechish algoritmi 1.26-rasmda keltirilgan. E’tibor bersak, jarayonni to‘xtatish sharti darajaga oshirish va faktorial hisoblash joriy qiymatlarining nisbat qiymatiga bog‘liq.



1.26-rasm. Hisoblash blok-sxemasi

3-*misol*. Funksiya uzluksiz va [a,b] oraliq chegaralarida har xil ishorali qiymatlarga ega deb faraz qilinadi. Oraliqni ikkiga bo‘lish usuli asosida f(x)=0 funksiyaning ildizini topish dasturini tuzing.

Masalani yechishdan avval oraliq chegarasidagi funksiya qiymatlarini moslash kerak, ya’ni x=a nuqtada funksiya manfiy, x= b nuqtada musbat qiymatga ega bo‘lish ta’minlanadi. Ularni joyma-joy almashtirish uchun “uch likopcha” usulidan foydalaniladi. So‘ngra oraliqni ikkiga bo‘lish usuli asosida f (x) = 0 funksiyaning ildizini aniqlash jarayoni amalga oshiriladi. Uning uchun, avvalo, s=(a+b)/2 o‘rta qiymat aniqlanadi va u=f (s) funksiya qiymatining ishorasi

aniqlanadi. Agar f (s)<0 bo‘lsa, u holda a=s, aks holda (f (s) > 0 bo‘lsa) – *b*=s deb qabul qilinadi. Bu jarayon |f (a)-f (b)|≤ε shart bajarilguncha davom etadi va funksiyaning ildizi x=(a+b)/2 deb hisoblanadi (bunda ε >0 - etarlikcha kichik musbat son).

Algoritmning so‘zlar orqali ifodalanishi usulidan foydalanamiz.

1) ma’lumotlarni kiritish (a, b, ε, f (x));
2) agar (f (a)>0 va f (b)<0)) bo‘lsa {c=a; a=b; b=c;}
3) ya=f (a);
4) yb=f (b);
5) agar (ya-yb) <=ε , u holda {*x*= (a+b)/2; => (10)}
6) ys=f ((a+b)/2);
7) agar (y3>0) bo‘lsa, u holda b=(a+b)/2, aks holda a=(a+b)/2;
8) ya=f (a);
9) yb =f (b);
10) muhrlash (x).

4-*misol*. Berilgan [a,b] oraliqda aniqlangan uzluksiz u= f (x) funksiya bilan OX o‘qi orasida hosil bo‘lgan S yuzani trapetsiya formulasi asosida taqribiy hisoblash algoritmi keltiriladi:

1) S = 0;
2) h = (b - a) / n;
3) i = 0;
4) S=S + (f (a+i*h)+ f (a+(i+1)*h)*h / 2;
5) i = i + 1;
6) agar i < n , u holda => (4);
7) muhrlash (S).

1.11. Ketma-ket yaqinlashuvchi yoki iteratsiali algoritmlar

Yuqori tartibli algebraik va transsendent tenglamalarni yechish usullari yoki algoritmlari ketma-ket yaqinlashuvchi – iteratsiali algoritmlarga misollar bo‘ladi. Ma’lumki, transsendent tenglamalarni yechishning quyidagi asosiy usullari mavjud:

- urinmalar usuli (Nyuton usuli),
- ketma-ket yaqinlashish usuli,
- vatarlar usuli,
- teng ikkiga bo‘lish usuli.

1-*misol*. Bizga $f(x)=0$ (1) transendent tenglama berilgan bo‘lsin. Faraz qilaylik, bu tenglama [a,b] oraliqda uzluksiz va f(a)*f(b)<0 shartni qanoatlantirsin. Ma’lumki, bunday holda berilgan tenglama [a,b] orilaqda kamida bitta ildizga ega bo‘ladi va u quyidagi formula orqali topiladi.

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)} \qquad n = 0,1,2,... \quad (2)$$

Boshlang‘ich X_0 qiymat $f(x_0)f''(x_0) < 0$ shart asosida tanlab olinsa, (2) iteratsiya albatta yaqinlashadi. Ketma-ketlik

$$\left|X_{n+1} - X_n\right| < \varepsilon$$

shart bajarilgunga qadar davom ettiriladi.

2-misol. Berilgan musbat *a* haqiqiy sondan kvadrat ildizini hisoblash algoritmi tuzing.

Bu masalani yechish uchun kvadrat ildizni *x* deb belgilab olib,

$$\sqrt{a} = x \qquad (3)$$

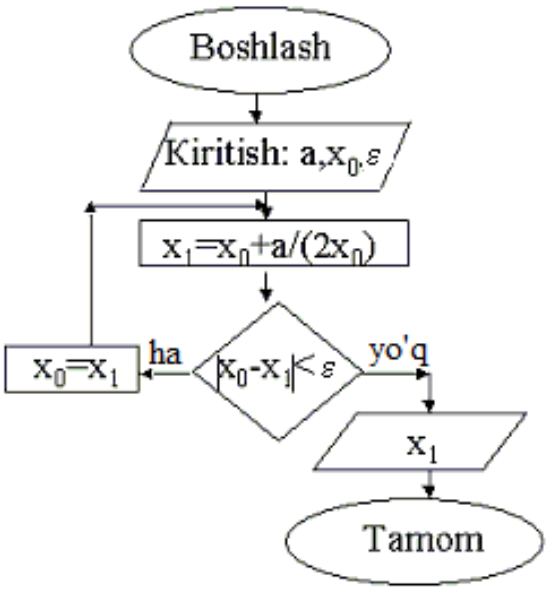
ifodani yozib olamiz. U holda (1) tenglamaga asosan:

$$f(x) = x^2 - a \qquad (4)$$

ekanligini topish mumkin. (4) ifodani (2) ga qo‘yib, quyidagi rekurrent formulani topish mumkin:

$$X_{n+1} = \frac{1}{2}(X_n + \frac{a}{2X_n}) \qquad (5)$$

Bu formulaga mos blok-sxema 1.17-rasmda keltirilgan. Musbat kichik son ε>0 - kvadrat ildizni topishning berilgan aniqligi. Eslatib o‘tamiz, algoritmda indeksli o‘zgaruvchilarga zarurat yo‘q.



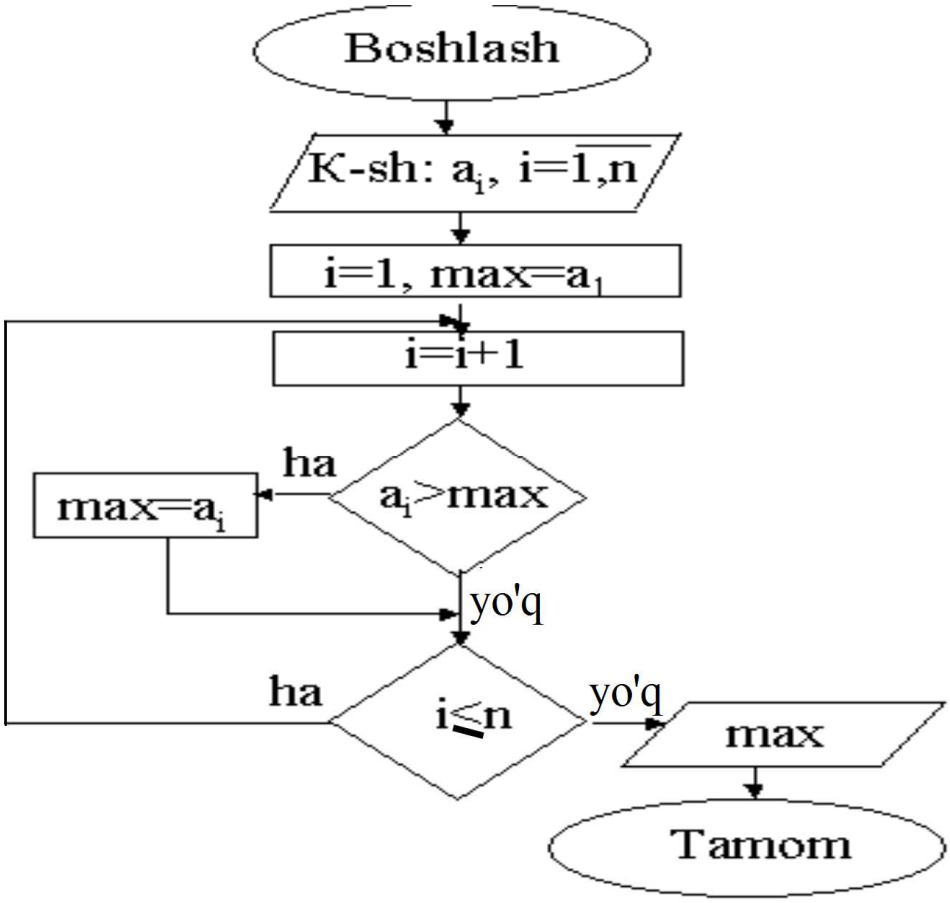
1.27-rasm. Berilgan musbat haqiqiy sondan kvadrat ildiz chiqarish blok-sxemasi

1.12. Algoritm ijrosini tekshirish

Kompyuter uchun tuzilgan algoritm ijrochisi-bu kompyuterdir. Biror dasturlash tilida yozilgan algoritm kodlashtirilgan oddiy ko‘rsatmalar ketma-ketliligа o‘tadi va kompyuter tomonidan avtomatik ravishda bajariladi. Metodik nuqtai-nazardan qaraganda, algoritmning birinchi ijrochisi sifatida o‘quvchining o‘zini olish muhim ahamiyatga ega. O‘quvchi tomonidan biror masalani yechish algoritmi tuzilganda bu algoritmni to‘g‘ri natija berishini tekshirish juda muhimdir. Buning yagona usuli o‘quvchi tomonidan algoritmni turli boshlang‘ich ma’lumotlarda qadamma – qadam bajarib (ijro etib) ko‘rishdir. Algoritmni bajarish natijasida xatolar aniqlanadi va to‘g‘rilanadi. Ikkinchi tomonidan, masalani yechishga qiynalayotgan o‘quvchi uchun tayyor algoritmni bajarish – masalani yechish yo‘llarini tushunishga xizmat qiladi.

Algoritm ijrosini quyidagi misolda ko‘raylik (1.28-rasm).

1-misol. Berilgan massiv A=$\{a_i\}$, $i = 1,....,n$ elementlari ichida eng kattasini topish algoritmini tuzaylik. Buning uchun, berilgan sonlardan birinchisi - $a_1$ni eng katta qiymat deb faraz qilaylik va uni *max* nomli yangi o‘zgaruvchiga uzataylik: *max=a1*. Parametr *i* ning qiymatini bittaga oshirib, ya’ni *i=i+1 da max*-ning qiymati a_i bilan taqqoslanadi va a_i katta bo‘lsa, u *max* o‘zgaruvchisiga uzatiladi va jarayonni shu tarzda to *i= n* bo‘lguncha davom ettiramiz. Bu fikrlar 1.28-rasmdagi blok-sxemada o‘z aksini topgan:



1.28-rasm. Vektor elementlarining eng kattasini topish blok-sxemasi

Endi, bu blok-sxema yoki algoritmning ijrosini $n = 3$ $a_1 = 3$, $a_2 = 5$, $a_3 = 1$aniq sonlarda ko‘rib o‘taylik:

1) *i=1*da *max=3* bo‘ladi.
2) *i=i+1=2* ni topamiz,
3) a_2>*max*, ya’ni *5>3*ni tekshiramiz, shart bajarilsa, max=5 bo‘ladi.

4) $i<n$, ya'ni $2<3$ ni tekshiramiz. Shart bajarilsa, i ni yana bittaga oshiramiz, va $i=3$ bo'ladi;
5) $a_3>max$, ya'ni $1>5$, ni tekshiramiz. Shart bajarilmadi, demak, keyingi
6) $i<n$ shartni, ya'ni $3<3$ni tekshiramiz. Shart bajarilmadi. Demak natiga $max=5$ chop etiladi.

Biz blok-sxemani tahlil qilish davomida uning to'g'riligiga ishonch hosil qildik. Endi ixtiyoriy n qiymat uchun bu blok-sxema bo'yicha eng katta elementni topish mumkin.

1-bob bo‘yicha savol va topshiriqlar

Savol va topshiriqlar [6, 7] adabiyotlardan tanlab olingan.

1. Tadqiq qilinadigan obyekt haqida tushuncha bering.
2. Jarayonni hal qilishdagi muammolar nimadan iborat?
3. Matematik model ta’rifini bering.
4. Diskret model nimadan iborat?
5. Masalani yechish usullari to‘g‘risida ma’lumot bering.
6. Dastur yaratish texnologiyalarini aytib bering.
7. Dasturni sozlash nimadan iborat?
8. Natija va uning tahlili nimadan iborat?
9. Hisoblash eksperimenti haqida tushuncha bering.
10. “Algoritm” tushunchasining ta’rifini bering.
11. Blok-sxemaning ta’rifini bering.
12. Blok-sxema asosiy figuralar ma’nosini ayting bering.
13. Qanday algoritmlar chiziqli deb ataladi.
14. Qanday algoritmlar tarmoqlanuvchi deb ataladi?
15. Qanday algoritmlar takrorlanuvchi deb ataladi?
16. Blok-sxemada ishlatiladigan asosiy bloklarni aytib bering.
17. Dastur tuzilishida qaysi bloklardan foydalaniladi?
18. Algoritmdagi xatolar qanday qilib to‘g‘rilanadi?
19. Chiziqli algoritmning ishlash prinsiplari.
20. Shartli algoritmning ishlash prinsiplari.
21. Shartli algoritm shakllarini ko‘rsating.
22. Takrorlanuvchi algoritmning ishlash usullari.
23. Ichma-ich joylashgan takrorlanuvchi algoritmlar nimadan iborat?
24. Qaysi masalalarda tarkibiy algoritmdan foydalaniladi?
25. Rekursiyaga oid algoritmlar nimadan iborat?
26. Soni noma’lum bo‘lgan takrorlash algoritmlar nimadan iborat?
27. Ketma-ket yaqinlashuvchi yoki iteratsion algoritmlar nimadan iborat?

28. Algoritm ijrosini tekshirish nimadan iborat?

29. Transsendent tenglamalarni yechishning urinmalar usuli (Nyuton usuli) bilan hisoblash algoritmini tuzing.

30. Transsendent tenglamalarni yechishning ketma-ket yaqinlashish usuli bilan hisoblash algoritmini tuzing.

31. Transsendent tenglamalarni yechishning vatarlar usuli bilan hisoblash algoritmini tuzing.

32. Transsendent tenglamalarni yechishning teng ikkiga bo‘lish usuli bilan hisoblash algoritmini tuzing.

33. $Y=a(b+cx)-dx$ formula bo‘yicha qiymat hisoblash algoritmini tuzing.

34. Bir to‘g‘ri chiziqda yotmaydigan uchta nuqta koordinatlari orqali o‘tuvchi aylanani yuzasini hisoblash algoritmini tuzing.

35. Bir to‘g‘ri chiziqda yotmaydigan uchta nuqta koordinatlari orqali yuzasini hisoblash algoritmini tuzing.

36. "Svetofordan (uchchiroqli) foydalanish" algoritmini tuzng.

37. Geron formulasi bo‘yicha uchburchak yuzasini hisoblash algoritmining 1.2 da keltirilgan masalaga oid blok-sxemasini tuzing.

38.2. $y=x^2+2x^4+b$ funksiyani $x=3$ qiymatida hisoblash algoritmining blok-sxemasini tuzing.

39. Quyidagi ifodani hisoblash algoritmini tuzing:

$y = (\sin(a^2) + \cos^2(a-b)) / \ln(a+b) + c^2$ bu erda

$a = (p+q) * b - c,\ b = \sin(e^2 - (p+q^3)),\ c = tg(3p+q) - b.$

40. Quyidagi ifodani hisoblash algoritmini tuzing:

$s = (tg(y^2) + |(x-y)^3|) / \ln(x+y^4)$, bu erda

$x = (a+b)^2 - (y-c^3),\ y = e^{(a+b)} - (c^2 - (a-b)^3).$

41. Quyidagi ifodalarni hisoblash algoritmini tuzing:

$z = (a-b)^3 + \lg^2(2a+3c) / (tg^2(c - \sin a))$ bu erda

$a = \sin((b-c^2))^3,\ b = tg(x+y^2),\ c = \cos(e^{(x-y)} + 5\sin(b+2)).$

42. Quyidagi ifodalarni hisoblash algoritmini tuzing:

$f = \lg(x^2 - y + z^3) + \cos^4(x + y)$ bu yerda

$x = (\cos(a^3 - b^2) + \sin(e^{a+b}))/(a + \cos(y + 2))$, $y = |a^2 - b|$, $z = (a + |y - x|)^2$

43. Haqiqiy *a* qiymat berilgan. Uchta ko‘paytirish amali yordamida a^8 qiymatini hisoblash algoritmini tuzing.
44. Berilgan *x* uchun quyidagi ifodaning qiymatini hisoblash algoritmini tuzing: $A = |x - 2|/(2 \cdot x + \sqrt{x} - 2)$.
45. Berilgan ikki (*x1, y1*) va (*x2, y2*) nuqtalar orasidagi masofani hisoblash algoritmini tuzing.
46. Geron formulasi asosida uchlari koordinatalari bilan berilgan uchburchak yuzasini hisoblash algoritmini tuzing.
47. Diskreminant hisoblash asosida uchlari koordinatalari bilan berilgan uchburchak yuzasini hisoblash algoritmini tuzing.
48. Teng bo‘laklarga bo‘lish usuli yordamida [a,b] oralig‘ida *f(x) = 0* tenglamaning yagona ildizini berilgan aniqlik bilan hisoblash algoritmbi tuzing.
49. Berilgan 4 ta - *a,b,c,d* haqiqiy sonlar orasidan eng katta va eng kichik qiymatlar o‘rtasidagi ayirmani aniqlang.
50. 100 ballik shkala bo‘yicha talabalarning to‘plagan ballari klaviatura orqali kiritiladi. Ballar taqsimotining qabul qilingan usuliga ko‘ra talabalarning ballar bo‘yicha baholarini aniqlash dasturini tuzing.
51. Xizmatchining oylik maoshidan o‘sib boruvchi shkala bo‘yicha ushlab qolinadigan daromad solig‘ini hisoblang.
52. Fibonachchi sonlar qatori: 1,1,2,3,5,… dan dastlabki n – ta sonlarning yig‘indisini aniqlang .
53. Aylananing yuzasi *S* va kvadratning yuzasi *R* berilgan. Kvadratning aylanaga sig‘ishni yoki sig‘masligi aniqlash algoritmini tuzing.
54. Berilagan - uchta - *a, b, s* sonlardan foydalanib tomonlarining uzunliklari shu sonlarga teng bo‘lgan uchburchakning mavjudligini aniqlang va shunday

uchburchakni yasash mumkin bo‘lsa, uning yuzasini hisoblash algoritmini tuzing.

55. $S = \frac{1}{n}\sum_{i=1}^{n} a_i - \min_{1 \le i \le n} a_i$ munosabatning hisoblash algoritmini tuzing.

56. Quyidagi munosabanning algoritmini tuzing: $P = \prod_{i=1}^{n} a_i - n * \max_{1 \le i \le n} a_i$.

57. Berilgan sonlarning eng kattasini topadigan algoritmini tuzing.

58. $R=(x-2)(x-4)(x-8)...(x-64)$ hisoblash algoritmini tuzing. (x-haqiqiy son).

59. Ikkita n va m natural sonning eng katta umumiy bo‘luvchisini topish algoritmi (Evklid algoritmi) tuzingn.

60. To’rt xonali sonlar orasidan avvalgi ikkita raqamli yig‘indisi, keyingi 2 ta raqamli yig‘indisiga teng bo‘lgan sonlarni va miqdorini hisoblash algoritmi tuzilsin.

61. Berilgan ***n***x***n*** o‘lchovli $a_{i\ j}$ matritsaning satr elementlarining yig‘indisini hisoblash algoritmini tuzing..

62. **n**x**m** o‘lchovli a_{ij} matritsaning elementlarining eng katta va kichik elementlari hisoblash algoritmini tuzing..

63. $S = \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$ munosabat cheksiz kichik son ε > 0 aniqlikda hisoblash algoritmini tuzing.

64. Berilgan $a_1, a_2, a_3,...,a_n$ conlarning eng katta va eng kichik elementlarini bir vaqtda topadigan blok-sxema tuzing va uni $n=3$da tekshiring.

65. Berilgan $a_1, a_2, a_3,...,a_n$ sonlarni qiymatlari bo‘yicha o‘sish tartibida qayta joylashtirish algoritmini tuzing.

