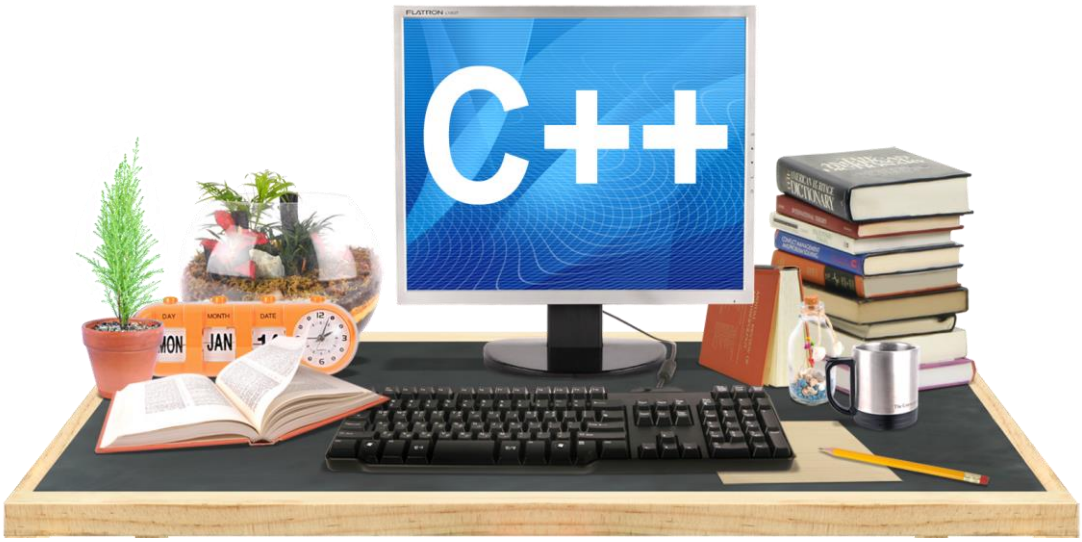
O`ZBEKISTON RESPUBLIKASI
OLIY VA O`RTA MAXSUS TA'LIM VAZIRLIGI
ANDIJON MASHINASOZLIK INSTITUTI

"AVTOMATIKA VA ELEKTROTEXNOLOGIYA"
FAKULTETI
"AXBOROT TEXNOLOGIYALARI"
KAFEDRASI

"C++ DASTURLASH TILI" NI O'RGANISH BO'YICHA



USLUBIY QO'LLANMA

Andijon-2015 yil

TASDIQLANGAN.
Andijon mashinasozlik instituti
o‘quv-uslubiy kengashi yig‘ilishida ko‘rib chiqilgan va tasdiqlangan.
(Kengashning 2015- yil __ _____dagi -sonli bayonnomasi)

Kengash raisi _____Q.Ermatov

MA’QULLANGAN.
Avtomatika va elektrotexnologiya fakulteti kengashi yig‘ilishida
muhokama qilingan va ma’qullangan.
(Kengashning 2015 - yil __ _____ dagi -sonli bayonnomasi)

Kengash raisi_____N. To’ychiboyev

TAVSIYA ETILGAN.
«Axborot texnologiyalari» kafedrasi
yig‘ilishida muhokama qilingan va foydalanishga tavsiya etilgan.
(Kafedra yig‘ilishining 2015 - yil __ _____dagi
-sonli bayonnomasi)

Kafedra mudiri _____X.Sarimsaqov

Tuzuvchilar:

J. Axmadaliyev – “Axborot texnologiyalari“ kafedrasi assistenti.

R. Holdarboyev– “Axborot texnologiyalari“ kafedrasi assistenti.

Taqrizchilar:

G’.Tojiboyev – “Axborot texnologiyalari“ kafedrasi dotsenti,

M.Jalilov– TATU Farg’ona filiali “Kompyuter tizimlari“ kafedrasi dotsenti.

Ushbu uslubiy qo’llanma “O`zbekiston Respublikasi Oliy va o`rta maxsus ta’lim vazirligining 2008- yil 7- martdagi 61- sonli buyrug`i bilan tasdiqlangan “Oliy ta’lim muassasalarida kurs loyihasini bajarish tartibi haqida nizom” ga asosan tayyorlangan bo`lib, unda kurs loyihasi mavzulari, ularni tayyorlash va rasmiylashtirish qoidalari keltirilgan.

MUNDARIJA

KIRISH

Insoniyat o’zining tarixiy taraqqiyoti jarayonida har xil ish qurollarini yaratgan. Bu ish qurollari uning jismoniy mehnatini yengillashtirishga xizmat qilgan. Bularga oddiy bolta, tesha, arradan tortib hozirgi zamon qudratli mashina va traktorlarini misol sifatida keltirish mumkin.

Inson bu davrda faqat mehnat qurollarini yaratish bilan chegaralanib qolmay, balki u o’zining aqliy mehnatini yengillashtirish qurollarini ham yaratdi. Bunga oddiy hisob-kitob toshlaridan tortib, hozirgi kunda ham o’z kuchi va qulayligini yo’qotmagan cho’tlar misol bo’la oladi.

XX asrning 30-40 yillariga kelib, EHMlarning birinchi loyihalari paydo bo’la boshladi. Birinchi EHM yaratish ishlarini 1937 yilda AQSHning Ayova shtatida joylashgan universitetning professori A. Atanasov boshladi. Millati bolgar bo’lgan bu olim yaratmoqchi bo’lgan EHM matematik-fizikaning ayrim masalalarini yechishga mo’ljallangan edi. Ammo ikkinchi jahon urushi bu ishlarni oxirigacha yetkazish imkonini bermadi. Atanasovning buyuk xizmatlari shundaki, u birinchi bo’lib EHMlarda ikkilik sanoq sistemasini qo’llashning qulayligini ko’rsatadi.

Axborot kommunikatsion texnonologiyalarini taraqqiy etishida bevosita dasturlash tillarining o’rni beqiyos. Ayniqsa, hozirgi davrga kelib C++, Java, Delphi dasturlash tillar yordamida shaxsiy kompyuterlar uchun amaliy dasturiy to’plamlardan tashqari SmartPhone va Planshetlar uchun operatsion tizim (iOS, Android, Windows mobile, Symbian va h.k) va ilovalar yaratilmoqda.

Informatsion texnologiyalarning yana bir muhim jihatlaridan biri shundaki, bu fan jadal sur’atlarda o‘sib, yil sayin yangidan-yangi yo‘nalishlarga, mutaxassisliklarga tarmoqlanib ketmoqda: algoritmik, mantiqiy, obyektga yo‘naltirilgan, vizual, parallel dasturlash texnologiyalari, animatsiya, multimediya, Web, ma’lumotlar bazasini boshqarish tizimlari, ko‘p prosessorli, neyron arxitekturali kompyuterlar va hokazo. Ko‘rinib turibdiki, informatika

meta fan darajasiga ko‘tarilib, uni bitta o‘quv kursi chegarasida to‘liq o‘zlashtirishning imkoni bo‘lmay qoldi.

Informatsion texnologiyalar sohasi bo‘yicha rus va ingliz tillarida qo‘llanmalar juda ko‘p chop etilmoqda. Oxirgi yillarda o‘zbek tilidagi qo‘llanmalar ham ko‘payib qoldi.

Ushbu taklif etilayotgan qo‘llanma asosan **C++** dasturlash tilini o‘rganmoqchi bo‘lganlar uchun mo‘ljallangan. Shu sababli qo’llanmada **C++** tiliga bog’liq boshlang’ich ma’lumotlar yoritilgan. Bu qo’llanmadan **C++** dasturlash tilini o‘rganuvchilar, dastur tuzishni o‘rganayotganlar hamda “Dasturlash asoslari”, “Informatika va dasturlash” fanlaridan olingan nazariy bilimlarni mustahkamlash uchun foydalanishlari hisobga olingan. Ushbu qo‘llanmaga kiritilgan ma’lumotlar dasturlashning bazaviy kursidagi deyarli barcha bo‘limlarini, ya’ni skalyar turlar va boshqaruv operatorlaridan tortib, ma’lumotlarning murakkab turlari kabilarni o‘z ichiga oladi.

Ushbu qo’llanma haqidagi tanqidiy fikr va mulohazalar jAxmadaliyev@umail.uz elektron manzili orqali mamnuniyat ila qabul qilinadi.

I BOB. C++ DASTURLASH TILINING BOSHLANG'ICH TUSHUNCHALARI

1.1. C++ dasturlash tilining elementlari

Hozirgi kunda juda ko‘p algoritmik tillar mavjud. Bular ichida **Java** va **C++** dasturlash tillari universal tillar hisoblanib, boshqa tillarga qaraganda imkoniyatlari kengroqdir. So‘ngi yillarda **Java** va **C++** dasturlash tillari juda takomillashib, tobora ommalashib bormoqda. Mazkur tillardagi vositalar zamonaviy kompyuter texnologiyasining hamma talablarini o‘z ichiga olgan va unda dastur tuzuvchi uchun ko‘pgina qulayliklar yaratilgan.

C++ 1980 yillar boshida **Bjarne Stroustrup** tomonidan **C** tiliga asoslangan tarzda tuzildi. **C++** juda ko’p qo’shimchalarni o’z ichiga olgan, lekin eng asosiysi u obyektlar bilan dasturlashga imkon beradi.

Dasturlarni tez va sifatli yozish hozirgi kunda katta ahamiyat kasb etmoqda. Buni ta’minlash uchun obyektli dasturlash g’oyasi ilgari surildi. Huddi 1970 yillar boshida strukturali dasturlash kabi, dasturlarni hayotdagi jismlarni modellashtiruvchi obyektlat orqali tuzish dasturlash sohasida inqilob qildi.

C++ dan tashqari boshqa ko'p obyektli dasturlshga yo’naltirilgan tillar paydo bo'ldi. Shulardan eng ko’zga tashlanadigani Xerox ning Palo Altoda joylashgan ilmiy-qidiruv markazida (PARC) tuzilgan Smalltalk dasturlash tilidir. Smalltalk da hamma narsa obyektlarga asoslangan. **C++** esa gibrid tildir. Unda **C** tiliga o’hshab strukturali dasturlash obyektlar bilan dasturlash mumkin.

C++ funksiya va obyektlarning juda boy kutubhonasiga ega. Yani **C++** dasturlash tilida dasturlashni o’rganish ikki qismga bo’linadi. Birinchisi bu **C++** tilini o’zini o’rganish, ikkinchisi esa **C++** ning standart kutubhonasidagi tayyor obyekt va funksiyalarni qo’llashni o’rganishdir.

C++ tiliga ko‘plab yangiliklar kiritilgan bo‘lib, tilning imkoniyati yanada kengaytirilgan. **C++** dasturlash tili ham boshqa dasturlash tillari kabi o‘z alfavitiga va belgilariga ega.

- Tillarda mavjud alfavit va leksemalarga quyidagilar kiradi:
  1. Katta va kichik lotin alfaviti harflari;
  2. Raqamlar - 0,1,2,3,4,5,6,7,8,9;
  3. Maxsus belgilar: " {} | [] () + - / % \ ; ' : ? <=>_ ! & ~ # ^ . *
- Alfavit belgilaridan tilning leksemalari shakllantiriladi:
  - ✓ Identifikatorlar;
  - ✓ Kalit (xizmatchi yoki zahiralangan) so‘zlar;
  - ✓ O‘zgarmaslar;
  - ✓ Amallar belgilanishlari;
  - ✓ Ajratuvchilar.

Bu tillarda tuzilgan dasturlarda izohlar istalgan joyda berilishi mumkin. Ular satriy va blokli ko‘rinishlarda bo’ladi. Satriy izohlar uchun “//”, blokli izohlar uchun “/*”, “*/” belgilari ishlatiladi.

1.2.Dastur tuzilmasi

C++ dasturlash tilida dastur quyidagi tarkibda tashkil topadi:

Direktivalar – funksiyalar kutubxonasini chaqirish. Ular maxsus **include** katalogida joylashgan va **.h** kengaytmali fayllar bo’ladi. **C++** tilida masalaning qo’yilishiga qarab kerakli kutubxonalar chaqiriladi. Bus esa dasturning xotirada egallaydigan joyini minimallashtiradi.

Masalan, ma’lumotlarni kiritish-chiqarish proseduralari uchun:

#include <stdio.h> tizimdan chaqirish

#include “stdio.h” joriy katalogdan chaqirish.

C++ dasturlash tili bilan ishlovchi eng sodda dasturlar Dev C++ va CodeBlocks dasturlaridir. Ularning tarkibida 300 dan ortiq kutubxonalar mavjud. Eng ko’p ishlatiladigan kutubxonalar quyidagilar:

#include<iostream.h>,

#include <math.h>

#include <conio.h>

#include <graphics.h>

#include <memory.h> va boshqalar

Makrolar **(#define**) – dastur bajarilishi davomida o'zgaruvchi ko'rsatilgan qiymatni qabul qilishi uchun (**const**). Unda makroning nomi va qiymati ko'rsatiladi. **Масалан:**

#define pi 3.1415

#define x 556

#define s[100]

#define M x*x*x

main () funksiyasi– asosiy degan ma'noni anglatadi. Bu funksiya "{" belgisidan boshlanadi va dasturning asosini tashkil etuvchi o'zgaruvchilarning toifalari ko'rsatiladi. Dastur "}" belgisi bilan yakunlanishi shart. Agar dasturda qism dasturlardan foydalanilayotgan bo'lsa, ularning nomlari va haqiqiqy parametrlari keltiriladi. So'ngra dasturning asosiy buyruqlari yoziladi. Agar buyruqlar murakkab bo'lsas, ular alohida "{ }" belgilari orasiga olingan bo'lishi kerak.

C++ tilida dasturning asosi bo'lmish buyruqlar kichik harflar bilan yoziladi. Buyruqlar nuqta-verguk bilan (;) yakunlanadi. Buyruqlar bir qator qilib yozilishi ham mumkin.

C++ dasturlash tilida dastur funksiya va funksiyalardan tashkil topadi. Agar dastur bir nechta funksiyalardan tashkil topgan bo'lsa, bir funksiyaning nomi main deb nomlanishi shart. Dastur aynan main funksiyasining birinchi operatoridan boshlab bajariladi.

C++ tilidagi dastur ko'rinishini quyidagi misol yordamida keltirib o'tamiz.

```
#include <iostream.h>                    // sarlavha faylni qo'shish
int main ()                              // bosh funksiya tavsifi
```

```
{                                   // blok boshlanishi
cout << “Salom Dunyo! “\n”;         //  satrni chop etish
return 0;                           // funksiya qaytaradigan qiymat
}                                   // blok tugashi
```

Dasturning 1-satrida **#include** direktivasi bo‘lib, dastur kodiga oqimli o‘qish/yozish funksiyalari va uning o‘zgaruvchilari e’loni joylashgan **iostream.h** sarlavha faylini qo‘shadi. Keyingi qatorlarda dasturning yagona, asosiy funksiyasi **main()** funksiyasi tavsifi keltirilgan. Shuni qayd etish kerakki, **C++** dasturida albatta **main()** funksiyasi bo‘lishi shart va dastur shu funksiyani bajarish bilan o‘z ishini boshlaydi.

Dastur tanasida konsol rejimi (***Consol*** *– rejimi bu MS DOS oynasi ko’rinishiga o’xshash oyna bo’lib, unda foydalanuvchi dastur tuzuishda faqat dastur kodlari bilan ishlaydi.* ***Graphic interface*** *– rejimida esa faqat tilning kodlari bilangina emas muhitning menyulari, komponentalari bilan ham ishlashi mumkin bo’ladi*) da belgilar ketma-ketligini oqimga chiqarish amali qo‘llanilgan. Ma’lumotlarni standart oqimga (ekranga) chiqarish uchun quyidagi format ishlatilgan:

cout << <ifoda>;

Bu yerda **<ifoda>** sifatida o‘zgaruvchi yoki sintsksisi 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).

cin << a;

Ma’lumotlarni klaviatura yordamida kiritish buyrug’i bo’lib, u ham **iostream.h** kutubxonasi tarkibidagi funksiya hisoblanadi.

1.3 Identifikatorlar va kalit so‘zlar.

Dasturlash tillarida identifikator tushunchasi mavjud bo‘lib, dasturda obyektlarni nomlash uchun ishlatiladi. O‘zgarmaslarni,

o‘zgaruvchilarni, belgi (metka), protsedura va funksiyalarni belgilashda ishlatiladigan nom **identifikatorlar** deyiladi. Identifikatorlar lotin alfaviti harflaridan boshlanib, qolgan belgilari harf yoki raqamlar ketma-ketligidan tashkil topgan bo'lishi mumkin. Masalan: axc, alfa.

Dasturlash tillarida dastur bajarilishi vaqtida qiymati o‘zgarmaydigan identifikatorlar **o‘zgarmaslar** deyiladi. O‘zgarmaslar beshta guruhga bo‘linadi – butun, haqiqiy (suzuvchi nuqtali), sanab o‘tiluvchi, belgi (literli) va satr («string», literli satr).

C++ tilida o‘zgarmas (**cons**) – bu fiksirlangan sonni, satrni va belgini ifodalovchi leksema hisoblanadi.

Kompilyator o‘zgarmasni leksema sifatida aniqlaydi, unga xotiradan joy ajratadi, ko‘rinishi va qiymatiga (turiga) qarab mos guruhlarga bo’ladi.

Butun o‘zgarmaslar: ular quyidagi formatlarda bo’ladi

- o‘nlik son;
- sakkizlik son;
- o‘n oltilik son.

O’nlik o’zgarmas 0 raqamidan farqli raqamdan boshlanuvchi raqamlar ketma-ketligi va 0 hisoblanadi: 0; 123; 7987; 11.

Manfiy o’zgarmas – bu ishorasiz o‘zgarmas bo‘lib, unga faqat ishorani o’zgartirish amali qo‘llanilgan deb hisoblanadi.

Sakkizlik o’ 0 raqamidan boshlanuvchi sakkizlik sanoq sistemasi (0,1,..,7) raqamlaridan tashkil topgan raqamlar ketma-ketligi:

023; 0777; 0.

O‘n oltilik o‘zgarmas 0x yoki 0X belgilaridan boshlanadigan o‘n oltilik sanoq sistemasi raqamlaridan iborat ketma-ketlik hisoblanadi:

0x1A; 0X9F2D; 0x23.

Harf belgilar ixtiyoriy registrlarda berilishi mumkin.

Kompilyator sonning qiymatiga qarab unga mos turni belgilaydi. Agar tilda belgilangan turlar dastur tuzuvchini qanoatlantirmasa, u oshkor ravishda turni

koʻrsatishi mumkin. Buning uchun butun oʻzgarmas raqamlari oxiriga, probelsiz l yoki L (1ong), u yoki U (unsigned) yoziladi. Zarur hollarda bitta oʻzgarmas uchun bu belgilarning ikkitasini ham ishlatish mumkin: 451u, 012Ul, 0xA2L.

