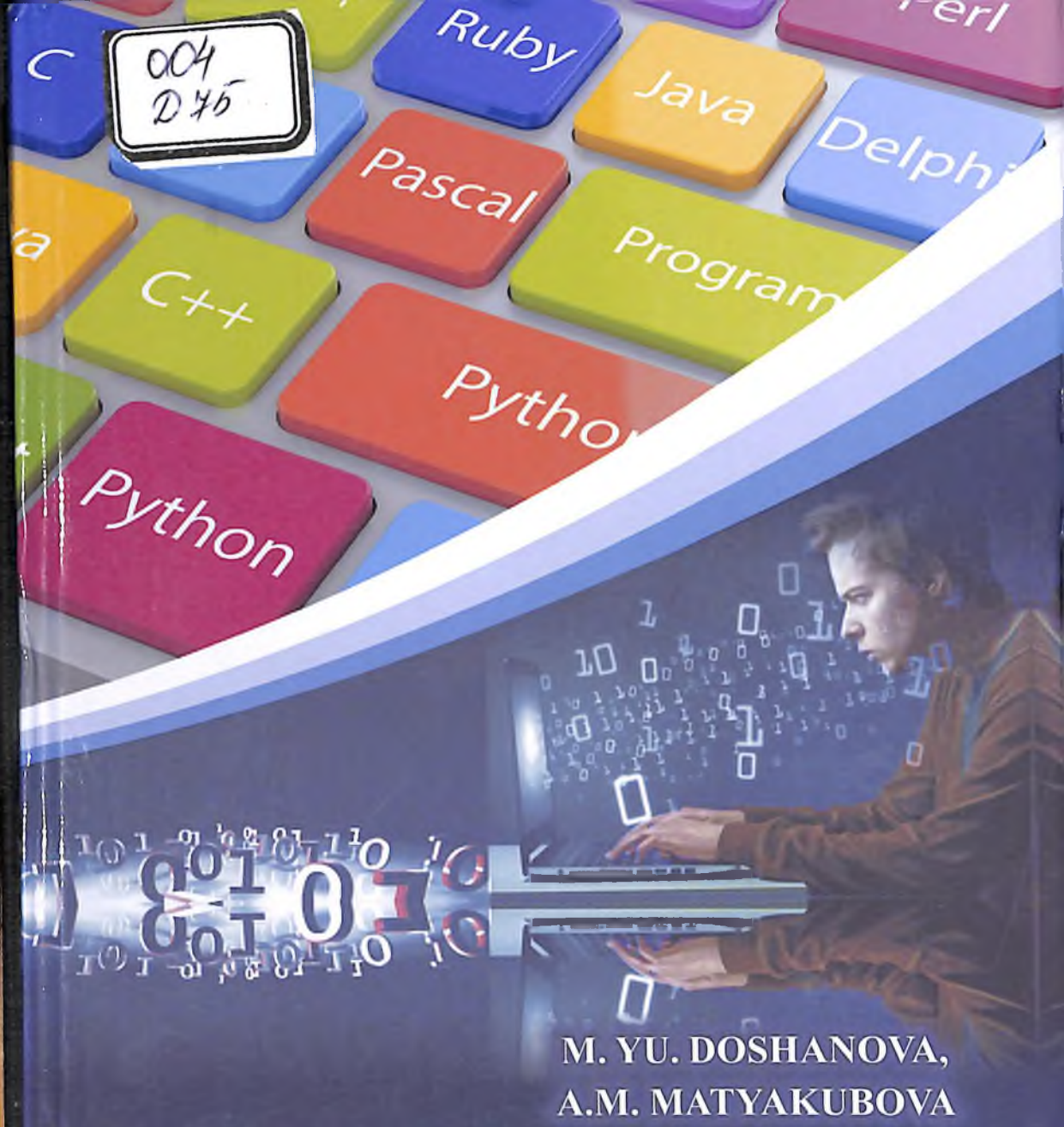


M. YU. DOSHANOVA,
A.M. MATYAKUBOVA

DASTURLASH

O‘ZBEKISTON RESPUBLIKASI AXBOROT TEXNOLOGIYALARI VA KOMMUNIKATSIYALARINI RIVOJLANTIRISH VAZIRLIGI

MUHAMMAD AL-XORAZMIY NOMIDAGI
TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI
“DASTURIY INJINIRING” FAKULTETI

“AXBOROT TEXNOLOGIYALARINING
DASTURIY TA’MINOTI” KAFEDRASI

M. YU. DOSHANOVA, A.M. MATYAKUBOVA

DASTURLASH

(O‘QUV QO‘LLANMA)

TOSHKENT – 2019

UO'K: 004.43
KBK: 32.973

M. Yu. Doshanova, A.M. Matyakubova. Dasturlash. O'quv qo'llanma. – T.: «Aloqachi», 2019. – 140 b.

ISBN 978–4493–5805–1–0

"Dasturlash" nomli o'quv qo'llanma kasb – hunar kollejlari o'quvchilari uchun mo'ljallangan. O'quv qo'llanmada C++ dasturlash tiliga kirish, murakkab tiplar, asosiy konstruksiyalardan foydalanish xususiyatlari, sinflar va ob'yektlar, sinflar va ob'yektlar bilan ishlash xususiyatlari, sinflarda vorislik, vorislikdan foydalanish xususiyatlari, funksiya va sinf shablonlari, dinamik sinflardan foydalanish xususiyatlari, fayllar bilan ishlash, oqimli sinflar ko'rib chiqilgan.

Shuningdek, dasturlashda ishlatiladigan istisnolarni boshqarish, fayllar va istisnolar bilan ishlash xususiyatlari, konteyner sinflar standart bibliotekasi, standart algoritmlar va ularning turlari, hodisalar asosida dasturlash masalalari yoritilgan.

Mazkur o'quv qo'llanma o'quvchilarga "Dasturlash" fanini o'zlashtirishda, keyinchalik esa ishlab chiqarish, loyihalash va tadqiqot ishlarida kerak bo'ladigan asosiy tushunchalarni o'rgatadi.

O'quv qo'llanma kasb – hunar kollejlarining "3330200 – Kompyuter injiniringi" tayyorlov yo'nalishi o'quvchilari uchun mo'ljallangan.

O'quv qo'llanma Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti ilmiy-uslubiy kengashining qarori bilan chop etishga tavsiya etildi.

UO'K: 004.43
KBK: 32.973

Taqrizchilar: Sh.M.G'ulomov; N.Latipova.
Mas'ul muharrir: A.A.Tulyaganov.

ISBN 978–4493–5805–1–0

KIRISH

"Kadrlar tayyorlash milliy dasturi" maqsadli ta'lim tizimini tubdan isloh qilish, o'quv yurtlarini yangi yo'nalishlarga solish va zamonaviy texnologiyalarda ishlay oladigan mutaxassislar tayyorlashdir.

