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

Samarqand davlat universiteti

Nazarov F.M., Nurmamatov M.

DASTURLASH ASOSLARI

Ma’ruzalar matni

Samarqand-2018

Mavzu1:C++ dastrulashning tili haqida umumiy tushunca. C++ amallari, operatorlari.

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 tili ga asoslangan tarzda tuzildi. C++ juda ko ’ p qo’ shimchalarni o ’ z ichiga olgan, lekin eng asosiysi u ob yektlar 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 struktu rali dasturlash kabi, dasturlarni hayotdagi jismlarni modellashtiruvchi ob yektlat 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 Xer ox 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 ob yektlarning juda boy kutubhonasiga ega. Yani C++ dasturlash tili da dasturlashni o ’ rganish ikki qismga bo ’ linadi. Birinchisi bu C++ tili ni o’ zini o’ rganish, ikkinchisi esa C++ ning standart kutubhonasidagi tayyor ob yekt va funksiyalarni qo ’ l lashni 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>,

7

#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

ko‘rsatishi mumkin. Buning uchun butun o‘zgarmas raqamlari oxiriga, probelsiz

l yoki L (1ong), u yoki U (unsigned) yoziladi. Zarur hollarda bitta o‘zgarmas

uchun bu belgilarning ikkitasini ham ishlatish mumkin: 451u, 012Ul, 0xA2L.

Haqiqiy o‘zgarmaslar: Haqiqiy o‘zgarmaslar – suzuvchi nuqtali son bo‘lib, u ikki xil formatda berilishi mumkin:

 O‘nlik fiksirlangan nuqtali formatda. Bu ko‘rinishda son nuqta orqali

ajratilgan butun va kasr qismlar ko‘rinishida bo’ladi. Sonning butun yoki kasr

qismi bo‘lmasligi mumkin, lekin nuqta albatta bo‘lishi kerak. Fiksirlangan

nuqtali o‘zgarmaslarga misollar: 24.56; 13.0; 66.; .87;

 eksponensial shaklda haqiqiy o‘zgarmas 6 qismdan iborat bo’ladi:

1) butun qismi (o‘nli butun son);

2) o‘nli kasr nuqta belgisi;

3) kasr qismi (o‘nlik ishorasiz o‘zgarmas);

4) eksponenta belgisi ‘e’ yoki ‘E’;

5) o‘n darajasi ko‘rsatkichi (musbat yoki manfiy ishorali o‘nli butun son);

6) qo‘shimcha belgisi (‘F’ yoki f , ‘L’ yoki ‘l’).

 Eksponensial shakldagi o‘zgarmas sonlarga misollar:

1E2; 5E+3; .25E4; 31.4E-1.

Belgi o‘zgarmaslar: Belgi o‘zgarmaslar qo‘shtirnoq (‘,’-apostroflar) ichiga

olingan alohida belgilardan tashkil topadi va u char kalit so‘zi bilan aniqlanadi.

Bitta belgi o‘zgarmas uchun xotirada bir bayt joy ajratiladi va unda butun son

ko‘rinishidagi belgining ASCII kodi joylashadi. Quyidagilar belgi

o‘zgarmaslarga
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
11
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

12

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> { <nom1> =<qiymat1>, <nom2> =

<qiymat2>,… <nom3>=<qiymat3> } ;

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

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

13

<nom> – butun qiymatli konstantalarning nomlari;

<qiymati> – 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,

14

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

Mavzu2: C++ ko’rsatkichlari va ilovalari. Massivlar.

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

70

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);}
```

Mavzu3: Funksiyalar. Standart funksiyalar.

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 uchbu satrni ko'rganida,standart kutubhonadan sin funksiyasini chaqiradi. Kirish qiymati sifatida m ni berdik. Javob, yani funksiyadan qaytgan qiymat k ga berildi.Funksiya agumentlari o'zgarmas sonlar (konstanta) 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 funksiya-lar kutubhonasining bazi bir a'zolarini beraylik. x va y o'zgaruvchilari double tipiga ega.

Funksiya	Aniqlanishi	Misol
ceil(x)	x ni x dan katta yoki unga teng b-n	ceil(12.6) = 13.0

	eng kichik butun songacha yahlitlaydi	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

FUNKSIYALARNING TUZILISHI

Funksiyalar dasturchi ishini juda yengillashtiradi. Funksiyalar yordamida programma modullashadi, qismlarga bo'limadi. Bu esa keyinchalik dasturni rivojlantirishni osonlashtiradi. 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 argument-laridir, 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 klaslarni (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 belgilana-digan joyga hech narsa yozilmaydi. Biz unday funksiyalarni keyinroq qo'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,
                      //      No1 bilan e'lonlari ham 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. Aslida () qavslar C++ da operatorlardir. Funksiya kirish parametrlarini har birini ayri-ayri yozish kerak, masalan yuqoridagi

```
float average(int a, int b, int c);
```

funksiyasini

```
float average(int a,b,c); // Hato!
```

deb yozishimiz hatodir.

Hali etib o'tganimizdek, funksiya kirish parametrlari ushbu funksiyaning lokal o'zgaruvchilaridir. Bu o'zgaruvchilarni funksiya tanasida boshqattan e'lon qilish sintaksis hatoga olib keladi. Bir dastur yozaylik.