Haqiqiy oʻzgarmaslar: Haqiqiy oʻzgarmaslar – suzuvchi nuqtali son boʻlib, u ikki xil formatda berilishi mumkin:

- ✓ Oʻnlik fiksirlangan nuqtali formatda. Bu koʻrinishda son nuqta orqali ajratilgan butun va kasr qismlar koʻrinishida bo'ladi. Sonning butun yoki kasr qismi boʻlmasligi mumkin, lekin nuqta albatta boʻlishi kerak. Fiksirlangan nuqtali oʻzgarmaslarga misollar: 24.56; 13.0; 66.; .87;
- ✓ eksponensial shaklda haqiqiy oʻzgarmas 6 qismdan iborat bo'ladi:
  1) butun qismi (oʻnli butun son);
  2) oʻnli kasr nuqta belgisi;
  3) kasr qismi (oʻnlik ishorasiz oʻzgarmas);
  4) eksponenta belgisi ‘e’ yoki ‘E’;
  5) oʻn darajasi koʻrsatkichi (musbat yoki manfiy ishorali oʻnli butun son);
  6) qoʻshimcha belgisi (‘F’ yoki f , ‘L’ yoki ‘l’).
- ✓ Eksponensial shakldagi oʻzgarmas sonlarga misollar:
  1E2; 5E+3; .25E4; 31.4E-1.

Belgi oʻzgarmaslar: Belgi oʻzgarmaslar qoʻshtirnoq (‘,’-apostroflar) ichiga olingan alohida belgilardan tashkil topadi va u char kalit soʻzi bilan aniqlanadi. Bitta belgi oʻzgarmas uchun xotirada bir bayt joy ajratiladi va unda butun son koʻrinishidagi belgining ASCII kodi joylashadi. Quyidagilar belgi oʻzgarmaslarga misol bo'ladi:

‘e’, ‘@’ , ‘7’, ‘z’, ‘w’, ‘+’, ‘sh’, ‘*’ , ‘a’, ‘s’.

1.1-jadval. **C++** tilida escape – belgilar jadvali

Escape belgilari	**Ichki kodi (16 son)**	**Nomi**	**Belgining nomlanishi va unga mos amal**
\\	0x5C	\	Teskari yon chiziqni chop etish
\’	0x27	‘	Apostrofni chop etish

Escape belgilari	Ichki kodi (16 son)	Nomi	Belgining nomlanishi va unga mos amal
\”	0x22	“	Qo‘shtirnoqni chop etish
\?	0x3 F	?	So‘roq belgisi
\a	0x07	be1	Tovush signalini berish
\b	0x08	Bs	Kursorni 1 belgi o‘rniga orqaga qaytarish
\f	0x0C	ff	Sahifani o‘tkazish
\n	0x0A	lf	Qatorni o‘tkazish
\r	0x0D	cr	Kursorni ayni qatorning boshiga qaytarish
\t	0x09	ht	Kursorni navbatdagi tabulyatsiya joyiga o‘tkazish
\v	0x0D	vt	Vertikal tabulyatsiya (pastga)
\000	000		Sakkizlik kodi
\xNN	0xNN		Belgi o‘n oltilik kodi bilan berilgan

Ayrim belgi o‘zgarmaslar ‘\’ belgisidan boshlanadi, bu belgi birinchidan, grafik ko‘rinishga ega bo‘lmagan o‘zgarmaslarni belgilaydi, ikkinchidan, maxsus vazifalar yuklangan belgilar – apostrof belgisi, savol belgisini (?), teskari yon chiziq belgisini (\) va ikkita qo‘shtirnoq belgisini (“) chop qilish uchun ishlatiladi. Undan tashqari, bu belgi orqali belgini ko‘rinishini emas, balki oshkor ravishda uning ASCII kodini sakkizlik yoki o‘n oltilik shaklda yozish mumkin. Bunday belgidan boshlangan belgilar escape ketma-ketliklar deyiladi (1.1-jadval).

Turlangan o‘zgarmaslar: Turlangan o‘zgarmaslar xuddi o‘zgaruvchilardek ishlatiladi va initsializatsiya qilingandan (boshlang‘ich qiymat berilgandan) keyin ularning qiymatini o‘zgartirib bo‘lmaydi

Turlangan o‘zgarmaslar **const** kalit so‘zi bilan e’lon qilinadi, undan keyin o‘zgarmas turi va albatta initsializatsiya qismi bo‘lishi kerak.

Misol tariqasida turlangan va literli o‘zgarmaslardan foydalangan holda radius berilganda aylana yuzasini hisoblaydigan dasturni keltiramiz.

```
#include <iosream.h>
int main (){
    const double pi=3.1415;
    const int radius=3;
    double square=0;
    square=pi*radius*radius;
    cout<<square<<'\n';
    return 0; }
```

Dastur bosh funksiyasining boshlanishida ikkita – pi va radius o‘zgarmaslari e’lon qilingan. Aylana yuzasini aniqlovchi square o‘zgarmas deb e’lon qilinmagan, chunki u dastur bajarilishida o‘zgaradi. Aylana radiusini dastur ishlashida o‘zgartirish mo‘ljallanmagan, shu sababli u o‘zgarmas sifatida e’lon qilingan.

Sanab o‘tiluvchi tur: Ko‘p miqdordagi, mantiqan bog‘langan o‘zgarmaslardan foydalanganda sanab o‘tiluvchi turdan foydalanish ma’qul. Sanab o‘tiluvchi o‘zgarmaslar **enum** kalit so‘zi bilan aniqlanadi. Mazmuni bo‘yicha bu o‘zgarmaslar oddiy butun sonlardir. Sanab o‘tiluvchi o‘zgarmaslar **C++** standarti bo‘yicha butun turdagi o‘zgarmaslar hisoblanadi. Har bir o‘zgarmasga (songa) mazmunli nom beriladi va bu identifikatorni dasturning boshqa joylarida nomlash uchun ishlatilishi mumkin emas. Sanab o‘tiluvchi tur quyidagi ko‘rinishga ega:

enum <Sanab o‘tiladigan tur nomi> { <nom_1> =<$qiymat_1$>, <nom_2> = <$qiymat_2$>,… <nom_3>=<$qiymat_3$> } ;

bu yerda, **enum** – kalit so‘z (inglizcha enumerate – sanamoq);

<Sanab o‘tiladigan tur nomi> – o‘zgarmaslar ro‘yxatining nomi;

<nom> – butun qiymatli konstantalarning nomlari;

<qiymat$_i$> – shart bo‘lmagan initsializatsiya qiymati (ifoda).

Dastur ishlashi mobaynida qiymatlari o‘zgarishi mumkin bo‘lgan identifikatorga **o‘zgaruvchilar** deyiladi.

Dasturlash tillarida dastur bajarilishi paytida qandaydir berilganlarni saqlab turish uchun o‘zgaruvchilar va o‘zgarmaslardan foydalaniladi. O‘zgaruvchi-dastur obyekti bo‘lib, xotiradagi bir nechta yacheykalarni egallaydi va berilganlarni saqlash uchun xizmat qiladi. O‘zgaruvchi nomga, o‘lchamga va boshqa atributlarga – ko‘rinish sohasi, amal qilish vaqti va boshqa xususiyatlarga ega bo’ladi. O‘zgaruvchilarni ishlatish uchun ular albatta e’lon qilinishi kerak. E’lon natijasida o‘zgaruvchi uchun xotiradan qandaydir soha zahiralanadi, soha o‘lchami esa o‘zgaruvchining aniq turiga bog‘liq bo’ladi. Shuni qayd etish zarurki, bitta turga turli apparat platformalarda turlicha joy ajratilishi mumkin.

C++ tilida o‘zgaruvchi e’loni uning turini aniqlovchi kalit so‘zi bilan boshlanadi va **‘=’** belgisi orqali boshlang‘ich qiymat beriladi (shart emas). Bitta kalit so‘z bilan bir nechta o‘zgaruvchilarni e’lon qilish mumkin. Buning uchun o‘zgaruvchilar bir-biridan **‘,’** belgisi bilan ajratiladi. E’lonlar **‘;’** belgisi bilan tugaydi. O‘zgaruvchi nomi 255 belgidan oshmasligi kerak.

O‘zgaruvchilarni e’lon qilish dastur matnining istalgan joyida amalga oshirilishi mumkin.

Dasturlash tillarida kalit so‘zlar mavjud bo‘lib ulardan boshqa maqsadlarda foydalanilmaydi. Quyida **C++** tilining kalit so‘zlarini alfavit tartibida keltiramiz.

C++ tilida: asm, auto, break, case, catch, char, class, const, continue, default, delete, do, double, else, enum, explicit, extern, float, for, friend, goto, if, inline, int, long, mutable, new, operator, private, protected, public, register, return, short, signed, sizeof, static, struct, swith, template, this,

throw, try, typedef, typename, union, unsigned, virtual, void, volatile, while.

Protsessor registrlarini belgilash uchun quyidagi so‘zlar ishlatiladi:

_AH, _AL, _AX, _EAX, _BN, _BL, _BX, _EVX, _CL, _CN, _CX, _ESX, _DN, _DL, _DX, _EDX, _CS, _ESR, EBP, _FS, __GS, _DI, _EDI, _SI, _ESI, __BP, SP, DS, _ES, SS, _FLAGS.

Bulardan tashqari «__» (ikkita tag chiziq) belgilaridan boshlangan identifikatorlar kutubxonalar uchun zahiralangan. Shu sababli ‘_’ va «_» belgilarni identifikatorning birinchi belgisi sifatida ishlatmagan ma’qul. Identifikator belgilari orasida bo’sh joy belgisi (probel) ishlatish mumkin emas, zarur bo‘lganda uning o‘rniga ‘_’ ishlatish mumkin.

Misol uchun: silindr_radiusi, aylana_diametri.

1.4.Ma'lumotlar turlari

C++ tilida ma’lumotlar uchun turlar quyidagicha bo’ladi.

C++ tilining tayanch turlari, ularning baytlardagi o‘lchamlari va qiymatlarining chegaralari 1.1-jadvalda keltirilgan.

1.2-jadval. C++ tilining tayanch turlari

Tur nomi	**Baytlardagi o‘lchami**	**Qiymat chegarasi**
bool	1	True yoki false
Unsigned short int	2	0..65535
Short int	2	-32768..32767
Unsigned long int	4	0..42949667295
Long int	4	-2147483648..2147483647
int(16 razryadli)	2	-32768.. 32767
Int (32 razryadli)	4	-2147483648..2147483647
Unsigned int (16 razryadli)	2	0..65535
Unsigned int (32 razryadli)	4	0..42949667295
Unsigned char	1	0..255
char	1	-128.. 127

Tur nomi	Baytlardagi o‘lchami	Qiymat chegarasi
float;	4	1.2E-38..3.4E38
double	8	2.2E-308..1.8E308
Long double(32 razryadli)	10	3.4e-4932..-3.4e4932
void	2 ёки 4	-

C++ tilida ham o’zgaruvchilarning turlari bir necha guruhlarga ajraladi. Ularni quyida qarab chiqamiz.

Butun son turlari. Butun son qiymatlarni qabul qiladigan o‘zgaruvchilar int(butun), short(qisqa) va 1ong(uzun) kalit so‘zlar bilan aniqlanadi. O‘zgaruvchi qiymatlari ishorali bo‘lishi yoki unsigned kalit so‘zi bilan ishorasiz son sifatida qaralishi mumkin.

Belgi turi. Belgi turidagi o‘zgaruvchilar char kalit so‘zi bilan beriladi va ular o‘zida belgining ASCII kodini saqlaydi. Belgi turidagi qiymatlar nisbatan murakkab bo‘lgan tuzilmalar – satrlar, belgilar massivlari va hokazolarni hosil qilishda ishlatiladi.

Haqiqiy son turi. Haqiqiy sonlar **float** kalit so‘zi bilan e’lon qilinadi. Bu turdagi o‘zgaruvchi uchun xotiradan 4 bayt joy ajratiladi va <ishora><tartib><mantissa> qolipida sonni saqlaydi. Agar kasrli son juda katta (kichik) qiymatlarni qabul qiladigan bo‘lsa, u xotirada 8 yoki 10 baytli ikkilangan aniqlik ko‘rinishida saqlanadi va mos **double** va **long double** kalit so‘zlari bilan e’lon qilinadi. Oxirgi holat 32-razryadli platformalar uchun o‘rinli.

Mantiqiy tur. Bu turdagi o‘zgaruvchi bool kalit so‘zi bilan e’lon qilinib, xotiradan 1 bayt joy egallaydi va 0 (**false**, yolg‘on) yoki (**true**, rost) qiymat qabul qiladi. Mantiqiy tur o‘zgaruvchilar qiymatlar o‘rtasidagi munosabatlarni ifodalaydigan mulohazalarni rost (**true**) yoki yolg‘on (**false**) ekanligi tavsifida qo‘llaniladi va ular qabul qiladigan qiymatlar matematik mantiq qonuniyatlariga asoslanadi.

Mantiqiy mulohazalar ustida uchta amal aniqlangan:

1) **inkor** – A mulohazani inkori deganda A rost bo‘lganda yolg‘on yoki yolg‘on bo‘lganda rost qiymat qabul qiluvchi mulohazaga aytiladi. **C++** tilida inkor – **‘!’** belgisi bilan beriladi. Masalan, A mulohaza inkori «!A» ko‘rinishida yoziladi;

2) **konyunksiya**- ikkita A va B mulohazalar konyunksiyasi yoki mantiqiy ko‘paytmasi «A && B» ko‘rinishga ega. Bu mulohaza faqat A va B mulohazalar rost bo‘lgandagina rost bo’ladi, aks holda yolg‘on bo’ladi (odatda **«&&»** amali «va» deb o‘qiladi). Masalan «bugun oyning 5- kuni va bugun chorshanba» mulohazasi oyning 5- kuni chorshanba bo‘lgan kunlar uchungina rost bo’ladi;

3) **dizyunksiya** – ikkita A va B mulohazalar dizyunksiyasi yoki mantiqiy yig‘indisi «A || B» ko‘rinishda yoziladi. Bu mulohaza rost bo‘lishi uchun A yoki B mulohazalardan biri rost bo‘lishi yetarli. Odatda **«||»** amali «yoki» deb o‘qiladi.

Yuqorida keltirilgan fikrlar asosida mantiqiy amallar uchun rostlik jadvali aniqlangan (1.2-jadval).

1.3-jadval. Mantiqiy amallar uchun rostlik jadvali

Mulohazalar		**Mulohazalar ustida amallar**		
A	**B**	**Not A**	**A and B**	**A or B**
		!A	**A&&B**	**A||B**
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Mantiqiy tur qiymatlari ustida mantiqiy ko‘paytirish, qo‘shish va inkor amallarini qo‘llash orqali murakkab mantiqiy ifodalarni qurish mumkin. Misol uchun, «x – musbat va uning qiymati [1..3] sonlar oralig‘iga tegishli emas» mulohazasini mantiqiy ifoda ko‘rinishi quyidagicha bo’ladi:

(x>0)&&(y<1|| y>3).

Void turi. C++ tilida **void** turi aniqlangan bo'lib bu turdagi dastur obyekti hech qanday qiymatga ega bo'lmaydi va bu turdan qurilmaning til sintsksisiga mos kelishini ta'minlash uchun ishlatiladi. Masalan, **C++** tili sintsksisi funksiya qiymat qaytarishini talab qiladi. Agar funksiya qiymat qaytarmaydigan bo'lsa, u void kalit so'zi bilan e'lon qilinadi.

Misollar.

```
int a=0 A=1; float abc=17.5;
double Ildiz;
bool ok=true;
char LETTER='z';
Void mening_funksiyam();  /*funksiya qaytaradigan  qiymat inobatga  olinmaydi */
```

Turni boshqa turga keltirish: C++ tilida bir turni boshqa turga keltirishning oshkor va oshkormas yo'llari mavjud. Umuman olganda, turni boshqa turga oshkormas keltirish ifodada har xil turdagi o'zgaruvchilar qatnashgan hollarda amal qiladi (aralash turlar arifmetikasi). Ayrim hollarda, xususan tayanch turlar bilan bog'liq turga keltirish amallarida xatoliklar yuzaga kelishi mumkin. Masalan, hisoblash natijasining xotiradan vaqtincha egallagan joyi uzunligi, uni o'zlashtiradigan o'zgaruvchi uchun ajratilgan joy uzunligidan katta bo'lsa, qiymatga ega razryadlarni yo'qotish holati yuz beradi.

Oshkor ravishda turga keltirishda, o'zgaruvchi oldiga qavs ichida boshqa tur nomi yoziladi:

```
#include <iostream.h>
int main()
{
int integer_1=54;
int integer_2;
float floating=15.854;
integer_1=(int) floating; // oshkor keltirish;
```

integer_2=(**int**) floating // oshkormas keltirish;

cout<<’’yangi integer (oshkor): ‘’<<integer_1<<’’\n’’;

cout<<’’yangi integer (oshkormas): ‘’<<integer_2<<’’\n’’;

return 0;

}

Dastur natijasi quyidagi ko‘rinishda bo’ladi:

Yangi integer (oshkor):15

Yangi integer (oshkormas):15

Masala. Berilgan belgining ASCII kodi chop etilsin. Masala belgi turidagi qiymatni oshkor ravishda butun son turiga keltirib chop qilish orqali yechiladi.

Dastur matni:

#include <iostream.h>

int main()

{

Unsigned char A;

Cout<<’’belgini kiriting:’’;

Cin>>A;

Cout<<’\’’<<A<<’’’-belgi ASCII kodi=’’(int)A<<’\n’;

Return 0;

}

Dasturning belgini kiriting so‘roviga

A <enter> amali bajarilsa, ekranga ‘A’-belgi ASCII kodi=65 satri chop etiladi.

1.5.Arifmetik amallar.

Berilganlarni qayta ishlash uchun dasturlash tillarida amallarning juda keng majmuasi aniqlangan. Amal - bu qandaydir harakat bo‘lib, u bitta (unar)

yoki ikkita (binar) operandlar ustida bajariladi, hisob natijasi uning qaytaruvchi qiymati hisoblanadi.

Tayanch arifmetik amallarga qo‘shish (+), ayirish (-), ko‘paytirish (*), bo‘lish (/) va bo‘lishdagi qoldiqni olish (%) amallarini keltirish mumkin.