Hozirgi kunda ta'lim tizimida yangi bosqich bo'lgan kasb-hunar ta'limini zarur me'yoriy hujjatlar, zamonaviy texnika va texnologiyalar bilan ta'minlash borasida bir qator ishlar amalga oshirildi. Jumladan, O'zbekiston Respublikasi Vazirlar Maxkamasining qarori bilan bir qator farmoyishlar tasdiqlandi. Ularda o'rta maxsus, kasb-hunar ta'limi muassasalarining bitiruvchilariga qo'yiladigan fundamental fanlar va aniq kasb soxasi doirasida nazariy va amaliy bilimlarga ega bo'lish, kompyuter va telekommunikatsiya vositalaridan foydalana olish, o'quv fanlari bo'yicha oliy ta'lim muassasalarida taxsil olish uchun zarur bo'lgan bilimlar majmuasiga ega bo'lish vazifasi qo'yilgan. Shu talabdan kelib chiqqan holda, kasb – hunar kollejlari uchun "Dasturlash" fanidan yangi takomillashgan o'quv qo'llanma ishlab chiqish dolzarb hisoblanadi.

Hozirgi paytda zamonaviy dasturlash tillaridan biri hisoblangan C++ dasturlash tilidan keng foydalaniladi. C++ dasturlash tili – bu Windows OT muhitida ishlashga mo'ljallangan. Mazkur dasturlash tili o'zining ba'zi bir xususiyatlari bilan boshqa dasturlash tiliaridan ajralib turadi. Shu sababdan bugungi kunda C++ dasturlash tili akademik litsey, kasb-hunar kollejlari hamda oliy o'quv yurti fan dasturlariga kiritilgan.

C++ dasturlash tili C tiliga asoslangan. Hozirda operatsion tizimlarning asosiy qismi C/C++ da yozilmoqda. C mashina arxitekturasiga bog'langan tildir. Lekin yaxshi rejalashtirish orqali dasturlarni turli kompyuter platformalarida ishlaydigan qilsa bo'ladi. 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 ob'yektli dasturlash g'oyasi ilgari surildi.

C++ tili gibrid tildir. Unda C ga o'xshab strukturali dasturlash yoki yangicha, ob'yektlar bilan dasturlash mumkin. Yangicha deyishimiz

ham nisbiydir. Ob'yektli dasturlash falsafasi paydo bo'lganiga ham yigirma yildan oshayapti. C++ funksiya va ob'yektlarning juda boy kutubxonasiga ega.

Ushbu o'quv qo'llanmada biz C++ dasturlash tilini o'rganishni ikki qismga bo'lib o'rganamiz. Birinchi qismda C++ ni o'zini o'rganamiz, ikkinchi qismida esa C++ ning standart kutubxonasidagi tayyor ob'yekt funksiyalarni qo'llashni o'rganamiz.

1 BOB. C++ DASTURLASH TILIGA KIRISH

1.1. O'zgaruvchilar va konstantalar

O'zgaruvchilar. O'zgaruvchi nomi ostiga chizish belgisi yoki lotin harfidan boshlanuvchi lotin harflari, arab raqamlari va ostiga chizish belgilari ketma ketligi ya'ni identifikatordir.

O'zgaruvchilarning quyidagi tiplari mavjuddir: *char*(simvol), *short*(qisqa butun), *int*(butun), *long*(uzun butun), *float*(haqiqiy), *double*(ikkilangan haqiqiy).

Butun sonlar ta'riflanganda qurilgan tiplar oldiga *unsigned* (ishorasiz) ta'rifi qo'shilishi mumkin. Ishorali ya'ni *signed* tipidagi sonlarning eng katta razryadi son ishorasini ko'rsatish uchun ishlatilsa *unsigned* (ishorasiz) tipdagi sonlarda bu razryad sonni tasvirlash uchun ishlatiladi

O'zgaruvchilar ta'rifi sodda shakli:

<tip> <o'zgaruvchilar_nomlari_ro'yxati >;

o'zgaruvchilarni ta'riflashda boshlangich qiymatlarini ko'rsatish mumkin.

<tip> <o'zgaruvchilar_nomlari_ro'yxati >= <initsializator>;

Bu usul initsializatsiya deyiladi.

Misollar:

float pi = 3.14 , as=1.3456;

unsigned int god = 1999;

Konstantalar. Konstanta bu o'zgartirish mumkin bo'lmagan qiymatdir. Konstantaning quyidagi turlari mavjud:

Butun. O'nlik - oldida 0 emas(masalan: 4, 1, 675); sakkizlik - oldida 0(masalan: 023, 05); o'n oltilik - oldida 0x yoki 0X(masalan: 0xC, 0X0AE);

Simvolli. Oddiy - 'q','s','7'; boshqaruvchi –'\n', '\0xF5';

Haqiqiy. Fiksirlangan nuqtali (masalan: 4.2, .0043, 84.); suzuvchi nuqtali (masalan: 5.7e6, .74e-5, 6A2) ;

Mantiqiy. Mantiqiy konstantalar *true*(rost) va *false*(yolg'on). Ichki ko'rinishi *false* – 0, ixtiyoriy boshqa qiymat *true* deb qaraladi.

Nomlangan konstantalar. Nomlangan konstantalar quyidagi shaklda kiritiladi:

Const tip konstanta_nomi=konstanta_qiymati.

Misol uchun:

Const double qim=2.718282;

Const long S=99999999;

Const K=765;

Yangi tip. Typedef ta'riflovchisi yangi tiplarni kiritishga imkon beradi. Misol uchun yangi COD tipini kiritish:

Typedef unsigned char COD;

COD simbol;

1.2. Amallar

Arifmetik amallar. Binar amallar: + qo'shish, – ayirish, * ko'paytirish, / bo'lish va % modul olish.

Misol uchun 20/3=6; (-20)/3=-6; 5%2=1;

Unar amallar: unar minus – va unar +; inkrement ++ va decrement --. Inkrement va dekrement prefiks ya'ni ++i, postfiks ya'ni i++ ko'rinishda ishlatishi mumkin. Masalan agar i=2, u holda 3+(++i)= 6, 3+i++=5 ga teng bo'ladi. Ikkala holda ham i=3 ga teng bo'ladi.

Nisbat amallari. Nisbat amallari qiymatlari mantiqiy. Katta >, kichik <, katta yoki teng >=, kichik yoki teng <=, teng == , teng emas !=.

Mantiqiy amallar. Mantiqiy amallar qiymatlari mantiqiydir. Mantiqiy amallar || (diz'yunksiya), && (kon'yunksiya), !(inkor).

Razryadli amallar. Razryadli amallar | (diz'yunksiya), & (kon'yunksiya), ^(XOR), !(inkor). Masalan 5 kodi 101 ga teng va 6 kodi 110 ga teng:

6&5= 4=100; 6|5= 7= 111; 6^5=3=011; ~6=4=010.

Chapga surish << amali. Masalan:5<<2=20 yoki 101<<2=10100.

O'ngga surish >> amali. Masalan 5>>2=1 yoki 101>>2=001=1.

Qiymat berish amali. Oddiy qiymat berish amali:

O'zgaruvchi_nomi = ifoda;

Z=4.7+3.34; C=y=f=4.2+2.8;

Murakkab qiymat berish amali:

O'zgaruvchi_nomi amal= ifoda;

Bu yerda amal quyidagi amallardan biri *,/,%,+,-, &,^,|, <<,>>.

Misol uchun: x+=4 ifoda x=x+4 ifodaga ekvivalentdir;

x>>=4 ifoda x=x>>4 ifodaga ekvivalentdir;

Shartli amal. Shartli amal ternar amal deyiladi:

<1-ifoda>?<2-ifoda>:<3-ifoda>

Misol: a<b?a:b .