66. Ikkita n va m natural sonlarining eng kata umumiy bo‘luvchisini topish (Evklid) algoritmiga blok-sxema tuzilsin.

67. Teng ikkiga bo‘lish usuli uchun blok-sxemani tuzing.

68. Vatarlar usuli uchun blok-sxema tuzing.

69. Ketma-ket yaqinlashish usuli uchun blok-sxema tuzing.

70. $S = \sum_{i=0}^{3}(2i+3)!$ hisoblash algoritmini tuzing.

71. $Y = \sum_{i=1}^{3}\frac{(2i+2)!}{(3i+4)!}$ hisoblash algoritmini tuzing.

2-BOB. C++ DASTURLASH TILI ASOSIY OPERATORLARI

2.1. C++ tilidagi dastur tuzilishi

C++ tilida dastur tuzilishi tushuntirish uchun sodda programma keltiramiz [3, 4-8 b.].

```
#include <iostream.h> // sarlavha faylni qo‘shish
int main()            // bosh funksiya tavsifi
{                     // asosiy  blok boshlanishi
 cout<<”Kompyuter olami!\n”;// satrni chop etish
 return 0;            // funksiya qaytaradigan qiymat
}                     // asosiy blok tugashi
```

Dasturning 1-satrida *“#include”* preprotsessor ko‘rsatmasi bo‘lib, dastur kodiga standart oqimli o‘qish-yozish funksiyalari va uning o‘zgaruvchilari e’loni joylashgan «iostream.h» sarlavha faylini qo‘shadi (mnemonika: ‘i’(*input*) - kiritish (o‘qish); ‘o’(*ouput*) - chiqarish (yozish); “*stream*”- oqim; ‘h’(*head*) – sarlavha). Kelishuv bo‘yicha standart oqim ekranga chiqarish hisoblanadi. Keyingi qatorlarda dasturning yagona, asosiy funksiyasi - *main*() funksiyasi tavsifi keltirilgan. Shuni qayd etish kerakki, S++ dasturida, albatta, *main*() funksiyasi bo‘lishi shart va dastur shu funksiyani bajarish bilan o‘z ishini boshlaydi. Funksiya nomi oldidagi “*int*” kalit so‘zi funksiya bajarilishi natijasida qaytaraladigan qiymat turini bildiradi. Bunday holat funksiyaning matematikadagi tavsifiga mos keladi. Keyingi qatordan funksiya tanasi - figurali qavsga olingan amallar ketma-ketligi keladi. Bizning holda funksiya tanasi ikkita amaldan iborat. Birinchisi, konsol rejimida belgilar ketma-ketligini oqimga chiqarish amali qo‘llangan. Buning uchun «iostream.h» sarlavha faylida aniqlangan *cout* obyektidan foydalanilgan. Uning formati quyidagi ko‘rinishda:

cout << <*ifoda*>;

Bu yerda “<<” – ma’lumot uzatish amali («..ga joylashtir»), <ifoda> sifatida o‘zgaruvchi yoki sintaksisi to‘g‘ri yozilgan va qandaydir qiymat qabul qiluvchi til

ifodasi kelishi mumkin (*keyinchalik, burchak qavs ichiga olingan o‘zbekcha satr ostini til tarkibiga kirmaydigan tushuncha deb qabul qilish kerak)*.

Ikkinchisi, funksiya o‘z ishini tugatganligini anglatuvchi va undan chiqishni amalga oshiruvchi “**return** 0;” operatoridir. Odatda, bajarilishi normal tugagan funksiyalar operatsion sistemaga 0 qiymatini qaytaradi. Shu qoidagi rioya qilgan holda dastur ham 0 qiymatini qaytaradi.

Bajariluvchi dasturni hosil qilish uchun dastur matni kompilyatsiya qilinishi kerak. Kompilyasiya jarayonining o‘zi ham ikkita bosqichdan tashkil topadi. Boshida preprotsessor ishlaydi, u matndagi kompilyatsiya direktivalarini bajaradi, xususan *#include* direktivasi bo‘yicha ko‘rsatilgan kutubxonalardan S++ tilida yozilgan modullarni dastur tarkibiga kiritadi. Shundan so‘ng kengaytirilgan dastur matni kompilyatorga uzatiladi. Kompilyator o‘zi ham dastur bo‘lib, uning uchun kiruvchi ma’lumot bo‘lib, C++ tilida yozilgan dastur matni hisoblanadi. Kompilyator dastur matnini leksema (atomar) elementlarga ajratadi va uni leksik, keyinchalik sintaktik tahlil qiladi. Leksik tahlil jarayonida u matnni leksemalarga ajratish uchun «probel ajratuvchisini» ishlatadi. Probel ajratuvchisiga - probel belgisi (‘␣’), ‘*t*’ - tabulyatsiya belgisi, ‘*n*’- keyingi qatorga o‘tish belgisi, boshqa ajratuvchilar va izohlar kiradi.

Dastur matni tushunarli bo‘lishi uchun izohlar ishlatiladi. Izohlar dastur amal qilishiga hech qanday ta’sir qilmaydi.

C++ tilida izohlar ikki ko‘rinishda yozilishi mumkin.

Birinchisida “**/***” dan boshlanib, “***/**” belgilar oralig‘ida joylashgan barcha belgilar ketma-ketligi izoh hisoblanadi, ikkinchisi «*satriy izoh*» deb nomlanadi va u “**//**” belgilardan boshlangan va satr oxirigacha yozilgan belgilar ketma-ketligi bo‘ladi. Izohning birinchi ko‘rinishida yozilgan izohlar bir necha satr bo‘lishi va ulardan keyin C++ operatorlari davom etishi mumkin.

Berilganlarni standart oqimdan (odatda, klaviaturadan) o‘qish quyidagi formatda amalga oshiriladi:

cin >> <*o‘zgaruvchi*>;

Bu yerda <o‘zgaruvchi> oqimdan qiymat qabul qiluvchi o‘zgaruvchining nomi.

int *n*;
cout << ” *n* = ”;
cin >> *n*;

Butun turdagi *n* o‘zgaruvchisi kiritilgan qiymatni o‘zlashtiradi [3, 11-12 b.]. Bir paytning o‘zida probel vositasida bir nechta va har xil turdagi qiymatlarni oqimdan kiritish mumkin. Qiymat kiritish “Enter” tugmasini bosish bilan tugaydi. Agar kiritilgan qiymatlar soni o‘zgaruvchilar sonidan ko‘p bo‘lsa, «*ortiqcha*» qiymatlar bufer xotirada saqlanib qoladi.

int x, y;
float z;
cin >> x >> y >> z;

SHuni qayd etish kerakki, oqimga qiymat kiritishda probel ajratuvchi hisoblanadi. Haqiqiy sonning butun va kasr qismlari ‘.’ belgisi bilan ajratiladi.

2.2.Taqqoslash amallari

C++ tilida qiymatlarni solishtirish uchun taqqoslash amallari aniqlangan [3, 22 b.]. Taqqoslash amali binar amal bo‘lib, quyidagi ko‘rinishga ega:

<*operand1*> <*taqqoslash amali*> < *operand2*>

Taqqoslash amallarining natijasi - taqqoslash o‘rinli bo‘lsa, *true* (rost), aks holda *false* (yolg‘on) qiymat bo‘ladi. Agar taqqoslashda arifmetik ifoda qatnashsa, uning qiymati 0 qiymatidan farqli holatlar uchun 1 deb hisoblanadi.

Taqqoslash amallari va ularning qo‘llanishi

Amallar	Qo‘llanishi	Mazmuni (o‘qilishi)
<	a<b	“*a* kichik b”
<=	a<=b	“*a* kichik yoki teng *b*”
>	a>b	“*a* katta *b*”
>=	a>=b	“*a* katta yoki teng *b*”
==	a==b	“*a* teng *b*”
!=	a!=b	“*a* teng emas *b*”

2.3. Mantiqiy operatorlar

Mantiqiy ko‘paytirish operatori ikkita ifodani hisoblaydi, agar ikkala ifoda *true* qiymat qaytarsa, *BA* operatori ham *true* qiymat qaytardi [3, 22 b.]:

Operator	Belgi	Namuna
BA	**&&**	*ifoda1* **&&** *ifoda2*
YOKI	**||**	*ifoda1* **||** *ifoda2*
INKOR	**!**	***!****ifoda*

Mantiqiy ko‘paytirish operatori **&&** belgi orqali belgilanadi.

ifoda1	ifoda1	ifoda1 && ifoda2
false (0)	*false* (0)	*false* (0)
false (0)	*true* (0 emas)	*false* (0)
true (0 emas)	*false* (0)	*false* (0)
true (0 emas)	*true* (0 emas)	*true* (1)

Mantiqiy qo‘shish operatori ham ikkita ifoda orqali hisoblanadi. Agarda ulardan birortasi *true* bo‘lsa, mantiqiy qo‘shish operatori *true* qiymat qaytaradi.

Mantiqiy qo‘shish operatori || belgi orqali belgilanadi.

ifoda1	ifoda1	ifoda1 || ifoda2
false (0)	*false* (0)	*false* (0)
false (0)	*true* (0 emas)	*true* (1)
true (0 emas)	*false* (0)	*true* (1)
true (0 emas)	*true* (0 emas)	*true* (1)

Mantiqiy inkor operatori tekshirilayotgan ifoda *yolg‘on* bo‘lsa, true qiymat qaytaradi. Agarda tekshirilayotgan ifoda *rost* bo‘lsa, inkor operatori *false* qiymat qaytaradi.

Mantiqiy *inkor* operatori **!** belgi orqali belgilanadi.

ifoda1	!ifoda1
false (0)	true (1)
true (0 emas)	false (0)

2.4. Inkrement va dekrement amallari

C++ tilida operand qiymatini birga oshirish va kamaytirishning samarali vositalari mavjud. Bular inkrement (++) va dekrement (--) unar amallardir.

`soni=soni+1;` kodi o‘rniga `soni++;` ko‘rinishida yoziladi.

Inkrement amali operand qiymatini bittaga oshirish uchun ishlatiladi. Dekrement amali esa operand qiymatini bittaga kamaytirish uchun ishlatiladi. Operandga nisbatan bu amallarning ikki xil ko‘rinishi: prefiks va postfiks ko‘rinishlari mavjud. Prefiks ko‘rinishda amal til ko‘rsatmasi bo‘yicha ish bajarilishidan oldin operandga qo‘llaniladi. Postfiks holatda esa amal til ko‘rsatmasi bo‘yicha ish bajarilgandan keyin operandga qo‘llaniladi.

Prefiks inkrement: `++variable`

Postfiks inkrement: `variable++`

Prefiks dekrement: `--variable`

Postfiks dekrement: `variable--`

“Prefiks” yoki “postfiks” amal tushunchasi faqat qiymat berish bilan bog‘liq ifodalarda o‘rinli.

```
x = 5;
y = ++x;
```

Ushbu misolda x ning qiymati besh. *y* ga *x* ning qiymatini yuklashdan oldin pre-inkrement amali qo‘llaniladi, *x* ning qiymati oltiga o‘zgaradi, so‘ngra *y* o‘zgaruvchisiga olti yuklanadi. Natijada, *x* ning qiymati ham, *y* ning qiymati ham oltiga teng bo‘ladi.

```
x = 5;
y = x++;
```

Ushbu misolda *x* ning qiymati besh. *y* ga *x* ning qiymatini yuklash jarayonida post-inkrement amali qo‘llaniladi, *x* ning qiymati *y* o‘zgaruvchisiga yuklanadi, so‘ngra *x* ning qiymati bittaga oshiriladi. Natijada, *x* ning qiymati - olti, *y* ning qiymati - beshga teng bo‘ladi.

Inkrement va dekrement amallarini murakkab ifodaning ichida ham ishlatish mumkin. Faqat bunda tushunarli bo‘lishi uchun inkrement va dekrement amallarini qavs ichiga olish maqsadga muvofiq.

```
a = 5;
b = 2 + (++a);
```

Birinchi ifodada a o‘zgaruvchisiga besh soni yuklanadi. Keyin esa, `2 + (++a)` ifodani bajarish jarayonida avval `a` o‘zgaruvchisining qiymati bittaga oshiriladi, so‘ngra uning qiymatiga ikki sonini qo‘shib, natija `b` o‘zgaruvchisiga yuklanadi. Natijada, `a` ning qiymati - olti, `b` ning qiymati - sakkizga teng bo‘ladi.

```
a = 5;
b = 2 + (a--);
```

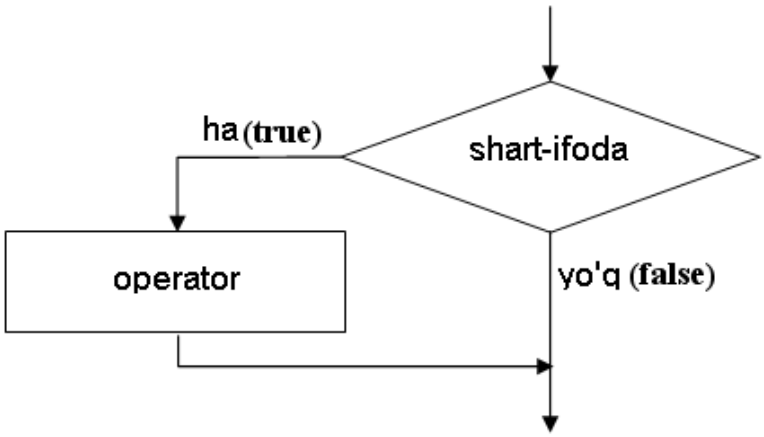
Birinchi ifodada `a` o‘zgaruvchisiga besh soni yuklanadi. Keyin esa, `2 + (a--)` ifodani bajarish jarayonida, avval, `a` o‘zgaruvchisining qiymati ifodaga qo‘yib hisoblanib, natija `b` o‘zgaruvchisiga yuklanadi, so‘ngra uning qiymati bittaga kamaytiriladi. Natijada, `a` ning qiymati to‘rt, `b` ning qiymati esa yettiga teng bo‘ladi.

2.5. Shart operatorlari

Tarmoqlanuvchi algoritmlarda tarmoqlanishni amalga oshirish uchun shartli operatordan foydalaniladi. **if** operatori qandaydir shartni rostlikka tekshirish natijasiga ko‘ra dasturda tarmoqlanishni amalga oshiradi [3, 34 b.]:

if (<*shart*>) <*operator*>;

Bu yerda <*shart*> har qanday ifoda bo‘lishi mumkin, odatda, u taqqoslash amali bo‘ladi. Agar shart 0 qiymatidan farqli yoki rost (*true*) bo‘lsa, <operator> bajariladi, aks holda, ya’ni shart 0 yoki yolg‘on (*false*) bo‘lsa, hech qanday amal bajarilmaydi va boshqaruv **if** operatoridan keyingi operatorga o‘tadi (agar u mavjud bo‘lsa). Bunday konstruksiya bir tomonlama tanlov deb ham ataladi. Ushbu holat quyidagi 2.1 - rasmda ko‘rsatilgan.



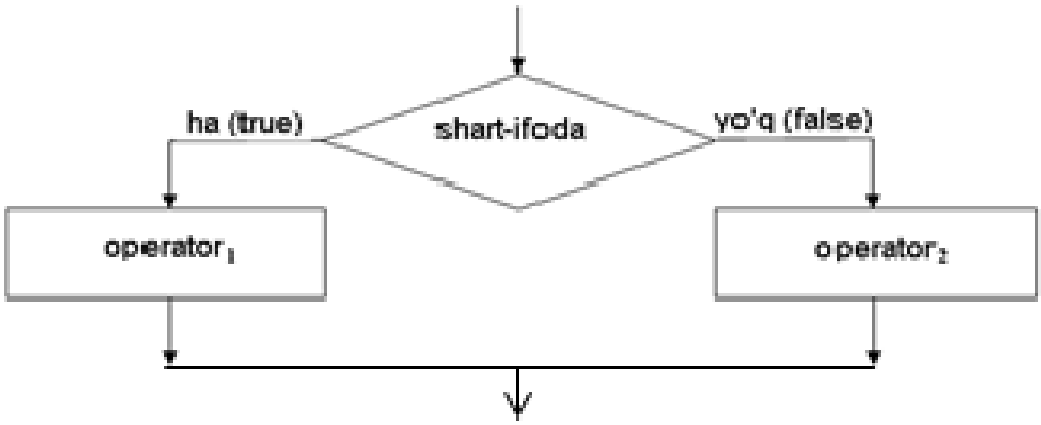
2.1.- if shart operatorining bajarilishi (bir tomonlama tanlov)

if - else operatori. Agar dastur bajarilishi jarayonida shartning natijasiga qarab u yoki bu amalni bajarish kerak bo‘lsa, shart operatorining ikki tomonlama tanlovli ko‘rinishidan foydalaniladi. Shart operatorining ikki tomonlama tanlovli ko‘rinishi - **if...else** quyidagicha sintaksisga ega:

if (<*shart-ifoda*>) **<***operator1***>; else <***operator2***>;**

Bu yerda <shart-ifoda> 0 qiymatidan farqli yoki true bo‘lsa, <$operator_1$>, aks holda <$operator_2$> bajariladi. **if...else** shart operatori mazmuniga ko‘ra algoritmning tarmoqlanuvchi blokini ifodalaydi: <*shart-ifoda*> - shart bloki (romb) va <$operator_1$> blokning «*ha*» shoxiga, <$operator_2$> esa blokning «*yo‘q*» shoxiga mos keluvchi amallar bloklari deb qarash mumkin.

Shart operatorining bajarilishi 2.2.- rasmda keltirilgan.



2.2.- if...else shart operatorining bajarilishi

C++ tilining qurilmalari operatorlarni blok ko‘rinishida tashkil qilishga imkon beradi. *Blok* – ‘{‘ va ‘}’ belgilari oralig‘iga olingan operatorlar ketma-ketligi bo‘lib, u kompilyator tomonidan yaxlit bir operator deb qabul qilinadi. Blok ichida e’lon operatorlari ham bo‘lishi mumkin va ularda e’lon qilingan o‘zgaruvchilar faqat shu blok ichida ko‘rinadi (amal qiladi), blokdan tashqarida ko‘rinmaydi. Blokdan keyin ‘;’ belgisi qo‘yilmasligi mumkin, lekin blok ichidagi har bir ifoda ‘;’ belgisi bilan yakunlanishi shart.

```
{ifoda_1; ifoda_2; ...ifoda_n;}
```

Misol tariqasida diskriminantni hisoblash usuli yordamida $ax^2+bx+c=0$ ko‘rinishidagi kvadrat tenglama ildizlarini topish masalasini ko‘raylik:

```
#include <iostream.h>
#include <smath.h>
int main()
{
float a, b, c;
float D, x1, x2;
cout << "ax^2+bx+c=0 tenglama ildizini topish. ";
cout << "\n a - koeffitsiyentini kiriting: ";
cin >> a;
```

```
cout << "\n b - koeffitsiyentini kiriting: ";
cin >> b;
cout << "\n c - koeffitsiyentini kiriting: ";
cin >> c;
D = b * b – 4 * a * c;
if (D < 0)
{
cout << "Tenglama haqiqiy ildizga ega emas!";
return 0;
}
if (D == 0)
{
cout << "Tenglama yagona ildizga ega: ";
x1 = -b / (2 * a);
cout << "\nx= " << x1;
}
else
{
cout << "Tenglama ikkita ildizga ega: ";
x1 = (-b + sqrt(D)) / (2 * a);
x2 = (-b - sqrt(D)) / (2 * a);
cout << "\nx1= " << x1;
cout << "\nx2= " << x2;
}
return 0;
}
```

Dastur bajarilganda, birinchi navbatda, tenglama koeffitsiyentlari - *a*, *b*, *c* o‘zgaruvchilar qiymatlari kiritiladi, keyin diskriminant D o‘zgaruvchi qiymati hisoblanadi. Keyin D qiymatining manfiy ekanligi tekshiriladi. Agar shart o‘rinli

bo‘lsa, yaxlit operator sifatida keluvchi ‘{‘ va ‘}’ belgilari orasidagi operatorlar bajariladi va ekranga “*Tenglama haqiqiy ildizga ega emas!*” xabari chiqadi va dastur o‘z ishini tugatadi (“**return** 0;” operatorini bajarish orqali). Diskriminant noldan kichik bo‘lmasa, navbatdagi shart operatori uni nolga tengligini tekshiradi. Agar shart o‘rinli bo‘lsa, keyingi qatorlardagi operatorlar bloki bajariladi – ekranga “*Tenglama yagona ildizga ega:*” xabari, x_1 hamda o‘zgaruvchi qiymati chop etiladi, aks holda, ya’ni diskriminantning qiymati noldan katta holati uchun **else** kalit so‘zidan keyingi operatorlar bloki bajariladi va ekranga “*Tenglama ikkita ildizga ega:*” xabari hamda x_1 va x_2 o‘zgaruvchilar qiymatlari chop etiladi. Shu bilan shart operatoridan chiqiladi va asosiy funksiyaning return ko‘rsatmasini bajarish orqali dastur o‘z ishini tugatadi.