Amallar qaytaradigan qiymatlarni o‘zlashtirish uchun **C++** tilida “=” va uning turli modifikatsiyalari ishlatilib, quyidagilar hisoblanadi: qo‘shish, qiymat berish bilan (+=); ayirish, qiymat berish bilan (-=); ko‘paytirish, qiymat berish bilan (*=); bo‘lish, qiymat berish bilan (/=); bo‘lish qoldig‘ini olish, qiymat berish bilan (%=) va boshqalar. Bu holatlarning umumiy ko‘rinishi:

<o‘zgaruvchi><amal>=<ifoda>;

Quyidagi dastur matnida ayrim amallarga misollar keltirilgan.

```
#include <iostream.h>
int main()
{
    int a=0 , b=4, c=90; char z='\t';
    a=b; cout<<a<<z;                    //a=4
    a=b+c+c+b; cout<<a<<z;              //a=4+90+90+4=188
    a=b-2; cout<<a<<z;                  //a=2
    a=b*3 cout<<a<<z;                   //a=4*3=12
    a=c/(b+6); cout<<a<<z;              //a=90/(4+6)=9
    cout<<a%2<<z;                       //9%2=1
    a+=b; cout<<a<<z;                   //a=a+b=9+4=13
    a*=c-50; cout<<a<<z;                //a=a*(c-50)=13*(90-50)=520
    a-=38; cout<<a<<z;                  //a=a-38=520-38=482
    a%=8; cout<<a<<z;                   //a=a%8=482%8=2
    return 0;
}
```

Dastur bajarilishi natijasida ekranda quyidagi sonlar satri paydo bo’ladi:

4 188 2 12 9 1 482 2

1.6.Razryadli mantiqiy amallar

Dastur tuzish tajribasi shuni ko‘rsatadiki, odatda qo‘yilgan masalani yechishda biror holat ro‘y bergan yoki yo‘qligini ifodalash uchun 0 va 1 qiymat qabul qiluvchi bayroqlardan foydalaniladi. Bu maqsadda bir yoki undan ortiq baytli o‘zgaruvchilardan foydalanish mumkin. Masalan, mantiqiy turdagi o‘zgaruvchini shu maqsadda ishlatsa bo’ladi. Boshqa tomondan, bayroq sifatida baytning razryadlaridan foydalanish ham mumkin. Chunki razryadlar faqat ikkita qiymatni – 0 va 1 sonlarini qabul qiladi. Bir baytda 8 razryad bo‘lgani uchun unda 8 ta bayroqni kodlash imkoniyati mavjud.

Faraz qilaylik, qo‘riqlash tizimiga 5 ta xona ulangan va tizim taxtasidagi 5 ta chiroqcha (indikator) xonalar holatini bildiradi: xona qo‘riqlash tizimi nazoratida ekanligini mos indikatorning yonib turishi (razryadning 1 qiymati) va xonani tizimga ulanmaganligini indikator o‘chganligi (razryadning 0 qiymati) bildiradi. Tizim holatini ifodalash uchun bir bayt yetarli bo’ladi va uning kichik razryadidan boshlab beshtasini shu maqsadda ishlatish mumkin:

7	6	5	4	3	2	1	0
			ind5	ind4	ind3	ind2	ind1

Masalan, baytning quyidagi holati 1, 4 va 5 xonalar qo‘riqlash tizimiga ulanganligini bildiradi:

7	6	5	4	3	2	1	0
x	x	x	1	1	0	0	1

Quyidagi jadvalda **C++** tilida bayt razryadlari ustida mantiqiy amallar majmuasi keltirilgan. .

1.4 –jadval. Bayt razryadlari ustida mantiqiy amallar

Amallar		Mazmuni
and	&	Mantiqiy VA (ko'paytirish)
or	|	Mantiqiy yoki (qo'shish))
xor	^	Istisno qiluvchi YOKI
not	~	Mantiqiy INKOR (inversiya)

C++ tilida razryadli mantiqiy amallarni qiymat berish operatori birgalikda bajarilishining quyidagi ko'rinishlari mavjud:

&= – razryadli VA qiymat berish bilan;

| = – razryadli YOKI qiymat berish bilan;

^= – razryadli istisno qiluvchi YOKI qiymat berish bilan.

Chapga va o'ngga surish amallari: Baytdagi bitlar qiymatini chapga yoki o'ngga surish uchun, **C++** tilida "<<" va ">>" amallari qo'llanilib, amaldan keyingi son bitlarni nechta o'rin chapga yoki o'nga surish kerakligini bildiradi.

Masalan:

unsigned char A=12; //$A=00001100_2=0x0C_{16}$

A=A<<2; // $A=00110000_2=0x30_{16}=48_{10}$

A=A>>3; //$A=00000110_2=0x06_{16}=6_{10}$

Razryadlarni n ta chapga (o'nga) surish sonni 2^n soniga ko'paytirish (bo'lish) amali bilan ekvivalent bo'lib va nisbatan tez bajariladi. Shuni e'tiborga olish kerakki, operand ishorali son bo'lsa, u holda chapga surishda eng chapdagi ishora razryadi takrorlanadi (ishora saqlanib qoladi) va manfiy sonlar ustida bu amal bajarilganda matematika nuqtai-nazardan xato natijalar yuzaga keladi:

unsigned char B=-120; // $B=10001000_2=0x88_{16}$

B=B<<2; // $B=00100000_2=0x20_{16}=32_{10}$

B=-120; // $B=10001000_2=0x88_{16}$

B=B>>3; // $B=11110001_2=0xF1_{16}=-15_{10}$

Shu sababli, bu razryadli surish amallari ishorasiz (unsigned) turdagi qiymatlar ustida bajarilgani ma’qul.

1.7.Taqqoslash amallari

Dasturlash tillarida qiymatlarni solishtirish uchun taqqoslash amallari aniqlangan (3.3-jadval). 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.

1.5-jadval. Taqqoslash amallari va ularning qo‘llanishi

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

1.8 Amallarning ustunliklari va bajarilish yo‘nalishlari

An’anaviy arifmetikadagidek dasturlash tillarida ham amallar ma’lum bir tartib va yo‘nalishda bajariladi. Ma’lumki, matematik ifodalarda bir xil ustunlikdagi (“приоритет”dagi) amallar uchrasa (masalan, qo‘shish va ayirish), ular chapdan o‘ngga bajariladi. Bu tartib dasturlash tillarida ham o‘rinli, biroq ayrim hollarda amal o‘ngdan chapga bajarilishi mumkin (qiymat berish amalida).

Ifodalar qiymatini hisoblashda amallar ustunligi hisobga olinadi, birinchi navbatda eng yuqori ustunlikka ega bo‘lgan amal bajariladi.

1.6-jadval. Operatorlarning tavsifi

Operator	Tavsifi	Ustunligi	Yo'nalishi
::	Ko'rinish sohasiga ruxsat berish	16	=>
[]	Massiv indeksi	16	=>
()	Funksiyani chaqirish	16	=>
->	Tuzilma yoki sinf elementini tanlash	16	=>
++	Postfiks inkrement	15	<=
--	Postfiks dekrement	15	<=
++	Prefiks inkrement	14	<=
--	Prefiks dekrement	14	<=
Sizeof	O'lchamni olish	14	<=
(<type>)	Turga akslantirish	14	
~	Bitli mantiqiy INKOR	14	<=
!	Mantiqiy inkor	14	<=
-	Unar minus	14	
+	Unar plyus	14	<=
&	Adresni olish	14	<=
*	Vositali murojaat	14	<=
new	Dinamik obyektni yaratish	14	<=
delete	Dinamik obyektni yo'q qilish	14	<=
casting	Turga keltirish	14	
*	Ko'paytirish	13	=>
/	Bo'lish	13	=>
%	Bo'lish qoldig'i	13	=>
+	Qo'shish	12	=>
-	Ayirish	12	=>
»	Razryad bo'yicha o'ngga surish	11	=>
«	Razryad bo'yicha chapga surish	11	

Operator	Tavsifi	Ustunligi	Yo'nalishi
<	Kichik	10	=>
<=	Kichik yoki teng	10	=>
>	Katta	10	=>
>=	Katta yoki teng	10	
==	Teng	9	=>
!=	Teng emas	9	=>
&	Razryadli VA	8	=>
^	Razryadli istisno qiluvchi YOKI	7	=>
|	Razryadli YOKI	6	=>
&&	Mantiqiy VA	5	=>
||	Mantiqiy yoki	4	=>
?:	Shart amali	3	<=
=	Qiymat berish	2	<=
*=	Ko'paytirish qiymat berish amali bilan	2	<=
/=	Bo'lish qiymat berish amali bilan	2	<=
%=	Modulli bo'lish qiymat berish amali bilan	2	<=
+=	Qo'shish qiymat berish amali bilan	2	
-=	Ayirish qiymat berish amali bilan	2	<=
<<=	Chapga surish qiymat berish amali bilan	2	<=
>>=	o'ngga surish qiymat berish amali bilan	2	<=
&=	Razryadli va qiymat berish bilan	2	<=
^=	Razryadli istisno qiluvchi yoki qiymat berish bilan	2	<=
|=	Razryadli yoki qiymat berish bilan	2	<=
Throw	Istisno holatni yuzaga keltirish	2	<=
,	Vergul	1	<=

Dastur tuzuvchisi amallarni bajarilish tartibini o’zgartirishi ham mumkin. Xuddi matematikadagidek, amallarni qavslar yordamida guruhlarga jamlash mumkin. Qavs ishlatishga cheklov yo’q .

Quyidagi dasturda qavs yordamida amallarni bajarish tartibini o’zgartirish ko’rsatilgan.

```
#include <iostream.h>
int main(){
    int x=0,y=0;
    int a=3, b=34, c=82;
    x=a*b+c;
    y=(a*(b+c));
    cout<<’’x=’’<<x<<’’n\’’;
    cout<<’’y=’’<<y<<’’n\’’; }
```

Dasturda amallar ustunligiga ko‘ra x qiymatini hisoblashda oldin a o’zgaruvchi b o’zgaruvchiga ko’paytiriladi va unga c o’zgaruvchining qiymati qo’shiladi. Navbatdagi ko‘rsatmani bajarishda esa birinchi navbatda ichki qavs ichidagi ifoda –(b+c) ning qiymati hisoblanadi, keyin bu qiymat a ga ko‘paytirilib, y o‘zgaruvchisiga o‘zlashtiriladi.

Dastur bajarilishi natijasida ekranga

x=184

y=348

satrlari chop etiladi.

II BOB OPERATORLAR

Dasturlash tili operatorlari yechilayotgan masala algoritmini amalga oshirish uchun ishlatiladi. Operatorlar chiziqli va boshqaruv operatorlariga bo'linadi. Aksariyat holatlarda operatorlar nuqtali vergul (';') belgisi bilan tugallanadi va u kompilyator tomonidan alohida operator deb qabul qilinadi (**C++** tilidagi for operatorining qavs ichida turgan ifodalari bundan mustasno). Bunday operator ifoda operatori deyiladi.

2.1. Qiymat berish, inkrement va dekrement operatorlari

Qiymat berish amallari guruhi, xususan, qiymat berish operatorlari ifoda operatorlari hisoblanadi, **C++** da qiymat berish operatori '=' ko'rinishida bo'ladi:

k=8;.

O'zgaruvchilarga qiymat berish, ya'ni ularning oldingi qiymatini o'zgartirishning boshqa usullari ham mavjud. Bu kabi ishlarni (i++;), (--j;), (k+=i;) operatorlari yordamida amalga oshirish mumkin.

Bo'sh va e'lon operatorlari: Dastur tuzish amaliyotida bo'sh operator – ';' ishlatiladi. Garchi bu operator hech qanday ish bajarmasa ham, hisoblash ifodalarining til qurilmalariga mos kelishini ta'minlaydi. Ayrim hollarda yuzaga kelgan «boshi berk» holatlardan chiqib ketish imkonini beradi.

C++ tilida o'zgaruvchilarni e'lon qilish ham operator hisoblanadi va ularga e'lon operatori deyiladi.

Kiritish va chiqarish operatorlari: Biror-bir masalani yechishning chiziqli bo'lgan algoritmiga dastur tuzishda algoritmdagi keltirilgan ketma-ketliklar asosida operatorlar yoziladi. Bunday dasturlarni tuzishda asosan o'zgaruvchilarga qiymatni kiritish, natijalarni chiqarish va shu bilan birga o'zlashtirish operatorlari ishlatiladi.

Dasturdagi o‘zgaruvchilar qiymatlarini dastur ichida o‘zlashtirish operatori yordamida ham berish mumkin. Lekin dasturga o‘zgaruvchining qiymatni tashqaridan kiritish ehtiyoji ham tug‘iladi.

C++ tilida o‘zgaruvchilar qiymatini klaviaturadan kiritish uchun **cin>>**<o‘zgaruvchi 1> **>>** <o‘zgaruvchi 2> **>>**... **>>** <o‘zgaruvchi n> kiritish oqimidan foydalaniladi.

Bu yerda <o‘zgaruvchi 1>, <o‘zgaruvchi 2>, ..., <o‘zgaruvchi n>lar qiymat qabul qiladigan o‘zgaruvchilar nomi;

Quyida keltirilgan jadvalda **C++** tilida o‘zgaruvchilarga qiymat kiritish tasvirlangan.

cin>>S1>>S2;

cin>>x1>>x2>>x3>>”\n”;

cin>>”\n”;

Bu yerda birinchi operator Sl va S2 o‘zgaruvchilar qiymatini klaviaturadan kiritadi. Ikkinchi operator esa xl, x2, x3 o‘zgaruvchilar qiymatini klaviauradan qabul qiladi va kiritishni keyingi qatorga o‘tkazadi. Oxirgi operator esa kiritishni kutadi va kursorni navbatdagi qatorga o‘tkazadi.

Chop etish operatori: C++ tilida oddiy ma’lumotlar, o‘zgaruvchilar va ifodalar qiymatini chop etish uchun **cout<<** <o‘zgaruvchi 1> **<<** <o‘zgaruvchi 2> **<<**...**<<** <o‘zgaruvchi n> kiritish oqimidan foydalaniladi.

Bu yerda <o‘zgaruvchi l>, <o‘zgaruvchi 2>, ..., <o‘zgaruvchi n> oddiy matnlar yoki o‘zgaruvchilar nomi;

Quyida keltirilgan jadvalda **C++** tilida oddiy ma’lumotlar, o‘zgaruvchilar va ifodalar qiymatini chop etish namunalari keltirilgan.

cout<<Summa<<”\n”;

cout<<”Natija yo‘q”;

cout<<”Tenglama yechimi x1=”<<x1<<”x2=” <<x2;

Oddiy ma'lumotlarni chiqarishda ularga matn deb qaraladi va u qo‘shtirnoq ichida yoziladi.

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

Operandga nisbatan bu amallarning prefiks va postfiks ko‘rinishlari bo’ladi. 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 yoki postfiks amal tushunchasi faqat qiymat berish bilan bog’liq ifodalarda o‘rinli: ;

```
x=y++;              // postfiks
index =--i;         // prefiks
count:++;           // unar amal,  "++count;  " bilan ekvivalent
abc--  ;            // unar amal,  "--abc;    " bilan ekvivalent
```

Bu yerda y o‘zgaruvchining qiymati x o‘zgaruvchisiga o‘zlashtiriladi va keyin bittaga oshiriladi, i o‘zgaruvchining qiymati bittaga kamaytirib, index o‘zgaruvchisiga o‘zlashtiriladi.

sizeof amali: Har xil turdagi o‘zgaruvchilar kompyuter xotirasida turli sondagi baytlarni egallaydi. Bunda, hattoki bir turdagi o‘zgaruvchilar ham qaysi kompyuterda yoki qaysi operatsion sistemada amal qilinishiga qarab turli o‘lchamdagi xotirani band qilishi mumkin.

Bu ishni **C++** tilida sizeof amali yordamida bajarish mumkin. Quyida keltirilgan dasturda kompyuterning platformasiga mos ravishda tayanch turlarining o‘lchamlari chop qilinadi.

```
int main(){
    cout<<’’int turining o‘lchami: "<<sizeof (int)<<"\n"
    cout<<’’float turining o‘lchami: "<<sizeof (float) <<"\n";
    cout<<’’double turining o‘lchami: "<<sizeof  (double)<<"\n";
    cout<<’’char turining o‘lchami: "<<sizeof (char) <<"\n";
```

return 0; **}**

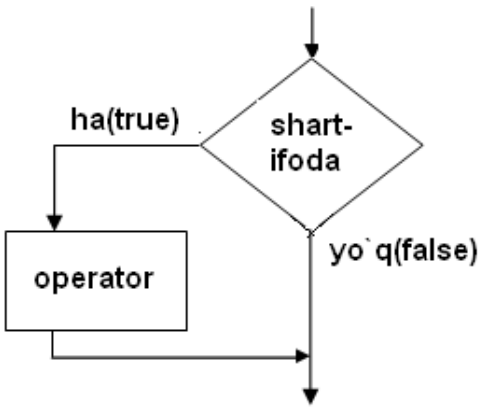
2.2.Shart operatorlari

Yuqorida mavzularda keltirilgan dasturlarda amallar yozilish tartibida ketma-ket va faqat bir marta bajariladigan holatlar, ya'ni chiziqli algoritmlar keltirilgan. Amalda esa kamdan-kam masalalar shu tariqa yechilishi mumkin. Aksariyat masalalar yuzaga keladigan turli holatlarga bog'liq ravishda mos qaror qabul qilishni (yechimni) talab etadi. **C++** tilida dasturning alohida bo'laklarining bajarilish tartibini boshqarishga imkon beruvchi qurilmalarning yetarlicha katta majmuasiga ega. Masalan, dastur bajarilishining birorta qadamida qandaydir shartni tekshirish natijasiga ko'ra boshqaruvni dasturning u yoki bu bo'lagiga uzatish mumkin (tarmoqlanuvchi algoritm). Tarmoqlanishni amalga oshirish uchun shartli operatordan foydalaniladi.

If operatori: If operatori qandaydir shartni rostlikka tekshirshi natijasiga ko'ra dasturda tarmoqlanishni amalga oshiradi:

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 (i (agar u mavjud bo'lsa). Ushbu holat 2.1 –rasmda ko'rsatilgan.



2.1-rasm. if() shart operatorining blok sxemasi

C++ tillarining qurilmalari operatorlarni blok ko‘rinishida tashkil qilishga imkon beradi. Blok **C++** tilida ‘{‘ va ‘}’ belgi oralig‘iga olingan operatorlar ketma-ketligi ko‘rinishida bo’ladi. Blok kompilyator tomonidan yaxlit bir operator deb qabul qilinadi. **C++** tilida 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.

Quyida keltirilgan dasturda **if** operatoridan foydalanish ko’rsatilgan.