Tiplar bilan ishlovchi amallar. Tiplarni o'zgartirish amali quyidagi ikki ko'rinishga ega:

Kanonik: (tip_nomi) operand; masalan: r=(unsigned long)1;

Funksional: tip_nomi (operand); masalan: z=double(1);

Xotiradagi hajmni hisoblash *sizeof* amalining ikkita ko'rinishi orqali amalga oshiriladi:

1 – ko'rinish: Sizeof ifoda;

Misol: Sizeof 3.14=8

2 – ko'rinish: Sizeof (tip);

Misol: Sizeof(char)=1

1.3. Ko'rsatkichlar va ilovalar

Ko'rsatkichlar ta'rifi. Ko'rsatkichlar qiymati konkret tipdagi ob'yektlar uchun xotirada ajratilgan adreslarga tengdir. Shuning uchun ko'rsatkichlar ta'riflanganda ularning adreslarini ko'rsatish shart. O'zgaruvchi ko'rsatkichlar quyidagicha ta'riflanadi.

<tip> * <ko'rsatkich nomi>

Misol uchun: int* lp, lk .

Ko'rsatkichlarni ta'riflaganda initsializatsiya qilish mumkindir. Initsializatsiya quyidagi shaklda amalga oshiriladi:

<tip> * <ko'rsatkich nomi>=<konstanta ifoda>

Konstanta ifoda sifatida & simvoli yordamida aniqlangan ob'yekt adresi keladi. Misol uchun:

```
char c='d'; char* pc=&c;
```

Ilovalar. Ilova biror ob'yektning o'zgacha nomidir. Ilovalar quyidagicha ta'riflanishi mumkin:

<tip>& <Ilova_nomi>=<ifoda>

<tip>& <Ilova_nomi>(<ifoda>)

Misol uchun:

int l=777; int& rl=l; int& pl(l)

Rl yoki pl qiymatlarini o'zgartirish avtomatik ravishda l ning ham qiymati o'zgaradi.

Ko'rsatkichlarga o'xshab ilovalarning qiymatlari ham adreslardir. Lekin ilovalarning qiymatlarini o'zgartirish mumkin emas va ilovalarga murojaat qilinganda avtomatik ravishda * qiymat olish amali bajariladi.

Ilovalar bilan ishlash qoidalari. Ilova o'zgaruvchi emasdir. Ilovaga bir marta qiymat bergandan so'ng uni o'zgartirish mumkin emas. Bundan tashqari ilovalar ustida quyidagi amallarni bajarish mumkin emas:

- C++ da adres olish operatori yordamida ilovaning adresini olish mumkin emas.
- Ilovaga ko'rsatkich qiymatini berish mumkin emas.
- Ilovalarni solishtirish mumkin emas.
- Ilovalar ustida arifmetik amallar bajarish mumkin emas.
- Ilovani o'zgartirish mumkin emas.

1.4. Dastur strukturasi

C++ tilidagi dastur quyidagi strukturaga ega:

#< preprotsessor direktivasi>

<funksiyalar>

void main ( ) { <operatorlar> }

preprotsessor direktivasi – dasturdagi almashtirishlarni uning kompilyatsiyasigacha boshqaradi.

1) *#define* – tekstdagi almashtirishlar qonun qoidasini aniqlaydi.

Misol:

#define ZERO 0.0

Bu dasturda tekstdagi har bir zero so‘zi 0.0 bilan almashtirilishini bildiradi.

2) *#include*<sarlavha fayl nomi> – standart bibliotekalar bilan birga yetkaziladigan sarlavxa faylidagi funksiyalarni ishlatishga mo‘ljallangan.

Misol:

#include< iostream.h >

1.5. Operatorlar

Kiritish-chiqarish operatorlari: *cin* – kiritish; *cout* – chiqarish.

Shartli operatorlar: *if* - shartli operator; *switch* – kalit bo‘yicha o‘tish.

Sikl operatorlari: *while* – oldingi shartli sikl; *do while* – keyingi shartli sikl; *for* – parametrli sikl.

O‘tish operatorlari: *break* – uzish operatori; *continue* – siklning keyingi iteratsiyasiga o‘tish operatori.

Misol:

```
# include <iostream.h>
 void main() {
int n;
do {
cout<<"\n Baxo ="; cin>>n;
switch(n) {
case 2:cout<<"\n qoniqarsiz"; break;
case 3:cout<<"\n qoniqarli"; break;
case 4:cout<<"\n yaxshi"; break;
case 5:cout<<"\n a'lo"; break;
default :cout<<"\n noto‘g‘ri kiritish"; break;
} }
while(n!=0) }
```

Misol:

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

```
 void main()      {
int k, n;
 cout<<"n ="; cin>>n;
for(int i= 0,s=0; i<n; i++) {
cin>>k;
if k<0 continue; s+=k;}
cout<<"s="<<s;
 }
```

1.6. Foydalanuvchi funksiyalari

Funksiya ta'rifi. Funksiyani C++ tilida quyidagi ikki sifatda qarash mumkin:

- hosila tiplaridan biri;
- dastur bajariluvchi minimal moduli.

Funksiya ta'rifining umumiy ko'rinishi quyidagichadir:

<tip> <funksiya nomi>(<formal_parametrlar_ta'rifi>)

Formal parametrlarga ta'rif berilganda ularga boshlangich qiymatlari ham ko'rsatilishi mumkin.

Funksiya qaytaruvchi ifoda qiymati funksiya tanasida

return <ifoda> ;

operatori orqali ko'rsatiladi. Misol:

```
float  min(float, float b) {
if (a<b)
return a;
return b;
}
```

Funksiyaga murojaat qilish quyidagicha amalga oshiriladi:

<Funksiya nomi> (<haqiqiy parametrlar ro'yxati>)

Misol:

```
        int x=5,y=6, z;
        z=min(x,y)
```

yoki

```
        int z=min(5,6)
```

yoki

int x=5; int z=min(x,6).

Funksiya ta'rifida formal parametrlar initsializatsiya qilinishi, ya'ni boshlang'ich qiymatlar ko'rsatilishi mumkin.

Misol uchun:

```
float  min(float a=0.0, float b=0) {
if (a<b) return a;
return b;
}
```

Bu funksiyaga quyidagicha murojaat qilish mumkin:

```
int y=6,z; z=min(,y)
```

yoki

```
int z=min(,6);
```

Prototip. Agar programmada funksiya ta'rifi murojaatdan keyin berilsa yoki funksiya boshqa faylda joylashgan bo'lsa, murojaatdan oldin shu funksiyaning prototipi joylashgan bo'lishi kerak. Prototip funksiya nomi va formal parametrlar tiplaridan iborat bo'ladi. Formal parametrlar nomlarini berish shart emas.

Misol uchun:

```
float  min(float, float);
```

Protseduralar. Funksiyaga parametrlar qiymat bo'yicha uzatiladi. Funksiyaga parametrlar qiymatlari uzatilishi haqiqiy parametrlar qiymatlarini funksiya tanasida o'zgartirish imkonini bermaydi. Bu muammoni hal qilish uchun ko'rsatkichlardan foydalanish mumkin.

Masalan:

```
void change (int &a, int &b) {
int r;
r = a;  a = b; b = r;
}
// funksiya chaqirig'i
change(a, b);
```

Inlayn funksiyalar. Dasturda funksiya ta'riflanganda kompilyator funksiya kodini bir marta mashina kodiga o'tkazadi va dasturga ma'lumotlarni stekka joylovchi instruksiyalar qo'shadi. Argumentlarni stekka qo'shish, funksiyaga o'tish va qaytish mashina vaqtini oladi.