```
//Funksiya bilan ishlash
# include <iostream.h>
int foo(int a, int b); //Funksiya prototipi,
                       //argumentlar ismi shart emas.
int main()
{
  for (int k = 1; k <6; k++){
    for (int l = 5; l>0; l--){
      cout << foo(k,l) << " ";        //Funksiya chaqirig'i.
    }//end for (l...)
    cout << endl;
  }//end for (k...)
return (0);
} //end main()
//foo() funksiyasining aniqlanishi
int foo(int c, int d)
{ //Funksiya tanasi
  return(c * d);
}
```

Ekranda:

```
5 4 3 2 1
10 8 6 4 2
15 12 9 6 3
20 16 12 8 4
25 20 15 10 5
```

Bizda ikki sikl ichida foo() funksiyamiz chaqirilmoqda. Funksiyaga k va l o'zgaruvchilarining nushalari uzatil-moqda. Nushalarning qiymati mos ravishda funksiyaning aniqlanishida berilgan c va d o'zgaruvchilarga berilmoqda. k va l ning nushalari deganimizda adashmadik, chunki ushbu o'zgaruvchilarining qiymatlari funksiya chaqirig'idan hech qanday ta'sir ko'rmaydi. C++ dagi funksiyalarning bir noqulay tarafi shundaki, funksiyadan faqat bitta qiymat qaytadi. Undan tashqari yuqorida ko'rganimizdek, funksiyaga berilgan o'zgaruvchilarning faqat nushalari bilan ish ko'rilarkan. Ularning qiymatini normal sharoitda funksiya ichida

o'zgartirish mumkin emas. Lekin bu muammolar ko'rsatkichlar yordamida osonlikcha hal etiladi. Funksiya chaqiriqlarida avtomatik ma'lumot tipining konversiyasi bajariladi. Bu amal kompilyator tomonidan bajarilganligi sababli funksiyalarni chaqirganda ehtiyot bo'lish kerak. Javob hato ham bo'lishi mumkin. Shu sababli kirish parametrlar tipi sifatida katta hajmli tiplarni qo'llash maqsadga muofiq bo'ladi. Masalan double tipi har qanday sonli tipdagi qiymatni o'z ichiga olishi mumkin. Lekin bunday qiladigan bo'lsak, biz tezlikdan yutqazishimiz turgan gap. Avtomatik konversiyaga misol keltiraylik.

```
 int division(int m, int k){
return (m / k);
}
```

dasturda chaqirsak:

```
...
float f = 14.7;
double d = 3.6;
int j = division(f,d); //f 14 bo'lib kiradi, d 3 bo'lib kiradi
                       // 14/3 - butun sonli bo'lish esa 4 javobini beradi
cout << j;
...
```

Ekranda:

```
 4
```

Demak kompilyator f va d o'zgaruvchilarining kasr qismlarini tashlab yuborar ekan. Qiymatlarni pastroq sig'imli tiplarga o'zgartirish hatoga olib keladi.

Mavzu4: Murakkab ma’lumotlarning turlari. Yangi ma’lumotlarning turlarini kiritish. Tuzilmalar va jamlanmalar.

Shu paytgacha ma'lumotlar tipi deganda butun son va kasrli son bor deb kelgan edik. Lekin bu bo'limda maylumotlar tipi tushunchasini yahshiroq ko'rib chiqish kerak bo'ladi. Chunki funksiyalar bilan ishlaganda argument kiritish va qiymat qaytarishga to'g'ri keladi. Agar boshidan boshlaydigan bo'lsak, kompyterda hamma turdagi ma'lumotlar 0 va 1 yordamida kodlanadi. Buning sababi shuki, elektr uskunalar uchun ikki holat tabiyi-dir, tok oqimi bor yoki yo'q, kondensatorda zaryad bor yoki yo'q va hakozo. Demak biz bu holatlarni oladigan jihozlarni bir quti deb faraz qilsak, quti ichida yo narsa bo'ladi, yo narsa bo'lmaydi. Mantiqan buni biz bir yoki nol deb belgilaymiz. Bu kabi faqat ikki holatga ega bo'lishi mumkin bo'lgan maylumot birligiga biz BIT deymiz. Bu birlik kichik bo'lgani uchun kompyuterda bitlar guruhi qo'llaniladi. Bittan keyingi birlik bu BAYT (byte). Baytni sakkizta bit tashkil etadi. Demak bir bayt yordamida biz 256 ta holatni kodlashimiz mumkin bo'ladi. 256 soni ikkining sakkizinchi darajasiga tengdir. Bitimiz ikki holatga ega bo'lgani uchun biz kompyuterni ikkili arifmetikaga asoslangan deymiz. Ammo agar kerak bo'lsa, boshqa sistemaga asoslangan mashinalarni ham qo'llash mumkin. Masalan uchli sanoq sistemasiga asoslangan kompyuterlar bor. Informatika faniga ko'ra esa, hisoblash mashinasi uchun eng optimal sanoq sistemasi e ga teng bo'lar ekan. Demak amaldagi sistemalar ham shu songa iloji borisha yaqin bo'lishi kerakdir. C/C++ da baytga

asoslangan tip char dir. char tipi butun son tipida bo'lib, chegaraviy qiymatlari -128 dan +127 gachadir. Bu tip lotin alifbosi harflarini va y ana qo'shimcha bir guruh simvollarni kodlashga qulay bo'lgan. Lekin hozirda milliy alifbelarni kodlash uchun 16 bitlik UNICODE qo'llanilmoqda. U yordamida 65536 ta simvolni ko'rsatish mumkin. char tipida o'zgaruvchi e'lon qilish uchun dasturda

char g, h = 3, s;

kabi yozish kerak.

O'zgaruvchilar vergul bilan ayriladi. E'lon bilan bir vaqtning o'zida boshlang'ich qiymat ham berish imkoni bor. Mashina ichida baytdan tashkil topgan boshqa kattaliklar ham bor. Ikki baytdan tuzilgan kattalik so'z (word) deyiladi, unda 16 bit bo'ladi. 4 ta bayt guruhi esa ikkili so'z (double word) bo'ladi. Bu birlik 32 bitli mashimalarda qo'llaniladi. Hozirda qo'llanilmoqda bo'lgan mashinalar asosan 32 bitlidir, masalan Pentium I/II/III sistemalari. C++ da butun sonlarning ikki tipi bor. Biri char - uni ko'rib chiqdik. Ikkinchisi int dir. Mashinalarning arhitekturasi qanday kattalikda bo'lsa, int tipining ham kattakigi huddi shunday bo'ladi. 16 bitlik mashinalarda int 16 bit edi. Hozirda esa int ning uzunligi 32 bitdir. int (integer - butun son) tipi charga o'hshaydi. Farqi bir baytdan kattaligidadir. 16 bitli int ning sig'imi -32768 dan +32767 gachadir. 