O‘z navbatida, <*operator*$_1$> va <*operator*$_2$> ham shartli operator bo‘lishi mumkin. Ifodadagi har bir **else** kalit so‘zi, oldindagi eng yaqin **if** kalit so‘ziga tegishli hisoblanadi (xuddi ochiluvchi va yopiluvchi qavslardek). Buni inobatga olmaslik mazmunan xatoliklarga olib kelishi mumkin.

?: shart amali. Agar tekshirilayotgan shart nisbatan sodda bo‘lsa, shart amalining “**?:**” ko‘rinishini ishlatish mumkin:

<shart ifoda> **?** <ifoda$_1$> **:** <ifoda$_2$>;

Shart amali **if** shart operatoriga o‘xshash holda ishlaydi: agar <shart ifoda> 0 qiymatidan farqli yoki *true* bo‘lsa, <ifoda$_1$>, aks holda <ifoda$_2$> bajariladi. Odatda, ifodalar qiymatlari birorta o‘zgaruvchiga o‘zlashtiriladi. Misol tariqasida ikkita butun son maksimumini topish masalasini ko‘raylik:

```
if (a >= b) max = a; else max = b;
```

Dasturni “**?:**” operatori yordamida quyidagicha yozish mumkin:

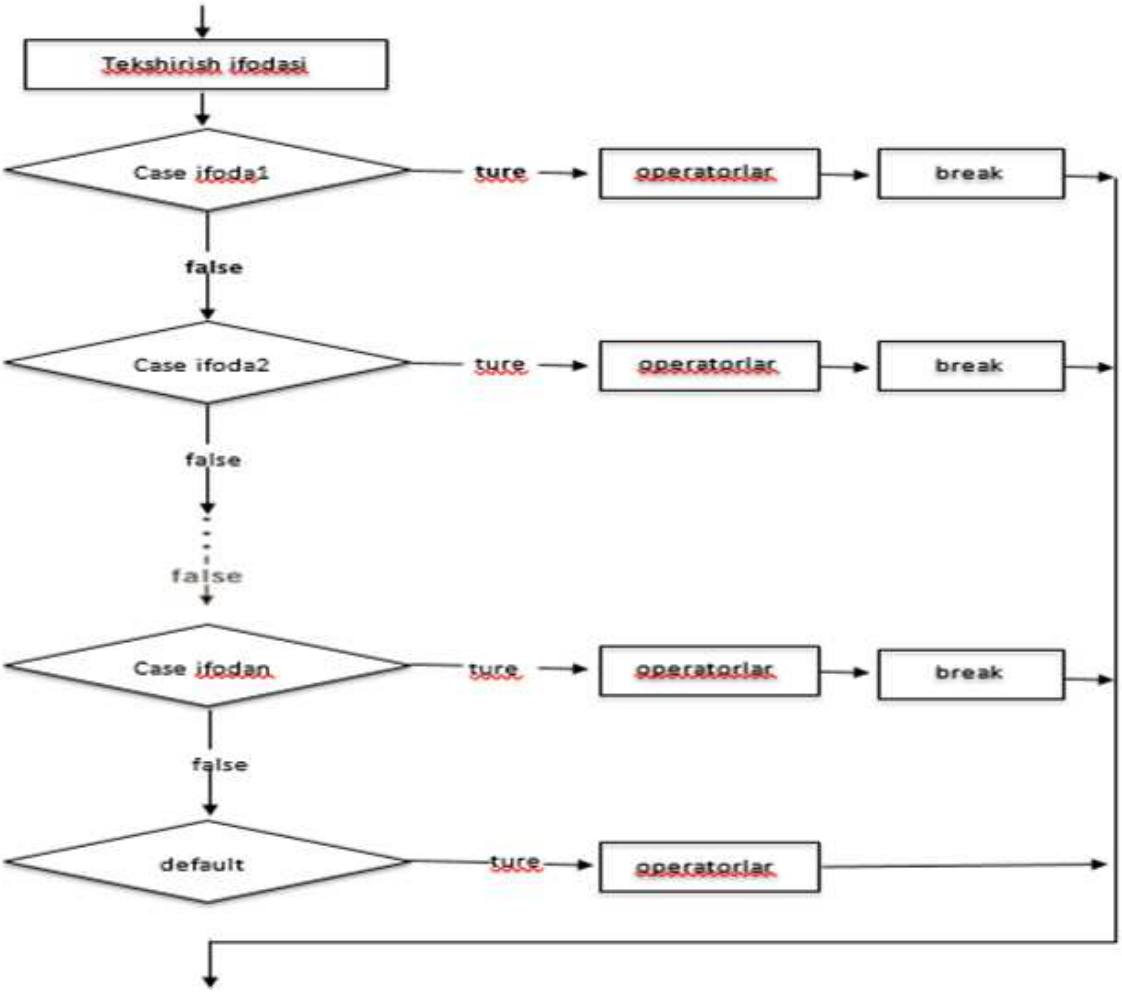
```
max = (a >= b) ? a : b;
```

Dasturdagi shart operatori qiymat berish operatorining tarkibiga kirgan bo‘lib, *a* o‘zgaruvchining qiymati *b* o‘zgaruvchining qiymatidan kattaligi tekshiriladi. Agar shart *rost* bo‘lsa, *max* o‘zgaruvchisiga *a* o‘zgaruvchi qiymatini, aks holda, *b* o‘zgaruvchining qiymatini o‘zlashtiradi va *max* o‘zgaruvchisining qiymati chop etiladi.

2.6. switch operatori

Shart operatorining yana bir ko‘rinishi - **switch** tarmoqlanish operatori bo‘lib, uning sintaksisi quyidagicha [3, 333-334 b.]:

```
switch (<ifoda>)
{
case <o‘zgarmas ifoda1> :
<operatorlar guruhi1>;
break;
case <o‘zgarmas ifoda2> :
<operatorlar guruhi2>;
break;
...
case <o‘zgarmas ifodan> :
<operatorlar guruhin>;
break;
default :
<operatorlar guruhin+1>;
}
```



2.3-rasm. Hisoblash algoritmi

Bu operator quyidagi amallarni bajaradi (2.3-rasm): birinchi navbatda, <*ifoda*> qiymati hisoblanadi, keyin bu qiymat **case** kalit so‘zi bilan ajratilgan <*o‘zgarmas ifoda*$_i$ > bilan solishtiriladi. Agar ular ustma-ust tushsa, shu qatordagi ‘:’ belgisidan boshlab, toki **break** kalit so‘zigacha bo‘lgan <*operatorlar guruhi*$_i$ > bajariladi va boshqaruv tarmoqlanuvchi operatordan keyin joylashgan operatorga o‘tadi. Agar <*ifoda*> birorta ham <*o‘zgarmas ifoda*$_i$ > bilan mos kelmasa, qurilmaning **default** qismidagi <*operatorlar guruhi*$_{i+1}$> bajariladi. Shuni qayd etish kerakki, qurilmada **default** kalit so‘zi faqat bir marta uchrashi mumkin.

2.7. Takrorlash operatorlari

Takrorlash operatori "*takrorlash sharti*" deb nomlanuvchi ifodaning rost qiymatida dasturning ma'lum bir qismidagi operatorlarni (takrorlash tanasini) ko'p marta takror ravishda bajaradi (iterativ jarayon).

Takrorlash shartini tekshirish takrorlash tanasidagi operatorlarni bajarishdan oldin tekshirilishi mumkin (**for**, **while** takrorlashlari) yoki takrorlash tanasidagi operatorlari bir marta bajarilgandan keyin tekshirilishi mumkin (**do-while**) [3, 41-42 b.].

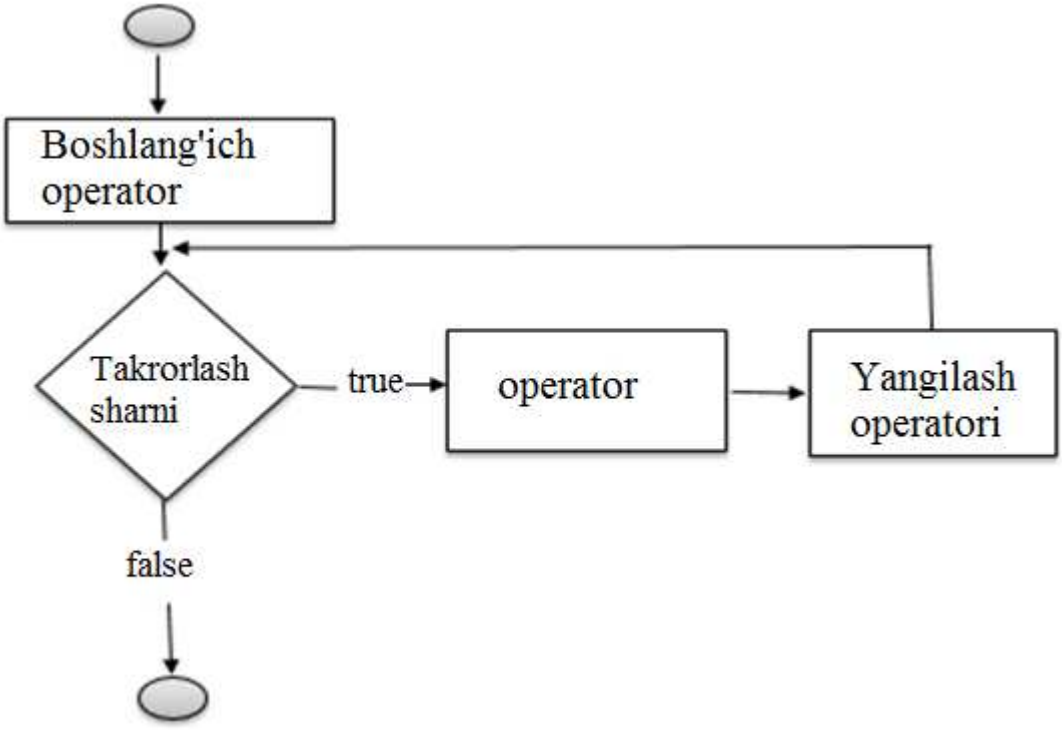
2.7.1. for takrorlash operatori

for takrorlash operatorining sintaksisi qo'yidagi ko'rinishga ega:

for (<*ifoda1*>; <*ifoda2*>; <ifoda3>)

<*operator yoki blok*>;

Uning bajarilishi 2.4 – rasmda keltirilgan.



2.4-rasm. Hisoblash algoritmi

Bu operator o‘z ishini <*ifoda*$_1$> ifodasini bajarishdan boshlaydi. Keyin takrorlash qadamlari boshlanadi. Har bir qadamda <*ifoda*$_2$> bajariladi, agar natija *0* qiymatidan farqli yoki *true* bo‘lsa, takrorlash tanasi - <*operator yoki blok*> bajariladi va oxirida <*ifoda*$_3$> bajariladi. Agar <*ifoda*$_2$> qiymati *0* (*false*) bo‘lsa, takrorlash jarayoni to‘xtaydi va boshqaruv takrorlash operatoridan keyingi operatorga o‘tadi.

Takrorlash operatorlarining qavs ichidagi ifodalariga izoh berish mumkin:

<*ifoda*$_1$> - takrorlash sanagichi vazifasini bajaruvchi o‘zgaruvchiga boshlang‘ich qiymat berishga xizmat qiladi va u takrorlash jarayoni boshida faqat bir marta hisoblanadi. Ifodada o‘zgaruvchi e’loni uchrashi mumkin va bu o‘zgaruvchi takrorlash operatori tanasida amal qiladi va takrorlash operatoridan tashqarida «ko‘rinmaydi»;

<*ifoda*$_2$> - takrorlashni bajarish yoki yo‘qligini aniqlab beruvchi mantiqiy ifoda, agar shart *rost* bo‘lsa, takrorlash davom etadi, aks holda yo‘q. Agar bu ifoda bo‘sh bo‘lsa, shart doimo *rost* deb hisoblanadi;

<*ifoda*$_3$> - odatda takrorlash sanagichining qiymatini oshirish (kamaytirish) uchun xizmat qiladi yoki unda takrorlash shartiga ta’sir qiluvchi boshqa amallar bo‘lishi mumkin.

C++ tilining qurilmalari operatorlarni blok ko‘rinishida tashkil qilishga imkon beradi. *Blok* – ‘{‘ va ‘}’ belgilari oralig‘iga olingan operatorlar ketma-ketligi bo‘lib, u kompilyator tomonidan yaxlit bir operator deb qabul qilinadi. Blok ichida e’lon operatorlari ham bo‘lishi mumkin va ularda e’lon qilingan o‘zgaruvchilar faqat shu blok ichida ko‘rinadi (amal qiladi), blokdan tashqarida ko‘rinmaydi. Blokdan keyin ‘;’ belgisi qo‘yilmasligi mumkin, lekin blok ichidagi har bir ifoda ‘;’ belgisi bilan yakunlanishi shart.

{*operator_1*; *operator_2*; ... *operator_n*;}

1-*misol*. Butun *n* sonining faktoriali hisoblanadigan yana bir misolni ko‘rib

chiqamiz. Faktorial 1dan n gacha bo‘lgan barcha sonlar ko‘paytmasini anglatadi va $n!$ ko‘rinishida yoziladi. Matematik yozuvi quyidagicha:

$$P = \prod_{i=1}^{n} i = 1 * 2 * 3 ... * n$$

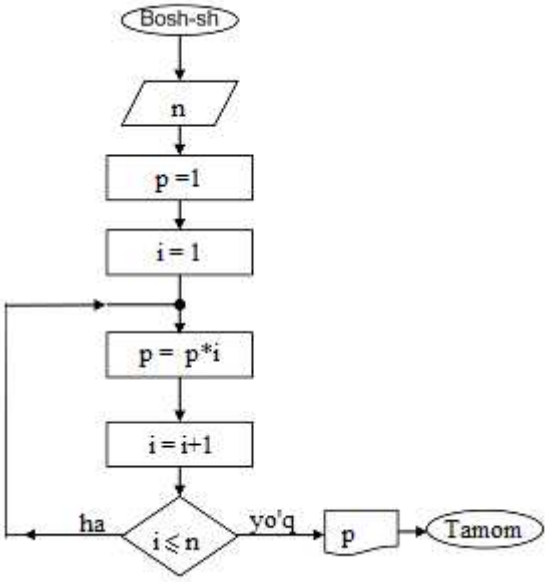
Takrorlanuvchi jarayonni tashkil etish, quyidagidan tashqari, yuqoridagisi bilan bir xil:

- ko‘paytirish jarayoni uchun boshlang‘ich qiymat berilishi;

`p = 1` ko‘rinishiga ega;

- natijani hisoblash `p = p * i` formulasi bo‘yicha amalga oshiriladi.

Shunday qilib, faktorialni hisoblash uchun takrorlanuvchi jarayonni tashkil etish blok-sxemasi quyidagi ko‘rinishga ega (2.5-rasm).



2.5-rasm. Hisoblash algoritmi

C++dasturlash tilidagi dastur:

```
#include <iostream.h>
int main()
{
int n=7;
int i, p;
p=1;
for (i =1; i<= n; i++)
p = p * i;
```

```
cout << ”p=”<<p;;
return 0;
}
```

2-misol. Haqiqiy x sonining n chi darajasini hisoblash $q = x^n$ misolini ko‘rib chiqamiz.

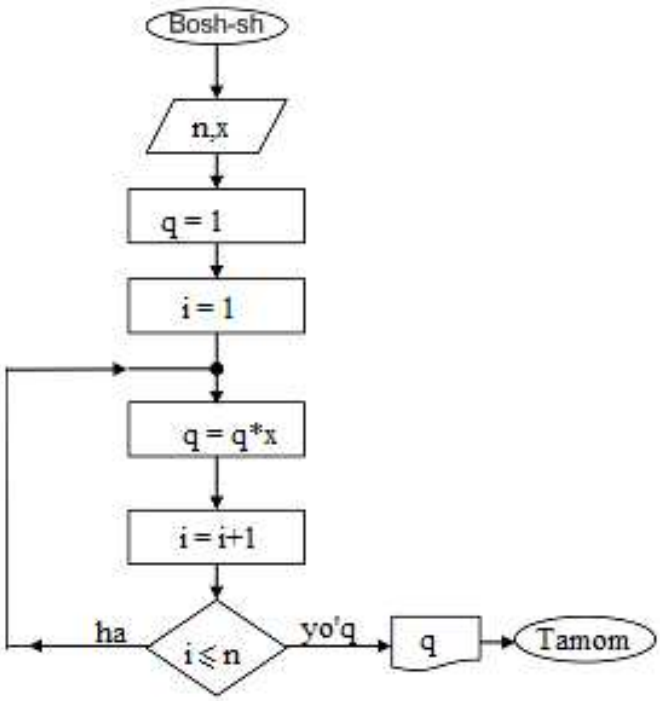
Takrorlanuvchi jarayonni tashkil etish, quyidagidan tashqari, yuqoridagisi bilan bir xil:

- ko‘paytirish jarayoni uchun boshlang‘ich qiymat berilishi

  `q = 1` ko‘rinishiga ega;

- natijani hisoblash `q = q * x` formulasi bo‘yicha amalga oshiriladi.

Shunday qilib, x-ning n chi darajasini hisoblash uchun takrorlanuvchi jarayonni tashkil etish blok-sxemasi quyidagi ko‘rinishga ega (2.6-rasm) .



2.6-rasm. Hisoblash blok-sxemasi

C++ dasturlash tilidagi dastur:

```
#include <iostream.h>
int main()
{
int n=7;
float x,q;
```

```
cin>>x;
q = 1;
for ( i=1;  i<= n;  i++)
q = q * x;
cout <<"q =" << q;
return 0;
}
```

3-misol. Quyidagi ifodani hisoblash kerak bo‘lsin:

$$S = \sum_{i=1}^{n} \frac{x^{i}}{i!}$$

bu ifodani quyidagi ko‘rinishda yozish mumkin:

$$s = x^1/1! + x^2/2! + \cdots + x^n / n!$$

for operatoridan foydalanib, bu jarayonga mos dastur quyidagi ko‘rinishga ega:

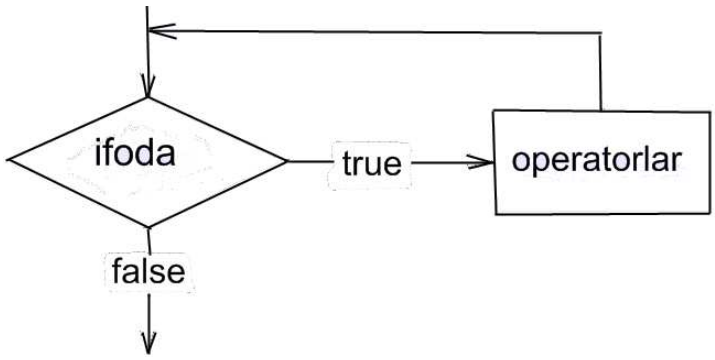
```
#include <iostream.h>
int main()
{
int n =7;
int i, p;
float x, q, s;
cin >> x;
s = 0;
q = 1;
p = 1;
for ( i=1; i <= n;  i++)
{
q = q * x;
p = p * i;
s = s + q / p;
}
cout<<" Miqdori= "<< s;
return 0;
}
```

2.7.2. `while` takrorlash operatori

`while` takrorlash operatori, operator yoki blokni takrorlash sharti yolg‘on (false yoki 0) bo‘lguncha takror bajariladi [3, 396-397 b.]. U quyidagi sintaksisga ega:

```
while (<ifoda>)
<operator yoki blok>;
```

Uning bajarilishi 2.7 - rasmda keltirilgan.



2.7. -rasm. Hisoblash blok-sxemasi

Agar <*ifoda*> rost qiymatli o‘zgarmas ifoda bo‘lsa, takrorlash cheksiz bo‘ladi. Xuddi shunday, <*ifoda*> takrorlash boshlanishida *rost* bo‘lib, uning qiymatiga takrorlash tanasidagi hisoblash ta’sir etmasa, ya’ni uning qiymati o‘zgarmasa, takrorlash cheksiz bo‘ladi.