C++ kompilyatori *inline* so'zini uchratsa bajariluvchi faylga har bir funksiyaga murojaat o'rniga funksiya operatorlarini qo'yadi. Shunday qilib dastur samaradorligini oshirish mumkindir.

Misol:

```
inline float Line(float x1,float y1,float x2=0, float y2=0)
{
return sqrt(pow(x1-x2)+pow(y1-y2,2));
}    //funksiya ikki nuqta orasidagi masofani qaytaradi.
```

Funksiyalarni qo'shimcha yuklash. Funksiyalarni qo'shimcha yuklashdan maqsad bir xil nomli funksiyaga har xil tipli o'zgaruvchilar bilan murojaat qilib qiymat olishdir. Kompilyator haqiqiy parametrlar ro'yxati va funksiya chaqirig'i asosida qaysi funksiyani chaqirish kerakligini o'zi aniqlaydi.

Misol uchun har xil o'zgaruvchilarni ko'paytirish uchun quyidagi funksiyalar kiritilgan bo'lsin:

```
float  min(float a, float b) {
if (a<b) return a;
return b;
}
int  min(int a, int b) {
if (a<b) return a;
return b;
}
```

Quyidagi murojaatlarning har biri to'g'ri bajariladi:

```
int m=min(6, x);
float c=min(5, y);
```

Rekursiv funksiyalar. Rekursiv funksiya deb o'ziga o'zi murojaat qiluvchi funksiyaga aytiladi. Misol uchun faktorialni hisoblash funksiyasini keltiramiz:

```
Long fact(int k) {
if (k==0) return 1;
return k*fact(k-1);
}
```

Nazorat savollari:

1. Asosiy tiplarni ko‘rsating.
2. Ko‘rsatkich ta’rifini keltiring.
3. Ko‘rsatkichlar bilan bog‘liq amallarni keltiring.
4. Ilova ko‘rsatkichdan qanday farq qiladi?
5. Funksiyalarni qo‘shimcha yuklash qanday amalga oshiriladi?
6. Rekursiv funksiya deb nimaga aytiladi?
7. Inlayn funksiyalar qanday maqsadda ishlatiladi?
8. Prototip nima ma’noni bildiradi?

2 BOB. MURAKKAB TIPLAR

2.1. Sanovchi tip

Agar har xil qiymatlarga ega bir nechta nomlangan konstantani kiritish kerak bo'lsa sanovchi tipdan foydalanish mumkin:

enum <tip_nomi > {<konstantalar ro'yxati >};

Konstantalar butun bo'lishi kerak va initsializatsiya qilinishi mumkin, agar initsializator mavjud bo'lmasa birinchi konstanta nol qiymat oladi, qolganlari bo'lsa oldingisidan birga ortiq bo'ladi.

Misol uchun:

```
enum {
        one=1, two=2, three=3;
        };
```

Agar son qiymatlari ko'rsatilmagan bo'lsa eng chapki so'zga 0 qiymati berilib qolganlariga tartib bo'yicha o'suvchi sonlar mos qo'yiladi:

```
enum {
        zero, one, two;
        };
```

Bu misolda avtomatik ravishda konstantalar quyidagi qiymatlarni qabul qiladi:

zero=0, one=1, two=2;

Konstantalar aralash ko'rinishda kiritilishi ham mumkin:

```
enum {
zero, one, for=4, five, seeks;
 }.
```

Bu misolda avtomatik ravishda konstantalar quyidagi qiymatlarni qabul qiladi:

zero=0, one=1, for=4; five=5,seeks=6;

Nomlangan sanovchi tip kiritib, shu tipdagi o'zgaruvchilar, ko'rsatkichlar va ilovalardan foydalanish mumkin. Masalan:

```
enum color (black, green, yellow, blue, red, white);
color col=red;
```

```
color* cp=&col;
if (*cp==green)
cout<<"yashil";
```

2.2. Strukturalar

Struktura – bu ma'lumotlarni bir butun nomlangan elementlar to'plamiga birlashtirishdir. Struktura elementlari (maydonlar) har xil tipda bo'lishi mumkin va ular har xil nomlarga ega bo'lishi kerak.

Strukturali tip quyidagicha aniqlanadi:

```
struct {
        <ta'riflar ro'yxati >
        }
```

Strukturada albatta bitta komponenta bo'lishi kerak. Struktura tipidagi o'zgaruvchi quyidagicha ta'riflanadi:

<struktura_nomi > <o'zgaruvchi>;

Struktura tipidagi o'zgaruvchi ta'riflanganda initsializatsiya qilinishi mumkin:

<struktura_nomi > <o'zgaruvchi>=<initsializator>;

Strukturani initsializatsiyalash uchun uning elementlarining qiymatlari figurali qavslarda tavsiflanadi.

Misollar:

```
1. struct Student {
 char name[20];
int kurs;
float rating; };
Student s={"Qurbonov",1,3.5};
2. struct {
char name[20];
char title[30];
float rate;
}employee={"Ashurov", "direktor",10000};
```

Strukturalarni o'zlashtirish. Bitta tuzilma tipdagi o'zgaruvchilar uchun o'zlashtirish operatsiyasi aniqlangan. Bunda har bir elementdan nusxa olinadi. Masalan:

Student ss=s;

Struktura elementlariga murojaat. Struktura elementlariga murojaat qilingan ismlar yordamida bajariladi:

<Struktura_nomi>.<element_nomi>

Masalan:

employee.name – «Ashirova» satriga ko'rsatkich;

employee.rate – 10000 qiymatga ega bo'lgan butun tipdagi o'zgaruvchi

Strukturaga ko'rsatkichlar. Strukturaga ko'rsatkichlar oddiy ko'rsatkichlar kabi tasvirlanadi:

Student *ps;

Strukturaga ko'rsatkich ta'riflanganda initsializatsiya qilinishi mumkin:

Student *ps=&mas[0];

Ko'rsatkich orqali struktura elementlariga ikki usulda murojaat qilish mumkin. Birinchi usul adres bo'yicha qiymat olish amaliga asoslangan bo'lib, u quyidagi shaklda qo'llaniladi:

(*strukturaga ko'rsatkich).element nomi;

Ikkinchi usul maxsus strelka (->) amaliga asoslangan bo'lib quyidagi ko'rinishga ega:

strukturaga ko'rsatkich->element nomi

Struktura elementlariga quyidagi murojaatlar o'zaro tengdir:

cin>>(*ps).name;

cin>>ps->title;

2.3. Bitli maydonlar

Bitli maydonlar strukturalarning xususiy xolidir. Bitli maydon ta'riflanganda uning uzunligi bitlarda ko'rsatiladi (butun musbat konstanta).

Misol:

```
struct {
int a:10;
int b:14}xx, *pxx;
. . . .
xx.a=1;
pxx=&xx;
pxx->b=8;
```

Bitli maydonlar ixtiyoriy butun tipga tegishli bo'lishi mumkin. Bitli maydonlar adresini olish mumkin emas. Xotirada bitli maydonlarni joylashtirish kompilyator va apparaturaga bog'liq.

2.4. Birlashmalar

Strukturalarga yaqin tushuncha bu birlashma tushunchasidir. Birlashmalar *union* xizmatchi so'zi yordamida kiritiladi. Misol uchun:

```
union {
 long h;
 int i,j;
 char c[4]}UNI;
```

Birlashmalarning asosiy xususiyati shundaki uning hamma elementlari bir xil boshlang'ich adresga ega bo'ladi.