32 bitli int esa -2 147 483 648 dan +2 147 483 647 gacha o'rin egallaydi. Bu ikki butun son tipidan tashqari C++ da ikki tur vergulli, (nuqtali) yani haqiqiy son tipi mavjud. Bulardan biri float, hotirada 4 bayt joy egallaydi. Ikkinchisi esa double, 8 bayt kattalikka ega. Bularning harakteristikalari quyidagi jadvalda berilgan. Ushbu tiplar bilan ishlaganda unsigned(ishorasiz, +/- siz), signed (ishorali) long (uzun) va short (qisqa) sifatlarini qo'llasa bo'ladi. unsigned va signed ni faqat butun son tiplari bilan qo'llasa bo'ladi. unsigned qo'llanganda sonning ishorat biti bo'lmaydi, ishorat biti sonning kattaligini bildirish uchun qo'llaniladi. Masalan char tipida 8 chi, eng katta bir odatda ishorat bitidir. Biz unsigned char ch; desak, ch o'zgaruvchimizga faqat 0 va musbat qiymatlarni berishimiz mumkin. Lekin oddiy char [-128;127] ichida bo'lsa, unsigned char [0;255] orasidagi qiymatlarni oladi, chunki biz ishorat biti ham qo'llamoqdamiz. Huddi shunday unsigned int da (4 baytli) qiymatlar [0;4 294 467 296] orasida yotadi.

signed ni ishlatib esa biz ochiqchasiga butun sonimizning ishorati bo'lishi kerakligini bildiramiz. Normalda agar signed yoki unsigned qo'yilmasa, tipimizning ishorasi bo'ladi. **long int** bilan qo'llanilganda 16 bitli int 32 ga aylanadi. Bu agar mashina 16 bitli bo'lsa, mashina 32 bitli arhitekturaga ega bo'lsa, int ning kattaligi 4 bayligicha qolaveradi. long double tipi esa 10 bayt joy oladi. Short sifati int bilan qo'llanilganda 32 bit emas, 16 bit joy egallashga boshlaydi. Tezlikni oshirish maqsadida kam joy egallaydigan ma'lumot tiplarini qo'llash maqsadga muofiqdir. Agar tipning nomi yozilmagan bo'lsa, o'zgaruvchi int tipiga ega deb qabul qilinadi.

Ma'lumot tiplarining nomlari	Sinonimlar	Keng tarqalgan harakteristikalari
long double		10 bayt, +/-3.4e-4932...+/-3.4e4932

double 8 bayt, +/-1.7e-308...+/-1.7e308
float 4 bayt, +/-3.4e-38...+/-3.4e38
unsigned long int unsigned long
long int long
unsigned int unsigned
int
unsigned short int unsigned short
short int short
unsigned char
short
char

char va **int** dan tashqari C++ da yana bir necha integral tiplar mavjud. Bulardan biri bool tipidir. bool tipi faqat ikki farqli qiymat olishi mumkin. Bittasi true (to'g'ri) ikkinchisi false (noto'g'ri). bool tipi mantiqiy arifmetika amallarini bajarganda juda qo'l keladi. bool tipi boshqa bir integral tipga asoslangan bo'lishiga qaramasdan (int yoki char), yuqoridagi ikki qiymatdan tashqari boshqa qiymat ololmaydi. bool tipi o'zgaruvchilari to'g'ri shaklda initsalizatsiya qilinmagan taqdirda, ularning qiymati hato ravishda na true va na false bo'lishi mumkin.Yana boshqa bir integral tip bu **wchar_t** dir (wide char type - keng simvol tipi). U ham ko'pincha boshqa bir butun son tipiga asoslanadi - bir baytdan kattaroq bo'lishi kerakligi uchun short int qo'llaniladi.**wchar_t** simvollar kodlanishida qo'llaniladi. Masalan C++ da UNICODE ni odatda wchar_t bilan kodlaymiz. Hozirda wchar_t ning kattaligi 16 bit, lekin yuqori kattaligi necha bit bo'lishi kerakligi standartda belgilanmagan. Butun sonlarni C++ da bir necha asosda berish mumkin. Hech qanday belgi qo'yilmasdan yozilgan son o'nlik asosda (decimal) deb qabul qilinadi.

Sakkizli asosdagi (octal) sonni berish uchun sondan oldin 0o yoki 0O belgilarini qo'yish kerak.O'n oltilik sistema-
dagi (hexadecimal) sonlar oldiga 0x yoki 0X lar yoziladi. Sakkizli sistemada qo'llaniladin raqamlar to'plami 0,1,2,3,4,5,6 va 7 dir. O'n oltilik asosda 0 dan 9 gacha sonlar, 10 - a, 11 - b, 12 - c, 13 - d, 14 - e va 15 uchun f qo'llaniladi. Harflar katta bo'lishi ham mumkin. Harflarning registorining (katta-kichikligi) farqi yo'q. Misol beraylik:

```
char d = 10, j = 0o11; // d 10 ga teng, j 9 ga teng.
int f = 0X10;          // f 16 ga teng
```

Butun son va kasr son tiplaridan tashqari C++ da void (bo'sh, hech narsa) tipi ham mavjud. Bu tipning oladigan qiymatlari bo'sh to'plamga tengdir. Void tipidagi ma'lumot chala tugallangan hisoblanadi. Boshqa turdagi ma'lumotni void ga keltirish mumkindir. Bu tip bilan ishlashni dasturlarimizda ko'rib chiqamiz.

MA'LUMOTLAR TIPINI KELTIRISH (DATA CASTING)

Gohida bir turdagi o'zgaruvchining qiymatini boshqa tipdagi o'zgaruvchiga berish kerak bo'ladi. Bu amal ma'lumot tipini keltirish (data type casting) deyiladi. Ko'p hollarda bu amal avtomatik ravishda, kompilyator tarafidan bajariladi. Masalan ushbu parchani ko'raylik:

```
 char c = 33;
int k;
 k = c;
```

Bu yerda k ning sig'imi c nikidan kattaroqdir. Shuning uchun c ning qiymatini k ga berishda hech qanday muammo paydo bo'lmaydi. Quyidagi misolni ko'raylik:

```
 int i = 5;
float f = 9.77;
float result;
 result = f + i;
```

C++ ning kompilyatori ikki turdagi o'zgaruvchilar bilan ishlay olmaydi. Shu sababli ifodadagi sig'imi kichik bo'lgan o'zgaruvchilar ifodadagi qatnashgan eng katta sig'imga o'tqaziladi. Bu ham avtomatik tarzda bajariladi. i o'zgaruvchimiz qiymati vaqtinchalik **float** tipidagi o'zgaruvchiga beriladi. Bu vaqtinchalik o'zgaruvchi esa f ga qo'shiladi. Chiqqan javob result ga beriladi. Yuqorida ko'rib chiqqanlarimiz kompilyator tarafidan bajariladi. Bu kabi tip o'zgarishlarini avtomatik konversiya(implicit conversion) deymiz. Lekin gohida to'g'ri kelmaydigan tiplarni birga qo'llashga to'g'ri keladi. Masalan **float** tipiga double tipni o'tqazish, char ga int va hokazo. Bu hollarda ochiq konversiya (explicit conversion) amalini bajarishimiz kerak. Buni bajarishning ikki uslubi bor. Birinchisi C da qo'llaniladigan yo'l, ikkinchisi C++ uslubi. C da tipni keltirish uchun o'zgaruvchi oldiga kerakli tipni () qavslar ichida yozamiz.

```
 int k = 100;
char s;
 s = (char)k;
```

Yuqorida k ning qiymatini char tipidagi vaqtinchalik o'zgaruvchiga berildi, keyin s ga ushbu o'zgaruvchi qiymatini berildi.Bu yerda etibor berilishi kerak bo'lgan narsa shuki, 100 char ga ham to'g'ri keladi. Agar k ning qiymati char oladigan qiymattan kattaroq/kichikroq bo'ladigan bo'lsa, bu hato olib keladi. Shu sababli C dagi tip keltirish nisbatan havfli hisoblanadi. Lekin albatta bilib qo'llanilsa katta yordam beradi. C++ da ma'lumotlar tipini keltirish uchun mahsus operatorlar kiritildi. C uslubidagi keltirish hamma sharoitda qo'llanilar edi. C++ ning keltirish operatorlari esa faqat o'ziga ajratilgan funksiyalarni bajaradi. Undan tashqari ular C dagi keltirishlardan ko'ra kuchsizroqdir. Shu sababli hato ehtimoli kamaytirildi. Yana bir afzallik tarafi shundaki, yangi stildagi keltirish operatorlari tip tekshirishlarini bajarishadi, agar noto'g'ri keltirish bajarilsa, bu sintaktik hatoga olib keladi.

Ular quyida berilgan:

```
 static_cast
dynamic_cast
const_cast
reinterpret_cast
```

static_cast ni ko'rib chiqaylik.

```
 int k = 200;
char h;
 h = static_cast<char>(k);
```

static_cast dan keyin kerakli tip nomi <> qavslar ichida beriladi, va tipi o'zgarishi kerak bo'lgan o'zgaruvchi () qavslar ichida parametr sifatida beriladi. Static_cast kompilyatsiya davrida tip keltirishlarida qo'llaniladi.**dynamic_cast** esa dastur ishlash davrida tip keltirishlari uchun qo'llaniladi.
const_cast esa o'zgaruvchilardan const (o'zgarmas) va volatile (o'zgaruvchan, uchuvchan) sifatlarini olib tashlashda qo'llaniladi. Odatda const o'zgaruvchining qiymatini o'zgartirib bo'lmaydi. Ushbu holda const_cast
qo'llaniladi. **reinterpret_cast** odatiy bo'lmagan keltirishlarni bajarishda qo'llaniladi (masalan void* ni int ga). reinterpret_cast o'zgaruvchining bitlarini boshqa ma'noda qo'llashga imkon beredi. Bu operator bilib
ishlatilinishi kerak. Ushbu operatorlar bilan keyinroq yanada yaqin tanishamiz.

Mavzu5:Tuzilmalar va massivlar. Tuzilmalar va funksiyalar.
FUNKSIYA ISMI YUKLANISHI

Bir hil ismli bir necha funksiya e'lon qilinishi mumkin. Bu C++ dagi juda kuchli tushunchalardandir. Yuklatilgan funksiyalarning faqat kirish parametrlari farqli bo'lishi yetarlidir. Qaytish parametri yuklatilishda ahamiyati yo'qdir. Yuklangan funksiyalar chaqirilganda, qaysi funksiyani chaqirish kirish parametrlarining soniga, ularning tipiga va navbatiga bog'liqdir. Yani ism yuklanishida funksiyaning imzosi rol o'ynidi. Agar kirish parametrlari va ismlari ayni funksiyalarning farqi faqat ularning qaytish qiymatlarida bo'lsa, bu yuklanish bo'lmaydi, kompilyator buni hato deb e'lon qiladi. Funksiya yuklanishi asosan ayni ishni yoki amalni farqli usul bilan farqli ma'lumot tiplari ustida bajarish uchun qo'llaniladi. Masalan bir fazoviy
jismning hajmini hisoblash kerak bo'lsin. Har bir jismning hajmi farqli formula yordamida, yani farqli usulda topiladi, bir jismda radius tushunchasi bor bo'lsa, boshqasida asos yoki tomon tushunchasi bor bo'ladi, bu esa farqli ma'lumot tiplariga kiradi. Lekin amal ayni - hajmni hisoblash. Demak, biz funksiya yuklanishi mehanizmini qo'llasak bo'ladi. Bir hil amalni bajaruvchi funksiyalarni ayni nom bilan atashimiz esa, dasturni o'qib tushunishni osonlashtiradi. Kompilaytor biz bergan funksiya imzosidan (imzoga funksiya ismi va kirish parametrlari kiradi, funksiyaning qaytish qiymati esa imzoga kirmaydi) yagona ism tuzadi, dastur ijrosi davruda esa funksiya chaqirig'idagi argumentlarga qarab, kerakli funksiyani chaqiradi. Yangi ismni tuzish operatsiyasi ismlar dekoratsiyasi deb ataladi. Bu tushunchalarni misolda ko'rib chiqaylik.