`while` takrorlash shartini oldindan tekshiruvchi takrorlash operatori hisoblanadi. Agar takrorlash boshida <*ifoda*> *yolg‘on* bo‘lsa, **`while`** operatori tarkibidagi <operator yoki blok> qismi bajarilmasdan cheklab o‘tiladi.

```
i = 0;
while ( i <= 20)
{
cout << i << " ";
i = i + 5;
}
cout << endl;
```

Dastur qismi ishlashi natijasi: 0 5 10 15 20

1-*misol.* Quyidagi ifodani hisoblash kerak bo‘lsin:

$$S = \sum_{i=1}^{n} \frac{x^{i}}{i!}$$

while operatoridan foydalangan holda, bu jarayonga mos dastur quyidagi ko‘rinishga ega:

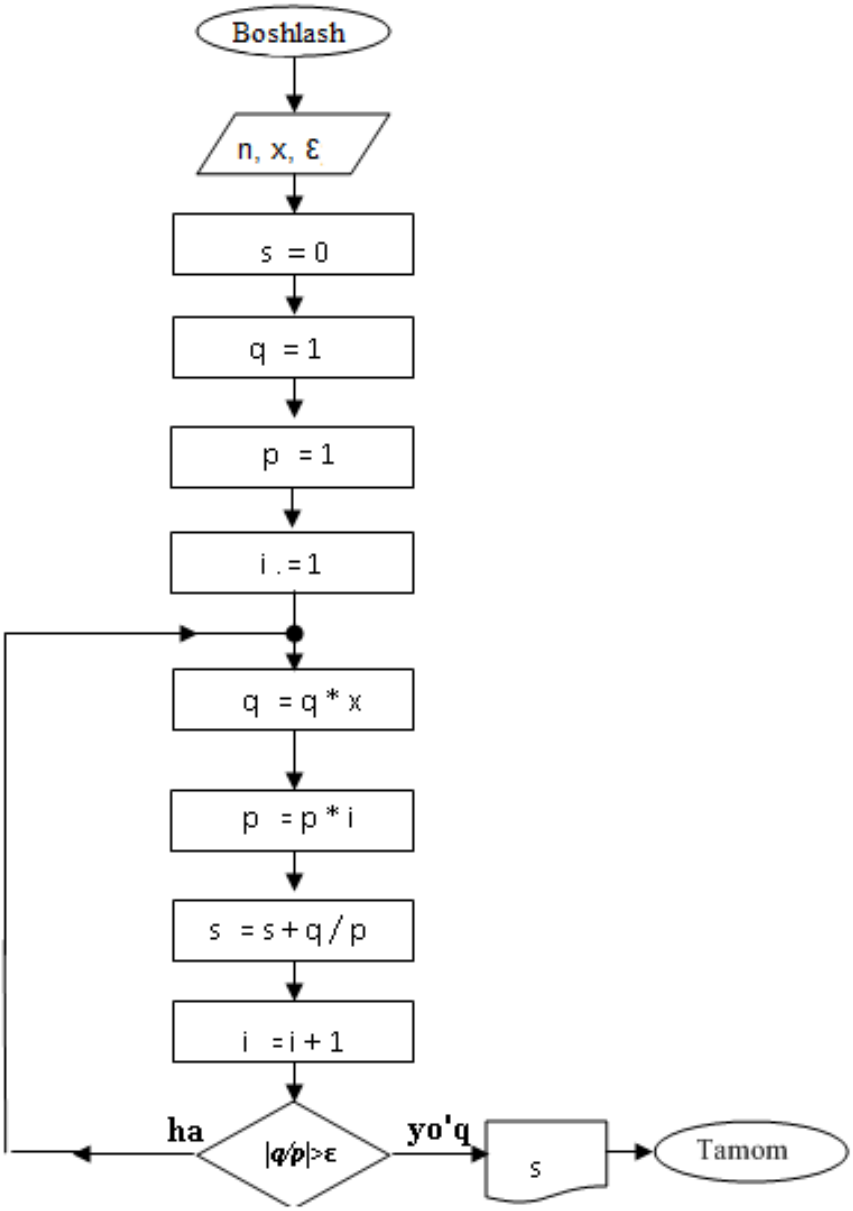
```
#include <iostream.h>
int main()
{
int n=7;
int i, p;
float x, q, s;
cin >> x;
s = 0;
q = 1;
p = 1;
i = 1;
while (  i <= n )
{
q = q * x;
p = p * i;
s = s + q / p;
i = i + 1;
}
cout<<” Miqdori= ” << s;
return 0;
}
```

2-misol. Musbat kichik son ε>0 aniqligida quyidagi munosabatni hisoblang:

$$s = x^1/1! + x^2/2! + \ldots + x^i/i! + \ldots .$$

Misolda cheksiz qatorning i - chi hadining absolyut qiymati ε>0 qiymatidan kichik bo‘lmaguncha yig‘indi davom ettirilishi kerak, ya’ni shart $|x^i/i!| > \varepsilon$ munosabat ko‘rinishida beriladi.

Misolni yechish algoritmining blok-sxemasi quyidagi ko‘rinishga ega (2.8-rasm):



2.8-rasm. Hisoblash algoritmi

while operatoridan foydalangan holda bu jarayonga mos dastur quyidagi ko‘rinishga ega:

```
#include <iostream.h>
int main()
{
int i, p;
float x, q, s, eps;
cin >> x>>eps;
s = 0;
q = 1;
```

```
p = 1;
i = 1;
while ( fabs (q / p) > eps)
{
q = q * x;
p = p * i;
s = s + q / p;
i = i + 1;
}
cout << ”Miqdori=” << s;
return 0;
}
```

while takrorlash operatori yordamida samarali dastur kodi yozishga yana bir misol.

3-*misol.* Ikkita natural sonning eng katta umumiy bo‘luvchisini (EKUB) Evklid algoritmi bilan topish masalasini keltirishimiz mumkin:

```
#include <iostream>
int main()
{
int a,b;
cout << "A va B natural sonlar EKUBini topish.\n";
cout << "A va B natural sonlarni kiriting: ";
cin >> a >> b;
while (a != b)
a > b ? a -= b : b -= a;
cout << "Bu sonlar EKUBi = " << a;
return 0;
}
```

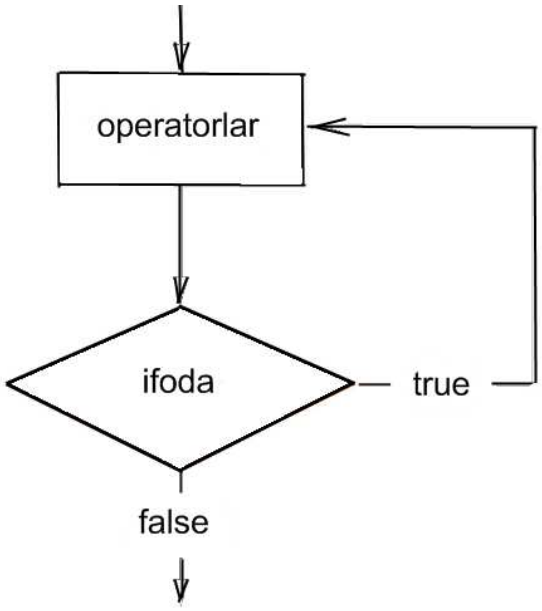
Butun turdagi *a* va *b* qiymatlari oqimdan o‘qilgandan keyin toki ularning qiymatlari o‘zaro teng bo‘lmaguncha takrorlash jarayoni ro‘y beradi. Takrorlashning har bir qadamida *a* va *b* sonlarning kattasidan kichigi ayriladi. Takrorlashdan keyingi ko‘rsatma vositasida a o‘zgaruvchisining qiymati natija sifatida chop etiladi.

2.7.3. `do-while` takrorlash operatori

`do-while` takrorlash operatori **`while`** operatoridan farqli ravishda, oldin operator yoki blokni bajaradi, keyin takrorlash shartini tekshiradi (2.9-rasm). Bu qurilma takrorlash tanasini kamida bir marta bajarilishini ta'minlaydi. **`do-while`** takrorlash operatori quyidagi sintaksisga ega:

```
do
<operator yoki blok>;
while (<ifoda>);
```

Bunday takrorlash operatorining keng qo‘llaniladigan holatlari - bu takrorlash boshlanmasdan turib, takrorlash shartini tekshirishning iloji bo‘lmagan holatlar hisoblanadi. Masalan, birorta jarayonni davom ettirish yoki to‘xtatish haqidagi so‘rovga javob olish va uni tekshirish zarur bo‘lsin. Ko‘rinib turibdiki, jarayonni boshlamasdan oldin bu so‘rovni berishning ma'nosi yo‘q. Hech bo‘lmaganda takrorlash jarayonining bitta qadami amalga oshirilgan bo‘lishi kerak.



2.9-rasm. Hisoblash blok-sxemasi

1-*misol.* Quyidagi ifodani hisoblang:

$$S = \sum_{i=1}^{n} \frac{x^{i}}{i!}$$

`do-while` operatoridan foydalangan holda, bu jarayonga mos dastur quyidagi ko‘rinishga ega:

```
#include <iostream.h>
#include <math.h>
int main()
{
int n=7;
int i, p;
float x, q, s;
cin >> x;
s = 0;
q = 1;
p = 1;
i = 1;
do
{
q = q * x;
p = p * i;
s = s + q / p;
i = i + 1;
while (  i <= n )
}
cout << ”Miqdori=” << s;
return 0;
}
```

2.8. Boshqaruvni uzatish operatorlari

Takrorlash operatorlarining bajarilishida shunday holatlar yuzaga kelishi mumkinki, unda qaysidir qadamda, takrorlash yakuniga yetkazilmay takrorlashdan chiqish zarurati bo‘lishi mumkin. Boshqacha aytganda, takrorlashni «uzish» kerak bo‘lishi mumkin. Bunda **break** operatoridan foydalaniladi. **break** operatorini takrorlash operatori tanasining ixtiyoriy (zarur) joylariga qo‘yish orqali shu joylardan takrorlashdan chiqishni amalga oshirish mumkin [3, 336-337 b.]. E’tibor beradigan bo‘lsak, **switch-case** operatorining tub mohiyatiga ham **break** operatorini qo‘llash orqali erishilgan [3, 397 b.].

Ichma-ich joylashgan takrorlash va **switch** operatorlarida **break** operatori faqat o‘zi joylashgan blokdan chiqish imkoniyatini beradi.

continue operatori xuddi **break** operatoridek takrorlash operatori tanasini bajarishni to‘xtatadi, lekin takrorlashdan chiqib ketmasdan keyingi qadamiga «sakrab» o‘tishini taminlaydi.

Nishon - bu davomida ikki ustma-ust joylashgan nuqta (‘**:**’) qo‘yilgan identifikatordir. Nishon bilan qandaydir operator belgilanadi va keyinchalik dasturning boshqa bir qismidan unga shartsiz o‘tish amalga oshiriladi. Nishon bilan har qanday operator belgilanishi mumkin, shu jumladan, e’lon operatori va bo‘sh operator ham. Nishon faqat funksiyalar ichida amal qiladi.

Nishonga shartsiz o‘tish **goto** operatori yordamida bajariladi [3, 398 b.]. **goto** operatori orqali faqat uning o‘zi joylashgan funksiya ichidagi operatorlarga o‘tish mumkin. **goto** operatorining sintaksisi quyidagicha:

```
goto <nishon>;
```

Shartsiz o‘tish operatori dasturni tuzishdagi kuchli va shu bilan birga, xavfli vositalardan biri hisoblanadi. Kuchliligi shundaki, uning yordamida algoritmning «boshi berk» joylaridan chiqib ketish mumkin. Ikkinchi tomondan, bloklarning ichiga o‘tish, masalan, takrorlash operatorlarining ichiga «sakrab» kirish kutilmagan holatlarni yuzaga keltirishi mumkin.

Garchi nishon yordamida dasturning ixtiyoriy joyiga o‘tish mumkin bo‘lsa ham, boshlang‘ich qiymat berish e’lonlaridan sakrab o‘tish man etiladi, lekin bloklardan sakrab o‘tish mumkin.

1-*misol.* Quyidagi dasturda ikkita natural sonning eng katta umumiy bo‘luvchisini (EKUB) topish masalasidagi takrorlash jarayonini goto operatori vositasida amalga oshirish ko‘rsatilgan:

```
#include <iostream>
int main()
{
int a,b;
cout << "A va B natural sonlar EKUBini topish.\n";
cout << "A va B natural sonlarni kiriting: ";
cin >> a >> b;
nishon:
if (a == b)
```

```
{
cout << "Bu sonlar EKUBi: " << a;
return 0;
}
a > b ? a =a - b : b =b - a;
goto nishon;
}
```

Dasturdagi nishon bilan belgilangan operatorda *a* va *b* sonlarning tengligi tekshiriladi. Agar ular teng bo‘lsa, ixtiyoriy bittasi, masalan, *a* soni EKUB bo‘ladi va funksiyadan chiqiladi. Aks holda, bu sonlarning kattasidan kichigi ayriladi va **goto** orqali ularning tengligi tekshiriladi. Takrorlash jarayoni *a* va *b* sonlar o‘zaro teng bo‘lguncha davom etadi.

2-misol. Tarmoqlanuvchi jarayonni o‘rganishdagi keyingi bosqich shart ikkitadan ortiq bo‘lgandagi holatidir. Bu holat uchun algoritm va dastur yozish zarur bo‘lsin:

$$z=\begin{cases} sin|a+b|, & agar\ a<b; \\ n*tga, & agar\ a=b; \\ a^2-b^2, & agar\ a>b. \end{cases}$$

Bu yerda, $n = 4$ – butun o‘zgarmas son,

a , b - haqiqiy o‘zgaruvchilar:

Bu yozuv quyidagicha o‘qiladi:

agar a<b **bo‘lsa, u holda** z = sin|a+b|, **aks holda**

agar a=b **bo‘lsa, u holda** z = n*tga, **aks holda** $z = a^2 - b^2$.

Topshiriqni yechish blok-sxemasi ko‘rinishi 2.10.-rasmda keltirilgan. Bu algoritmda quyidagi qo‘shimchalar mavjud:

- o‘zgarmas n=4;

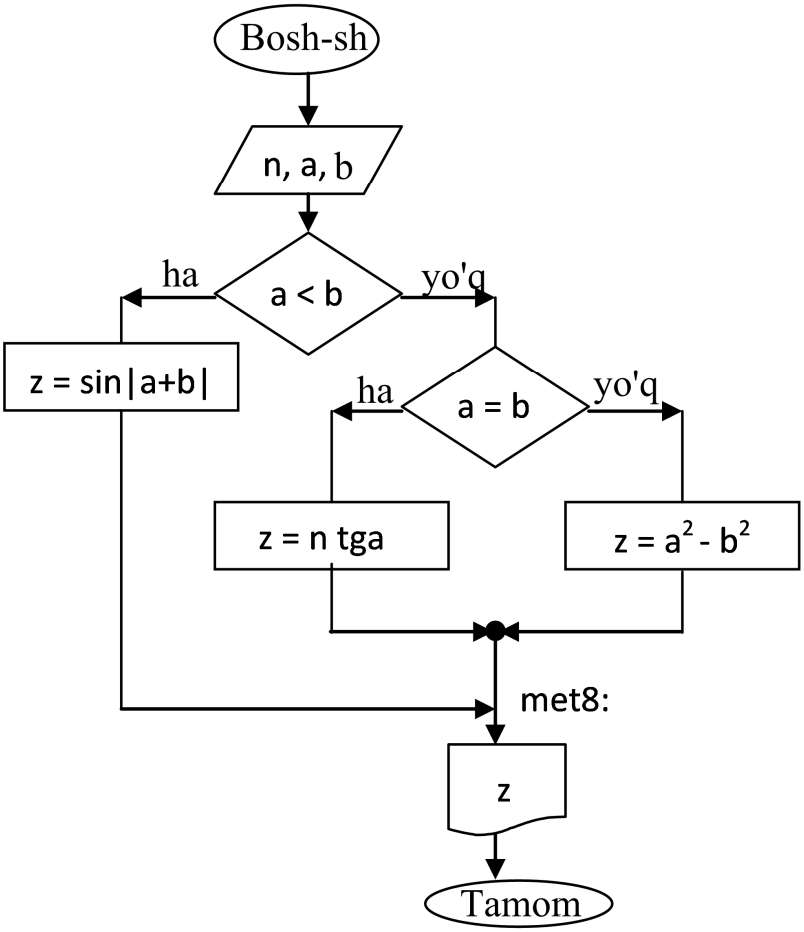
- $a < b$ sharti rost bo‘lganida $z = sin|a+b|$ operatori bajariladi, so‘ng boshqaruv blok-sxemada *met8* belgisi bilan belgilangan chop etish operatoriga uzatiladi.

Birinchi qo‘shimcha uchun dasturda o‘zgarmaslarni aniqlash blokidan foydalaniladi: **const int** *n=4***;** Bu yozuv translyatorga *n* - butun turdagi o‘zgarmas ekanligini va uning qiymati 4 ga tengligini ma’lum qiladi.

Ikkinchi qo‘shimcha:

- chop etish operatori uchun *met8* belgisini qo’uyish;

- belgi buo’ylab shartsiz o’tish operatoridan foydalanish bilan:



2.10-rasm. Hisoblash algoritmi

Bu qo‘shimchalarni e’tiborga olgan holda hisoblash dasturi quyidagi ko‘rinishga ega bo‘ladi.

```
#include <iostream.h>
#include <math.h>
int main()
{
  const int n = 4;
  float a, b, z;
  cin >> a >> b;
  if (a > b) { z = sin(fabs(a+b));  goto met8;}
  if (a==b) z=n*tan(a);
  else z=a*a-b*b;
met8: cout<<”z=”<< z;
  return 0;
```

```
}
```

O‘z navbatida, <operator$_1$> va <operator$_2$> ham shartli operator bo‘lishi mumkin. Ifodadagi har bir **else** kalit so‘zi, oldindagi eng yaqin **if** kalit so‘ziga tegishli hisoblanadi (xuddi ochiluvchi va yopiluvchi qavslardek).

Bu tarmoqlanuvchi algoritmni ichma-ich joylashgan shartli operatorlar ko‘rinishida quyidagicha yozish mumkin:

```
if ( a < b) z = sin(fabs(a+b)); else
if ( a==b) z = n*tan(a); else z = pow (a,2)-pow (b,2);
```

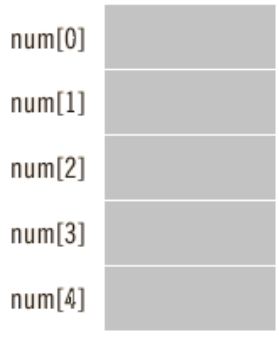
Bunda **if** operatori ichma-ich joylashgan ko‘rinishga ega bo‘ladi.

2.9. Statik massivlar

Massiv - bu fiksirlangan miqdordagi ayrim qiymatlarning (massiv elementlarining) tartiblangan majmuasidir. Barcha elementlar bir xil turda bo‘lishi kerak va bu tur element turi yoki massiv uchun tayanch tur deb nomlanadi. Dasturda ishlatiladigan har bir konkret massiv o‘zining individual nomiga ega bo‘lishi kerak. Bu nom to‘liq o‘zgaruvchi deyiladi, chunki uning qiymati massivning o‘zi bo‘ladi. Massivning har bir elementi massiv nomi hamda kvadrat qavsga olingan va element selektori deb nomlanuvchi indeksni ko‘rsatish orqali oshkor ravishda belgilanadi [4, 486-494 b.].

Murojaat sintaksisi: <*massiv nomi* >[<*indeks*>]

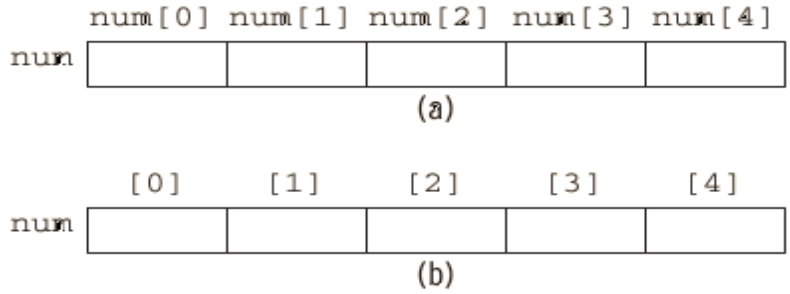
Bu ko‘rinishga xususiy o‘zgaruvchi deyiladi, chunki uning qiymati massivning alohida elementidir. Massivining alohida elementlariga indeksli o‘zgaruvchilar orqali murojaat qilish mumkin (2.11-rasm).



2.11-rasm. Massiv elementlari ketma-ketligi

Massiv indeksi sifatida butun son qo‘llaniladi. Umuman olganda, indeks sifatida butun son qiymatini qabul qiladigan ixtiyoriy ifoda ishlatilishi mumkin va uning qiymati massiv elementi nomerini aniqlaydi. Dasturdagi bir indeksli o‘zgaruvchi orqali massivning barcha elementlarini belgilash (aniqlash) mumkin bo‘ladi.

Massivning elementlariga murojaat indekslari orqali bo‘ladi. Indeks sifatida butun turdagi o‘zgaruvchilardan foydalanish mumkin.



C++ tilida massiv indeksi doimo 0 dan boshlanadi va uning eng katta qiymati massiv e’lonidagi uzunlikdan bittaga kam bo‘ladi.