Masalan:

```
union {
char s[10];
int x;
}u1;
```

Quyidagi dastur yordamida bu xususiyatni tekshirish mumkin:

```
enum paytype{CARD, CHECK};
struct {
paytype ptype
union {
 char card[25];
 long check;      };
}info;
```

```
switch (info.ptype){
case CARD:cout<<"\nKarta bilan to'lash:"<<info.card; break;
case CHECK:cout<<"\nChek bilan to'lash:"<<info.check; break;
}
```

2.5. Massivlar va satrlar

Massivlarni ta'riflash. Massiv indeksli o'zgaruvchidir.

Massivning sodda ta'rifi:

<tip> <o'zgaruvchi_nomi>>[<konstanta_ifoda>] = <initsializator>;

Massivning indekslar qiymati har doim 0 dan boshlanadi.

Ko'p o'lchovli massiv initsializatsiya qilinganda massivning birinchi indeksi chegarasi ko'rsatilishi shart emas, lekin qolgan indekslar chegaralari ko'rsatilishi shart.

Misol uchun:

```
int a[6];
float b[8],c[100];
double d[] = {1, 2, 3, 4, 5};
int A [20][10];
int A [30][20][10];
int A [3][3] = {0,1,2,3,4,5,6,7,8,9,10,11};
int A[ ][3] = { {0,1,100}, {200,210,300}, {1000, 2000, 2100}};
```

Satrlar. Satrli konstanta ikkilik qavslarga olingan simvollar ketma ketligidir. Satrli konstanta oxiriga avtomatik ravishda satr ko'chirish '\n' simvoli qo'shiladi.

Satr qiymati simvolli konstanta bo'lgan simvolli massiv sifatida ta'riflanadi.

Misol uchun:

```
Char capital[]="TASHKENT";
Char capital[]={'T','A','S','H','K','E','N','T','\n'};
char A[ ][9] = { "Tashkent", "Samarkand", "Xiva"};
```

Massivlar va satrlar funksiya parametrlari sifatida. Funksiyalarda massivlar argument sifatida ishlatilganda ularning birinchi indeksi chegarasini ko'rsatish shart emas, qolganlarini chegarasini ko'rsatish

shart. Massivlar ilova bo'yicha uzatiladi, ya'ni ularning qiymati funksiyada o'zgarishi mumkin.

Misol:

```
int sum (int n, int a[] ) {
int i, int s=0;
for( i=0; i<n; i++ )
s+=a[i];
return s; }
```

Satrlar parametrlar sifatida char[] tipidagi bir o'lchovli massivlar sifatida uzatilishi mumkin. Bu holda satr uzunligini aniq ko'rsatish shart emas. Misol:

```
int strlen ( char a[]) {
int i=0;
while(a[i++]);
return i;
}
```

2.6. Dinamik massivlar

O'zgaruvchi o'lchamli massivlarni shakllantirish ko'rsatkichlar va xotirani dinamik taqsimlash vositalari yordamida tashkil etiladi.

Xotirani dinamik taqsimlash uchun *new* va *delete* operatsiyalaridan foydalaniladi. Operatsiya

new <tip_nomi> (<initsializator>)

tip ismi orqali belgilangan ma'lumotlar tipiga mos keluvchi o'lchamli bo'sh xotira qismini ajratish va unga murojaat etish imkonini beradi. Ajratilgan xotira qismiga initsializator orqali aniqlangan qiymat kiritiladi. Xotira ajratilsa xotira ajratilgan qismining bo'sh adresi qaytariladi, agarda xotira ajratilmasa *NULL* qaytariladi.

New operatsiyasi orqali oldindan ajratilgan xotira qismi *delete* operatsiyasi yordamida bo'shatiladi. Misollar:

```
int *i;
i=new int(10);
delete i;
```

O‘zgaruvchilar massiviga xotira ajratishga imkon beradi.

Misollar:

```
int *mas=new[5];
delete [] mas;
```

Skalyar o‘zgaruvchilarga xotira ajratilish 1 – misolda ko‘rsatilgan.

Matritsani shakllantirishdan oldin bir o‘lchovli massivlarga ko‘rsatuvchi ko‘rsatkich massivlar uchun xotira ajratadi, keyin esa parametrli siklda bir o‘lchovli massivlarga xotira ajratadi. Misol:

```
int n,m;
cin>>n;
matr=new int*[n];
for (i=0;i<n;i++) {
cin>>m;
matr[i]=new int[m];
```

Xotirani bo‘shatish uchun bir o‘lchovli massivlarni bo‘shattiruvchi siklni bajarish zarur.

```
for (int i=0;i<n;i++)
delete matr[i];
```

keyin esa matr ko‘rsatkich ko‘rsatgan xotira bo‘shattiriladi.

```
delete [] matr;
```

2.7. Satr murakkab tip sifatida

String tipi. Satrlar bilan ishlash uchun standart bibliotekaga kiruvchi *string* murakkab turidan foydalanish qulaydir.

Bu tipdan foydalanish uchun quyidagi sarlavxali faylni ulash lozim:

```
#include <string.h>
```

Satrlarni ta’riflashga misollar:

```
string st( "BAXO \n" ); // simvollar satri bilan initsiallash
string st2; // bo‘sh satr
string st3( st ); // shu tipdagi o‘zgaruvchi bilan initsiallash
```

Satrlar ustida amallar. Satrlar ustida quyidagi amallar aniqlangan:

- qiymat berish (=);
- ikki amal ekvivalentligini tekshirish uchun (==) va (!=);

• konkatenatsiya yoki satrlarni ulash (+);
• qiymat berib qo'shish amali (+=)
• indeks olish ([]).

Usullar. Satr uzunligini aniqlash uchun *size()* funksiyasidan foydalaniladi(uzunlik tugallovchi simvolni hisobga olmaydi).

```
cout << "uzunlik "<< st << ": " << st.size();
```

Maxsus *empty()* usuli agar satr bo'sh bo'lsa *true* qaytaradi, aks holda *false* qaytaradi:

```
if ( st.empty() ) // to'g'ri: bo'sh
```

Nazorat savollari:

1. Sanovchi tip nima uchun qo'llaniladi?
2. Struktura elementlariga qanday murojaat qilinadi?
3. Bitli maydonlar qaysi tipga tegishli?
4. Birlashmalar asosiy xossalarini ko'rsating.
5. Amallar *new* va *delete* nima uchun ishlatiladi?
6. Dinamik massivlarni hosil qilish usullarini ko'rsating.
7. Satr simvolli massivdan qanday farq qiladi?
8. Massivlarni initsializatsiya qilish usullarini ko'rsating.
9. Satrlarni initsializatsiya qilish usullarini ko'rsating.
10. Qanday qilib massivlar formal parametrlar sifatida ishlatilishi mumkin?

3 BOB. ZAMONAVIY DASTURLASH TILLARIDA ASOSIY KONSTRUKSIYALARDAN FOYDALANISH XUSUSIYATLARI

3.1. Delphi tilida ma'lumotlar turlari

Ma'lumotlar turlarini Delphi tilida umumiy holda ikkiga ajratish mumkin:

- standart turlar. Bu turlar oldindan Delphi tili tomonidan aniqlangan bo'ladi;
- dasturchi tomonidan kiritiladigan (aniqlanadigan) turlar.