```
// Yuklatilgan funksiyalarni qo'llash
 # include <iostream.h>
# include <math.h>
   // Yangi ismlar sohasini aniqladik
namespace
  mathematics {
     const double Pi = 3.14159265358979;
 double hajm(double radius); // sharning hajmi uchun - 4/3 * Pi * r^3
double hajm(double a, double b, double s) // kubning hajmi uchun - abc
```

```
 }
 using namespace mathematics;
 int main()
{
   double d = 5.99; // sharning radiusi
   int x = 7, y = 18, z = 43;
    cout << "Sharninig hajmi: " << hajm(d) << endl;
   cout << "Kubning hajmi: " << hajm(x,y,z) << endl;
   return (0);
}
 double mathematics::hajm(double radius) {
   return (  (Pi * pow(radius,3) * 4.0) / 3.0  );
}
 double mathematics::hajm(double a, double b, double c) {
   return ( a * b * c );
}
```

Ekranda:

```
 Sharning hajmi: 900.2623
Kubning hajmi: 5418
```

Yuqoridagi dasturda yangi ismlar sohasini aniqladik, unda Pi konstantasini e'lon qildik. shaqning hajmini hisoblashda standart kutubhonadagi pow() funksiyasini ishlatdik, shu sababli <math.h> e'lon faylini # include ifodasi bilan kiritdik. Ismlar sohasida joylashgan funksiyalarni aniqlash uchun, yani ularning tanasini yozish uchun biz ilarning to'liq ismini berishimiz kerak. Albatta, agar funksiya ismlar sohasining ichida aniqlangan bo'lsa, tashqarida boshqattan yozib o'tirishning hojati yo'q. hajm() funksiyalarining to'liq ismi mathematics::hajm(...) dir. :: operatori sohalarni bog'lovchi operatordir. Yangi ismlar sohasini faqatgina misol tariqasida berdik, uni funksiya yuklanishlari bilan hech qanday aloqasi yo'qdir. Funksiya ismlari yuklanishi, ko'rib turganimizdek, juda qulay narsadir. Funksiya yuklanishini qo'llaganimizda, funksiyalar argumentlarining berilgan qiymatlarini ehtiyotkorlik bilan qo'llashimiz lozim. Masalan bizda ikkita funksiyamiz bor bo'lsin.

```
foo(int k = 0); // berilgan qiymati 0
foo();
```

Bu ikki funksiya yuklatilgan. Lekin agar biz birinchi funksiyani dasturda argumentsiz chaqirsak, kompilyator qaysi funksiyani chaqirishni bilmaydi, va shu sababli hato beradi. Biroq bu deganimiz funksiya yuklanishi bilan berilgan qiymatlar qo'llanilishi mumkin emas deganimiz emas, eng muhimi funksiya chaqirig'ini ikki hil tushunish bo'lmasligi kerak.

FUNKSIYA SHABLONLARI

Funksiya shablonlari (function templates) ham funksiya yuklanishiga o'hshash tushunchadir. Bunda eng asosiy farq funksiya shablonlarida amal ham bir hil yo'l bilan bajariladi. Masalan bir necha sonlar ichidan eng kattasini topish kerak bo'lsin. Sonlar to'plami faqat tipi bilan farqlanadi, int, double yoki float. Ishlash

algoritmi esa aynidir. Bu holda biz funksiyalarni yuklab o'tirmasdan, shablon yozib qo'ya qolamiz. Funkisya shabloni yoki yuklanishisiz ham bu masalani yechish mumkinku degan savol paydo bo'ladi. Masalan, agar biz kiradigan parametrlarning hammasini long double qilsak, istalgan sonli tipdagi argumentni bera olamiz, chunki kompilyator o'zi avtomatik ravishda kirish tiplarini *long double* ga o'zgartiradi. Lekin, agar biz bunday funksiya yozadigan bo'lsak, hotiradan va tezlikdan yutqizamiz. Dasturimizda faqat char tipidagi, bir baytli qiymatlar bilan ishlashimiz mumkin. long double esa 10 bayt, va eng katta sonni aniqlash uchun sonlarni solishtirganimizda, long double qiymatlarni solishtirish char tipidagi qiymatlarni solishtirishdan ko'ra ancha ko'p vaqt oladi. Qolaversa, har doim ham kompilyator tiplarni biridan ikkinchasiga to'g'ri keltira oladi. Shablonlarning strukturasi bilan tanishaylik. Bizning funksiya ikkita kirish argumentini bir biriga qo'shsin, va javobni qaytarsin.