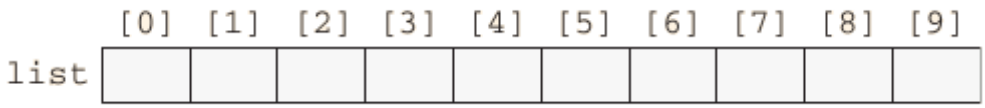
Massiv e’loni quyidagicha bo‘ladi:

<*tur*><*nom*> [<*uzunlik*>]={*boshlang‘ich qiymatlar}*.

Bu yerda <*uzunlik*> - o‘zgarmas ifoda (konstanta).

Misol: **int** *list* [10];

Bu yerda *list* nomli massiv elementlari 10 ta bo‘lsa, uning elementlari `list[0], list[1], list[2],…, list[9]` bo‘ladi, boshqacha aytganda massiv elementlari 10 ta:



Agar `list[5]=34;` bo‘lsa, 34 qiymat massivning 5-e’lementiga joylashtiriladi:

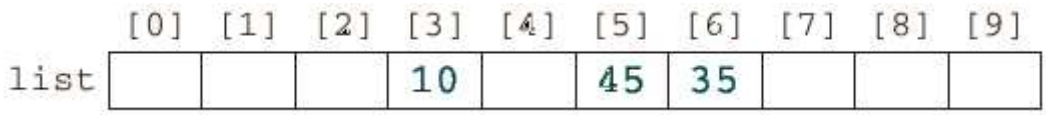
Bu yerda *i* indeks **-** butun turdagi o‘zgaruvchi: *list*[3]=63;

Buni quyidagicha tushunish mumkin *i*=3; *list*[*i*]=63;

Quyidagi misollarni ko‘raylik:

```
list [3]=10;
list [6]=35;
list [5]= list [3] + list [6];
```

Yuqoridagi misolda birinchi ***list*** massivining uchinchi e’lementiga 10 qiymatini o‘zlashtiradi massivning oltinchi elementiga 35 qiymatini o‘zlashtiradi va massivning uchinchi va oltinchi elementlari yig‘indisini massivning beshinchi elementiga yuklaydi:



Odatda massiv quyidagicha e’lon qilinadi:

```
const int  n = 10;
int list [n];
```

Bu yerda birinchi o‘rinda butun turdagi o‘zgarmas e’lon qilingan, so‘ng massiv e’lon qilinib, o‘lchamlari o‘rnatilgan.

C++ tilida massivlar elementining turiga cheklovlar qo‘yilmaydi, lekin bu turlar chekli o‘lchamdagi obyektlarning turi bo‘lishi kerak.

Ikki o‘lchamli massivning sintaksisi quyidagi ko‘rinishda bo‘ladi:

<tur> <nom> [<uzunlik >] [<uzunlik>];

Masalan, 10x5 o‘lchamli haqiqiy sonlar massivining e’loni:

```
float a [10][5];
```

E’lon qilingan ***a*** massivning ko‘rinishi quyidagi 2.12-rasmda keltirilgan.

	[0]	[1]	[2]	[3]	[4]
[0]					
[1]					
[2]					
[3]					
[4]					
[5]					
[6]					
[7]					
[8]					
[9]					

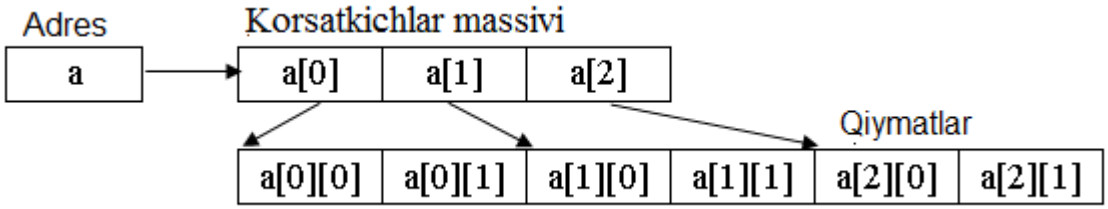
2.12 rasm. Massiv umumiy ko‘rinishi

$$
\begin{array}{cl}
 & \qquad\qquad\qquad\qquad\qquad\qquad\quad j \\
 & a_0: \ (a_{0\,0}, \ a_{0\,2} \ \ldots \ \ldots \ a_{0\,18}, \ a_{0\,19}), \\
 & a_1: \ (a_{1\,0}, \ a_{1\,1}, \ \ldots \ \ldots \ a_{1\,18}, \ a_{1\,19}), \\
 & \ldots \\
i & a_i: \ (\ldots, \ \ldots, \ \ldots \ a_{ij} \ \ldots \ \ldots, \ \ldots), \\
 & \ldots \\
 & a_9: \ (a_{9\,0}, \ a_{9\,1}, \ \ldots \ \ldots \ a_{9\,18}, \ a_{9\,19}).
\end{array}
$$

E’ndi adres nuqtai nazaridan ko‘p o‘lchamli massiv elementlariga murojaat qilishni ko‘raylik. Quyidagi e’lonlar berilgan bo‘lsin:

```
int a [3][2];
float b [2][2][2];
```

Birinchi e’londa ikki o‘lchamli massiv, ya’ni 3 satr va 2 ustundan iborat matritsa e’lon qilingan, ikkinchisida uch o‘lchamli massiv e’lon qilingan. ***a*** massivining elementlariga murojaat sxemasi:



Massivlar ustida bajariladigan asosiy amallar berilganlarni massiv elementlari yuklash, massiv elementlari ustida amallar bajarish va massiv elementini yig‘indisini, o‘rta arifmetigini, massiv elementlarini chop qilish va boshqa amallarni bajarish mumkin. Bunda massivnig har bir elementiga murojaat qilishga to‘g‘ri keladi, buni boshqarish oson. Misol sifatida massiv e’loni quyidagicha bo‘lsin:

```
const int  n = 100;
int list [n];
```

Quyidagi takrorlash operatori orqali massivning har bir elementiga murojaat qilishimiz mumkin bo‘ladi va murojaat massivning birinchi elementidan boshlanadi:

```
for ( i = 0;  i < n;  i++)
```

Massiv elementlari ustida amallar bajarishimiz uchun berilganlarni massivning har bir elementiga o‘qib olishimiz kerak. Bu *cin* operatori orqali amalga oshiriladi. Misol sifatida quyidagi ifoda massivning *n* =100 ta elementlarini o‘qib oladi:

```
for  ( i = 0;  i < n;  i++) cin >> list[i];
```

a. Massivni initsializatsiya qilish: Quyida *s* massivning har bir elementiga 0.0 qiymat bilan initsializatsiya qilinmoqda:

```
for  ( i=0;  i<n;  i++)  s[i]=0.0;
```

b. Massiv elementlarini o‘qib olish: quyidagi misolda klaviaturadan kiritilayotgan berilganlarni `s` massivining har bir elementiga o‘qib olinmoqda:

```
for  ( i = 0;  i < n;  i++) cin >> s [i];
```

c. Massiv elementlarini chop qilish: quyidagi ifodada massivning har bir elementi probel belgisi yordamia ajratilib chop qilinmoqda:

```
for ( i = 0; i < n; i++) cout << s [i]<< " ";
```

d. Massivning elementlari yig‘indisi va massiv elementlarining o‘rta arifmetigini topish: Bunday holda butun turdagi *sum* o‘zgaruvchisi olinib, unga *0* berib qo‘yiladi va takrorlash operatori orqali indeks o‘zgaruvchisi *0* dan *n-1* gacha

o‘zgaradi. Massivning har bir elementini *sum* o‘zgaruvchisiga yig‘ib boramiz. Natijani *n* ga bo‘lsak, o‘rta arifmetigi kelib chiqadi:

```
int  i, n=10;
float d, sum = 0;
for ( i = 0;  i < n;  i++)
sum = sum + s [i];
d = sum / n;
```

1-*misol*. Massiv elementlaridan eng kattasini topish [2, 214 b.]. Bu misolda butun turdagi *i* o‘zgaruvchisini olib, unga 0 berib qo‘yamiz va takrorlash operatorini 1 dan boshlaymiz va massivning nolinchi indeksli elementi bilan qolgan o‘rindagi elementlarni solishtirib chiqamiz, agar undan katta element chiqib qolsa, *max* o‘zgaruvchisiga katta element qiymatini olamiz va tekshirishni davom ettiramiz. Tekshirish natijasida bizda eng katta element qiymati hosil bo‘ladi va uni chop qilamiz:

```
max = s [0];
for ( i = 1; < n;  i++)
if ( max < s [i] ) max = s[i];
```

Bu algoritm massivning quyidagi qiymatlarida tekshirib ko‘raladi:

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
S	12.50	8.35	19.60	25.00	14.00	39.43	35.90	98.23	66.65	35.64

Har bir qadam bajarilishini ko‘ramiz:

index	max	max	s[i]	max<s[i]
1	0	12.50	8.35	12.50 < 8.35 false;
2	0	12.50	19.60	12.50 < 19.60 true; max = 19.60
3	2	19.60	25.00	19.60 < 25.00 true; max = 25.00
4	3	25.00	14.00	25.00 < 14.00 false;
5	3	25.00	39.43	25.00 < 39.43 true; max = 39.43
6	5	39.43	35.90	39.43 < 35.90 false
7	5	39.43	98.23	39.43 < 98.23 true; max = 98.23
8	7	98.23	66.45	98.23 < 66.65 false
9	7	98.23	35.64	98.23 < 35.64 false

for operatori bajarilgandan so‘ng massivning eng katta elementi *max* = 98.23 ga teng bo‘ladi.

2-misol. Biz massiv e’lon qilish, massiv elementlarini o‘qib olish, massiv elementlaridan kattasini topish va elementlari yig‘indisini hisobsh algoritmlarini ko‘rdik. Shundan so‘ng, C++ da dasturini ifodalaymiz:

```
#include <iostream>
int main()
{
const int  n = 5; //Massiv elementlari beshta bo‘lsin
int item [n];
int sum;
int  i;
cout << "Beshta sonni kiriting: ";
sum = 0;
for ( i = 0; i < n;  i++)
{
cin >> item [i];
sum = sum + item[i];}
cout << endl;
cout << "Elementlarning yig‘indisi: " << sum << endl;
cout << " Qiymatlarni teskari tartibda chop qilish: ";
for  ( i = n-1;  i >= 0;  i--)
cout << item [i]<< " ";
cout << endl;
return 0;
}
```

Dasturni ishga tushiramiz.

Beshta qiymatni kiritaylik: 12 76 34 52 89

Massiv elementlari yig‘indisi: 263

Massiv elemetlarini teskari tartibda chop etish: 89 52 34 76 12.

3-*misol*. Massivlar elementlarini qayta ishlash bo‘yicha masalalarni yechishni o‘rganishdagi keyingi qadam - bu $B = \{b_i\}$ massivining maksimal

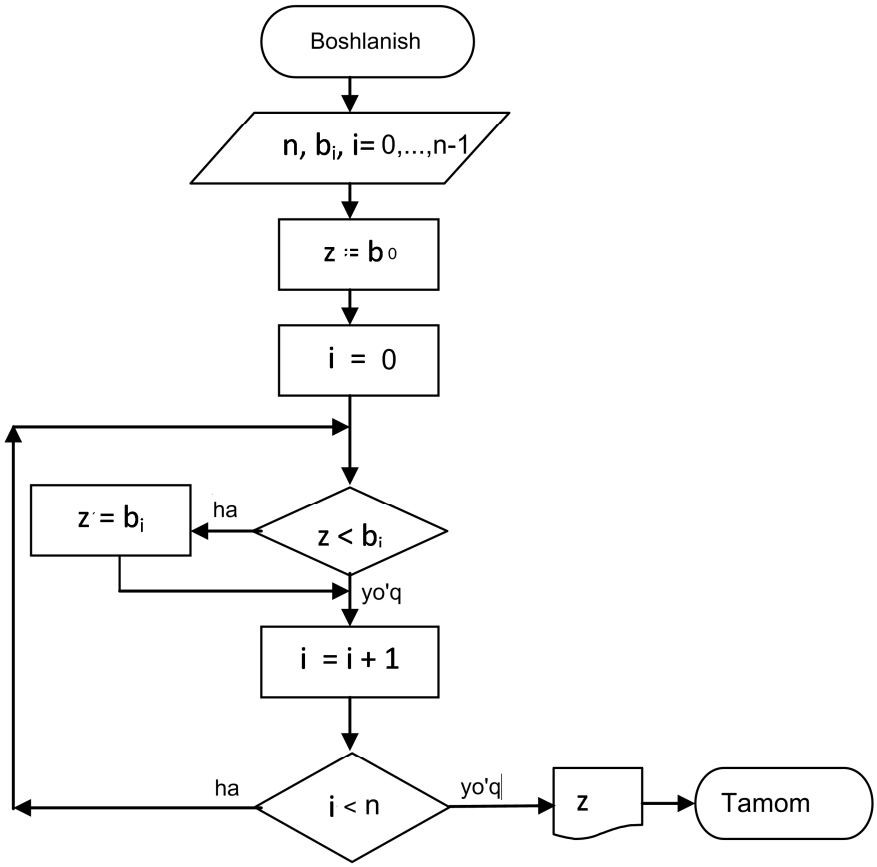
(minimal) elementi va uning o‘rnini (indeksini) aniqlash bilan bog‘liq masalani ko‘rib chiqishdir.

Mazkur masalaning matematik ifodasi quyidagi ko‘rinishga ega:

$$z = \max_{0 \le i < n} b_i , \quad \text{m} = 8;$$

Massiv elementlari ichida maksimal elementni aniqlash uchun quyidagi tadbirni amalga oshirish zarur. Avval massivning birinchi elementini maksimal element deb taxmin qilamiz. So‘ng taxmin qilingan maksimal element boshqa elementlar bilan solishtiriladigan takrorlash jarayonini tashkil etamiz. Agar massivning keyingi elementi maksimal deb belgilangan elementdan katta bo‘lsa, bu element maksimal deb belgilanadi. Takrorlashning yakunida o‘zgaruvchining qiymati maksimal elementga muvofiq keladi.

Maksimal elementni topish algoritmi blok-sxemasi quyidagi ko‘rinishga ega (2.13-rasm).



2.13-rasm. Hisoblash blok-sxemasi

Bu blok-sxemaga mos keluvchi dastur quyidagi ko‘rinishga ega:

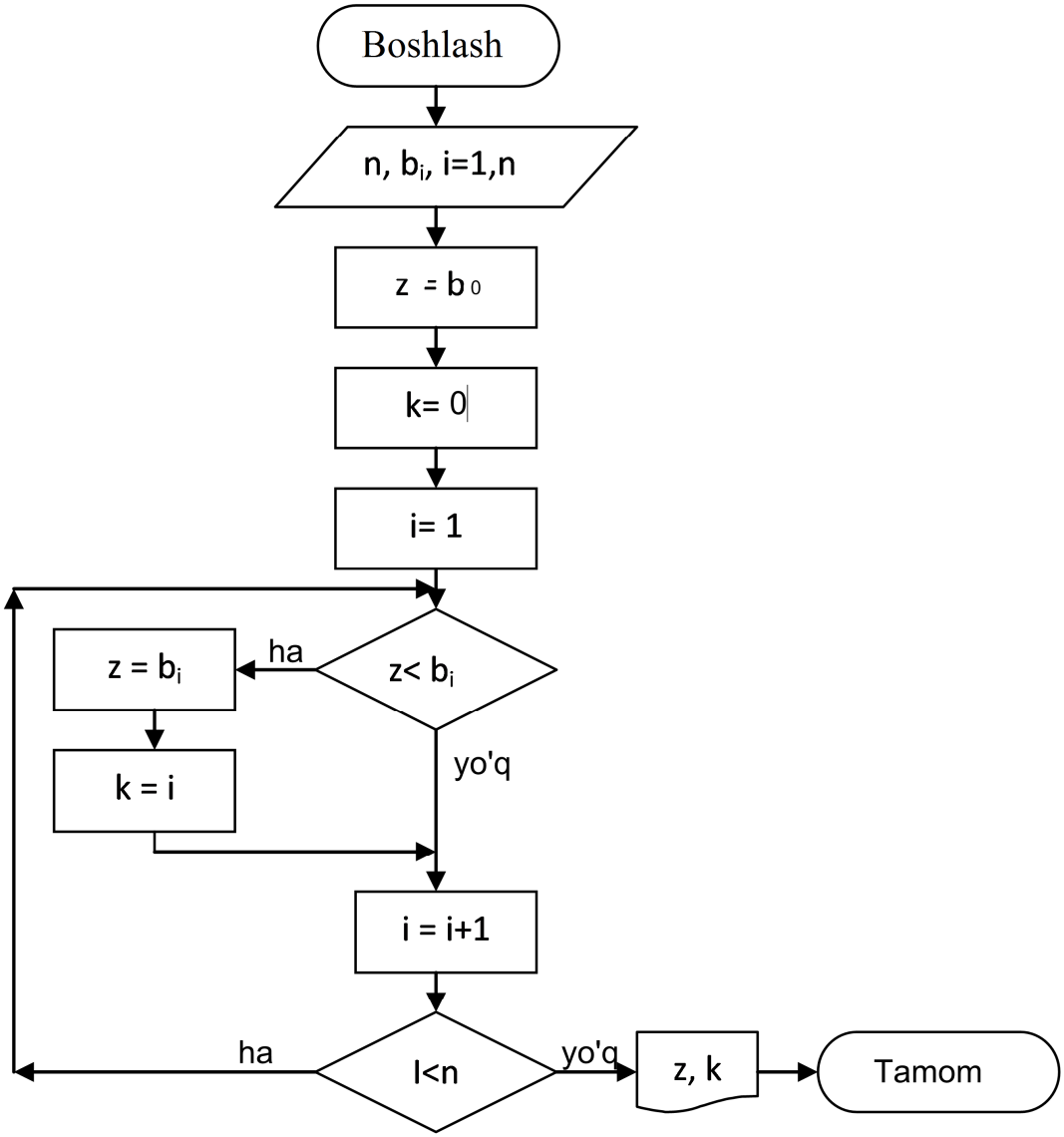
```
#include <iostream.h>
int main()
{
const int n=8;
int b[n],z;
for(int i=0; i<n; i++)
cin>>bi];
z=b[0];
for(int i=1;i<n;i++)
 if(z<b[i]) z=b[i];
cout<<” max= ” << z;
return 0;
}
```

Minimal elementni aniqlash uchun munosabat belgisi “**<** “kichikni “**>**” kattaga o‘zgartirishning o‘zi kifoya.

4-*misol*. Massivning maksimal elementi joylashgan joyni, ya’ni uning indeksini aniqlash uchun algoritmga ko‘rib chiqiladigan elementning indeksini belgilaydigan o‘zgaruvchini boshqarish operatoriga qo‘shishning o‘zi kifoya:

1) $k = 1$ (birinchi elementni maksimal deb taxmin qilamiz);

2) $k = i + 1$ (agar ko‘rib chiqilayotgan element taxmin qilinayotgan maksimumdan katta bo‘lsa, u ko‘rib chiqilgan elementlar ichida maksimali bo‘ladi). Qo‘shimchalarni hisobga olgan holda blok-sxemasi keltiramiz (2.14 - rasm).



2.14-rasm. Hisoblash algoritmi

Bu algoritmga mos keluvchi dastur quyidagi ko‘rinishga ega:

```
#include <iostream.h>
int main()
{
const int n=8;
int b[n],z,k=0;
for(int i=0; i<n; i++)
cin>>b[i];
z=b[0];
for (int i=1; i<n; i++)
if (z<b[i]) {z=b[i]; k=i+1;}
cout<<”max=”<< z <<” k=”<< k;
return 0;
}
```

5-*misol*. Massivning elementlarini kamayish tartibida joylashtirish algoritmi va dasturini yaratish uchun yuqorida keltirilgan massiv elementlari ichida maksimal qiymatli elementi va uning indeksini aniqlash algoritmidan foydalaniladi va quyidagi amallar ketma-ketligi bajariladi:

1) i=1;
2) massivning i-chidan to n-chi elementlari orasidagi eng katta elementi - z va uning indeksi - k aniqlanadi;
3) “uch likopcha” usuli asosida i-chi va maksimal qiymatli element joyma-joy almashtiriladi: $c=b[i]$; $b[i]= z$; $b[k]=c$, bunda c - yordamchi o‘zgaruvchi;
4) $i=i+1$;
5) agar $i<n$ bo‘lsa, u holda =>(2).

Natijada $b=\{b_i\}$ – massivda a massiv elementlari kamayish tartibida joylashtiriladi.

Bu algoritmga mos keluvchi dastur quyidagi ko‘rinishga ega.

```
#include <iostream.h>
int main()
{
const int n=8;
int k,z,c,max;
int b[n];
for (int i=0; i<n; i++)
cin>>b[i];
for (int i=0; i<n-1; i++)
{
z=b[i]; k= i;
for (int j=i+1; j<n; j++)
if (z<b[j]) {z=b[j]; k=j;}
c=b[i]; b[i]= z; b[k]=c;    //uch likopcha usuli
}
for (int i=0; i<n; i++)
cout<<b[i]<<”  ”;
return 0;
}
```

6-*misol*. Vektorlarning skalyar ko‘paytmasini hisoblash masalasi.