Standart turlar tarkibiga quyidagilar kiradi: butun, haqiqiy, belgili (simvol), qator (satr), mantiqiy, ko'rsatkichli va tanlash.

Dasturchi turlarni dasturning var bo'limida o'zgaruvchilarni tavsiflashda aniqlaydi yoki maxsus turlarni aniqlash uchun bo'lim bo'lgan - turlarni tavsiflash *type* bo'limida aniqlaydi.

Bu bo'lim umumiy holda quyidagicha bo'ladi.

```
Type
    <tur nomi>=<turning tavsifi>;
```

Misol:

```
Type
    TColor=(Red,Blue,Black);
  Var Color1,Color2,Color3: TColor;
```

Type bo'limida dasturchi tomonidan yangi TColor nomli tur kiritilmoqda va u Red, Blue, Black mumkin bo'lgan qiymatlarni qabul qilishi mumkin.

Var bo'limida dasturchi tomonidan turi aniqlangan uchta Color1, Color2, Color3 o'zgaruvchilar tavsiflanmoqda.

Bu o'zgaruvchilarni to'g'ridan to'g'ri quyidagicha ham tavsiflash mumkin.

```
Var Color1, Color2, Color3: (Red, Blue, Black);
```

Standart turlarni *type* bo'limida tavsiflash shart emas, ularni to'g'ridan to'g'ri *var* bo'limida tavsiflash mumkin.

Butun turlar. Butun turlar butun sonlarni tasvirlash uchun ishlatiladi. Jadvalda Delphi 7 da ishlatiladigan butun turlar ro'yxati keltirilgan.

Tur	O'zgarish diapazoni	O'lcham (baytda)
Integer	-2147483648..2147483647	4
Cardinal	0..4294967295	4
Shortint	-128..127	1
Smallint	-32768..32767	2
Longint	-2147483648..2147483647	4
Int64	$-2^{63}..2^{63}-1$	8
Byte	0..255	1
Word	0..65535	2
LongWord	0..4294967295	4

Haqiqiy turlar. Haqiqiy turlar haqiqiy sonlarni tasvirlash uchun ishlatiladi. Jadvalda Delphi 7 da ishlatiladigan haqiqiy turlar ro'yxati keltirilgan.

Tur	O'zgarish diapazoni	O'lcham (baytda)
Real	$5.0*10^{-324}..1.7*10^{308}$	8
Real48	$2.9*10^{-39}..1.7*10^{38}$	6
Single	$1.5*10^{-45}..3.4*10^{38}$	4
Double	$5.0*10^{-324}..1.7*10^{308}$	8
Extended	$3.6*10^{-4951}..1.1*10^{4932}$	10
Comp	$-2^{63}+1..2^{63}-1$	8

Belgili turlar. Ma'lumotlarning belgili turlari faqat bitta belgini saqlash uchun xizmat qiladi. Jadvalda Delphi 7 da ishlatiladigan belgili turlar ro'yxati keltirilgan.

Tur	O'lcham (baytda)
Char	1
ANSChar	1
WideChar	2

Mantiqiy turlar. Mantiqiy turlar chin (*true*) yoki yolg'on (*false*) qiymatning birini qabul qiladi. Jadvalda Delphi 7 da ishlatiladigan mantiqiy turlar ro'yxati keltirilgan.

Tur	O'lcham (baytda)
Boolean	1
ByteBool	1
WordBool	2
LongBool	4

O'zgartirish funksiyalari

Funksiya	Qiymati
Chr(n)	Kodi n ga teng simvol
IntToStr(k)	Butun k ni tasvirlovchi satr
FloatToStr (n)	Haqiqiy n ni tasvirlovchi satr
FloatToStrF(n, f , k, m)	Haqiqiy n ni tasvirlovchi satr. Bunda: f - format; k - aniqlik; m - kasr qismidagi raqamlar soni
StrToInt (s)	Satrni butun songa o'tkazish
StrToFloat (s)	Satrni haqiqiy songa o'tkazish
Round (n)	Haqiqiy sonni yaxlitlash
Trunc (n)	Haqiqiy son kasr qismini olib tashlash
Frac(n)	Kasrli sonning kasr qismi
Int (n)	Kasr sonning butun qismi

3.2. Dasturchi tomonidan kiritiluvchi turlar

Delphi tili dasturchiga o'zining turlarini kiritishga imkon beradi. Bu turlar standart turlarga yoki avval kiritilgan turlarga asoslangan bo'lib quyidagi turlarga tegishli bo'lishi mumkin:

- sanovchi;
- interval;
- murakkab tur (yozuv).

Sanoqli turlar. Sanoqli turlar tartiblangan qiymatlar to'plamini ishlatadi.

Tur =(1 qiymat, 2 qiymat, ... , I qiymat)

Masalan:

```
Type
  Color=(black, green, yellow, blue, red, white);
  Fam=(Petrov, Sidorov, Raximov, Sobirov);
  DayOfWeek=(mon, tue, wed, thu, fri, sat, sun);
```

Bu yerda

- Color sanoq turi beshta ranglar ketma ketligini aniqlaydi.
- Fam sanoq turi to'rtta familiyani aniqlaydi.
- DayOfWeek sanoq turi hafta nomlarini aniqlaydi.

Odatda Delphi tilida turlar nomlari T harfidan boshlanadi (*Type* — tip so'zidan).

Yangi tur ta'riflangandan so'ng shu turga tegishli o'zgaruvchini ta'riflash mumkin.

Masalan:

```
type
TDayOfWeek = (MON, TUE, WED, THU, FRI, SAT, SUN) ;
var
ThisDay, LastDay: TDayOfWeek;
```

Sanovchi tur ta'rifi qiymatlar o'zaro munosabatini ko'rsatadi. Eng chap element minimal, eng o'ng element maksimal hisoblanadi. Yuqorida kiritilgan DayOfWeek turi elementlari uchun quyidagi munosabat o'rinli:

MON < TUE < WED < THU < FRI < SAT < SUN

Interval (diapazon) turi. Interval (diapazon) turi beriladigan qiymatga chegara qo'yadi.

```
Type
    <tur nomi>=<minimal>..<maksimal>;
```

Masalan:

```
Type
 Color=red..green; // rangga chegara
 Digit=0..9; // butun sonlarga chegara
 Symb='A'..'Z'; // harflarga chegara
```

Haqiqiy turlarga chegara qo'yilmaydi. Interval tur ta'rifida nomlangan konstantalardan foydalanish mumkin. Quyidagi misolda interval tur TIndex ta'rifida HBOUND nomlangan konstantadan foydalanilgan:

```
const
HBOUND=100;
type
      TIndex=l..HBOUND;
```

Interval turdan massivlarni ta'riflashda qulaydir:

```
type
      TIndex =1 .. 100;
var
      tab1 : array[TIndex] of integer; i:TIndex;
```

Butun son turidan tashqari asos tur sifatida sanovchi turdan foydalanish mumkin. Quyidagi dastur qismida TMonth sanovchi tur asosida interval tur TSammer ta'riflangan:

```
type
    TMonth = (Jan, Feb, Mar, Apr, May, Jun,
    Jul, Aug, Sep, Oct, Nov, Dec);
    TSammer = Jun.. Aug;
```

Yozuv. Dasturlash amaliyotida standart ma'lumotlardan tashkil topgan murakkab ma'lumotlar bilan ishlashga to'g'ri keladi. Misol uchun talaba to'g'risidagi ma'lumot ismi sharifi, tug'ilgan yili, adresi, kursi, guruhi va hokazolardan iborat bo'lishi mumkin. Bunday ma'lumotlarni ta'riflash uchun Delphi da yozuv (*record*) lardan foydalaniladi.