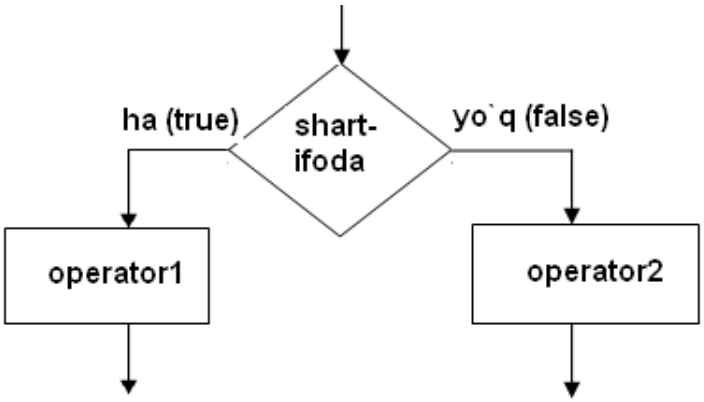
```
#include <iostream.h>
int main()
{
        int b;
        cin>>b;
        if (b>0)
                {
                 //b>0 shart bajarilgan holat
                cout<<''b- musbat son'';
                }
        if (b<0)
        cout<<''b – manfiy son'';
        //b<0 shart bajarilgan holat
        return 0;
}
```

Dastur bajarilishi jarayonida butun turdagi b o‘zgaruvchi e’lon qilingan va uning qiymati klaviaturadan o‘qiladi. Keyin b qiymatini 0 sonidan kattaligi tekshiriladi, agar shart bajarilsa(**true**) , u holda ekranga “b – musbat son” xabari chiqadi. Agar shart bajarilmasa, bu operatorlar cheklab o‘tiladi. Navbatdagi shart operatori b o‘zgaruvchi qiymatini manfiylikka tekshiradi, agar shart bajarilsa, ekranga “b – manfiy son” xabari chiqadi.

If – else operatori: Shart operatorining **if – else** ko‘rinishi quyidagicha:

if (<shart-ifoda>) <operator$_1$>; **else** <operator$_2$>;.

Bu yerda <shart-ifoda> 0 qiymatidan farqli yoki **true** bo‘lsa <operator$_1$>, aks holda <operator$_2$> bajariladi. **If-else** shart operator mazmuniga ko‘ra algoritmning tarmoqlanuvchi blokini ifodalaydi: <shart-ifoda> - shart bloki (romb) va <operator$_1$> blokning «ha» tarmog‘iga, <operator$_2$> esa blokning «yo‘q» tarmog‘iga mos keluvchi amallar bloklari deb qarash mumkin.



2.2-rasm. **if(), else** shart operatorining blok sxemasi

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

```
#include <iostream.h>
#include <math.h>
int main()
{
    float a,b,c;
    float D,x1,x2;
    cout<<''ax^2+bx+c=0; tenglama ildizini topish.'';
    cout<<''\n a- koiffitsiyentni kiriting: '';
    cin>>a;
    cout<<''\n b- koeffitsientni kiriting: '';
    cin>>b;
    cout<<''\n c- koeffitsientni 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;
        return o;
        }
        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 koeffitsientlari – a, b, c o‘zgaruvchilar qiymatlari kiritiladi, keyin diskriminant – D o‘zgaruvchining qiymati hisoblanadi. Keyin D o‘zgaruvchining manfiy ekanligi tekshiriladi. Agar shart o‘rinli bo‘lsa, yaxlit operator bajariladi va ekranga “Tenglama haqiqiy ildizlarga 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, hamda x1 o‘zgaruvchi qiymati chop qilinadi va dastur shu yerda o‘z ishini tugatadi, aks holda, ya’ni D qiymatni noldan katta holati uchun **else** kalit so‘zidan keyingi operatorlar bloki bajariladi va ekranga “Tenglama ikkita ildizga ega:“ xabari, hamda x1 va x2 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.

Masalan:

```
if(x==1)
if(y==1) cout<<”x=1 va y=1”;
else cout <<’’x< >1”;
```

Bu misolda «x< >1» xabari x qiymatini 1 ga teng va y qiymatini 1 ga teng bo‘lmagan holda ham chop etiladi. Quyidagi variantda ushbu mazmunan xatolik bartaraf etilgan:

```
if (x==1)
{
    if (y==1) cout<<”x=1 va y=1”;
}
else  cout<<”x< >1”;
```

C++ tilida shart operatorida umumiy bo‘lgan o‘zgaruvchilarni e’lon qilish man etiladi, lekin undagi bloklarda o‘zgaruvchilarni e’lon qilish mumkin va bu o‘zgaruvchilar faqat blok ichida amal qiladi. Quyidagi misolda bu holat bilan bog‘liq xatolik ko‘rsatilgan:

```
    if (j>0) {int i; i=2*j;}
```

else i=-j; //xato, chunki I blokdan tashqarida ko‘rinmaydi

Masala. Berilgan to‘rt xonali ishorasiz sonning boshidagi ikkita raqamining yig‘indisi qolgan raqamlar yig‘indisiga teng yoki yo‘qligi aniqlansin (raqamlar yig‘indisi deganda ularga mos son qiymatlarining yig‘indisi tushuniladi). Sonning raqamlarini ajratib olish uchun butun sonlar arifmetikasi amallaridan foydalaniladi:

```
#include <iostream.h>
int main()
{
    Unsigned int n,a3,a2,a1,a0;          //n=a3a2a1a0 ko’rinishida
    cout<<’’\nn-qiymatini kiriting:’’;
    cin>>n;
    If(n<1000| n>9999)
        {
        cout<<’’kiritilgan son 4 xonali emas!’’;
        return 1;
        }
    a3=n/1000;
    a2=n%1000/100;
    a1=n%100/10;
    a0=n%10;
    if(a3+a2==a1+a0) cout<<’’a3+a2=a1+a0’’;
    else cout<<’’a3+a2<>a1+a0’’;
    return 0;
}
```

Dastur ishorasiz butun son kiritishni taklif qiladi. Agar kiritilgan son 4 xonali bo‘lmasa (n<1000 yoki n>9999), bu haqda xabar beriladi va dastur o‘z ishini tugatadi. Aks holda n sonining raqamlari ajratib olinadi, hamda boshidagi ikkita raqamning yig’indisi – (a3+a2) qolgan ikkita raqamlar yig‘indisi – (a1+a0)

bilan solishtiriladi va ularning teng yoki yo‘qligiga qarab mos javob chop qilinadi.

?: shart amali: **C++** tilida **“?”** amali ham aniqlangan bo‘lib 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 ko‘raylik.

```
#include <iostream.h>
int main()
{
int a,b,c;
cout<<”a va b sonlar maksimumini topish dastursi.’’;
cout<<’’\n a- qiymatni kiriting: ‘’;
cin>>a;
cout<<’’\n b- qiymatni kiriting: ‘’;
cin>>b;
c=a>b?a:b;
cout<<’’\nSonlar maksimumi: ‘’<<c;
return 0;
}
```

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 c o‘zgaruvchiga a o‘zgaruvchi qiymati, aks holda b o‘zgaruvchining qiymati o‘zlashtiriladi va c o‘zgaruvchining qiymati chop etiladi.

?: amalining qiymat qaytarish xossasidan foydalangan holda, uni bevosita cout ko‘rsatmasiga yozish orqali ham qo‘yilgan masalani yechish mumkin:

```
#include <iostream.h>
int main()
{
int a,b;
cout<<”a va b sonlar maksimumini topish dastursi.’’;
cout<<’’\n a- qiymatni kiriting: ‘’;
cin>>a;
cout<<’’\n b- qiymatni kiriting: ‘’;
cin>>b;
c=a>b ? a : b;
cout<<’’\nSonlar maksimumi: ‘’<<(a>b) ?a:b;
return 0;
}
```

2.3. Tanlash operatori

Shart operatorining yana bir ko‘rinishi tanlash tarmoqlanish operatori bo‘lib, uning sintsksisi quyidagacha:

switch (<ifoda>)

{

case <o‘zgarmas ifoda$_1$> : <operatorlar guruhi$_1$>; **break**;

case <o‘zgarmas ifoda$_2$> : <operatorlar guruhi$_2$>; **break**;

…

case <o‘zgarmas ifoda$_n$> : <operatorlar guruhi$_n$>; **break**;

default:: <operatorlar guruhi$_{n+1}$>;

}

C++ tilida bu operator quyidagicha amal qiladi: 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 keyingi joylashgan operatorga o‘tadi. Agar <ifoda> birorta ham <o‘zgarmas ifoda$_i$> bilan mos kelmasa, qurilmaning **default** qismidagi <operatorlar guruhi$_{n+1}$> bajariladi. Shuni qayd etish kerakki, qurilmada **default** kalit so‘zi faqat bir marta uchrashi mumkin.

Tanlash operatorini qo‘llashga doir misolni qarab chiqaylik. Klaviaturadan kiritilgan “Jarayon davom etilsinmi?” so‘roviga foydalanuvchi tomonidan javob olinadi. Agar ijobiy javob olinsa, ekranga “Jarayon davom etadi!” xabari chop etiladi va dastur o‘z ishini tarmoqlanuvchi operatordan keyingi operatorlarni bajarish bilan davom ettiradi, aks holda “Jarayon tugadi!” javobi beriladi va dastur o‘z ishini tugatsin. Bu masala uchun tuziladigan dastur foydalanuvchining ‘y’ yoki ‘Y’ javoblari jarayonni davom ettirishni bildiradi, boshqa belgilar esa tugatishni anglatadi.

```
#include <iostream.h>
int main()
{
        char javob=' ';
        cout<<''jarayon davom etsinmi?('y','Y'):'';
        cin>>javob;
        switch(javob)
        {
                case 'y':
                case 'Y':
                        cout<<''jarayon davom etadi!\n'';
                        break;
```

```
            default:
                cout<<"jarayon tugadi!\n";
        return 0;
        }                                          //jarayon
    return 0;
    }
```

Tanlash operatorining **C++** tilidagi ko‘rinishida **break** va **default** kalit so‘zlarini ishlatmasa ham bo’ladi. Ammo bu holda operatorning mazmuni buzilishi mumkin. Masalan, **default** qismi bo‘lmagan holda, agar <ifoda> birorta <o‘zgarmas ifoda$_i$> bilan ustma-ust tushmasa, operator hech qanday amal bajarmasdan boshqaruv tanlash operatordan keyingi operatorga o‘tadi. Agar **break** bo‘lmasa, <ifoda> birorta <o’zgarmas ifoda$_i$> bilan ustma-ust tushgan holda, unga mos keluvchi operatorlar guruhini bajaradi va «to‘xtamasdan» keyingi qatordagi operatorlar guruhini bajarishga o‘tib ketadi. Masalan, yuqoridagi misolda **break** operatori bo‘lmasa va jarayonni davom ettirishni tasdiqlovchi (‘Y’) javob bo‘lgan taqdirda ekranga

Jarayon davom etadi!

Jarayon tugadi!

xabarlari chiqadi va dastur o‘z ishini tugatadi (**return** operatorining bajarilishi natijasida).

Tanlash operatori sanab o‘tiluvchi turdagi o‘zgarmaslar bilan birgalikda ishlatilganda samara beradi. Quyidagi dasturda ranglar gammasini toifalash masalasi yechilgan.

```
#include <iostream.h>
int main()
{
    enum Ranglar{qizil, tuq_sariq,sariq,yashil,kuk,zangori,binafsha};
    Ranglar rang;                          //...
```

```
    switch(rang)
{
    case qizil:
    case tuq_sariq:
    case sariq;
        cout<<''Issiq gamma tanlandi. \n'';
        break;
    case yashil:
    case kuk:
    case zangori:
    case binafsha:
        cout<<''Sovuq gamma tanlandi.\n'';
        break;
    default:
        cout<<''Kamalak bunday rangga ega emas. \n'';
}
return 0;
 }
```

Dastur bajarilishida boshqaruv tanlash operatorga kelganda, rang qiymati qizil yoki tuq_sariq yoki sariq bo‘lsa, “Issiq gamma tanlandi’’ xabari, agar rang qiymati yashil yoki kuk yoki zangori yoki binafsha bo‘lsa, ekranga “Sovuq gamma tanlandi” xabari, agar rang qiymati sanab o‘tilgan qiymatlardan farqli bo‘lsa, ekranga “Kamalak bunday rangga ega emas “ xabari chop etiladi va dastur o‘z ishini tugatadi.

C++ tilidagi tanlash operatorida e’lon operatorlari ham uchrashi mumkin. Lekin **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 mumkin:

```
    int k=0,n=0;
```

```
cin>>n;
switch (n)
{
int=10;                 //xato,bu operator hech qachon bajarilmaydi
case 1:
        int j=20;       //agar n=2 bo‘lsa,bu e’lon bajarilmaydi
case 2:
        k+=i+j;         //xato, chunki i,j o‘zgaruvchilar noma’lum
}
cout<<k;
```

Tanlash operatorini qo‘llashga doir yana bir masalani qarab chiqamiz.

Masala. Quyida, sanab o‘tiluvchi turlar va shu turdagi o‘zgaruvchilar e’lon qilingan:

```
enum Birlik{detsimetr, kilometr, metr, millimetr, santimetr}
float y; Birlik r;
```

Birlikda berilgan x o‘zgaruvchisining qiymat metrlarda chop qilinsin.

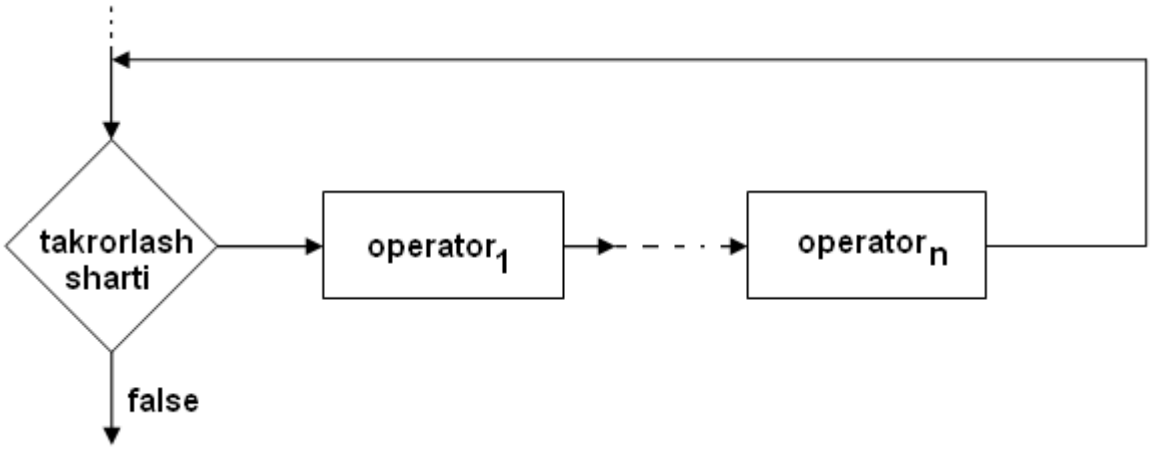
```
#include <iostream.h>
int main()
{
enum Birlik {detsimetr, kilometr, metr, millimetr, santimetr};
float x,y;
Birlik r;
cout<<’’uzunlikni kiriting: x=’’;
cin>>x;
cout<<’’ uzunlik birliklari\n’’;
cout<<’’ 0-detsimetr\n’’;
cout<<’’1-kilometr\n’’;
cout<<’’2-metr\n’’;
cout<<’’3-millimetr\n’’;
```

```
cout<<''4-santimetr\n'';
cout<<''uzunlikni birligini tanlang; r='';
cin>>r;
switch(r)
{
        case detsimetr:y=x/10; break;
        case kilometr: y=x*1000; break;
        case metr: y=x; break;
        case millimetr: y=x/1000; break;
        case santimetr: y=x/100; break;
        default:
        cout<<''uzunlik birligi noto'g'ri kiritildi!'';
        return 0;
}
cout<<y<<''metr'';
return 0;
 }
```

2.4. Takrorlash operatorlari

Dastur bajarilishini boshqarishning boshqa bir kuchli mexanizmlaridan biri – takrorlash operatorlari hisoblanadi.

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



2.3-rasm. Takrorlash operatorining blok sxemasi

Takrorlash o‘zining kirish va chiqish nuqtalariga ega, lekin chiqish nuqtasining bo‘lmasligi mumkin. Bu holda takrorlashga cheksiz takrorlash deyiladi. Cheksiz takrorlash uchun takrorlashni davom ettirish sharti doimo rost bo’ladi.

Takrorlash shartini tekshirish takrorlash tanasidagi operatorlarni bajarishdan oldin tekshirilishi mumkin (**for**, **while** takrorlashlari) yoki takrorlash tanasidagi operatorlari bir marta bajarilgandan keyin tekshirilishi mumkin (repeat-until, do-while).

Takrorlash operatorlari ichma-ich joylashgan bo‘lishi ham mumkin.

for takrorlash operatori: C++ dasturlash tilida for takrorlash operatorining sintsksisi quyidagi ko‘rinishga ega:

for (<ifoda1>; <ifoda2>;<ifoda3>) <operatorlar guruhi>;

Bu operator o‘z ishini <ifoda1> ifodasini bajarishdan boshlaydi. Keyin takrorlash qadamlari boshlanadi. Har bir qadamda <ifoda2> bajariladi, agar natija 0 qiymatidan farqli yoki true bo‘lsa, takrorlash tanasi - <operatorlar guruhi> bajariladi va oxirida <ifoda3> bajariladi. Agar <ifoda2> qiymati 0 (**false**) bo‘lsa, takrorlash jarayoni to‘xtaydi va boshqaruv takrorlash operatoridan keyingi operatorga o‘tadi. Shuni qayd qilish kerakki, <ifoda2> ifodasi vergul bilan ajratilgan bir nechta ifodalar birlashmasidan iborat bo‘lishi mumkin, bu holda oxirgi ifoda qiymati takrorlash sharti hisoblanadi. Takrorlash tanasi

sifatida bitta operator, jumladan bo‘sh operator bo‘lishi yoki operatorlar guruhi kelishi mumkin.

Misol uchun 1 dan 20 gacha bo‘lgan butun sonlar yig‘indisini hisoblash masalasini ko‘raylik.