```
template <class T>
T summa(T a, T b) {
  return ( a + b);
}
```

Shablon funksiya e'loni va aniqlanishidan oldin template <> ifodasi yoziladi, <> qavslardan keyin nuqta-vergul (;) qo'yilmaydi. <> qavslar ichida funksiya kirish parametrlari, chiqish qiymati va lokal o'zgaruvchilar tiplari beriladi. Ushbu formal tiplarning har birining oldida class yoki typename (tip ismi) so'zi qo'yilish kerak. Yuqoridagi misolda T ning o'rniga istalgan boshqa identefikator qo'yish mumkin. Misollar beraylik.

```
template <class javob, class uzunlik, class englik, class balandlik>
javob hajmKub(uzunlik a, englik b, balandlik c);
 template <typename T>
T maximum(T k, T l);
```

Yuqorida yozgan shablonimizni qo'llagan holga bir misol keltiraylik.

```
 // Shablonlar bilan ishlash
 # include <iostream.h>
 template <class T>
T summa(T a, T b) {
  return ( a + b );
}
 int main()
{
int x = 22, y = 456;
float m = .01, n = 56.90; // kasrli sonda nuqtadan oldingi (butun qismdagi)
                          // nolni berish shart emas: ... m = .01 ...
  cout << "int: 22 + 456 = " << summa(x,y) << endl;
  cout << "float: 0.01 + 56.90 = " << summa(0.01,56.90) << endl;
  return (0);
}
```

Ekranda:

```
 int: 22 + 456 = 478
```

float: 0.01 + 56.90 = 56.91

Shablonlarni funksiyalardan tashqari klaslarga ham qo'llasa bo'ladi. Ko'rib turganimizdek, shablonlar faqat bir marotaba yoziladi. Keyin esa mos keladigan tiplar qo'yilib, yozilib ketilaveradi. Aslida shablonlar C++ ning standartida juda ko'p qo'llanilgan. Agar bilib ishlatilsa, shablonlar dasturchining eng kuchli quroliga aylanishi mumkin. Biz keyinroq yana
shablonlar mavzusiga qaytamiz.

Funksiyalarga massivlarni kirish argument sifatida berish uchun parametr e'lonida [] qavslar qo'yiladi. Masalan:

```
...
void sortArray(int [], int );  // funksiya e'loni
void sortArray(int n[], int hajm) { // funksiya aniqlanishi
 ...
}
...
```

Dasturda esa, funksiya chaqirilganda, massivning faqat ishmi beriladi halos, [] qavslarning keragi yo'q.

```
 int size = 10;
int array[size] = {0};
 ...
void sortArray(array, size); // funksiya chaqirig'i,
                    // faqat massiv ismi - array berildi
...
```

Funksiyaga massivlarni berganimizda, eng katta muammo bu qanday qilib massivdagi elementlari sonini berishdir. Eng yaxshi usul bu massiv kattaligini qo'shimcha kirish parametri orqali funksiyaga bildirishdir. Bundan tashqari, massiv hajmini global konstanta orqali e'lon qilishimiz mumkin. Lekin bu ma'lumotni ochib tashlaydi, global sohani ortiqcha narsalar bilan to'ldirib tashlaydi. Undan tashqari massiv hajmini funksiyaning o'ziga yozib qoyishimiz mumkin. Biroq bunda bizning funksiyamiz faqat bitta kattalikdagi massivlar bilan ishlaydigan bo'lib qoladi. Yani dasturimiz dimamizmni yo'qotadi. Klaslar yordamida tuzilgan massivlar o'z hajmini biladi. Agar bunday ob'ektlarni qo'llasak, boshqa qo'shimcha parametrlarni qo'llashimizning keragi yo'q. Funksiyalarga massivlar ko'rsatkich ko'rinishida beriladi. Buni C++, biz ko'rsatmagan bo'lsak ham, avtomatik ravishda bajaradi. Agar massivlar qiymat bo'yicha chaqirilganda edi, har bir massiv elementining nushasi olinishi kerak bo'lardi, bu esa dastur ishlash tezligiga salbiy ta'sir ko'rsatar edi. Lekin massivning alohida elementi argument o'rnida funksiyaga berilganda, ushbu element, aksi ko'rsatilmagan bo'lsa, qiymat bo'yicha beriladi. Masalan:

```
 ...
double m[3] = {3.0, 6.88, 4.7};
void foo(double d){
   ...
```

```
}
...
int main()
{
...
void foo(m[2]); // m massivining uchinchi elementining qiymati - 4.7 berildi
...
return (0);
}
```

Agar kiritilayatgan massiv funksiya ichida o'zgarishi ta'qiqlansa, biz funksiya massiv parametri oldiga const sifatini qo'ysak bo'ladi:

```
foo(const char []);
```

Bunda funksiyaga kiradigan massiv funksiya tomonidan o'zgartirilmaydi. Agar o'zgartirishga urinishlar bo'lsa, kompilyator hato beradi. Massivlar va funksiyalarning birga ko'llanilishiga misol beraylik.

```
// Massiv argumentli funksiyalar
 # include <istream.h>
 const int arraySize = 10;
 double ortalama(int m[], int size) {
   double temp = 0;
    for (int i = 0; i < size; i++) {
      temp += m[i];
   }
   return ( temp / size );
}
 void  printArray(const int n[], int size, int ortalama) {
   for (int i = 0; i < size; i++) {
      cout << n[i]; << endl;
   }
   cout << "O'rtalama: " << ortalama << endl;
}
 int main()
{
   int m[10] = {89,55,99,356,89,335,78743,44,767,346};
   printArray(m, arraySize, ortalama(m, arraySize)) ;
   return (0);
}
```

Ekranda:

```
 89
55
99
356
89
335
```

78743
44
767
346
O'rtalama: 8092.3

Mavzu6: Ob’ektli dasturlash tillari. Ob’ektga yo’naltirilgan dasturlash. Sinf va ob’ekt tushinchasi. Vorislik, inkapsulyasiya va polimorfizm

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.

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.
74
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>

76
}
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

Mavzu7: Sinflar va ob'ektlar. Interfeyslar. Konstruktorlar va destruktorlar. Sinf strukturaning kengaytmasi sifatida.

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();
…
};
```

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 ko‘rishi tushuniladi. SHunday qilib, polimorfizm yordamida bitta nom turli xulq -atvorni bildirishi mumkin.

Vorislik polimorfizmning ayrim turlaridan foydalanish uchun zarurdir. Aynan o‘rindoshlik imkoniyati mavjud bo‘lgani uchun, polimorfizmdan foydalanish mumkin bo‘ladi. Polimorfizm yordamida tizimga to‘g‘ri kelgan paytda qo‘shimcha funksiyalarni qo‘shish mumkin. Dasturni yozish paytida hatto taxmin qilinmagan funksionallik bilan yangi sinflarni qo‘shish mumkin, buning ustiga bularning hammasini dastlabki dasturni o‘zgartirmay turib ham amalga oshirish mumkin. YAngi talablarga osongina moslasha oladigan dasturiy vosita deganda, mana shular tushuniladi.

Polimorfizmning uchta asosiy turi mavjud:

- qo‘shilish polimorfizmi;
- parametrik polimorfizm;
- ortiqcha yuklanish;

Qo‘shilish polimorfizmini ba’zida sof polimorfizm deb ham ataydilar. Qo‘shilish polimorfizmi shuning bilan qiziqarliki, uning tufayli tarmoq sinf nushalari o‘zini turlicha tutishi mumkin. Qo‘shilish polimorfizmidan foydalanib, yangi tarmoq sinflarni kiritgan xolda, tizimning xulq-atvorini o‘zgartirish mumkin. Uning bosh afzalligi shundaki, dastlabki dasturni o‘zgartirmay turib, yangi xulq-atvorni yaratish mumkin.

Aynan polimorfizm tufayli joriy qilishdan takroran foydalanishni vorislik bilan aynanlashtirish kerak emas. Buning o‘rniga vorislikdan avvalam bor o‘zaro almashinish munosabatlari yordamida polimorf xulq-atvorga erishish uchun foydalanish lozim. Agar o‘zaro almashinish munosbatlari to‘g‘ri belgilansa, buning ortidan albatta takroran qo‘llash chiqib keladi. Qo‘shilish polimorfizmidan foydalanib, bazaviy sinfdan, har qanday avloddan, shuningdek bazaviy sinf qo‘llaydigan 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.