Yozuv bu – alohida nomlangan har xil turli komponentalardan iborat murakkab turdir.

Har qanday tur kabi, "yozuv" *type* bo'limida ta'riflanishi lozim. Bu ta'rif umumiy ko'rinishi:

```
Nom = record
1_ Maydon: 1_Tip; 2_ Maydon: 2_Tip;...; K_ Maydon: K_Tip; end;
```

Ta'riflarga misollar:

```
type
    TPerson = record
    f_name: string[20];
    l_name: string[20];
day: integer;
month: integer;
year: integer;
address: string[50]; end;
TDate = record
    day: integer; month: integer; year: integer;
end;
```

Yozuv turidagi o'zgaruvchini quyidagicha ta'riflash mumkin:

```
var
    student : TPerson; birthday : TDate;
```

Yozuv elementiga (maydoniga) murojaat qilish uchun yozuv nomi va nuqtadan so'ng maydon nomini ko'rsatish kerak. Masalan:

```
Writeln('Ism: ', student.f_name + #13 + 'Adres: ', student.address);
```

Instruksiya ekranga student o'zgaruvchi-yozuvning f_name (nom) va address (adres) maydonlarini chiqaradi.

Ba'zida o'zgaruvchi-yozuv turi o'zgaruvchilar e'lon qilish bo'limida e'lon qilinadi. Bu holda yozuv turi o'zgaruvchi nomidan so'ng ko'rsatiladi. Misol uchun student yozuvi *var* bo'limida quyidagicha ta'riflanishi mumkin:

```
student: record
    f_name:string[20];
    l_name:string[20];
    day:integer;
    month:integer;
```

```
        year:integer;
        address:string[50];
    end;
```

With instruksiyasi. With instruksiyasi dasturda maydonlar nomlarini o'zgaruvchi – yozuv nomini ko'rsatmasdan ishlatishga imkon beradi. Umumiy holda *with* instruksiyasi quyidagi ko'rinishga ega:

```
with nom do
    begin
{ dastur instruksiyasi } end;
```

Misol uchun dasturda quyidagi yozuv ta'riflangan bo'lsin:

```
student: record
    f_name: string[30];
    l_name: string[20];
    address: string[50];
end;
```

va studentlar to'g'risidagi ma'lumotlar E1, E2 va E3 o'zgaruvchilarda joylashgan bo'lsin. U holda

```
student.f_name := El;
student.l_name := E2;
student.address := E3;
```

instruksiyalar o'rniga quyidagi instruksiyani yozish mumkin:

```
with student do begin
f_name := E1; l_name := E2; address := E3;
end;
```

Nazorat savollari:

1. O'zgaruvchilarning qanday turlari mavjud?
2. Mantiqiy o'zgaruvchilar qanday qiymat qabul qiladi?
3. Qanday turlarni o'zgartirish funksiyalari mavjud?
4. Delphi tilida dastur qanday strukturaga ega?
5. With instruksiyasi dastur tuzishda qanday imkoniyat yaratadi?
6. Yozuv qanday tur?
7. Dasturchi tomonidan kiritiluvchi turlar qanday turlar?

4 BOB. SINFLAR VA OB'YEKTLAR

4.1. Sinf

Sintaksis bo'yicha, C++ da sinf – bu mavjud bo'lgan tiplar asosida yangi yaratilgan strukturalangan tipdir.

Sinf ta'rifi sodda shakli:

<sinf_tipi> <sinf_nomi>{<sinf_komponentlari_ro'yxati>};

bu yerda:

sinf_tipi – class, struct, union xizmatchi so'zlaridan biri;

sinf_nomi – identifikator;

sinf_komponentlari_ro'yxati – sinfga tegishli ma'lumotlar va funksiyalar ta'rifi.

Funksiya – bu ob'yektlar ustida bajariladigan operatsiyalarni aniqlovchi sinf usuli.

Ma'lumotlar – bu ob'yekt strukturasini hosil qiluvchi maydon.

Usullar sinfdan tashqarida aniqlanganda ularning nomlarini kvalifikatsiya qilish (ixtisoslashtirish) kerak. Usulning ko'rimlilik soxasini aniqlaydigan uning bunday kvalifikatsiya sintaksisi quyidagi ko'rinishga ega:

<sinf nomi>::<usul nomi>

Sinf ichida aniqlangan usullar ko'zda tutilgan bo'yicha joylashtiriluvchi (inline) funksiya hisoblanadi. Sinf tashqarisida aniqlangan usullarni oshkor ravishda joylashtiriluvchi deb ta'riflanishi lozim.

Sinf ob'yekti (sinf nusxasi) ni ta'riflash uchun quyidagi konstruksiyadan foydalaniladi:

<sinf_nomi> <ob'yekt_nomi>;

Ob'yekt orqali maydonlarga va usullarga quyidagicha murojaat qilish mumkin:

< ob'yekt_nomi >. <maydon_nomi>

< ob'yekt_nomi >. <usul_nomi>

4.2. Komponentalarga murojaat huquqlari

Komponentalarga murojaat huquqi murojaat spetsifikatorlari yordamida boshqariladi: *public*, *private*, *protected*.

Umumiy (*public*) komponentalar dasturni ixtiyoriy qismida murojaat huquqiga ega. Ulardan, ixtiyoriy funksiya ushbu sinf ichida va sinf tashqarisida foydalansa ham bo'ladi.

Xususiy (*private*) komponentalar sinf ichida murojaat huquqiga ega, lekin sinf tashqarisidan esa murojaat qilish mumkin emas. Komponentalardan ular tavsiflangan sinfdagi funksiya - a'zolari yoki "do'stona"- funksiyalar orqali foydalanish mumkin.

Himoyalangan (*protected*) komponentalar sinf ichida va hosila sinflarda murojaat huquqiga ega.

Agar sinf ta'rifida *class* so'zi ishlatilgan bo'lsa hamma komponentalari xususiy hisoblanadi, agar *struct* so'zi ishlatilgan bo'lsa hamma komponentalar umumiy hisoblanadi.

4.3. Konstruktor

Konstruktor – bu sinf ob'yektlarini avtomatik initsializatsiya qilish uchun ishlatiladigan maxsus komponentali funksiya.

Konstruktorlar ko'rinishi quyidagicha bo'lishi mumkin:

<Sinf nomi> (<formal parametrlar ro'yxati>)

{<konstruktor tanasi>}

Bu komponenta funksiya nomi sinf nomi bilan bir xil bo'lishi lozim.

Dasturchi tomonidan ko'rsatilmagan holda ham *new* operator yordamida sinf ob'yekti yaratilganda yoki xotirada joylashganda konstruktor avtomatik ravishda chaqiriladi.

Konstruktor ob'yekt uchun xotirada joy ajratadi va ma'lumotlar – sinf a'zolarini initsializatsiyalaydi.

Konstruktor bir nechta xususiyatlarga ega:

- Konstruktorlar uchun qaytariluvchi tiplar, hatto *void* tipi ham ko'rsatilmaydi.