```
#include <iostream.h>
int main(){
        int Summa=0;
        for (int i=1; i<=20; i++)
        Summa+=i;
        cout<<''yig'indi='' <<Summa;
        return 0;
}
```

Dasturdagi takrorlash operatori o‘z ishini, **i** takrorlash parametriga (takrorlash sanagichiga) boshlang‘ich qiymat – 1 sonini berishdan boshlaydi va har bir takrorlash qadamidan (itaratsiyadan) keyin uning qiymati bittaga oshadi. Har bir takrorlash qadamida takrorlash tanasidagi operator bajariladi, ya’ni summa o‘zgaruvchisiga **i** ning qiymati qo‘shiladi. Takrorlash sanagichi **i** ning qiymati 21 bo‘lganda “i<=20” takrorlash sharti (0-qiymati) bo’ladi va takrorlash tugaydi. Natijada boshqaruv takrorlash operatoridan keyingi operatorga o‘tadi va ekranga yig’indi chop etiladi.

Yuqorida keltirilgan misolga qarab takrorlash operatorlarining qavs ichidagi ifodalariga izoh berish mumkin:

<ifoda1> - takrorlash sanagichi vazifasini bajaruvchi o‘zgaruvchisiga boshlang’ich qiymat berishga xizmat qiladi va u takrorlash jarayoni boshida faqat bir marta hisoblanadi. Ifodada o‘zgaruvchi e’loni uchrash mumkin va bu o‘zgaruvchi takrorlash operatori tanasida amal qiladi va takrorlash operatoridan tashqarida «ko‘rinmaydi» (**C++** Builder kopiilyatori uchun);

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 sanagichning qiymatini oshirish (kamaytirish) uchun xizmat qiladi yoki unda takrorlash shartiga ta’sir boshqa amallar bo‘lishi mumkin.

Takrorlash operatorida qavs ichidagi ifodalar bo‘lmasligi mumkin, lekin sintsksis: ‘;’ bo‘lmasligiga ruxsat bermaydi. Shu sababli sodda ko‘rinishdagi takrorlash operatori quyidagicha bo’ladi:

for(;;) **cout** <<”Cheksiz takrorlash…’’;

Agar takrorlash jarayonida bir nechta o‘zgaruvchilarning qiymati sinxron ravishda o‘zgarishi kerak bo‘lsa, <ifoda$_1$> va <ifoda$_3$> ifodalarida zarur operatorlarni ‘,’ bilan yozish orqali bunga erishish mumkin:

```
for(int  i=10 , j=2 ; i<=20 ; i++ , j=i+10)
{
…
}
```

Takrorlash operatorining har bir qadamida **j** va **i** o‘zgaruvchi qiymatlari mos ravishda o‘zgarib boradi.

For operatorida takrorlash tanasi bo‘lmasligi ham mumkin. Masalan, dastur bajarilishini ma’lum bir muddatga «to‘xtab» turish zarur bo‘lsa, bunga takrorlashni hech qanday qo‘shimcha ishlarni bajarmasdan amal qilishi orqali erishish mumkin:

```
#include <iostream .h>
int main(){
int delay;
…
for (delay=5000; delay>0; delay--);          //  bo’sh operator
…
```

```
return 0;
}
```

Yuqorida keltirilgan 1 dan 20 gacha bo‘lgan sonlar yig‘indisini bo‘sh tanali (bo‘sh operatorli) takrorlash operatori orqali hisoblash mumkin:

```
…
for (int i=1;  i<=20;  summa+=i++) ;
…
```

Takrorlash operatori tanasi sifatida operatorlar guruhi ishlatishini faktorialni hisoblash misolida ko‘rsatish mumkin:

```
#include <iostream.h>
int main(){
        int a;
        unsigned long fact=1;
        cout<<’’butun sonni kiriting:_’’;
        cin>>a;
        if ((a>=0)&&(a<33)){
                for (int i=1; i<=a; i++) fact*=i;
        cout<<a<<’’factorial’’<<fact<<” ga teng \n”;
        }
return 0;
}
```

Dastur foydalanuvchi tomonidan 0 dan 33 gacha oraliqdagi son kiritilganda amal qiladi, chunki 34! Qiymati **unsigned long** uchun ajratilgan razryadlarga sig‘maydi.

Masala. Takrorlash operatorining ichma-ich joylashuviga misol sifatida raqamlari bir-biriga o‘zaro teng bo‘lmagan uch xonali natural sonlarni o‘sish tartibida chop qilish masalasini ko‘rishimiz mumkin:

```
#include <iostream.h>
int main(){
```

```
unsigned char a2,a1,a0;          // uch xonali son raqamlari
for (a2=' 1' ;a2<=' 9' ;a2++)    //sonning 2-o‘rindagi raqami
for (a1=' 0' ;a1<=' 9' ;a1++)    //sonning 1-o‘rindagi raqami
for (a0=' 0' ;a0<=' 9' ;a0++)    //sonning 0-o‘rindagi raqami
                                 // raqamlarni o‘zaro teng emasligini
                                 tekshirish
if(a0!=a1 && a1!=a2 && a0!=a2)   //o‘zaro teng emas
cout<<a2<<a1<<a0<<'\n' ;
return 0;
}
```

Dasturda uch xonali sonning har bir raqami takrorlash operatorlari yordamida hosil qilinadi. Birinchi, tashqi takrorlash operatori bilan 2-xonadagi raqam (a2 takrorlash parametri) hosil qilinadi. Ikkinchi, ichki takrorlash operatorida (a1 takrorlash parametri) son ko‘rinishining 1-xonasidagi raqam va nihoyat, unga nisbatan ichki bo‘gan a0 parametrli takrorlash operatorida 0-xonadagi raqamlar hosil qilinadi. Har bir tashqi takrorlashning bir qadamiga ichki takrorlash operatorining to‘liq bajarilishi to‘g‘ri keladi.

While takrorlash operatori: While takrorlash operatori, operator yoki blokni takrorlash sharti yolg’on (**false** yoki 0) bo‘lguncha takror bajaradi.

While sikl operatori takrorlanishlar soni oldindan aniq bo‘lmagan hollarda takrorlanishni biror-bir shart asosida bajaradi. Berilgan shart oldin tekshiriladi va keyin shartning rost yoki yolg‘onligiga qarab kerakli operatorlar ketma-ketligi bajariladi. Bu operatorning **C++** tilidagi sintaksisini keltiramiz:

while (<ifoda>) <operatorlar guruhi>;

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 <operatorlar guruhi> qismi bajarilmasdan chetlab o’tiladi.

Ayrim hollarda <ifoda> qiymat berish operatori ko‘rinishida kelishi mumkin. Bunda qiymat berish amali bajariladi va natija 0 solishtiriladi. Natija noldan farqli bo‘lsa, takrorlash davom ettiriladi.

Agar rost ifodaning qiymati noldan farqli o‘zgarmas bo‘lsa, cheksiz takrorlash ro‘y beradi. **Masalan:** **While**(1); // cheksiz takrorlash

Xuddi **for** operatoridek, ‘,’ yordamida <ifoda> da bir nechta amallar sinxron ravishda bajarilishi mumkin. Masalan, son va uning kvadratlarini chop qiladigan dasturda ushbu holat ko‘rsatilgan:

```
#include <iostream.h>
int main(){
int n, n2;
cout<<’’sonni kiriting(1..10):_’’;
cin>>n;
n++;
while(n--,n2=n*n, n>0)
cout<<’’n=’’<<n<<’’ n^2=’’<<n2<<end1;
return 0;
}
```

Dasturdagi takrorlash operatori bajarilishida **n** soni 1 gacha kamayib boradi. Har bir qadamda **n** va uning kvadrati chop qilinadi. Shunga e’tibor berish kerakki, shart ifodasida operatorlarni yozilish ketma-ketligining ahamiyati bor, chunki, eng oxirgi operator takrorlash sharti sifatida qaraladi va n ning qiymati 0 bo‘lganda takrorlash tugaydi.

While takrorlash operatori yordamida samarali dastur kodi yozishga misol sifatida ikkita natural sonlarning eng katta umumiy bo'luvchisi (EKUB)ni Evklid algoritmi bilan topish masalasini ko'rishimiz mumkin:

```
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
        cin<<'' bu  sonlar EKUBi=''<<a;
        return 0;
 }
```

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

Sharti keyin tekshiriladigan sikl operatori: Sharti keyin tekshiriladigan sikl operatori ham takrorlanishlar soni oldindan aniq bo'lmagan hollarda takrorlanishni biror-bir shart asosida bajaradi. Oldin sikl tanasidagi operatorlar ketma-ketligi bajariladi. Berilgan shart keyin tekshiriladi.

C++ tilida sharti keyin tekshiriladigan operator sifatida **do-while** operatori ishlatiladi. **Do-while** takrorlash operatori quyidagi sintaksisga ega:

do <operatorlar guruhi>; **while** (<ifoda>);

```
#include <iostream.h>
int main()
{
```

```
char javob;
do{
        …                                                    // sikl tanasi
        cout<<'' jarayonni to'xtashish(N):_'';
        cin>>javob;
}
while(javob !="N")
return 0;
}
```

Dastur toki "jarayonni to'xtash (N):_''so'roviga (N) javobi kiritilmaguncha davom etadi.

Shuni ta'kidlash kerakki **do-while** operatori **while** kalit so'zidan keying ifodaning qiy mati rost bo'lsa, takrorlanishni davom ettiradi, aks holda keying operatorlar bajarilishi davom etadi.

Masala. Har qanday 7 dan katta butun sondagi pul miqdoriga ega kupyuralarni 3 va 5 so'mlik kupyuralarda berish mumkinligi isbotlansin. Qo'yilgan masala p=3n+5m tenglamasini qanotlantiruvchi m, n sonlar juftliklarini topish masalasidir (p-pul miqdori). Bu shartning bajarilishini m va n o'zgaruvchilarining mumkin bo'lgan qiymatlarining barcha kombinatsiyalarida tekshirish zarur bo'ladi.

```
#include <iostream.h>
int main(){
unsigned int pul;                    //pu1-  kiritiladigan pul miqdori
unsigned int n3,m5;                  //n-3 so'mliklar , m-5 so'mliklar soni
bool xato=fa1se;                     //pu1  qiymatini kiritilgandagi xatolik
do{
        if (xato) cout<<''kiritilgan pul qiymati 7 dan kichik !'';
        xato=true ;                  //keyingi takrorlash xato hisoblanadi
        cout<<''\npul qiymatini kiriting (>7):'';
```

```
    cin>>pul;
}
while (pul<=7);                    // toki 7 sonidan katta son kiritulguncha
n3=0 ;                             //birorta ham 3 so'mlik yo'q
do{
m5=0;                              // birorta ham 5 so'mlik yo'q
do
{
if (3*n3+5*m5==pul)
cout<<n3<<''ta 3 so'mlik + ''<<m5<<'' ta 5 so'mlik\n'';
m5++                               //5 so'mliklar bittaga oshiriladi
}
while(3*n3+5*m5<=pul);
n3++;                              //3 so'mliklar bittaga oshiriladi
}
while(3*n3<=pul);
return 0;}
```

Dastur pul qiymatini kiritishni so‘raydi(pu1 o‘zgaruvchisiga). Agar **pul** qiymati 7 sonidan kichik bo‘lsa, bu haqda xabar beriladi va takror ravishda qiymat kiritish talab qilinadi. **Pul** qiymati 7 dan katta bo’lganda, 3 va 5 so‘mliklarning mumkin bo‘lgan to‘la kombinatsiyasini amalga oshirish uchun ichma-ich takrorlashlar amalga oshiriladi. Tashqi takrorlash **n3** (3 so‘mliklar miqdori) bo‘yicha, ichki takrorlash esa **m5** (5 so‘mliklar miqdori) bo‘yicha, toki bu miqdordagi pullar qiymati **pul** qiymatidan oshib ketmaguncha davom etadi. Ichki takrorlashda **m5** o’zgaruvchisining har bir qiymatida «3*n3+5*m5=**pu1**» sharti tekshiriladi, agar u o‘rinli bo‘lsa, yechim varianti sifatida **n3** va **m5** o’zgaruvchilar qiymatlari chop etiladi.

Pul qiymati 30 so‘m kiritilganda, ekranga

0 ta 3 so’mlik +6 ta 5 so’mlik chop etiladi.

5 ta 3 so’mlik +6 ta 5 so’mlik

10 ta 3 so’mlik +0 ta 5 so’mlik

yechim variantlari chop etiladi.

2.5. Goto operatori va nishonlar.

Dasturda shunday holatlar bo’ladiki, operatorlarning bajarilishiga qarab dasturning u yoki bu qismiga to‘g‘ridan-to‘g‘ri bajarishni uzatish ehtiyoji tug‘iladi. Bunday holatlarda shartsiz o‘tish operatoridan foydalanish mumkin.

C++ tilida shartsiz o‘tish operatorining sintsksisi quyidagicha:

Goto <nishon>;

Bu yerda <nishon> - belgi(metka) bo‘lib identifikator bo‘lishi mumkin. Goto - o‘tish ma’nosini bildiradi.

C++ tillarida e’lon qilingan nishonlar qayerda e’lon qilinishiga qarab faqat e’lon qilingan (funksiya, qism dastur) sohada ko‘rinadi.

Goto operatorida qo‘llaniladigan identifikatorlar **C++** tilida Delphidagi kabi e’lon qilinmaydi.

Shuni ta’kidlash lozimki **C++** tilida dastur tuzish jarayonidagi ayrim hollarda goto operatoridan foydalanib «sakrab o‘tishi» hisobiga xatoliklar yuzaga kelishi mumkin. Masalan,

int i=0;

i++; **if**(i) **goto** m;

int j;

m: j+=I;

Bu misoldagi **goto** operatorining bajarilishi xatolikka olib keladi, chunki j e’lon qilinmay qoladi.

Shartsiz o‘tish operatori dastur tuzishdagi kuchli va shu bilan birga xavfli vositalardan biri hisoblanadi. Kuchliligi shundaki, u 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. Shu sababli, imkon qadar **goto** operatoridan foydalanmaslik kerak, ishlatilgan taqdirda ham qo’yidagi qoidalarga amal qilish zarur: blok ichiga, **if…else** va tanlash operatorlari ichiga hamda takrorlash operatorlari tanasiga tashqaridan kirish mumkin emas.

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.

Xususan, nishon yordamida ichki blokdan tashqi blokka va tashqi blokdan ichki blokka o‘tishga **C++** tili ruxsat beradi:

```
{…
goto ABC:
…
{int i=15;
…
ABC:
…
goto XYZ;
int y=10;
…
goto KLM;
…}
…
int k=0;
…
KLM:
…}
```

Lekin, yuqorida keltirilgan misoldagi barcha o‘tishlar mazmunan xato hisoblanadi.

Quyidagi dasturda ikkita natural sonlar EKUBini topish masalasidagi takrorlash jarayonini nishon va **goto** operatori vositasida amalga oshirish ko‘rsatilgan:

```
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-=b:b-=a;
    goto nishon;
}
```

Dasturdagi nishon bilan belgilangan operatorda **a** va **b** o‘zgaruvchilarining qiymati tengligi tekshiriladi. Agar ular teng bo‘lsa, ixtiyoriy bittasi, masalan **a** o‘zgaruvchisidagi son **EKUB** bo’ladi va dastur ishini yakunlaydi. Aks holda, bu sonlarning kattasidan kichigi ayriladi va goto orqali ularning tengligi tekshiriluvchi shart operatoriga o‘tiladi. Takrorlash jarayoni a va b sonlar o‘zaro teng bo‘lguncha davom etadi.

Shuni qayd etish kerakki, bu masalani takrorlash operatorlari yordamida bajarish ancha samarali hisoblanadi.

2.6. Break va continue operatorlari

Takrorlash operatorlarining bajarilishida shunday holatlar yuzaga kelishi mumkinki, unda qaysidir qadamda, takrorlashni yakuniga yetkazmasdan 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. E’tibor beradigan bo’lsak **switch-case** operatorining tub mohiyatiga ham **break** operatorini qo’llash orqali erishilgan.

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

Quyidagi dasturda ikkita ichma-ich joylashgan takrorlash operatoridan foydalangan holda foydalanuvchi tomonidan kiritilgan qandaydir sonni 3 va 7 sonlariga nisbatan qanday oraliqqa tushishi aniqlanadi .Tashqi takrorlashda ‘’son kiriting (0-to’xtash):_’’ so‘rovi beriladi va javob javob_son o‘zgaruvchisiga o‘qiladi. Agar son noldan farqli bo‘lsa, ichki takrorlash operatorida bu sonning qandaydir tushishi aniqlanib, shu haqida xabar beriladi va ichki operatoridan chiqiladi. Tashqi takrorlashdagi so‘rovga javob tariqasida 0 kiritilsa, dastur o‘z ishini tugatadi.

```
#include <iostream.h>
int main(){
int javob_son=0;
do{
        while (javob_son){
                if (javob_son<3){
                        cout<<’’3 kichik !’’;
                        break;
                }
                if(3<=javob_son&& javob_son<=7){
                        cout<<’’3 va 7 oralig’da !’’;
                        break;
                }
                if (javob_son>7){
                        cout<<”7 dan katta !”;
```

```
                    break;
                }
            }
            cout<<"\nSon kiriting (0-to'xtash):_";
            cin>>javob_son;
        }
        while(javob_son !=0)
        return 0
        }
```

Amaliyotda **break** operatoridan cheksiz takrorlashdan chiqishda foydalaniladi.

```
For (;;){                       // 1-shart
if (…){
        …
        break ;
}
if (…)
{                               // 2- shart
        …
        break;
}
…
}
```

Bu misolda cheksiz **for** takrorlashidan 1 yoki 2- shart bajarilganda chiqiladi,

Masala. Ishorasiz butun sonlar ketma-ketligi 0 qiymati bilan tugaydi. Bu yerda 0 ketma-ketlik hadi hisoblanmaydi. Ketma-ketlikni kamaymaydigan holda tartiblangan yoki yo‘qdigi aniqlansin.

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

```
int main(){
unsigned int Ai_1=0,Ai;
cout<<'' sonlar ketma-ketligini kiriting''
cout<<(0-tugash alomati):\n'';
cin>>Ai;                          //  ketma-ketlikning birinchi hadi
while(Ai){
        Ai_1>Ai;
        cin>>Ai;                  //  navbatdagi had
        if (Ai_1>Ai) break;
}
if (Ai_1)
        {
                cout<<''ketma-ketlik kamaymaydigan holda tartiblangan'';
                if(!Ai)cout<<''emas!'';
                else cout<<''!'';
        }
}
else cout<<''ketma-ketlik bo'sh!'';
return 0;
}
```

Dastur ishga tushganda, boshida ketma-ketlikning birinchi hadi alohida o‘qib olinadi (**Ai** o‘zgaruvchisiga). Keyin **Ai** qiymati nolga teng bo’lmaguncha takrorlash operatori amal qiladi. Takrorlash tanasida **Ai** qiymati oldingi qiymat sifatida **Ai_1** o‘zgaruvchisida eslab qolinadi va navbatdagi had **Ai** o‘zgaruvchisiga o‘qiladi. Agar oldingi had navbatdagi haddan katta bo‘lsa, **break** operatori yordamida takrorlash jarayoni uziladi va boshqaruv takrorlashdan keyingi shart operatoriga o‘tadi. Bu yerdagi shart operatorlari mazmuni quyidagicha agar **Ai_1** noldan farqli bo‘lsa, ketma-ketlikning kamida bitta hadi kiritilgan bo’ladi (ketma-ketlik mavjud) va oxirgi kiritilgan had tekshiriladi. O’z navbatida agar **Ai**

noldan farqli bo‘lsa, bu holat had1ar o’rtasida kamaymaslik sharti bajarilmaganligi sababli hadlarni kiritish jarayoni uzilganligini bildiradi va bu haqda xabar chop etiladi. Aks holda ketma-ketlikni kamaymaydigan holda tartiblangan bo’ladi.

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

Continue operatorini qo‘llanishiga misol tariqasida 2 va 50 sonlar oralig‘idagi tub sonlarni topadigan dastur matnini keltiramiz.