Vektorni vektorga skalyar ko‘paytmasi – s=A*B tasvirlanadi.

Bu yerda: $A = \{ a_i \}, B = \{ b_i \}, 0 \le i < n$, s – skalyar.

Hisoblash formulasi:

$$s = \sum_{i=0}^{n-1} a_i * b_i$$

Mos dastur matni:

```
#include <iostream.h>
int main()
{
const int n=6;
int  i;
float  s;
float a[n], b [n];
for ( i=0;  i < n;  i++)
cin >> a  [i], b [ i];
s = 0;
for ( i=0; i < n; i++)
s=s+a[i] * b[i];
cout << ”s=”, <<s;
return 0;
}
```

7-*misol*. Matritsani vektorga ko‘paytmasi – C=A*B ni hisoblash masalasini ko‘raylik [2, 75-76 b.]. Bu yerda:

$A=\{a_{ij}\}$, $b=\{b_j\}$, $c=\{c_i\}$, $0 \le i < m$, $0 \le j < n$.

Hisoblash formulasi:

$$c_i = \sum_{j=0}^{n-1} a_{ij} b_j$$

Mos dastur matni:

```
int  main()
{
```

```
const int n=4, m=5;
float a[m][n], b[n],c[m];
int  i, j; float s;
for ( i=0; i<m; i++)
for ( j=0; j<n; i++) cin >>a [i][j];
for ( i=0; i<m; i++) cin>>b[i];
for ( i=0; i<m; i++)
{
s=0;
for ( j=0; j<n; j++)
s= s + a[i][j]*b[j];
c [i] = s;
}
for ( i=0; i<m; i++)
cout<<"\t c"<<i<<"]="<<c[i];
return 0;
}
```

8-*misol*. Matritsani matritsaga ko‘paytmasi – C=A*B ni hisoblash masalasi ko‘riladi [3, 277-278 b.].

Bu yerda: $A=\{a_{ik}\}$, $B=\{b_{kj}\}$, $C=\{c_{ij}\}$, $0 \le i<m$, $0 \le j<n$, $0 \le i< l$.

Hisoblash formulasi:

$$c_{ij} = \sum_{k=0}^{l-1} a_{ik} * b_{kj}$$

Mos dastur matni:

```
#include <iostream.h>
int main()
{
const int n = 3; m = 4; l = 2;
int  i, j, k;
float  s;
```

```
float a [n][m], b [m][l], c [n][l];
for ( i = 0;  i < n;  i++)
for ( j = 0; i < m;  i++)
cin >> a[i][j];
for ( j = 0;  j < m; j++)
for ( k = 0; k < l; k++)
cin >> b [j][k];
for ( i = 0; i < n;  i++)
for ( k = 0; k < l;  k++)
{
s = 0;
for ( j=0; j < m; j++)
s = s + a[i][j] * b[j][k];
c [i] [k]=s;
}
for ( i = 0; i < n; i++)
for ( k = 0; k < l; k++)
cout<< c[i][k];
return 0;
}
```

9-m*isol*. A={a_{ij}} matritsa sart elementlari ko‘paytmalarining yig‘indisini hisoblash algoritmini tuzish talab qilinsin. Bu masalaning matematik modeli quyidagicha ko‘rinishga ega [5, 114-115 b.]:

$$S = \sum_{i=0}^{n-1} \prod_{j=0}^{m-1} a_{ij}.$$

Mos dastur matni:

```
#include <iostream.h>
int main()
{
const int n=3; m=4;
int  i, j;
float  s,p;
float a [n] [m];
for ( i=0;  i < n;  i++)
```

```
for ( j =0; i < m;  i++)
cin >> a[i][j];
s = 0;
for ( i = 0;  i < n;  i++)
{
p =1;
for ( j=0;  j < m; j++)
p = p * a [i][j];
s = s + p;
}
cout<< ”s=”, s;
return 0;
}
```

10-*misol*. A={a_{ij}} matritsaning “egar” nuqtasini aniqlang. Matritsaning “egar” nuqtasi deganda bir vaqtda *i*-chi satr elementlari ichida eng katta va j-chi ustun elementlari ichida eng kichik bo‘lgan a_{ij} elementidir. Agar matritsa elementlari har xil kiymatli bo‘lsa, u holda “egar” nuqtasi yagona bo‘ladi yoki mavjud emas. Demak, masalaning yechish algoritmi, avvalo, tashqi takror jarayonida har bir *i*-satr bo‘yicha eng katta elementining ustun indeksi aniqlanib, shu ustun elementlar ichida eng kichik elementining indeksi *k = i* ga tengligi tekshirishdan iborat bo‘ladi. Agar bu shart hech bir shartda bajarilmasa, demak bu matritsada “egar” nuqta mavjud emas.

Jarayon quyidagi amallar ketma-ketligida bajariladi:

1) *kiritish (n, m, $a_{i\,j}$)*
2) *p1=false;*
3) *i=1;*
4) *t=0;*
5) *p=$a_{i\,1}$;*
6) *k=1*
7) *j=2;*
8) *agar p < $a_{i\,j}$ bo‘lsa, u holda { p = $a_{i\,j}$; k = j };*

9) j=j+1;

10) agar j <= m bo‘lsa, u holda =>(8);

11) i=i+1;

12) agar i <= n bo‘lsa, u holda =>(4);

13) l=1;

14) agar p < a_{lk} bo‘lsa, u holda t=t+1;

15) agar (t = n) bo‘lsa, u holda {p1=true; muhrlash (i, k, p)}.

16) l=l+1;

17) agar (l <= n) bo‘lsa, u holda =>(14);

18) agar (p1 = false) u holda muhrlash (egar nuqta yo’q).

Bu algoritmga mos dasturning ko‘rinishi:

```
#include <iostream.h>
#include <conio.h>
#include <vcl.h>
#pragma hdrstop
#pragma argsused
int main()
{
const int n=3,m=3;
int a[n][m], p, t, k, p1=0;
for (int i=0; i<n; i++)
for (int j=0; j<m; j++) cin>>a[i][j];
for (int i=0; i<n; i++)
{
t=0;
p=a[i][0];k=0;
for (int j=1;j<m;j++)
if (p<a[i][j]){p=a[i][j];k=j;
}
```

```
for (int l=0;l<n;l++)
if (p<a[l][k])t=t+1;
if (t==n-1){p1=1; cout<<i<<"   "<<k<<"    "<<p;}
}
if (!p1)cout << " e'gar nuqta yoq ";
getch();
return 0;
}
```

2.10. Funksiyalar bilan ishlash

Funksiya bu – C++ tilida masala yechishdagi kalit elementlardan biridir. Funksiyalar modullar deb ham ataladi. Funksiyalar oldindan aniqlangan va foydalanuvchi tomonidan aniqlanadigan funksiyalarga bo‘linadi.

Oldindan aniqlangan funksiyalar, asosan, tilning turli kutubxona fayllari orqali aniqlanadi. Ularga matematik funksiyalar, turlarni tekshirish funksiyalari, belgi va satrlar bilan ishlash funksiyalari misol bo‘ladi. Masalan:

Funksiya ishlatilishi	Kutubxona fayli	Bajaradigan amali
`abs(x)`	`<cmath>`	*x* butun sonining absolyut qiymatini qaytaradi
`fabs(x)`	`<cmath>`	*x* haqiqiy sonining absolyut qiymatini qaytaradi
`log(x)`	`<cmath>`	*x* sonining natural logarifmini qaytaradi
`pow(x, y)`	`<cmath>`	x^y hisoblaydi
`sqrt(x)`	`<cmath>`	*x* sonining kvadrat ildizini qaytaradi
`islower(x)`	`<cctype>`	*x* qiymatining kichik harfligini tekshiradi
`isupper(x)`	`<cctype>`	*x* qiymatini katta harfligini tekshiradi
`tolower(x)`	`<cctype>`	*x* qiymatini kichik harf ko‘rinishiga aylantiradi
`toupper(x)`	`<cctype>`	*x* qiymatini katta harf ko‘rinishiga aylantiradi

Dasturda ishlatiladigan har qanday foydalanuvchi tomonidan aniqlanadigan funksiyalar e'lon qilinishi kerak. Funksiyalar qiymat qaytaruvchi va qiymat qaytarmaydigan funksiyalar ko'rinishida bo'ladi.

Odatda funksiyalar e'loni sarlavha fayllarda e'lon qilinadi va **#include** direktivasi yordamida dastur matniga qo'shiladi.

Funksiya e'lonini *funksiya prototipi* tavsiflaydi (ayrim hollarda *signatura* deyiladi). Funksiya prototipi quyidagi ko'rinishda bo'ladi:

<*qaytaruvchi qiymat turi*> <*funksiya nomi*>(<*parametrlar ro'yxati* >);

Bu yerda <*qaytaruvchi qiymat turi*> - funksiya ishlashi natijasida u tomonidan qaytaradigan qiymatning turi. Agar qaytariladigan qiymat turi ko'rsatilmagan bo'lsa, kelishuv bo'yicha funksiya qaytaradigan qiymat turi **int** deb hisoblanadi, <*parametrlar ro'yxati*> - vergul bilan ajratilgan funksiya parametrlarining turi va nomlari ro'yxati. Parametr nomini yozmasa ham bo'ladi. Ro'yxat bo'sh bo'lishi ham mumkin. Funksiya prototiplariga misollar:

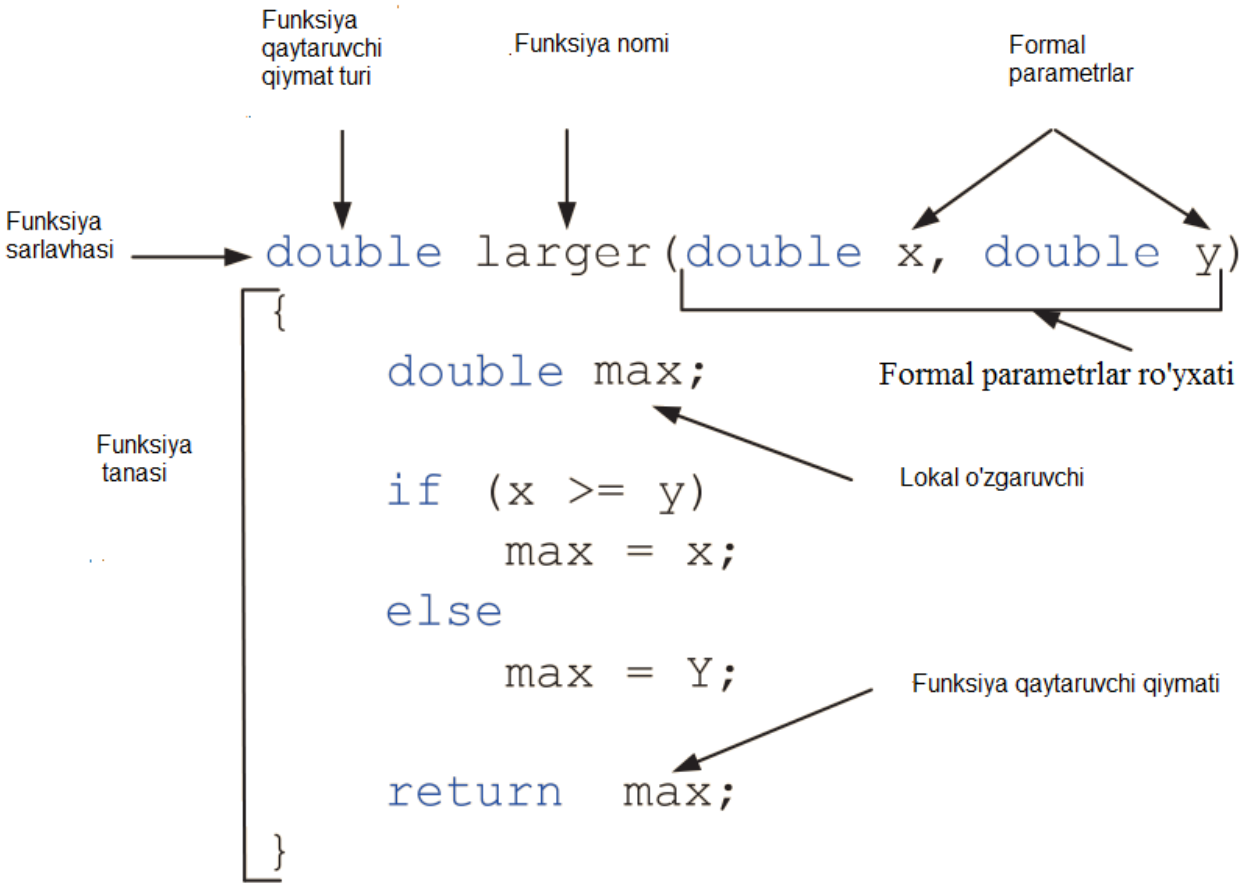
```
int almashsin(int, int);
double max(double x, double y);
void func();
```

Funksiya prototipi tushirib qoldirilishi mumkin, agar dastur matnida funksiya aniqlanishi uni chaqiradigan funksiyalar matnidan oldin yozilgan bo'lsa. Lekin bunday holat yaxshi uslub hisoblanmaydi, ayniqsa, o'zaro bir-biriga murojaat qiluvchi funksiyalarni e'lon qilishda muammolar yuzaga kelishi mumkin.

Funksiya aniqlanishi - funksiya sarlavhasida va figurali qavsga ('{', '}') olingan qandaydir amaliy mazmunga ega tanadan iborat bo'ladi. Agar funksiya qaytaruvchi turi **void** turidan farqli bo'lsa, uning tanasida, albatta, mos turdagi parametrga ega **return** operatori bo'lishi shart. Funksiya tanasida bittadan ortiq **return** operatori bo'lishi mumkin. Ularning ixtiyoriy birortasini bajarish orqali funksiyadan chiqib ketiladi. Agar funksiyaning qiymati dasturda ishlatilmaydigan bo'lsa, funksiyadan chiqish uchun parametrsiz **return** operatori ishlatilishi mumkin yoki umuman, **return** ishlatilmaydi. Oxirgi holda funksiyadan chiqish - oxirgi yopiluvchi qavsga yetib kelganda ro'y beradi.

Funksiya dasturning birorta modulida yagona ravishda aniqlanishi kerak, uning e'loni esa funksiyani ishlatadigan modullarda bir necha marta yozilishi mumkin. Funksiya aniqlanishida sarlavhadagi barcha parametrlar nomlari yozilishi shart.

Odatda dasturda funksiya ma'lum bir ishni amalga oshirish uchun chaqiriladi. Funksiyaga murojaat qilganda, u qo‘yilgan masalani echadi va o‘z ishini tugatishida qandaydir qiymatni natija sifatida qaytaradi (2.15 rasm).



2.15-rasm. Hisoblash algoritmi

Funksiyani chaqirish uchun uning nomi va undan keyin qavs ichida argumentlar ro‘yxati beriladi:

<*funksiya nomi*>(<argument1>, <argument2>,…,<argumentn >);

Bu yerda har bir <argument> - funksiya tanasiga uzatiladigan va keyinchalik hisoblash jarayonida ishlatiladigan o‘zgaruvchi, ifoda yoki o‘zgarmasdir. Argumentlar ro‘yxati bo‘sh bo‘lishi mumkin.

C++ tilidagi har qanday dasturda, albatta, ***main()*** bosh funksiyasi bo‘lishi kerak. Ayni shu funksiyani yuklagich tomonidan chaqirilishi bilan dastur bajarilishi boshlanadi.

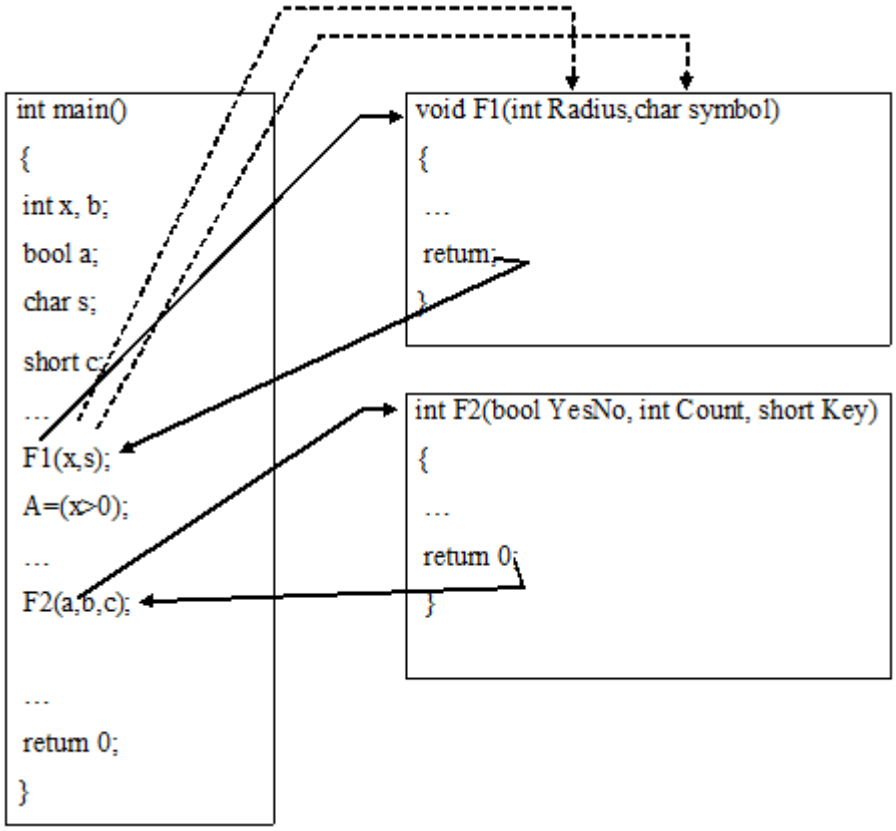
2.16-rasmda bosh funksiyadan boshqa funksiyalarni chaqirish va ulardan qaytish sxemasi ko‘rsatilgan.

Dastur *main()* bosh funksiyasini bajarishdan boshlanadi va «**F1(x,s);**» - funksiya chaqirishgacha davom etadi, keyinchalik boshqaruv **F1()** funksiya tanasidagi amallarni bajarishga o‘tadi. Bunda *Radius* parametrining qiymati sifatida funksiya *x* o‘zgaruvchi qiymatini, **symbol** parametri sifatida *s* o‘zgaruvchisining qiymati ishlatiladi. Funksiya tanasi **return** operatorigacha bajariladi. **return** operatori boshqaruvni ***main()*** funksiyasi tanasidagi **F1**() funksiyasi chaqirilgan operatordan keyingi operatorga o‘tishini ta’minlaydi, ya’ni funksiyadan qaytish ro‘y beradi. Shundan keyin ***main()*** funksiyasi operatorlari bajarilishda davom etadi va «**F2(a,b,c)**» - funksiya chaqirishi orqali boshqaruv **F2()** funksiya tanasiga o‘tadi va hisoblash jarayonida mos ravishda **YesNo** sifatida *a* o‘zgaruvchisining, **Count** sifatida *b* o‘zgaruvchisining va **Key** sifatida *c* o‘zgaruvchisining qiymatlari ishlatiladi. Funksiya tanasidagi **return** operatori yoki oxirgi operator bajargandan keyin avtomatik ravishda bosh funksiyaga qaytish amalga oshiriladi (2.16-rasm).

Funksiyaning dasturdagi o‘rnini yanada tushunarli bo‘lishi uchun son kvadratini hisoblash masalasida funksiyadan foydalanishni ko‘raylik.

Funksiya prototipini sarlavha.h sarlavha faylida joylashtiramiz:

int Son_Kvadrati(int);



2.16- rasm. Bosh funksiyadan boshqa funksiyalarni chaqirish va ulardan qaytish sxemasi

Asosiy dasturga ushbu sarlavha faylini qo‘shish orqali Son_Kvadrati() funksiya e’loni dastur matniga kiritiladi:

```
#include <iostream>
using namespace std;
#include "sarlavha.h"
int main()
{
int Uzgaruvchi=5;
cout << Son_Kvadrati(Uzgaruvchi);
return 0;
}
int Son_Kvadrati(int x) return x*x;
```

Xuddi shu masalani sarlavha faylidan foydalanmagan holda funksiya e’lonini dastur matniga yozish orqali ham hal qilish mumkin:

```
#include <iostream>
using namespace std;
int Son_Kvadrati(int);
```

```
int main()
{
int Uzgaruvchi = 5;
cout << Son_Kvadrati(Uzgaruvchi);
return 0;
}
int Son_Kvadrati(int x){ return x*x;}
```

Dastur ishlashida o‘zgarish bo‘lmaydi va natija sifatida ekranga 25 soni chop etiladi.

O‘zgaruvchilar funksiya tanasida yoki undan tashqarida e’lon qilinishi mumkin. Funksiya ichida e’lon qilingan o‘zgaruvchilarga *lokal o‘zgaruvchilar* deyiladi. Bunday o‘zgaruvchilar xotiradagi dastur stekida joylashadi va faqat o‘zi e’lon qilingan funksiya tanasida amal qiladi. Boshqaruv asosiy funksiyaga qaytishi bilan lokal o‘zgaruvchilar uchun ajratilgan xotira bo‘shatiladi (o‘chiriladi).

Har bir o‘zgaruvchi o‘zining amal qilish sohasi va yashash vaqti xususiyatlari bilan xarakterlanadi.

O‘zgaruvchining *amal qilish sohasi* deganda, o‘zgaruvchini ishlatish mumkin bo‘lgan dastur sohasi (qismi) tushuniladi. Bu tushuncha bilan o‘zgaruvchining *ko‘rinish sohasi* uzviy bog‘langan. O‘zgaruvchi amal qilish sohasidan chiqqanda ko‘rinmay qoladi. O‘zgaruvchining *yashash vaqti* deb, u mavjud bo‘lgan dastur bo‘lagining bajarilishiga ketgan vaqt intervaliga aytiladi.

Global o‘zgaruvchilar dastur matnida funksiya aniqlanishidan tashqarida e’lon qilinadi va e’lon qilingan joyidan boshlab dastur oxirigacha amal qiladi.

Dastur matnida global o‘zgaruvchilarni ular e’lonidan keyin yozilgan ixtiyoriy funksiyada ishlatish mumkin. Shu sababli, global o‘zgaruvchilar dastur matnining boshida yoziladi. Funksiya ichidan global o‘zgaruvchiga murojaat qilish uchun funksiyada uning nomi bilan mos tushadigan lokal o‘zgaruvchilar bo‘lmasligi kerak. Agar global o‘zgaruvchi e’lonida unga boshlang‘ich qiymat berilmagan bo‘lsa, ularning qiymati *0* hisoblanadi. Global o‘zgaruvchining amal qilish sohasi uning ko‘rinish sohasi bilan ustma-ust tushadi.

1-*misol*. Ikki butun son yig‘indisi qiymatini qaytaruvchi funksiyani keltiramiz.

```
#include <iostream. h>
int sum (int a, int b); //funksiya e’lon qilinishi
int main ( )
{
int a=2, b=3, c, d;
c = sum (a, b); // funksiyaga murojaat qilish
cin >> d;
cout << sum (c, d); // funksiyaga murojaat qilish
return 0;
}
int sum (int a, int b) // funksiyani aniqlash
{
return a+b; // funksiya tanasi
}
```

Massiv elementlarining yig‘indisini hisoblash misolida funksiya parametri sifatida massivlarni uzatishga doir masalani keltiramiz.

```
#include <iostream. h>
int summa (cont int* mas, const int n);
int const n = 5;
int main ( )
{
int marksn]={3, 4, 5, 4, 4};
cout << ”massiv elementlari yig‘indisi:”<<summa(marks,n);
return 0;
}
int sum (const int* mas, const int n)
{
int i;
int s = 0;
for ( i = 0; i < n; i ++)
s = s + mas  i ];
return s;
}
```

2-*misol*. Funksiya uzluksiz va oraliq chegaralarida turli xil ishorali qiymatga ega deb faraz qilinadi. Oraliqni ikkiga bo‘lish usuli asosida f(x)=0 funksiyaning [a,b] oraliqdagi ildizini topish dasturini tuzish.

Algoritmning so‘zlar orqali ifodalanishi usulidan foydalanamiz. Masalani yechishdan avval oraliq chegarasidagi funksiya qiymatlarini moslash kerak, ya’ni $x=a$ nuqtada funksiya manfiy, $x= b$ nuqtada musbat qiymatga ega bo‘lishi ta’minlanadi.Ularni joyma-joy almashtirish uchun “uch likopcha” usulidan foydalaniladi.So‘ngra oraliqni ikkiga bo‘lish usuli asosida $f(x)=0$ funksiyaning ildizini aniqlash jarayoni amalga oshiriladi. Uning uchun avvalo $s=(a+b)/2$ o‘rta qiymat aniqlanadi va $u=f(s)$ funksiya qiymatining ishorasi aniqlanadi. Agar $f(s)<0$ bo‘lsa, u holda $a=s$, aks holda $(f(s)>0$ bo‘lsa) – $b=s$ deb qabul qilinadi. Bu jarayon $|f(a)-f(b)|<\varepsilon$ shart bajarilgancha davom etadi va funksiyaning ildizi $x=(a+b)/2$ deb hisoblanadi (bunda $\varepsilon>0$ - yetarlicha kichik musbat son).

Asosiy dasturda funksiya tashkil qilish bilan bajarish:

```
#include <iostream.h>
#include <math.h>
#include <conio.h>
float f(float x);
int main()
{
float a,b,c,y1,y2,y3,eps=0.0001;
cin>>a>>b;
if (f(a)>0 && f(b)<0){c=a; a=b; b=c;}
y1 = f(a);
y2 = f(b);
cout<<"\n y1 = " << y1 << "   y2 = " << y2;
while(fabs(y1-y2)>=eps)
{
y3=f((a+b)/2);
if(y3>0) b=(a+b)/2; else a=(a+b)/2;
y1=f(a);
y2=f(b);
cout<<"\n y1 = "<<y1<<"   y2 ="<<y2;
}
```

```
  cout<<"\n"<<(a+b)/2;
  getch();
  return 0;
  }
  float f(float x)
  {return  x*x-4; }
```

3-*misol*. Eksponensial z = e^x funksiya qiymatini ε>0 musbat kichik son aniqligida taqribiy hisoblashda funksiyadan foydalanish masalasi ko‘riladi [5, 142-152 b.].

Ma’lumki natija $e^x = x^1/1! + x^2/2! + \ldots + x^i/i! + \ldots$ taqribiy munosabat asosida hisoblanadi. Cheksiz qatorning *i*-chi hadining absolyut qiymati ε>0 qiymatidan kichik bo‘lmaguncha yig‘indi davom ettirilishi kerak, ya’ni jarayon tugallanishi sharti $|x^i/i!| >$ ε munosabat ko‘rinishida beriladi.

Dastur matni.

```
#include <iostream.h>
float fun (float, float);
int main()
{
float x, e,q;
cin >>x>>e;
q=fun(x,e);
cout <<”natija = “, q;
}
float fun (float y, float eps);
{
int i, p;
float q, s;
s = 0;
q = 1;
p = 1;
i = 1;
while ( fabs (q / p) > eps)
{
q = q * y;
p = p * i;
```

```
    s = s + q / p;
    i = i + 1;
    }
    return s;
    }
```

4-*misol*. Berilgan [a,b] oraliqda aniqlangan uzluksiz y= f (x) funksiya bilan OX o‘qi orasida hosil bo‘lgan S yuzani trapetsiya formulasi asosida taqribiy hisoblash algoritmini yaratamiz:

1) S = 0;
2) h = (b - a) / n;
3) i = 0;
4) S=S + ((f (a+i*h)+ f (a+(i+1)*h))/ 2) *h;
5) i = i + 1;
6) **agar** i < n , **u holda** => (4);
7) muhrlash (S).

Asosiy dasturda funksiya tashkil qilish bilan bajarish:

```
#include <iostream.h>
#include <math.h>
#include <conio.h>
float f(float x);
int main()
{
float s, h;
int n;
s=0;
h=(b-a)/n;
for (int i=0; i<=n; i++)
s=s+((f(a+i*h)+ f(a+(i+1)*h)/2)*h);
  cout<<"\n"<<s;
  getch();
  return 0;
  }
  float f(float)
```

```
return x*x-4;
```

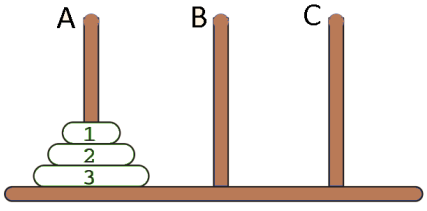
5-*misol*. Faktorial hisoblash misolida rekursiya asosida hisoblash funksiya ko‘rinishini keltiramiz:

```
long fact (long n)
{
if (n==0 || n==1) return 1;
return (n * fact (n - 1) );
}
```

6-*misol*. «Xanoy minorasi» [4, 1003-1007 b.].

Quyida rekursiya bilan samarali echiladigan «Xanoy minorasi» masalasini ko‘raylik (2.17-rasm).

Masala. Uchta A, B, C qoziq va n ta har xil o‘lchamli halqalar mavjud. Halqalarning o‘lchami o‘sish tartibida 1 dan n gacha tartiblangan. Avvaliga barcha halqalar A qoziqqa 2.17-rasmdagidek joylashtirilgan. A qoziqdagi barcha halqalarni C qoziqqa, yordamchi B qoziqdan foydalangan holda, quyidagi qoidalarga amal qilgan holda o‘tkazish talab etiladi: halqalarni bittadan ko‘chirish kerak va katta o‘lchamli halqani kichik o‘lchamli halqa ustiga qo‘yish mumkin emas.



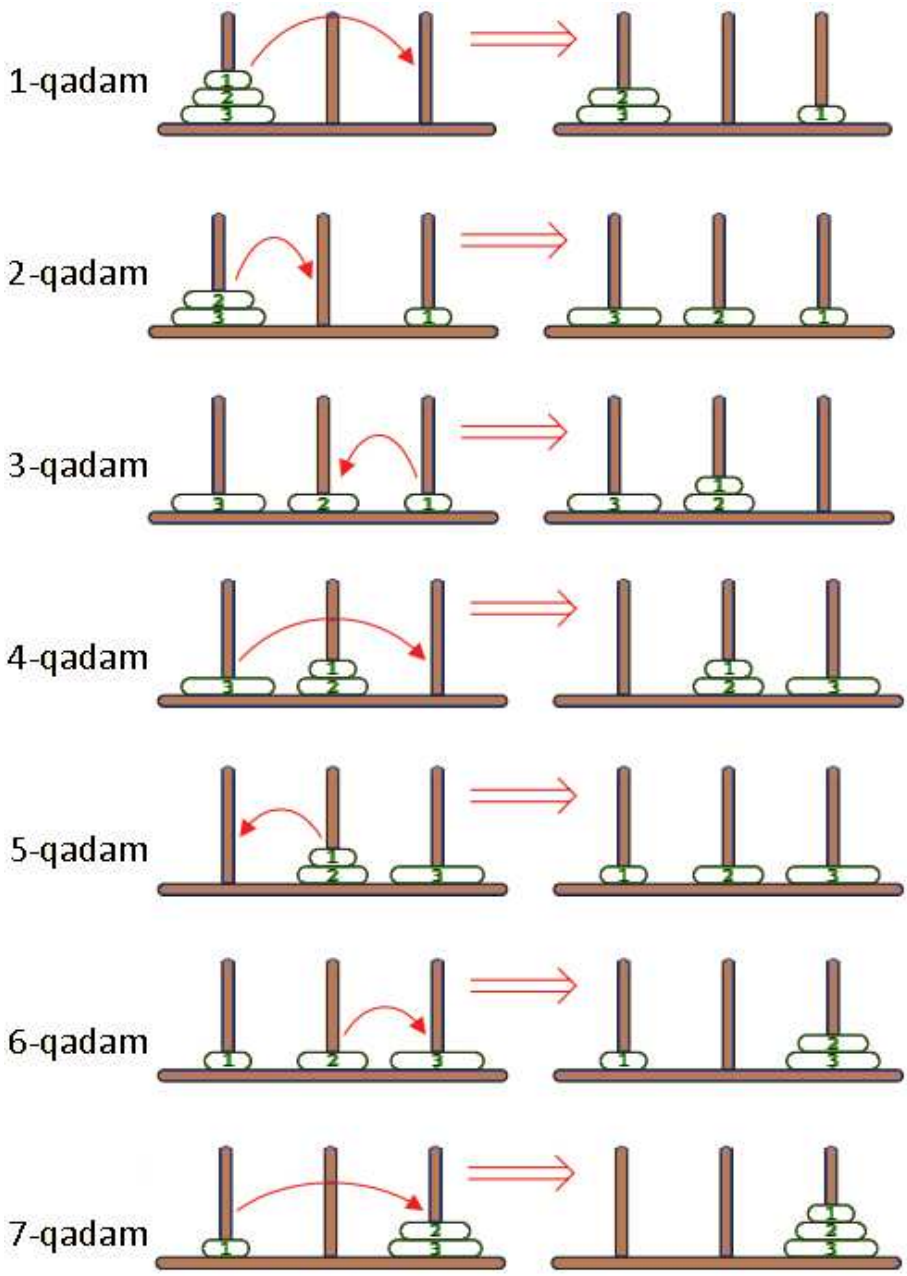
2.17. Xanoy minorasi masalasi

```
#include <iostream>
using namespace std;
void Xanoy(int n, char a = 'A', char b = 'C',
                  char c = 'B')
{
if(n)
```

```
{
     Xanoy(n-1, a, c, b);
     cout << "Halqa " << a << " dan " << b
          << " ga o'tkazilsin\n";
     Xanoy(n-1, c, b, a);
}
}
int main()
{
unsigned int Halqalar_Soni;
cout << "Xanoy minorasi masalasi" << endl;
cout << "Halqalar sonini kiriting: ";
cin >> Halqalar_Soni;
Xanoy (Halqalar_Soni);
return 0;
}
```

Halqalar soni 3 bo‘lganda (`Halqalar_Soni=3`) dastur ekranga halqalarni ko‘chirish bo‘yicha amallar ketma-ketligini chop etadi (2.18-rasm):

```
Halqa A dan C ga o‘tkazilsin
Halqa A dan B ga o‘tkazilsin
Halqa C dan B ga o‘tkazilsin
Halqa A dan C ga o‘tkazilsin
Halqa B dan A ga o‘tkazilsin
Halqa B dan C ga o‘tkazilsin
Halqa A dan C ga o‘kazilsin
```



2.18. Xanoy minorasi masalasini yechish algoritmi

Tahlil qilib ko‘rilsa, uchta halqani A qoziqdagi barcha halqalarni C qoziqqa o‘tkazish uchun $2^3-1=7$ ta jarayon bajarildi. Halqalar soni 64ta bo‘lganda bu jarayonlar soni $2^{64}-1$ ga teng bo‘ladi.

$$2^{10} = 1024 \approx 1000 = 10^3$$

Bundan kelib chiqadiki:

$$2^{64} = 2^4 \times 2^{60} \approx 2^4 \times 10^{18} = 1.6 \times 10^{19}$$

Bir yildagi sekundlar soni 3.2×10^7 ga teng. Bir dona diskni bir qoziqdan boshqasiga olib o‘tish uchun bir sekund sarflanadi deb hisoblansa, quyidagiga kelish mumkin:

$1.6 \times 10^{19} = 5 \times 3.2 \times 10^{18} = (3.2 \times 10^{7}) \times (5 \times 10^{11})$

Barcha 64ta diskni A qoziqdan C qoziqqa olib o‘tish uchun (5×10^{11}) yil kerak bo‘ladi. Kompyuter bir sekundda bir milliard (10^{9}) operatsiya bajara oladi deb hisoblansa, bir yilda quyidagicha operatsiya bajara oladi:

$(3.2 \times 10^{7}) \times 10^{9} = 3.2 \times 10^{16}$

64ta diskni A qoziqdan C qoziqqa olib o‘tish uchun kompyuterga quyidagicha miqdorda vaqt kerak bo‘ladi:

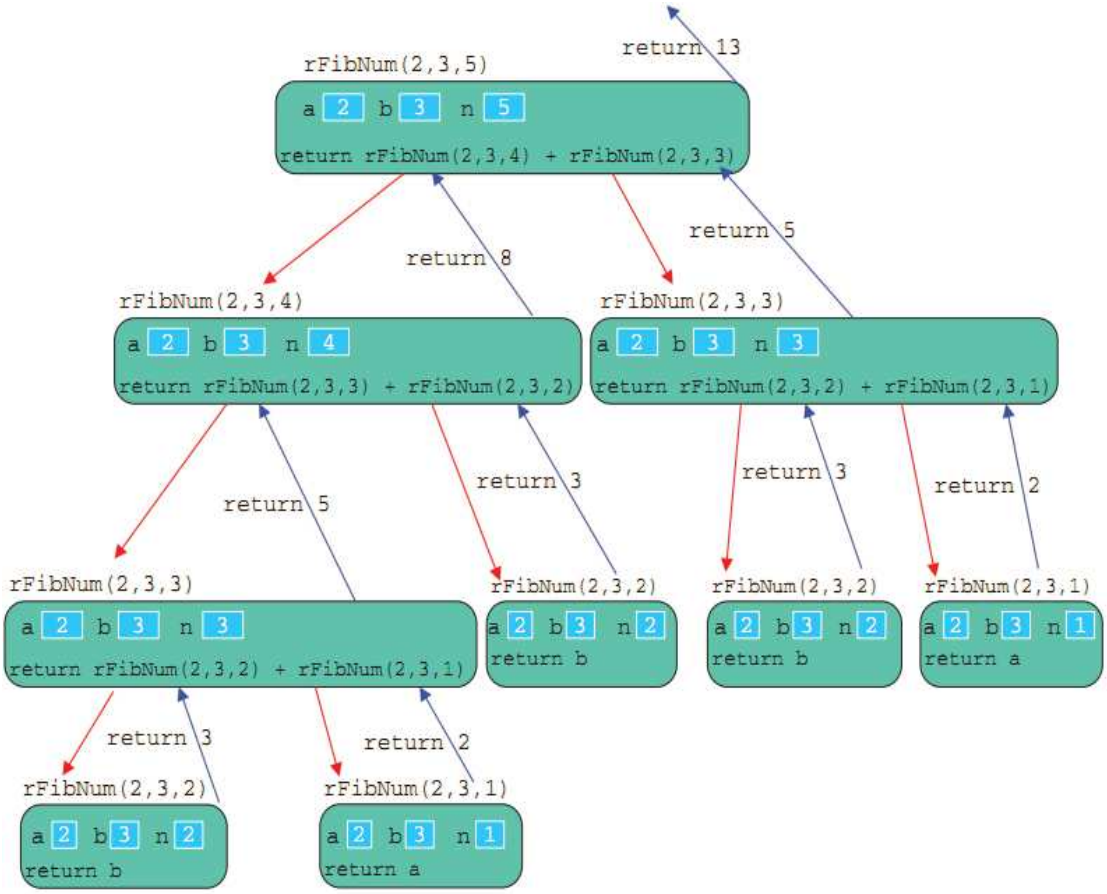
$2^{64} \approx 1.6 \times 10^{19} = 1.6 \times 10^{16} \times 10^{3} = (3.2 \times 10^{16}) \times 500$

Ya’ni, 500 yil vaqt kerak bo‘ladi.

7-*misol*. Fibonachchi sonlarini topish masalasi. Fibonachchi sonlarini topishda har bir had o‘zidan oldingi ikki had yig‘indisiga teng. Birinchi va ikkinchi hadi oldindan ma’lum bo‘ladi. Rekursiyani qo‘llagan holda Fibonachchi sonlarini topish formulasini quyidagicha yozish mumkin:

$$rFibNum(a,b,n) = \begin{cases} a & agar \quad n = 1, \\ b & agar \quad n = 2, \\ rFibNum(a,b,n-1) + rFibNum(a,b,n-2) & agar \quad n > 2. \end{cases}$$

Rekursiyaning bajarilish qadamlari 2.19–rasmda keltirilgan.



2.19 – rasm. Rekursiyaning bajarilish qadamlari

C++ tilida dastur quyidagicha tasvirlanadi.

```
#include <iostream>
using namespace std;
int rFibNum(int a, int b, int n);
int main()
{
int firstFibNum;
int secondFibNum;
int n;
cout << "Birinchi Fibonachchi sonini kiriting: ";
cin >> firstFibNum;
cout << endl;
cout << "Ikkinchi Fibonachchi sonini kiriting: ";
cin >> secondFibNum;
cout << endl;
```

```
    cout << "Qidirilayotgan Fibonachchi soni o’rnini
kiriting: ";
    cin >> n;
    cout << endl;
    cout << n << " – o’rindagi Fibonachchi soni: "
         << rFibNum(firstFibNum, secondFibNum, n) <<endl;
    return 0;
    }
    int rFibNum(int a, int b, int n)
    {
    if (n == 1)
         return a;
    else if (n == 2)
         return b;
    else
         return rFibNum(a, b, n - 1)
                   + rFibNum(a, b, n - 2);
    }
```

Funksiyaga murojaat qilish uchun quyidagini yozish kerak:

```
rFibNum(2, 3, 5) << endl;
```

Rekursiya chiroyli, ixcham ko‘ringani bilan xotirani tejash va hisoblash vaqtini qisqartirish nuqtai nazaridan uni imkon qadar iterativ hisoblash bilan almashtirilgani ma’qul. Masalan, x haqiqiy sonining n-darajasini hisoblashning quyidagi echim varianti nisbatan kam resurs talab qiladi (n – butun ishorasiz son):

```
double Butun_Daraja(double x, int n)
{
 double p=1;
 for (int i=1; i<=n; i++)p*=x;
 return p;
}
```

Ikkinchi tomondan, shunday masalalar borki, ularni yechishda rekursiya juda samarali, hattoki yagona usuldir. Xususan, grammatik tahlil masalalarida rekursiya juda ham qulay hisoblanadi.