```
#include <iostream.h>
int main(){
bool bulinadi=false;
        for (int i=2; i<50; i++){
                for (int j=2; j<i/2; j++){
                        if (i%j) continue;
                        else {
                                bulinadi=true;
                                break;
                        }
                }               // break bajarilganda boshqaruv o’tadigan joy
                if (!bulinadi ) cout <<i<<” “;
                bulinadi=false;
        }
return 0;
}
```

Keltirilgan dasturda qo‘yilgan masala ichma-ich joylashgan ikkita takrorlash operatorlari yordamida yechilgan. Birinchi takrorlash operatori 2 dan 50 gacha sonlarni hosil qilishga xizmat qiladi. Ichki takrorlash esa har bir hosil qilinayotgan sonni 2 sonidan toki shu sonning yarmigacha bo‘lgan sonlarga

bo‘lib, qoldig‘ini tekshiradi, agar qoldiq 0 sonidan farqli bo‘lsa, navbatdagi songa bo‘lish davom etadi, aks holda bo‘linadi o‘zgaruvchisiga true qiymat berib, ichki takrorlash uziladi (son o‘zining yarmigacha bo‘lgan qandaydir songa bo‘linar ekan, demak u tub emas va keyingi sonlarga bo‘lib tekshirishga hojat yo‘q). Ichki takrorlashdan chiqqandan keyin bo‘linadi qiymati false bo‘lsa (bo`linadi), son tub bo’ladi va u chop qilinadi.

III BOB FAYLLAR BILAN ISHLASH

3.1 Fayllar bilan ishalash uchun ilk sozlash

C++ dasturlash tilida fayllar bilan ishlash **ftsream** kutubxonasidagi biron bir sinflar yordamida amalga oshiriladi.

ftsream kutubxonasi fayllarni o’qib olish javob beradigan **ifstream** sinfiga hamda faylga axborotni yozib qo’yilishiga javob beradigan **oftsream** sinfiga ega.

Biron-bir faylni yozish yoki o’qish maqsadida ochish uchun , oftsream turdagi yoki mos holda iftsream turdagi o’zgaruvchini e’lon qilish kerak. Bunday o’zgaruvchini e’lon qilishda fayl nomi o’zgaruvchi nomidan keyin qavs ichida berilgan belgilar massivi ko’rinishida uzatiladi.

Masalan, C diskda joylashgan ‘text.txt’ faylini ochish kerak. Buning uchun kodning quyidagi fragmenti qo’llaniladi:

```
ifstream ifI (‘’C:\text.txt’’);
ofstream ofI(‘’C:\text.txt’’):
char s[20]=’’C:\text.txt’’;
ifstream ifJ (s);
```

Bu yerda ifI, ifJ va ofI - o’zgaruvchilar nomi bo’lib, ular orqali fayl bilan ma’lumotlarni ayirboshlash amalga oshiriladi. Agar fayl xam dasturning bajarilayotgan fayli joylashtirilgan papkada bo’lsa, u xolda faylning nomi to’liq ko’rsatilmasligi mumkin (faqat fayl nomi, unga borish yo’lisiz). Bundan tashqari fayl nomini to’g’ridan-to’ri ko’rsatish o’rniga, uning nomidan iborat belgilar massivlarini ko’rsatish mumkin.

3.2 Faylga yozish

Axborotni faylga yozish uchun put komandasidan foydalanish mumkin. Bu komanda orqali standart turdagi yakka o'zgaruvchi yoki biron-bir belgilar massivi uzatiladi. Belgilar massivi uzatilgan xolda xam massivdagi belgilar sonini uzatish kerak.

Bundan tashqari ''<<''operatoridan foydalanish mumkin. Bu operatordan kodning bitta satrida turli turdagi qiymatlarni uzatgan xolda ko'p martalab foydalanish mumkin. Satr xaqida gap ketganda, chiqarish satr oxiri belgisi, ya'ni '\n' paydo bo'lishidan oldin amalga oshiriladi. Belgisiz turga ega bo'lgan barcha o'zgaruvchilar oldin belgilarga o'zgartirib olinadi.

```
ifstream ofI (''C:\text.txt'');
char a='M';
ofI.put(s);
char s[9]=''The text'';
ofI.put(s,9);
ofI<<''The text'';
int i=100;
ofI<<i;
char ss[]=''Salom, Dunyo!'';
int k=200;
ofI<<'' Salom, Dunyo!''<<k<<ss<<200;
```

3.3 Fayldan o'qish

Axborotni fayldan o'qib olish uchun ''>>'' operatoriga ekvivalent bo'lgan **get** funksiyasi qo'llanadi. **Put** funksiyasi kabi, **get** funksiyasi xam har qanday o'zgaruvchilarning standart turlari yoki / va belgilar massivlari bilan ishlay oladi. Shuningdek **get** gaxar jixatdan ekvivalent bo'lgan **getline** funksiyasi mavjud: farqi faqat shundaki, **getline** funksiyasi satr oxiridan oxirgi belgini qaytarmaydi.

```
ifstream ofI (''C:\text.txt'');
char s; char ss[9];
s=ofI.get ();
cout<<s;
ofI.get(s);
cout<<s;
ofI.getline(ss,9);
cout<<ss;
ofI>>ss;
cout<<ss;
```

3.4 Fayl oxirini aniqlash

Fayl ichidagisini, fayl oxiri uchramaguncha, o'qish dasturdagi oddiy fayl operatsiyasi xisoblanadi. Fayl oxirini aniqlash uchun, dasturlar oqim ob'ektining **eof** funksiyasidan foydalanishlari mumkin. Agar fayl oxiri hali uchramagan bo'lsa, bu funksiya 0 qiymatini qaytarib beradi, agar fayl oxiri uchrasa, 1 qimatini qaytaradi. **While** siklidan foydalanib, dasturlar, fayl oxirini topmagunlaricha, quyida ko'rsatilganidek, uning ichidagilarini uzluksiz o'qishlari mumkin:

```
while (! Input_file.eof())
{
  //Operatorlar
}
```

Ushbu xolda dastur, **eof** funksiyasi yolg'on (0) ni qaytarguncha, siklni bajarishda davom etadi.

Xuddi shunday, keyingi dastur - WORD_EOF.CPP fayl ichidagisini bitta so'z bo'yicha bir martada, fayl oxiri uchramaguncha, o'qiydi:

```
#include <iostream.h>
#include <fstream.h>
```

```
void main(void)
{
  ifstream input_file("BOOKINFO.DAT");
  char word[64] ;
  while (! input_file.eof())
  {
    input_file >> word;
    cout << word << endl;
  }
}
```

IV BOB FUNKSIYALAR VA MASSIVLAR

4.1 Funksiya va uning tuzilishi.

C++ da dasturlashning asosiy bloklaridan biri funksiyalardir. Funksiyalar dasturchi ishini juda yengillashtiradi. Funksiyalar yordamida programma modullashadi, qismlarga bo'linadi. Bu esa keyinchalik dasturni rivojlantirishni osonlashtiradi. Bunda dasturchi yozgan funksiyalar C++ ning standart kutubhonasi va boshqa kutubhonalar ichidagi funksiyalar bilan birlashtiriladi. Bu esa ishni osonlashtiradi. Ko'p holda dasturda takroran bajariladigan amalni funksiya sifatida yozish va kerakli joyda ushbu funksiyani chaqirish mumkin. Dastur yozilish davrida hatolarni topishni yengillashtiradi. Bir misolda funksiyaning asosiy qismlarini ko'rib chiqaylik.

```
int foo(int k, int t) {
        int result;
        result = k * t;
        return (result);
}
```

Yuqoridagi **foo** funksiyamizning ismi, () qavslar ichidagi parametrlar – **int** tipidagi **k** va **t** lar kirish argumentlaridir, ular faqat ushbu funksiya ichida

ko'rinadi va qo'llaniladi. Bunday o'zgaruvchilar lokal (**local** - mahalliy) deyiladi. result **foo**() ning ichida e’lon qilinganligi uchun u ham lokaldir. Demak biz funksiya ichida o'zgaruvchilarni va sinflarni (**class**) e’lon qilishimiz mumkin ekan. Lekin funksiya ichida boshqa funksiyani e’lon qilib bo'lmaydi. **foo**() funksiyamiz qiymat ham qaytaradi. Qaytish qiymatining tipi **foo**() ning e’lonida eng boshida kelgan - **int** tipiga ega. Biz funksiyadan qaytarmoqchi bo'lgan qiymatning tipi ham funksiya e’lon qilgan qaytish qiymati tipiga mos kelishi kerak - ayni o'sha tipda bo'lishi yoki o'sha tipga keltirilishi mumkin bo'lgan tipga ega bo'lishi shart. Funksiyadan qiymatni return ifodasi bilan qaytaramiz. Agar funksiya hech narsa qaytarmasa e’londa void tipini yozamiz. Yani:

```
void funk(){
  int g = 10;
  cout << g;
  return;
}
```

Bu funksiya **void** (bo’sh, hech narsasiz) tipidagi qiymatni qaytaradi. Boshqacha qilib aytganda qaytargan qiymati bo’sh to’plamdir. Lekin funksiya hech narsa qaytarmaydi deya olmaymiz. Chunki hech narsa qaytarmaydigan mahsus funksiyalar ham bor. Ularning qaytish qiymati belgilanadigan joyga hech narsa yozilmaydi.

Biz unday funksiyalarni keyinroq ko’rib chiqamiz. Bu yerda bir nuqta shuki, agar funksiya mahsus bo’lmasa, lekin oldida qaytish qiymati tipi ko’rsatilmagan bo'lsa, qaytish qiymati **int** tipiga ega deb qabul qilinadi.

void qaytish tipli funksiyalardan chiqish uchun **return**; deb yozsak yetarlidir. Yoki **return** ni qoldirib ketsak ham bo’ladi.

Funksiyaning qismlari bajaradan vazifasiga ko’ra turlicha nomlanadi. Yuqorida korib chiqqanimiz funksiya aniqlanishi (**function definition**) deyiladi, chunki biz bunda funksiyaning bajaradigan amallarini funksiya nomidan keyin, {} qavslar ichida aniqlab yozib chiqyapmiz. Funksiya aniqlanishida {} qavslardan

oldin nuqta-vergul (;) qo'yish hatodir. Bundan tashqari funksiya e'loni, prototipi yoki deklaratsiyasi (**function prototype**) tushunchasi qo'llaniladi.

Bunda funksiyaning nomidan keyin hamon nuqta-vergul qo'yiladi, funksiya tanasi esa berilmaydi. **C++** da funksiya qo'llanilishidan oldin uning aniqlanishi yoki hech bo'lmaganda e'loni kompilyatorga uchragan bo'lishi kerak. Agar funksiya e'loni boshqa funksiyalar aniqlanishidan tashqarida berilgan bo'lsa, uning kuchi ushbu fayl ohirigacha boradi. Biror bir funksiya ichida berilgan bo'lsa kuchi faqat o'cha funksiya ichida tarqaladi. E'lon fayllarda aynan shu funksiya e'lonlari berilgan bo'ladi. Funksiya e'loni va funksiya aniqlanishi bir-biriga mos tushishi kerak.

Funksiya e'loniga misol:

```
double square(char, bool);
float average(int a, int b, int c);
```

Funksiya e'lonlarda kirish parametrlarining faqat tipi yozish kifoya, huddi square() funksiyasidek. Yoki kiruvchi parametrlarning nomi ham berilishi mumkin, bu nomlar kompilyator tarafidan etiborga olinmaydi, biroq dasturning o'qilishini ancha osonlashtiradi.

Bulardan tashqari **C++** da funksiya imzosi (**function signature**) tushunchasi bor. Funksiya imzosiga funksiya nomi, kiruvchi parametrlar tipi, soni, ketma-ketligi kiradi. Funksiyadan qaytuvchi qiymat tipi imzoga kirmaydi.

```
int foo();                  //No1
int foo(char, int);         //No2
double foo();               //No3 - No1 funksiya bilan imzolari ayni.
void foo(int, char);        //No4 - No2 bilan imzolari farqli.
char foo(char, int);        //No5 - No2 bilan imzolari ayni.
int foo(void);              //No6 - No1 va No3 bilan imzolari ayni,
```

Yuqoridagi misolda kirish parametrlari bo'lmasa biz () qavsning ichiga **void** deb yozishimiz mumkin (No6 ga qarang). Yoki () qavslarning quruq o'zini yozaversak ham bo'ladi (No1 ga qarang).

Yana bir tushuncha - funksiya chaqirig'idir. Dasturda funksiyani chaqirib, qo'llashimiz uchun uning chaqiriq ko'rinishini ishlatamiz. () qavslari funksiya chaqirig'ida qo'llaniladi. Agar funksiyaning kirish argumentlari bo'lmasa, () qavslar bo'sh holda qo'llaniladi.

4.2 Matematik kutubhona funksiyalari

Standart kutubhonaning matematik funksiyalari ko'pgina amallarni bajarishga imkon beradi. Biz bu kutubhona misolida funksiyalar bilan ishlashni ko'rib chiqamiz.

Masalan bizning dasturimizda quyidagi satr bor bo'lsin:

double = k;
int m = 123;
k = **sin**(m);

Kompilyator ushbu satrni ko'rganida, standart kutubhonadan **sin** funksiyasini chaqiradi. Kirish qiymati sifatida **m** ni berdik. Javob, ya'ni funksiyadan qaytgan qiymat **k** ga berildi. Funksiya agumentlari o'zgarmas sonlar (**const**), o'zgaruvchilar, ifodalar va boshqa mos keluvchi qiymat qaytaradigan funksiyalar bo'lishi mumkin. **Masalan**:

int g = 49, k = 100;
cout << "4900 ning ildizi -> "<< **sqrt**(g * k);

Ekranda:

4900 ning ildizi -> 70;

Matematik funksiyalar aksariyat hollarda **double** tipidagi qiymat qaytarishadi.

Kiruvchi argumentning tipi sifatida esa **double** ga keltirilishi mumkin bo'lgan tip beriladi. Bu funksiyalarni ishlatish uchun **math.h** (yangi ko'rinishda **cmath**) e'lon faylini **include** bilan asosiy dastur tanasiga kiritish kerak.

Quyida matematik funksiyalar kutubhonasining bazi bir a'zolarini beraylik. ***x*** va **y** o'zgaruvchilari **double** tipiga ega.

4.1–jadval. Matematik funksiyalar

Funksiya	Aniqlanishi	Misol
ceil(x)	x ni x dan katta yoki unga teng b-n eng kichik butun songacha yahlitlaydi	ceil(12.6) = 13.0 ceil(-2.4) = -2.0
cos(x)	x ning trigonometrik kosinusi (x radianda)	cos(0.0) = 1.0
exp(x)	e ning x chi darajasi (eskponetsial f-ya)	exp(1.0) = 2.71828 exp(2.0) = 7.38906
fabs(x)	x ning absolut qiymati	x>0 => abs(x) = x x=0 => abs(x) = 0.0 x<0 => abs(x) = -x
floor(x)	x ni x dan kichik bo'lgan eng katta butun songacha yahlitlaydi	floor(4.8) = 4.0 floor(-15.9) = -16.0
fmod(x,y)	x/y ning qoldig'ini kasr son tipida beradi	fmod(7.3,1.7) = 0.5
log(x)	x ning natural lagorifmi (e asosiga ko'ra)	log(2.718282)= 1.0
log10(x)	x ning 10 asosiga ko'ra lagorifmi	log10(1000.0)=3.0
pow(x,y)	x ning y chi darajasini beradi	pow(3,4)= 81.0 pow(16,0.25) = 2
sin(x)	x ning trigonometrik sinusi (x radianda)	sin(0.0)= 0.0
sqrt(x)	x ning kvadrat ildizi	sqrt(625.0)= 25.0
tan(x)	x ning trigonometrik tangensi (x radianda)	tan(0.0) = 0

4.3 Algoritm kutubxonasi funksiyalari

Har bir funksiya – funksiyalar shabloni yoki funksiyalar shabloni to’plami yordamida ifodalanadi. SHunday qilib, funksiya har xil tipdagi qiymatlarga ega bo’lgan har xil konteynerlar bilan ishlay oladi. Barcha funksiyalarni argumentlari (begin, end) yarim oraliqlar bo’ladi.

O’zgartirmaydigan funksiyalar

1. Oraliqdagi elementlarni o’zgartirmaydigan funksiya.

for_earch() oraliqning xar bir elementi uchun operatsiyalarni bajaradi

find() qiymatni oraliqdagi birinchi kirishini topadi

find_if() oraliqda predikatga birinchi moslashuvini topadi

count() qiymatni ketma-ketlikka kirishini xisoblaydi

count_if() oraliqda predikatni bajarilishini xisoblaydi

min_element() oraliqdagi eng kichik qiymat

max_element() oraliqdagi eng katta qiymat

2. Oraliqdagi elementlarni boshqa oraliqga nusxasini olib o’tish funksiyalari.

copy() birinchi elementdan boshlab oraliqni nusxasini oladi

copy_bacwards() oxirgi elementdan boshlab oraliqni nusxasini oladi

replace_copy() ko’rsatilgan qiymatga egabo’lgan elementlarni almashtirib nusxasini oladi.

replace_copy_if() predikatni bajarish jarayonida elementlarni almashtirgan xolda oraliqni nusxasini oladi

remove_copy() ko’rsatilgan qiymatga ega bo’lgan elementlarni o’chirgan xolda oraliqni nusxasini oladi

remove_copy_if() predikatni bajarish jarayonida elementlarni o’chirgan xolda oraliqni nusxasini oladi

unique_copy() teng bo’lgan qoshni elementlarni o’chirgan xolda oraliqni nusxasini oladi

rotate_copy() nusxasini olish jarayonida elementlarni sikl bo’yicha suradi

3. ikkita oraliqni solishtirish funksiyalari

search() oraliqni birinchi kiritilishini topadi

find_end() oraliqni oxirgi kiritilishini topadi

equal ikkita oraliqni tengligini tekshiradi

mismatch ikkita oraliqdagi farqlanadigan birinchi elementni qaytaradi

lexicographical_compare() ikkita oraliqni leksikografik solishtirilishi

4. Saralangan oraliqda topish funksiyalar

lower_bound() saralangan oraliqda qiymatni birinchi kirishini topadi

upper_bound() ko'rsatilgan qiymatdan katta bo'lgan birinchi elementni topadi

binary_search() saralangan oraliqda ko'rsatilgan element mavjudligini aniqlaydi

5. ikkita saralagan oraliqni solishtirish funksiyai

includes() bitta oraliqni ikkinchi oraliqga tegishliligini (kirishini) tekshirish.

6. ikkita saralangan oraliq ustidagi funksiyalar

set_union() oraliqlarni birlashtirish

set_intersection() oraliqlarni o'zaro kesishi

set_difference() oraliqlarni ayirmasi

set_symmetric_difference() oratiqlarni simmetrik ayirmasi

merge() ikkita oraliqni birlashtirish

O'zgartiruvchi funksiyalar

1. Oraliqda elementlarni almashtirish funksiyalari

fill() ko’rsatilgan qiymatdagi barcha elementlarni almashtiradi

replace() ko’rsatilgan qiymatli elementlarni almashtiradi

replace_if() predikat bajarilganda elementlarni almashtiradi

2. Oraliqda elementlarni o’chirish funksiyalari

remove() ko’rsatilgan qiymatdagi barcha elementlarni o’chiradi

remove_if() predikat bajarilganda elementlarni o’chiradi

unique() teng bo’lgan qo’shni elementlarni o’chiradi

3. Oraliqda elementlarni joyini almashtirish funksiyalari

reverse() elementlarni ketma-ketlik tartibini teskarisiga almashtiradi

rotate() **s**ikl bo’yicha elementlarni siljitadi

partition() elementlar tartibini o’zgartiradi, bunda kriteriyaga javob beradigan elementlar oldida bo’ladi

partition_stable() partition() ga o’xshash, lekin kriteriyaga javob beradigan va javob bermaydigan elementlarni ketma-ketlik tartibini saqlaydi

next_permutation() leksikografik tartibdagi keyingi almashuv

prev_permutation() l eksikografik tartibdagi oldingi almashuv

4. Oraliqdagi elementlarni saralash funksiyalari.

sort() oraliqni saralaydi

partial_sort() oraliqning qismini saralaydi

stable_sort() teng elementlarni ketma-ketlik tartibini saqlagan xolda oraliqni saralaydi

5. ikkita oraliqlar uchun o’zgarituvchi funksiyalar

transform() elementlarni modifikatsiyalaydi (va nusxasini oladi), xamda ikkita oraliqdagi elementlarni birlashtiradi

swap_ranges ikkita oraliqni joyini almashtiradi

6. Interval uchun sonli funksiyalar.

accumulate() elementlarning barcha qiymatlarini birlashtiradi (yig'indini, ko'paytmani va x.k. xisoblaydi)

inner_product() ikkita oraliqdagi barcha elementlarni birlashtirish (skalyar ko'paytmasini va x.k.larni xisoblaydi)

adjacent_difference() xar bir elementni uni oldingi qiymati bilan birlashtirish (ayirmasi va x.k.larni xisoblaydi.)

partial_sum() xar bir elementni ularning oldingi qiymatlari bilan birlashtiradi (barcha xususiy yig'indilarni va x.k. xisoblaydi)

4.4 Massivlar tushunchasi. Massivlar bilan ishlash.

Bu qismda dasturdagi ma'lumot strukturalari bilan tanishishni boshlaymiz. Dasturda ikki asosiy tur ma'lumot strukturalari mavjuddir. Birinchisi statik, ikkinchisi dinamikdir. Statik deganimizda hotirada egallagan joyi o'zgarmas, dastur boshida beriladigan strukturalarni nazarda tutamiz. Dinamik ma'lumot tiplari dastur davomida o'z hajmini, egallagan hotirasini o'zgartirishi mumkin.

Agar struktura bir hil kattalikdagi tiplardan tuzilgan bo'lsa, uning nomi massiv (**array**) deyiladi. Massivlar dasturlashda eng ko'p qo'laniladigan ma'lumot tiplaridir. Massivlar hotirada ketma-ket joylashgan, bir tipdagi o'zgaruvchilar guruhidir.

Alohida bir o'zgaruvchini ko'rsatish uchun massiv nomi va kerakli o'zgaruvchi indeksini yozamiz. C/C++ dagi massivlardagi elementlar indeksi har doim noldan boshlanadi. Bizda **char** tipidagi **m** nomli massiv bor bo'lsin va uning 4 dona elementi mavjud bo'lsin. Sxemada bunday ko'rsataylik:

m[0] -> 4

m[1] -> -44

m[2] -> 109

m[3] -> 23

Ko’rib turganimizdek, elementga murojaat qilish uchun massiv nomi va [] qavslar ichida element indeksi yoziladi. Bu yerda birinchi element qiymati 4, ikkinchi element - 1 nomerli indeksda -44 qiymatlari bor ekan. Ohirgi element indeksi n-1 bo'ladi (n - massiv elementlari soni).

[] qavslar ichidagi indeks butun son yoki butun songa olib keluvchi ifoda bo'lmog'i lozim. **Masalan**:

```
int k = 4, l = 2;
m[ k-l ] = 77;                  // m[2] = 77
m[3] *= 2;                      // m[3] = 46
double d = m[0] * 6;            // d = 24
cout << m[1];                   // Ekranda: -44
```

Massivlarni ishlatish uchun ularni e’lon qilish va kerak bo'lsa massiv elementlarini initsalizatsiya qilish kerak. Massiv e’lon qilinganda kompilyator elementlar soniga teng hajmda hotira ajratadi. Masalan yuqorida qo’llanilgan **char** tipidagi **m** massivini e’lon qilaylik.

```
char m[4];
```

Bu yerdagi 4 soni massivdagi elementlar miqdorini bildiradi. Bir necha massivni e'londa bersak ham bo'ladi:

```
int m1[4], m2[99], k, l = 0;
```

Massiv elementlari dastur davomida initsalizatsiya qilishimiz mumkin, yoki boshlang’ich qiymatlarni e’lon vaqtida, {} qavslar ichida ham bersak bo’ladi. {} qavslardagagi qiymatlar massiv initsalizaytsiya ro’yhati deyiladi.

```
int n[5] = {3, 5, -33, 5, 90};
```

Yuqorida birinchi elementning qiymati 3, ikkinchiniki 5 ... ohirgi beshinchi element qiymati esa 90 bo’ldi.

Misol:

```
double array[10] = {0.0, 0.4, 3.55};
```

Bu yerdagi massiv tipi **double** bo'ldi. Ushbu massiv 10 ta elementdan iboratdir.

{} qavslar ichida esa faqat boshlangich uchta element qiymatlari berildi. Bunday holda, qolgan elementlar avtomatik tarzda nolga tenglashtiriladi. Bu yerda aytib o'tishimiz kerakki, {} qavslar ichida berilgan boshlangish qiymatlar soni massivdagi elementlar sonidan katta bo'lsa, sintaksis hatosi vujudga keladi.

Masalan:

```
char k[3] = {3, 4, 6, -66, 34, 90};                    // Hato!
```

Uch elementdan iborat massivga 6 dona boshlangich qiymat berilyapti, bu hatodir. Boshqa misolni ko'rib chiqaylik:

```
int w[] = {3, 7, 90, 78};
```

w nomli massiv e'lon qilindi, lekin [] qavslar ichida massivdagi elementlar soni berilmadi. Bunday holda necha elementga joy ajratishni kompilyator {} qavslar ichidagi boshlangich qiymatlar miqdoriga qarab biladi. Demak, yuqoridagi misolda w massivimiz 4 dona elementdan iborat bo'ladi.

E'lon davridagi massiv initsalizatsiya ro'yhati dastur ijrosi vaqtidagi initsalizatsiyadan ko'ra tezroq ishlaydigan mashina kodini vujudga keltiradi. Bir misol keltiraylik.

```
#include <iostream.h>
#include <iomanip.h>
const int massiv = 8;                          // massiv kattaligi uchun konstanta
int k[massiv];
char c[massiv] = {5,7,8,9,3,44,-33,0};         // massiv initsalizatsiya ro'yhati
int main(){
    for (int i = 0; i < massiv; i++) {
        k[i] = i + 1;                          // dastur ichida inisalizatsiya
    }
    for (int j = 0; j < massiv; j++) {
        cout << k[j] << setw(4) << c[j] << endl;
    }
return (0);}
```

V OBYEKTGA MO'LJALLANGAN DASTURLASH ASOSLARI

5.1 Obyektga mo'ljallangan dasturlash asoslari va asosiy tamoyillari

Obyektga mo‘ljallangan yondoshuv dasturiy tizimlarni dasturlash tiliga bog‘liq bo‘lmagan holda yaratishda modellardan sistematik foydalanishga asoslangan. Har bir model uning o‘zi aks ettirayotgan predmetning hamma xususiyatlarini ifodalay olmaydi, u faqat ba’zi juda muhim belgilarini ifodalaydi. Demak model o‘zi aks ettirayotgan predmetga nisbatan ancha sodda bo‘ladi. Bizga shu narsa muhimki model endi formal konstruksiya hisoblanadi: modellarning formalligi esa ular orasidagi formal bog‘lanishlarni aniqlashni va ular orasida formal operatsiyalar bajarishni ta’minlaydi. Bu ish modellarni ishlab chiqishni va o‘rganishni hamda kompyuterda realizatsiya qilishni osonlashtiradi. Xususan esa, modellarning formal xarakteri yaratilayotgan dasturning formal modelini olishni ta’minlaydi.

Shunday qilib, obyektga mo‘ljallangan yondoshuv quyidagi murakkab muammolarni hal qilishda ishlatiladi:

- dasturiy ta’minotning murakkabligini pasaytiradi;
- dasturiy ta’minotning ishonchliligini oshiradi;
- dasturiy ta’minotning alohida komponentalarni modifikatsiya qilishni osonlashtiradi;
- alohida komponentalardan qayta foydalanishni ta’minlaydi.

Obyektga mo‘ljallangan yondoshuvning sistemali qo‘llanilishi yaxshi tuzilmalangan, ishlatishda barqaror bo‘lgan, oson modifikatsiya qilinuvchi dasturiy sistemalarni yaratish imkoniyatini beradi. Aynan ana shu imkoniyatlar dasturchilarni obyektga mo‘ljallangan yondoshuvdan foydalanishga juda ham qiziqtirmoqda. Obyektga mo‘ljallangan yondoshuvli dasturlash hozirgi vaqtda eng tez rivojlanayotgan dastur yozish texnologiyasi hisoblanadi. Obyektga mo‘ljallangan yondoshuv ikkita kismga bo‘linadi:

- Obyektga mo‘ljallangan dasturlar yaratish;

- Obyektga mo‘ljallangan dasturlash tillari.

Obyektga mo‘ljallangan dasturlash tillari oxirgi vaqtlarda juda ommaviylashgan tillarga kiradi. Bunday tillarga quyidagilar kiradi: C++, Visual C++, Visual Basic, Java, PHP va boshqalar. C++ eng ko‘p tarqalgan obyektga mo‘ljallangan dasturlash tillariga kiradi.

Obyektga mo‘ljallangan dasturlashda dastur obyektlarni va ularning xususiyatlarini (atributlarini) va ularni birlashtiruvchi sinflarni tavsiflashga olib kelinadi. SHu jumladan obyektlar ustida operatsiyalar (usullar) aniqlashga olib kelinadi.

Atributlar va usullarni tadqiq qilish asosida bazaviy sinflar va ularning hosilalarini yaratish imkoniyati to‘g‘iladi.

Obyektga mo‘ljallangan dasturlashning yana bir nazariy jihatdan juda muhim va zarur xususiyatlaridan biri hodisalarni ishlash mexanizmi hisoblanadi, ular yordamida obyektlar atributlari qiymatlari o‘zgartiriladi. Obyektga mo‘ljallangan dasturlashda avval yaratilgan obyektlar bibliotekasi va usullaridan foydalanish hisobiga obyektga yo‘naltirilgan dasturlashda ancha mehnat tejaladi.

Obyektlar, sinflar va usullar polimorfizm bo‘lishlari mumkin, bu esa dasturiy vositaning qulay foydalanishligi va universalligini ta’minlaydi.

5.2 Sinf tushunchasi

Sinf. Har bir sinf sinflar tabaqalanishida (ierarxiyasida) ma’lum o‘rinni egallaydi. Masalan, barcha soatlar vaqtni o‘lchash asboblari sinfiga (tabaqalanishda ancha yuqori turgan) mansub, soatlar sinfining o‘zi esa xuddi shu mavzudagi ko‘plab hosila variatsiyalarini o‘z ichiga oladi. SHunday qilib, har qanday sinf obyektlarning biron-bir kategoriyasini aniqlaydi, har qanday obyekt esa biron-bir sinf ekzemplyari (nusxasi)dir.

Sinf jismoniy mohiyatga ega emas, tuzilmaning e’lon qilinishi uning eng yaqin analogiyasidir. Sinf obyektni yaratish uchun qo‘llangandagina, xotira ajralib chiqadi. Bu jarayon ham sinf nusxasini yaratish deb ataladi.

C++tilining har qanday obyekti bir hil atributlarga, shuningdek ushbu sinfning boshqa obyektlari bilan funksionallikka ega. O‘z sinflarini yaratish hamda ushbu sinflar obyektlarining xulq-atvori uchun to‘liq mas’uliyat dasturchi zimmasiga yuklanadi. Biron-bir muhitda ishlar ekan, dasturchi standart sinflarning kattagina kutubxonasi (masalan, C++ Builder Visual Komponentlar Kutubxonasi)ga kirish huquqiga ega bo‘ladi.

Abstraksiya – bu identifikatorlardan farqli bo‘lgan istalgan dasturlash tili ifodasi hisoblanadi.

Garchi obyektga mo‘ljallanganliklar inkapsulyasiyalashdan foydalanishga yordam bersa-da, biroq ular inkapsulyasiyalashni kafolatlamaydi. Tobe va ishonchsiz kodni yaratib qo‘yish oson. Samarali inkapsulyasiyalash – sinchkovlik bilan ishlab chiqish xamda abstraksiya va tajribadan foydalanish natijasidir. Inkapsulyasiyalashdan samarali foydalanish uchun dasturni ishlab chiqishda avval abstraksiyadan va uning bilan bog‘liq konsepsiyalardan foydalanishni o‘rganib olish lozim.

Abstraksiya murakkab masalani soddalashtirish jarayonidir. Muayyan masalani echishga kirishar ekansiz, siz barcha detallarni hisobga olishga o‘rinmaysiz, balki echimni osonlashtiradiganlarini tanlab olasiz.

Aytaylik, siz yo‘l harakati modelini tuzishingiz kerak. SHunisi ayonki, bu o‘rinda siz svetoforlar, mashinalar, shosselar, bir tomonlama va ikki tomonlama ko‘chalar, ob-havo sharoitlari va h.k. sinflarini yaratasiz. Ushbu elementlarning har biri transport harakatiga ta’sir ko‘rsatadi. Biroq bu o‘rinda hasharotlar va qushlar xam yo‘lda paydo bo‘lishi mumkin bo‘lsa-da, siz ularning modelini yaratmaysiz. Inchunin, siz mashinalar markalarini ham ajratib ko‘rsatmaysiz. Siz haqiqiy olamni soddalashtirasiz hamda uning faqat asosiy elementlaridan foydalanasiz. Mashina - modelning muhim detali, biroq bu Kadillakmi yoki boshqa biron markadagi mashinami, yo‘l harakati modeli uchun bu detallar ortiqcha.

Abstraksiyaning ikkita afzal jihati bor. Birinchidan, u masala echimini soddalashtiradi. Muhimi yana shundaki, abstraksiya tufayli dasturiy ta’minot komponentlaridan takroran foydalanish mumkin. Takroran qo‘llanadigan komponentlarni yaratishda ular odatda g‘oyat ixtisoslashadi. Ya’ni komponentlar biron-bir ma’lum masala echimiga mo‘ljallangani, yana ular keraksiz o‘zaro bog‘liqlikda bo‘lgani sababli. dastur fragmentining boshqa biron o‘rinda takroran qo‘llanishi qiyinlashadi. Imkoni boricha bir qator masalalarni echishga qaratilgan obyektlarni yaratishga harakat qiling. Abstraksiya bitta masala echimidan ushbu sohadagi boshqa masalalarni ham echishda foydalanish imkonini beradi.

Sinflarni yozishda biz funksiyalarni yozishdagi tartib qoidalarga rioya qilamiz. Sinfning birinchi qatoriga kalit so‘z **class** va sinf nomi, so‘ngra yangi qatordan figurali qavslar ochiladi va uning ichiga sinf usullari va atributlari yoziladi.

Sinf quyidagi seksiyalarga ega bo‘lishi mumkin:

1. **private** (private, ichki).
2. **protected** (protected, himoyalangan qism).
3. **public** (public, umumiy).

Endi bazaviy sinfning umumiy yozilish sintaksisini quyidagicha yozish mumkin:

class className {

private:

<privat berilmalar a’zolari> <private konstruktorlar> <privat usullar>

protected:

<Himoyalangan a’zo berilmalar> <Himoyalangan konstruktorlar>

<Himoyalangan usullar>

public:

<Umumiy dostupli hususiyatlar> <Umumiy dostupli a’zo berilmalar> <Umumiy huquqli konstruktorlar va destruktorlar> <Umumiy huquqli usullar>

}

C++ ning bazaviy sinflarining seksiyalariga quyidagicha huquqlar aniqlangan:

1. **Private** seksiyasi – shu sinfning faqat usullariga dostupni aniqlaydi. Hosilaviy sinflar uchun privat usullarga dostup berilmaydi.

2. Himoyalangan **protected** nomlari faqat shu sinf usullariga va shu sinf hosila sinfi usullariga dostup beradi.

3. Umumiy huquqli **public** nomlari hamma turdagi sinflarning usullariga dostup beradi.

Sinflarni aniqlashda seksiyalardan foydalanishning asosiy qoidalari:

1. Seksiyalar istalgan tartibda e'lon qilinishlari mumkin, hatto qayta tavsiflashlar ham uchrashi mumkin.

2. Agar seksiya nomlangan bo'lmasa, u holda kompilyator sinfda oxirgi aniqlangan nomlarni **private** berilma deb qabul qiladi.

3. Agar biz a'zo berilmalarga dostupni cheklamokchi bo'lsak ularni umum dostupli seksiyaga joylashtirmasligimiz lozim.

5.3 Abstraksiya

Abstraksiya – bu identifikatorlardan farqli bo'lgan istalgan dasturlash tili ifodasi hisoblanadi.

Obyektga mo'ljallangan dasturlashda har bir obyekt prinsipial dinamik mohiyatga ega, ya'ni u vaqtga bog'lik holda va unga nisbatan tashqi faktorlar ta'sirida o'zgaradi. Boshqacha aytganda obyekt ma'lum bir darajada o'zini tutishiga ega. Obyektga mo'ljallangan dasturlashda abstraksiya OMD ning modeli hisoblanadi. Sinf umumiy xususiyatlar va hulk-atvorga ega bo'lgan obyektlarni birlashtiradi. Bitta sinfga mansub obyektlar bir xil xususiyatlarga ega bo'lib, bir xil xatti-xarakat namoyon etadi.

Sinflar shablon (qolip)ga o'xshaydi: ular obyektlarning ekzemplyarlarini tayyorlash uchun qo'llanadi. Belgilar - sinfning tashqaridan ko'rinib turgan xususiyatlari. Obyekt ichki o'zgaruvchiga bevosita kirishni takdim etganda yoki

usul yordamida qiymatni kaytargandagina, oʻz belgilarini namoyon kilishi mumkin.

Hulq-atvor - xabarga yoki holatning oʻzgarishiga javoban obyekt tomonidan bajariladigan xatti-xarakatlar. U obyekt nima qilayotganini bildiradi.

Bir obyekt ikkinchi obyekt ustida xatti-xarakatlar bajarib, uning xulk-atvoriga ta’sir koʻrsatishi mumkin. «Xatti-xarakat» atamasi oʻrniga «usulni chakirish», «funksiyasini chakirish» yoki «xabarni oʻzatish» atamalari koʻllanadi. Muximi bu atamalarning qaysi biri qullanayotganida emas, albatta, muximi bu xatti-xarakatlar obyekt hulk-atvorini namoyon qilishga da’vat etishidadir.

Obyektlar oʻrtasida aloqa obyektga moʻljallangan dasturlashning muhim tarkibiy qismidir. Obyektlar oʻzaro aloqasining ikkita asosiy usuli mavjuddir.

Birinchi usul: obyektlar biri ikkinchisidan mustaqil ravishda mavjud boʻladi. Agar alohida obyektlarga oʻzaro aloqa kerak boʻlib qolsa, ular bir-birlariga xabar joʻnatadi.

Obyektlar bir-birlari bilan xabarlar yordamida aloqa qiladi. Xabar olgan obyekt ma’lum xatti-xarakatlarni bajaradi.

Xabar uzatish bu obyekt xolatini oʻzgartirish maqsadida uslubni chaqirib olish yoki xulk-atvor modellaridan birini koʻllashning oʻzginasidir.

Ikkinchi usul: obyekt tarkibida boshka obyektlar boʻlishi mumkin. Xuddi OMDda boʻlganidek, dastur obyektlardan tashkil topganidek, obyektlar ham, oʻz navbatida, agregatsiya yordamida boshqa obyektlardan jamlanishi mumkin. Ushbu obyektlarning har bittasida uslub va belgilarga ega boʻlgan interfeys mavjud boʻladi.

Xabar - obyektga moʻljallangan yondoshuvning muhim tushinchasi. Xabarlar mexanizmi tufayli obyektlar oʻz mustakilligini saqlab qolishi mumkin. Boshqa biron obyektga xabar joʻnatayotgan obyekt uchun xabar olgan obyekt talabdagi xatti-xarakatni qanday bajarishi unchalik muhim emas. Unga xatti-xarakat bajarilganligining oʻzi muhimdir.

5.4 Vorislik

Vorislik. Vorislik mavjud bo‘lgan sinfning ta’rifi asosida yangi sinfni yaratish imkonini beradi. Yangi sinf boshqasi asosida yaratilgach, uning ta’rifi avtomatik tarzda mavjud sinfning barcha hususiyatlari, hulq-atvori va joriy qilinishiga vorislik qiladi. Avval mavjud bo‘lgan sinf interfeysining barcha metodlari va xususiyatlari avtomatik tarzda voris interfeysida paydo bo‘ladi. Vorislik voris sinfida biron-bir jixatdan to‘g‘ri kelmagan xulq-atvorni avvaldan ko‘ra bilish imkonini beradi. Bunday foydali xususiyat dasturiy ta’minotni talablarning o‘zgarishiga moslashtirish imkonini beradi. Agar o‘zgartirishlar kiritishga ehtiyoj tug‘ilsa, bu holda eski sinf funksiyalariga vorislik qiluvchi yangi sinf yozib qo‘ya qolinadi. Keyin o‘zgartirilishi lozim bo‘lgan funksiyalarga qaytadan ta’rif beriladi hamda yangi funksiyalar qo‘shiladi. Bunday o‘rniga o‘rin qo‘yishning mazmuni shundan iboratki, u dastlabki sinf ta’rifini o‘zgartirmay turib, obyekt ishini o‘zgartirish imkonini beradi. Axir bu holda qayta test sinovlaridan puxta o‘tkazilgan asosiy sinflarga tegmasa ham bo‘ladi. Agar siz ko‘p martalab qo‘llash yoki boshqa biron maqsadlarga ko‘ra vorislikni qo‘llashga ahd qilsangiz, avval har gal qarang - vorislik-sinf bilan vorislikni berayotgan sinfning turlari o‘zaro mos keladimi. Vorislikda turlarining mos kelishi ko‘pincha «Is-a» testi deb ataladi. Ikkita sinf bir hil turga ega bo‘lgandagina, o‘zaro «Is-a» munosabatida turibdi deb hisoblanadi. Birinchi sinf o‘zida ikkinchi sinfning ekzemplyariga ega bo‘lgandagina ikkita sinf o‘zaro «Xas-a» munosabatida turibdi deb hisoblanadi.

Boshqa sinfga vorislik bo‘layotgan sinf voris berayotgan sinf bilan shunday munosabatda bo‘lmog‘i lozimki, bunda natijaviy munosabatlar o‘z ma’nosiga ega bo‘lmog‘i, ya’ni vorislik tabaqalanishiga amal qilinishi kerak.

Vorislik yordamida qurilgan sinf metodlar va xususiyatlarning uchta ko‘rinishiga ega bo‘lishi mumkin:

- o‘rniga o‘rin qo‘yish (almashtirish): yangi sinf ajdodlarining metodi yoki xususiyatini shunchaki o‘zlashtirib olmaydi, balki unga yangi ta’rif ham beradi;
- yangi: yangi sinf butunlay yangi metodlar yoki xususiyatlarni qo‘shadi;
- Rekursiv: yangi sinf o‘z ajdodlari metodlari yoki xususiyatlarini to‘g‘ridan-to‘g‘ri olib qo‘ya qoladi.

Obyektga mo‘ljallangantillarning ko‘pchiligi ta’rifni ma’lumot o‘zatilgan obyektdan qidiradilar. Agar u erdan ta’rif topishning iloji bo‘lmasa, biron ta’rif topilmaguncha, qidiruv tabaqalar bo‘yicha yuqoriga ko‘tarilaveradi. Ma’lumotni boshqarish aynan shunday amalga oshiriladi hamda aynan shu tufayli o‘ringa o‘rin qo‘yish jarayoni ish ko‘rsatadi.

Voris sinflar himoyalangan kirish darajasiga ega bo‘lgan metodlar va xususiyatlarga kirish huquqini olishlari mumkin. Bazaviy sinfda faqat avlodlar foydalanishi mumkinligi aniq bo‘lgan metodlargagina himoyalangan kirish darajasini bering. Boshqa hollarda xususiy yoki ommaviy kirish darajasidan foydalanish lozim. Bunday yondoshuv barcha sinflarga, shu jumladan, tarmoq sinflarga ham kirish xuquqi berilganidan ko‘ra, mustahkamroq konstruksiyani yaratish imkonini beradi.

Vorislik uch asosiy hollarda qo‘llanadi:

- ko‘p martalab foydalanishda;
- ajralib turish uchun;
- turlarni almashtirish uchun.

Vorislikning ayrim turlaridan foydalanish boshqalaridan ko‘ra afzalroq hisoblanadi. Vorislik yangi sinfga eski sinfning amalda qo‘llanishidan ko‘p martalab foydalanish imkonini beradi. Kodni qirqib tashlash yoki kiritish o‘rniga, vorislik kodga avtomatik tarzda kirishni ta’minlaydi, ya’ni kodga kirishda, u yangi sinfning bir qismidek olib qaraladi. Ko‘p martalab qo‘llash uchun vorislikdan foydalanar ekansiz, siz voris qilib olingan realizatsiya (joriy

qilinish) bilan bog‘liq bo‘lasiz. Vorislikning bu turini ehtiyotkorlik bilan qo‘llash lozim. Yaxshisi bu o‘rinda «Xas-a» munosabatidan foydalanish kerak.

Farqlash uchun vorislik faqat avlod-sinf va ajdod-sinf o‘rtasidagi farqlarni dasturlash imkonini beradi. Farqlarni dasturlash g‘oyat qudratli vositadir. Kodlash hajmining kichikligi va kodning oson boshqarilishi loyiha ishlanmasini osonlashtiradi. Bu holda kod satrlarini kamroq yozishga to‘g‘ri keladiki, bu qo‘shiladigan xatolar miqdorini ham kamaytiradi.

Vorislik obyektga mo‘ljallangan dasturlashning muhim xususiyatlariga kiradi. Biz V sinfi A sinfini meroslashini ko‘rsatish uchun (V sinfi A sinfidan tashkil etilgan) V sinfini aniqlashda sinf nomidan keyin ikki nuqta quyiladi va so‘ngra V vorislanayotgan sinflar keltiriladi:

```
class A { public:A();
        A();
        MethodA();
};
Class B: public A{ public: B();
        …
};
```

5.5 Polimorfizm

Polimorfizm. Agar inkapsulyasiyalash va vorislikni obyektga mo’ljallangan yondashuvning foydali vositalari sifatida olib qarash mumkin bo‘lsa, polimorfizm - eng universal va radikal vositadir. Polimorfizm inkapsulyasiyalash va vorislik bilan chambarchas bog‘liq, boz ustiga, polimorfizmsiz obyektga mo’ljallangan yondashuv samarali bo‘lolmaydi. Polimorfizm - obyektga mo’ljallangan yondashuv paradigmasida markaziy tushunchadir. Polimorfizmni egallamay turib, obyektga mo’ljallangan yondashuvdan samarali foydalanish mumkin emas.

Polimorfizm shunday holatki, bunda qandaydir bitta narsa ko‘p shakllarga ega bo‘ladi. Dasturlash tilida «ko‘p shakllar» deyilganda, bitta nom avtomatik

mexanizm tomonidan tanlab olingan turli kodlarning nomidan ish koʻrishi tushuniladi. SHunday qilib, polimorfizm yordamida bitta nom turli xulq-atvorni bildirishi mumkin.

Vorislik polimorfizmning ayrim turlaridan foydalanish uchun zarurdir. Aynan oʻrindoshlik imkoniyati mavjud boʻlgani uchun, polimorfizmdan foydalanish mumkin boʻladi. Polimorfizm yordamida tizimga toʻgʻri kelgan paytda qoʻshimcha funksiyalarni qoʻshish mumkin. Dasturni yozish paytida hatto taxmin qilinmagan funksionallik bilan yangi sinflarni qoʻshish mumkin, buning ustiga bularning hammasini dastlabki dasturni oʻzgartirmay turib ham amalga oshirish mumkin. YAngi talablarga osongina moslasha oladigan dasturiy vosita deganda, mana shular tushuniladi.

Polimorfizmning uchta asosiy turi mavjud:

- qoʻshilish polimorfizmi;
- parametrik polimorfizm;
- ortiqcha yuklanish;

Qoʻshilish polimorfizmini ba'zida sof polimorfizm deb ham ataydilar. Qoʻshilish polimorfizmi shuning bilan qiziqarliki, uning tufayli tarmoq sinf nushalari oʻzini turlicha tutishi mumkin. Qoʻshilish polimorfizmidan foydalanib, yangi tarmoq sinflarni kiritgan xolda, tizimning xulq-atvorini oʻzgartirish mumkin. Uning bosh afzalligi shundaki, dastlabki dasturni oʻzgartirmay turib, yangi xulq-atvorni yaratish mumkin.

Aynan polimorfizm tufayli joriy qilishdan takroran foydalanishni vorislik bilan aynanlashtirish kerak emas. Buning oʻrniga vorislikdan avvalam bor oʻzaro almashinish munosabatlari yordamida polimorf xulq-atvorga erishish uchun foydalanish lozim. Agar oʻzaro almashinish munosbatlari toʻgʻri belgilansa, buning ortidan albatta takroran qoʻllash chiqib keladi. Qoʻshilish polimorfizmidan foydalanib, bazaviy sinfdan, har qanday avloddan, shuningdek bazaviy sinf qoʻllaydigan metodlardan takroran foydalanish mumkin.

Parametrik polimorfizmdan foydalanib, turdosh metodlar va turdosh (universal) turlar yaratish mumkin. Turdosh metodlar va turlar dalillarning ko‘plab turlari bilan ishlay oladigan dasturni yozish imkonini beradi. Agar qo‘shilish polimorfizmidan foydalanish obyektni idrok etishga ta’sir ko‘rsatsa, parametrik polimorfizmdan foydalanish qo‘llanayotgan metodlarga ta’sir ko‘rsatadi. Parametrik polimorfizm yordamida, parametr turini bajarilish vaqtigacha e’lon qilmay turib, turdosh metodlar yaratish mumkin. Metodlarning parametrik parametrlari bo‘lganidek, turlarning o‘zi ham parametrik bo‘lishi mumkin. Biroq polimorfizmning bunday turi barcha tillarda xam uchrayvermaydi (C++da mavjud).

Ortiqcha yuklanish yordamida bitta nom turlicha metodlarni bildirishi mumkin. Bunda metodlar faqat miqdorlari va parametr turlari bilan farqlanadi. Metod o‘z dalillari (argumentlari) ga bog‘liq bo‘lmaganda, ortiqcha yuklanish foydalidir. Metod o‘ziga xos parametrlar turlari bilan cheklanmaydi, balki har xil turdagi parametrlarga nisbatan ham qo‘llanadi. Masalan max metodini ko‘rib chiqaylik. Maksimal - turdosh tushuncha bo‘lib, u ikkita muayyan parametrlarni qabul qilib, ularning qaysi biri kattaroq ekanini ma’lum qiladi. Ta’rif butun sonlar yoki suzuvchi nuqtali sonlar qiyoslanishiga qarab o‘zgarmaydi.

Polimorfizmdan samarali foydalanish sari qo‘yilgan birinchi qadam bu inkapsulyasiyalash va vorislikdan samarali foydalanishdir. Inkapsulyasiyalashsiz dastur osongina sinflarning joriy qilinishiga bog‘liq bo‘lib qolishi mumkin. Agar dastur sinflarning joriy qilinish aspektrlaridan biriga bog‘liq bo‘lib qolsa, tarmoq sinfda bu joriyni to‘g‘rilash mumkin bo‘lmaydi.

Vorislik - qo‘shilish polimorfizmining muhim tarkibiy qismi. Hamma vaqt bazaviy sinfga imkon darajada yaqinlashtirilgan darajada dasturlashga o‘ringan holda, o‘rinbosarlik munosabatlarini o‘rnatishga harakat qilish kerak. Bunday usul dasturda ishlov berilayotgan obyektlar turlari miqdorini oshiradi.

Puxta o‘ylab ishlab chiqilgan tabaqalanish o‘rinbosarlik munosabatlarini o‘rnatishga yordam beradi. Umumiy qismlarni abstrakt sinflarga olib chiqish

kerak hamda obyektlarni shunday dasturlash kerakki, bunda obyektlarning ixtisoslashtirilgan nushalari emas, balki ularning o‘zlari dasturlashtirilsin. Bu keyinchalik har qanday voris sinfni dasturda qo‘llash imkonini beradi.

Biroq ko‘p o‘rinlarda tajribasiz loyihachilar polimorfizmni ko‘chaytirish maqsadida hulq-atvorni juda baland tabaqaviy darajaga olib chiqishga urinadilar. Bu holda har qanday avlod ham bu xulq-atvorni ushlab tura oladi. SHuni esdan chiqarmaslik kerakki, avlodlar o‘z ajdodlarining funksiyalarini chiqarib tashlay olmaydilar. Dasturni yanada polimorfizm qilish maqsadida puxta rejalashtirilgan vorislik tabaqalarini bo‘zish yaramaydi.

Hamma narsaning xisob-kitobi bor. Xaqiqiy polimorfizmning kamchiligi shundaki, u unumdorlikni pasaytiradi. Polimorfizmdan foydalanganda dasturni bajarish paytida tekshiruvlar o‘tkazish talab qilinadi. Bu tekshiruvlar turlari statik ravishda berilgan qiymatlarga ishlov berishga qaraganda ko‘proq vaqtni talab qiladi.

ADABIYOTLAR

1. Стенли Липпман. Язык программирование C++. Базовой курс. Вильямс - М.: 2014.
2. Сидхарма Рао. Освой самостоятельно C++ за 21 день. Вильямс - М.: 2013.
3. Никита Культин. Microsoft Visual C++ в задачах и примерах. БХВ-Петербург - Петербург.:2010.
4. Б. Страуструп. Язык программирования C++. Специальное издание.- М.:ООО «Бином-Пресс», 2006.-1104 с.
5. Павловская Т.А. C++. Программирование на языке высокого уровня – СПб.: Питер. 2005.- 461 с.
6. Подбельский В.В. Язык C++.- М.; Финансы и статистика- 2003 562с.
7. Павловская Т.С. Щупак Ю.С. C/C++. Структурное программирование. Практикум.-СПб.: Питер,2002-240с
8. Павловская Т.С. Щупак Ю.С. C++. Объектно- ориентированное программ-мирование. Практикум.-СПб.: Питер,2005-265с
9. Глушаков С.В., Коваль А.В., Смирнов С.В. Язык программирования C++: Учебный курс.- Харьков: Фолио; М.: ООО «Издательство АСТ», 2001.-500с.
10. Ш.Ф. Мадрахимов, С. М. Гайназаров C++ тилида программалаш асослари. Т. 2009.
11. www.Intuit.ru. Интернет-Университет информационных технологий. Москва.
12. Пильшиков В.Н. Упражнения по языку Паскаль-М.: МГУ, 1986.
13. Абрамов С.А.,Гнезделова Капустина Е.Н.и др. Задачи по программ-мированию. - М.: Наука, 1988.
14. Вирт Н. Алгоритмы + структуры данных = программа.-М.:Мир,1985.-405с.
15. Информатика. Базовой курс. Учебник для Вузов., Санк-Петербург, 2001. под редакцией С.В.Симоновича.
16. Informatika va programmalsh.O’quv qo’llanma. Mualliflar: A.A.Xaldjigitov, Sh.F.Madraximov, U.E.Adamboev, O’zMU, 2005 yil, 145 bet.
17. O.Shukurov, F.Qorayev, E.Eshboyev, B.Shovaliyev “Programmalashdan masalalar to’plami”. Toshkent 2008,160 bet.