8-*misol*. Funksiyalarda bir o‘lchovli sonli massivlar argument sifatida ishlatilganda ularning chegarasini ko‘rsatish shart emas. Shu sababli **a** massiv funksiyaning parametrida **a** [ ] ko‘rinishida berilsa, bu ko‘rinish **a** massivning bosh elementi adresini anglatadi. Masalan, massiv elementlari yig‘indisini hisoblash uchun tuzilgan funksiyaning parametrida to‘g‘ridan to‘g‘ri **a** [ ] ifodani ko‘rsatish yetarli:

```
#include <iostream>
#include <conio.h>
int sum (int m, int a[]);
int main ()
{
const int m = 6;
int b[m];
int  i,  p;
for ( i = 0; i < m;  i++ )
cin >> b[i];
p = sum( m,b);
cout << " p=  "<< p;
getch();
return 0;
}
int sum ( int n, int a[] )
{
int i;
int s = 0;
for ( i = 0;  i < n;  i++)
s = s + a[i];
return s;
}
```

9-*misol*. Ikkinchi usul funksiya parametridagi massivni yangi tur kiritish asosida shakillantirishdan iborat [3, 349-350 b.]. Bu usulda massiv *typedef* yangi

tur **vec** [n] e'lon qilish orqali, funksiyada **a** parametrini **vec** turida deb ko'rsatish yetarli**:**

```
#include <conio.h>
#include <iostream.h>
#include <vcl.h>
const int n=30;
typedef int vec [n];
int fmax(int m, vec a);
int main ()
{
vec b;
int kmax, i, m1=6;
for ( i=0; i<m1;  i++ )
cin >> b[i];
kmax= fmax(m1,b);
cout<<"\n  max=  "<<kmax;
getch();
return 0;
}
int fmax(int m, vec a)
{
int p, i;
p = a [0];
for ( i=0; i<m;  i++ )
if ( p < a[i] ) p = a[i];
return p;
}
```

2- bob bo‘yicha savol va topshiriqlar

Savol va topshiriqlar [6, 7] adabiyotlarlan tanlab olingan.

1. Yuqori darajadagi dasturlash tillari nima?
2. Kompilyatsiya jarayonini necha bosqichdan tashkil topadi?
3. Qanday belgilar bilan tugagan barcha belgilar ketma-ketligi izoh hisoblanadi?
4. C++ tilining kalit so‘zlariga qaysilar kiradi?
5. Protsessor registrlarini belgilash uchun qaysi so‘zlar ishlatiladi?
6. Kompilyatsiya jarayoni nima?
7. Kiritilgan qiymatni o‘zgaruvchi turiga mos kelishini qanday tekshirish mumkin?
8. Identifikator nima?
9. Sintaksis – qanday tilning qoidalari?
10. C++ instruksiyalari qanday belgi bilan tugallanishi zarur?
11. Tilning ma’nosini beruvchi qoidalar to‘plami qanday nomlanadi?
12. **int** turidagi son, odatda, nima uchun ishlatiladi?
13. Agar dasturda sintaktik xatolar bo‘lsa, kompilyator bu haqida xabar beradimi?
14. Arifmetik operatorlar bajarilish ketma-ketligi qoidasi qanday qoida?
15. Figurali qavslar nima uchun ishlatiladi?
16. Vergul (“**,**”), odatda, nima uchun ishlatiladi?
17. “**double**” kalit so‘zi nima uchun ishlatiladi?
18. Haqiqiy son turi nima?
19. Berilganlarning strukturalashgan turlari qanday?
20. Butun son turlari nima?
21. Suzuvchi nuqtali turlar nima?
22. Sanab o‘tiluvchi tur nima?
23. Haqiqiy sonlar qanday kalit so‘z bilan e’lon qilinadi?
24. O‘nlik fiksirlangan nuqtali format deganda nima tushuniladi?
25. Eksponensial shaklda haqiqiy o‘zgarmas nechta qismdan iborat bo‘ladi va ularga misol keltiring.
26. Kompilyator nimaga qarab unga mos turni belgilaydi?
27. Opaliqni teng ikkiga bo’ljsh usuli asosida f(x)=0 tenglamani yeching?

28. [a,b] oraliqda aniqlangan va uzliksiz f(x) funksiya xosil qiluvchi uzani to'g'ri to'rtburchak usulida hisoblash dasturi yarating.
29. Qaysi operator yordamida oshkor ravishda bir turni boshqa turga keltirish mumkin?
30. C++ tilida tuzilgan dasturning asosiy maqsadi nima?
31. O‘zgaruvchi nima?
32. O‘zgaruvchilarga ifoda qanday belgi orqali yuklanadi?
33. C++ tilida `num = num + 2;` ko‘rinishidagi ifoda nimani bildiradi?
34. Kod qismidagi o‘zgaruvchilarning kompilyator uchun qanday ketma-ketlikda qiymat olishlari jadvalini yozing.
35. C++ tilida bir turni boshqa turga keltirishning qanday yo‘llari mavjud?
36. Inkrement va dekrement amallari nima?
37. “Prefiks” yoki “postfiks” amal tushunchasi qanday ifodalarda o‘rinli?
38. C++ tilida ixtiyoriy (tayanch va hosilaviy) turdagi o‘zgaruvchilarning o‘lchami qanday amal yordamida aniqlanadi?
39. Qo‘yilgan masalani yechishda biror holat ro‘y bergan yoki yo‘qligini ifodalash uchun 0 va 1 qiymat qabul qiluvchi nimalardan foydalaniladi?
40. C++ tilida bayt razryadlari ustida mantiqiy amallar majmuasi jadvalini ko‘rsating.
41. [a,b] oraliqda aniqlangan va uzliksiz f(x) funksiya xosil qiluvchi uzani trapetsia usulida hisoblash dasturi yarating.
42. Baytdagi bitlar qiymatini chapga yoki o‘ngga surish uchun mos ravishda qaysi amallari qo‘llaniladi?
43. Taqqoslash amali qanday amal va u qanday ko‘rinishga ega?
44. Taqqoslash amallarining natijasi - taqqoslash o‘rinli bo‘lsa yoki o‘rinli bo‘lmasa qanday qiymat bo‘ladi?
45. Ifodalar qiymatini hisoblashda nima hisobga olinadi?
46. O‘qish oqimi nima?
47. Baytlar-bu... .
48. Input stream nima?
49. Output stream nima?
50. C++ dastur sarlavhasida qaysi fayldan foydalanish kerak?
51. Qiymat dastur orqali o‘qib olinganida nimalar qiymatlarning ajratuvchisi sifatida qabul qilinadi?

52. Qaysi kalit so‘z orqali kiritish oqimidagi bir nechta funksiyalardan foydalanish mumkin?
53. Berilganlar oqimidan faqat kerakli qismini kiritish kerak bo‘lsa, unda kiritish oqimining qaysi funksiyasidan foydalanish kerak?
54. **fixed** manipulyatori nimani chop etadi?
55. Mantiqiy qo‘shish operatori nechta ifoda orqali hisoblanadi?
56. Mantiqiy inkor operatori tekshirilayotgan ifoda yolg‘on bo‘lsa qanday qiymat qaytaradi?
57. **if** operatori nima?
58. C++ tilining qurilmalari operatorlarni blok ko‘rinishida tashkil qilishga imkon beradimi? Buni tushuntirib bering.
59. *Blok* nima?
60. Shart operatorida e’lon qilish operatorlarini ishlatish mumkinmi?
61. <operator$_1$> va <operator$_2$> shartli operator bo‘lishi mumkinmi?
62. Agar tekshirilayotgan shart nisbatan sodda bo‘lsa nima ishlatish mumkin?
63. **switch** tarmoqlanish operatori nima?
64. **break** va **default** kalit so‘zlari nima uchun ishlatiladi?
65. **switch** operatorida e’lon operatorlari ham uchrashi mumkinmi?
66. **switch** operatori bajarilishida “*sakrab o‘tish*” holatlari bo‘lishi hisobiga blok ichidagi ayrim e’lonlar bajarilmasligi va buning oqibatida dastur ishida xatolik ro‘y berishi mumkinmi?
67. **switch** operatori nima uchun ishlatiladi?
68. Sanab o‘tiluvchi turlar va shu turdagi o‘zgaruvchilarga misol keltiring.
69. Mantiqiy operatorlarga nimalar kiradi?
70. <operand$_1$> <taqqoslash amali> < operand$_2$> - quyidagi amal nimani anglatadi?
71. **“**&&” “||” “!” operatorlari nimani anglatadi?
72. (x==3) && (y==5) agar x va y qiymatlari har xil bo‘lsa qanday qiymat qaytaradi?
73. Mantiqiy ko‘paytirish operatori qanday belgi orqali belgilanadi?
74. Mantiqiy qo‘shish operatori qanday belgi orqali belgilanadi?
75. Beshta sonning o‘rta arifmetigi kanday topiladi?
76. Cheksiz takrorlash uchun takrorlashni davom ettirish sharti qanday?
77. Takrorlash operatorida ham bloklardan foydalanish mumkinmi?

78. C++ tilining qurilmalari operatorlarni blok ko‘rinishida tashkil qilishga imkon beradimi?
79. Agar <ifoda> *rost* qiymatli o‘zgarmas ifoda bo‘lsa, takrorlash kanday bo’ladi?
80. Takrorlash operatorlarining bajarilishida kanday holatlar yuzaga kelishi mumkin?
81. Takrorlash operatori ichma-ich joylashgan bo‘lishi mumkunmi?
82. **do-while** takrorlash operatori kanday vazifani bajaradi?
83. **while** takrorlash shartini oldindan tekshiruvchi takrorlash operatori hisoblanadimi?
84. **else** operatori kanday vazifani bajaradi?
85. **continue** operatori kanday vazifani bajaradi?
86. **break** operatori kanday vazifani bajaradi?
87. **while** operatori kanday vazifani bajaradi?
88. **for** operatori kanday vazifani bajaradi?
89. **if** operatori kanday vazifani bajaradi?
90. **include** operatori kanday vazifani bajaradi?
91. Ayrim hollarda, **goto** operatorining «sakrab o‘tishi» hisobiga xatoliklar yuzaga kelishi mumkunmi?
92. Takrorlash operatorida qavs ichidagi ifodalar bo‘lmasligi mumkin, lekin sintaksisida ‘;’ bo‘lmasligiga ruxsat beriladimi?
93. Massiv deb nimaga aytiladi?
94. Massiv indeksi sifatida qanday son ishlatiladi?
95. Dasturda ishlatiladigan har bir konkret massiv qanday nomga ega?
96. Massiv elementiga murojaat qilish qanday amalga oshiriladi?
97. C++ tilida massivlar elementining turiga cheklovlar qo’yiladimi?
98. Ikki o‘lchamli massivning sintaksisi qanday ko’rinishda bo‘ladi?
99. **cout** operatori qanday vazifani bajaradi?
100. **cin** operatori qanday vazifani bajaradi?
101. **for** operatori qanday vazifani bajaradi?
102. **while** operatori qanday vazifani bajaradi?
103. **do-while** operatori qanday vazifani bajaradi?
104. **char** operatori qanday vazifani bajaradi?
105. **iostream** operatori qanday vazifani bajaradi?

106. Massiv – bu?
107. Palindrom deb nimaga aytiladi?
108. Misolda massiv elementlar soni keltirilmagan bo‘lsa, massiv elementlar soni qanday aniklanadi?
109. **int** operatori qanday vazifani bajaradi?
110. **gets** operatori qanday vazifani bajaradi?
111. Funksiya bu…?
112. Funksiyalar modullar deb ham atalishi mumkunmi?
113. C++ tilida funksiya chaqirilganda ayrim argumentlarni tushirib qoldirish mumkunmi va bunga qanday erishish mumkun?
114. Lokal o‘zgaruvchilar o‘zlari e’lon qilingan funksiya yoki blok chegarasida ko‘rinish sohasiga ega bo’ladimi?
115. Funksiyalar qanday turlarga bo‘linadi?
116. Kelishuv bo‘yicha qiymat berishning nechta sharti bor?
117. C++ tilidagi har qanday dasturda qaysi funksiya bosh funksiya bolishi kerak?
118. Funksiyalar qanday ko‘rinishda bo‘ladi?
119. Funksiya qanday aniklanadi?
120. Kompilyator ishlashi natijasida har bir funksiya qanday ko‘rinishda bo‘ladi?
121. Lokal o‘zgaruvchi yashash vaqti qanday aniklanadi?
122. **include** operatori qanday funksiyani bajaradi?
123. **cout** operatori qanday funksiyani bajaradi?
124. **cin** operatori qanday funksiyani bajaradi?
125. **if** operatori qanday funksiyani bajaradi?
126. **else** operatori qanday funksiyani bajaradi?
127. Ayrim algoritmlar berilganlarning qanday turdagi qiymatlari uchun qo‘llanishi mumkin?
128. Qayta yuklanuvchi funksiyalardan foydalanishda qanday qoidalarga rioya qilish kerak?
129. **inline** operatori qanday funksiyani bajaradi
130. **float** operatori qanday funksiyani bajaradi
131. Qiymatni hisoblash uchun funksiyaning «oldingi qiymati» ma’lum bo‘lishi kerakmi?
132. Matematikada manfiy bo‘lmagan butun sonlarning faktorialini aniqlash

qaysi formula yordamida amalga oshiriladi?

133. Rekursiya deb nimaga aytiladi?
134. Rekursiya uchun qanday aniqlanishlar o‘rinli?
135. Agar faktorial funksiyasiga `n>0` qiymat berilsa, qanday holat ro‘y beradi?
136. Rekursiv funksiyalarning to‘g‘ri amal qilishi uchun qanday chaqirishlarning to‘xtash sharti bo‘lishi kerak?
137. Har bir rekursiv murojaat qo‘shimcha xotira talab qiladimi?
138. Rekursiya chiroyli, ixcham ko‘ringani bilan xotirani tejash va hisoblash vaqtini qisqartirish nuqtai nazaridan uni imkon qadar iteraktiv hisoblash bilan almashtirilgani ma’qulmi?
139. Rekursiya qanday to‘xtatiladi?
140. Har bir rekursiv formula nechta ifodaga ega bo‘lishi kerak?

GLOSSARIY

Termin	Terminology	O‘zbek tilidagi sharhi
dasturlash tillari	programming languages	dastur ta’minotini yaratish jarayonini osonlashtirish uchun yaratilgan tillar
include		preprotsessor direktivasi, kutubxona fayllarini dasturga ulash uchun ishlatiladi
cout		ekranga chiqarish oqimi
kengaytma	extension	fayllarning turli dasturlarga tegishliligini aniqlovchi fayl ko‘rinishining qismi
kompilyatsiya	compilation	bajariluvchi fayl hosil bo‘lish jarayoni
leksema	lexeme	tilning ajralmaydigan qismlari
identifikator	identifier	katta va kichik lotin harflari, raqamlar va tag chiziq (‘_’) belgilaridan tashkil topgan va raqamdan boshlanmaydigan belgilar ketma-ketligi
dekompozitsiya		murakkab jarayonni bir nechta sodda jarayonlar ketma-ketligi bilan tasvirlash
sintaktik qoidalar	sintaktik rules	grammatik qoidalarga o‘xshash qoidalar to‘plami
semantika	semantics	tilning ma’nosini beruvchi qoidalar to‘plami
cin		ekrandan kiritish oqimi
berilganlar	variable	dastur ishlashi uchun kerakli qiymatlar
int		butun son ko‘rinishidagi berilganlarning turi
double		haqiqiy son ko‘rinishidagi berilganlarning turi
char		belgi ko‘rinishidagi berilganlarning turi
bayt		kompyuter xotirasi o‘lchov birligi
unar amal	unar	bitta operand ustida bajariluvchi amal
binar amal	binary	ikkita operand ustida bajariluvchi amal
o‘zgaruvchi	variable	berilganlarni saqlab turish uchun ishlatiluvchi til birligi
konstanta	const	dastur davomida qiymati o‘zgarmaydigan berilgan
sizeof		o‘zgaruvchi turining xotiradagi hajmini aniqlash
inkrement	increment	o‘zgaruvchining qiymatini bittaga oshirish

dekrement	decrement	o‘zgaruvchining qiymatini bittaga kamaytirish
prefiks	prefix	operatorning o‘zgaruvchidan oldin joylashgan ko‘rinishi
postfiks	postfix	operatorning o‘zgaruvchidan keyin joylashgan ko‘rinishi
bit	bit	eng kichik o‘lchov birligi
razryad	discharge	bitlardan (0 yoki 1) tashkil topgan indikator
setw		o‘zgaruvchini belgi bilan to‘ldirib chiqarish
if		shart operatori
else		shart yolg‘onligini aniqlovchi operator
switch		bir nechta konstanta bilan tekshirish operatori
case		konstantalar bilan tekshirish operatori
default		konstantaga teng bo‘lmaganda bajarish operatori
for		takrorlash qadami bilan beriladigan takrorlash operatori
while		sharti oldin tekshiriladigan takrorlash operatori
do-while		sharti keyin tekshiriladigan takrorlash operatori
break		takrorlashni to‘xtatish operatori
continue		takrorlashni to‘xtatish va boshqarishni keyingi qadamga o‘tkazish operatori
cheksiz takrorlash	endless loop	takrorlashni to‘xtatish shartining mavjud emasligi
iostream		kiritish-chiqarish oqimlari bilan ishlaydigan kutubxona
parametr	parametr	funksiya ishlashi uchun kerak berilganlar
argument	argument	funksiyaga parametriga jo‘natiladigan qiymat
sarlavha fayli	header file	funksiyalar e’loni yozilgan fayl
amal qilish sohasi		o‘zgaruvchini ishlatish mumkin bo‘lgan dastur sohasi
return		funksiya qiymatini qaytaruvchi operator
rekursiya	recursion	funksiya tanasida funksiyani chaqirish
funksiya	function	programmaning alohida bo‘lagi, asosiy qism tomonidan chaqirib ishlatiladi

FOYDALANILGAN ADABIYOTLAR

1. O’zbekiston Respublikasi Birinchi Prezidentining “Zamonaviy axborot-kommunikatsiya texnologiyalarni yanada joriy etish va rivojlantirish chora-tadbirlari to’g’risida”gi PQ-1730-sonli Qarori. 2012-y., 21 mart.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein *Introduction to Algorithms.* Third Edition. The MIT Press Cambridge, Massachusetts London, England, 2009. – 1312 p.
3. Scheinerman Edwant *C++ for Mathematicians. An Introduction for Students and Professionals.* Chapman&Hall/CRC, Taylor&Francis Group, LLC, Boca Raton, London, New York, 2006. - 496 p.
4. D.S. Malik *C++ Programming: From Problem Analysis to Program Design*. Seventh Edition. Course Technology, 2014.-1488 p.
5. Герберд Шилдт *C++базовый курс*.3-е издание. Перевод с англ. –М.: Изд. дом «Вильямс», 2010. - 624 с.
6. Культин Н.Б. C++Builder в задачах и примерах. -СПб.: БХВ-Петербург, 2005. -336 с.
7. Madraximov Sh.F., Ikramov A.M., Babajanov M.R. *C++ tilida programmalash bo’yicha masalalar to’plami. O’quv qo’llanma*. T., O’zbekiston Milliy universiteti, “Universitet” nashriyoti, 2014. - 160 b.

Ilmiy jurnallar

1. Science of Computer Programming
2. Scientific Programming

Davriy nashrlar

1. Programming and Computer Software

Internet manbalari

1. http://cppstudio.com – C++ tilida dasturlash bo‘yicha namunalar izohlari bilan keltirilgan
2. http://cplusplus.com – C++ tilida mavjud konstruksiyalar ta’rifi, ishlatish namunalari bilan keltirilgan.

3. http://compteacher.ru/programming – dasturlash bo‘yicha video darsliklar mavjud.
4. http://www.intuit.ru/ – internet universitet, dasturlash bo‘yicha yozma va video ma’ruzalar o‘qish, test sinovlaridan o‘tish va sertifikat olish imkoniyati mavjud.
5. http://www.ziyonet.uz – dasturlash asoslari bo‘yicha referatlar topish mumkin.