- Konstruktor adresini hisoblash mumkin emas. Konstruktor parametri sifatida o'z sinfining nomini ishlatish mumkin emas, lekin bu nomga ko'rsatkichdan foydalanish mumkin.
- Konstruktorlar vorislikga ega emas.

Konstruktorlar ixtiyoriy sinflar uchun doimo mavjud, lekin agarda u ko'rsatilgan holda tavsiflanmagan bo'lsa u avtomatik ravishda yaratiladi. Ko'rsatilmagan holda parametrsiz konstruktor va nusxa konstruktori yaratiladi. Agarda konstruktor ochiq holda tavsiflangan bo'lsa, unda ko'rsatilmagan holda konstruktor yaratilmaydi. Ko'rsatilmagan holda umumiy (*public*) konstruktorlar yaratiladi.

Konstruktorni oddiy komponenta funksiya sifatida chaqirib bo'lmaydi. Konstruktorni ikki xil shaklda chaqirish mumkin:

Birinchi shakl ishlatilganda haqiqiy parametrlar ro'yxati bo'sh bo'lmasligi lozim. Bu shakldan yangi ob'yekt ta'riflanganda foydalaniladi:

Konstruktorni ikkinchi shaklda chaqirish nomsiz ob'yekt yaratilishiga olib keladi. Bu nomsiz ob'yektdan ifodalarda foydalanish mumkin.

Misol:

```
include<iostream.h>
class complex {
 double re, im;  // private ko'zda tutilgan bo'yicha
 public:
 void show();
 complex(double re1 = 0.0, double im1 = 0.0){re = re1; im = im1;}
};
 inline void complex::show() { cout << "re=" << re<<"im=" << im;}
 void main() {
 complex ss (5.9,0.15);
 complex aa = complex (5.9,0.15);
  aa.show();
  }
```

Konstruktor yordamida ob'yekt ma'lumotlarni initsializatsiyalashni ikkita usuli mavjud:

Birinchi usulda parametrlar qiymatlari konstruktor tanasiga uzatiladi. Ikkinchi usulda esa ushbu sinfdagi initsializatorlar ro'yxatidan foydalanish nazarda tutilgan:

<nom> (<ifoda>)

Misol:

```
class A {
  int i; float e; char c;
public:
  A(int ii, float ee, char cc) : i(8), e( i * ee + ii ),s(ss){}
...
};
```

4.4. Ob'yektlar massivi

Ob'yektlar massivi ta'riflash uchun sinf ko'zda tutilgan (parametrsiz) konstruktorga ega bo'lishi kerak.

Ob'yektlar massivi ko'zda tutilgan konstruktor tomonidan yoki har bir element uchun konstruktor chaqirish yo'li bilan initsializatsiya qilinishi mumkin.

```
class complex a[20]; // parametrsiz konstruktorni chaqirish
class complex  b[2]={complex (10),complex (100)};//oshkor chaqirish
```

4.5. Nusxa olish konstruktori

Ko'rsatilmagan holda T::T(const T&) ko'rinishidagi nusxa konstruktori yaratiladi, bu yerda T – sinf ismi. Har gal sinfga tegishli ob'yektlarning nusxasi olinayotganda nusxa konstruktori chaqiriladi. Xususan:

a) ob'yekt funksiyaga qiymati bo'yicha uzatilsa;

b) funksiya qiymatlarini qaytaruvchi vaqtinchalik ob'yektlarni yaratishda;

v) boshqa ob'yektni initsializatsiyalash uchun ob'yektdan foydalanishda.

Agarda sinfda aniq bir nusxalash konstruktori ochiq ko'rsatilgan bo'lmasa, unda yuqorida aytib o'tilgan uch holatdan bittasi mavjud bo'lsa ob'yektni nusxalash bit bo'yicha amalga oshiriladi. Lekin bit bo'yicha nusxalash har doim ham adekvat deb hisoblanmaydi. Xuddi shu hollarda xususiy nusxalash konstruktorini belgilamoq lozim.

4.6. Destruktor

Sinfning biror ob'yekti uchun ajratilgan xotira ob'yekt yo'qotilgandan so'ng bo'shatilishi lozimdir. Sinflarning maxsus komponentalari destruktorlar bu vazifani avtomatik bajarish imkonini yaratadi. Destruktorni standart shakli quyidagicha:

~ <sinf_nomi> ( ) {<destruktor tanasi>}

Destruktor parametr yoki qaytariluvchi qiymatga ega bo'lishi mumkin emas (xatto *void* tipidagi). Dastur ob'yektni o'chirganda destruktor avtomatik chaqiriladi.

Agar sinfda destruktor ochiq ko'rsatilmagan bo'lsa, unda kompilyator ko'rsatilgan ob'yekt egallagan xotirani bo'shatuvchi destruktorni generatsiyalaydi. Boshqa ob'yektlar egallagan xotirani bo'shatmoqchi bo'lsak, destruktorni ochiq aniqlashimiz lozim.

"Simvol satri" sinfi.

```
class string {
char *ch;
 // matnli satrga ko'rsatkich
  int len;
  // matnli satrni uzunligi
public:
  // konstruktorlar
  // bo'sh satr – ob'yektni yaratish
  string(int N = 80): len(0){
ch = new char[N+1];
ch[0] = '\0';}
  // berilgan satr bo'yicha ob'yekt yaratadi
  string(const char *arch){
```

```
len = strlen(arch);
ch = new char[len+1];
strcpy(ch,arch); }
 // funksiya – komponentalar
 // murojaatni satr uzunligiga qaytaradi
 int& len_str(void){
return len;
}
 // ko'rsatkichni satrga qaytaradi
 char *str(void){
return ch;
}
//Nusxa olish konstruktori
string(const string& st){
len=strlen(st.len);
ch=new char[len+1];
strcpy(ch,st.ch);
}
//Destruktor
~string(){
delete []ch;
        }
. . .};
```

4.7. Ob'yektlarda ko'rsatkichlar

Ob'yektlar aniqlangandan so'ng shu ob'yektlarga ko'rsatkichlar belgilash mumkin. Masalan:

```
complex A(5.2,2.7);
complex* PA=&A;
```

Ob'yektning umumiy elementlariga murojaat uchun -> operatsiyani yoki ism almashtirish va nuqta operatsiyasidan foydalanish mumkin.

```
*PA.real() yoki PA->real;
```

4.8. This ko'rsatkichi

Agarda konkret ob'yektga ishlov berish uchun sinf a'zosi – funksiya chaqirilsa, unda shu funksiyaga ob'yektga belgilangan ko'rsatkich avtomatik va ko'rsatilmagan holda uzatiladi. Bu ko'rsatkich *this* ismiga ega va x* *this* kabi har bir funksiya uchun ko'rsatilmagan holda belgilanadi.

X sinfni ekvivalent ko'rinishda shunday tavsiflash mumkin:

```
class x {
int m;
public:
int readm() {
return this->m; }
};
```

A'zolarga murojaat etishda *this* dan foydalanish ortiqcha. Asosan *this* bevosita ko'rsatkichlar bilan manipulyatsiya qilish uchun a'zo funksiyalarini yaratilishida foydalaniladi.

4.9. Sinfning statik komponentalari

Sinf kompanentasi yagona bo'lib hamma yaratilgan ob'yektlar uchun umumiy bo'lishi uni statik element sifatida ta'riflash ya'ni *static* atributi orqali ta'riflash lozimdir. Ob'yektlarni yaratishda sinf statik ma'lumotlari takrorlanmaydi, ya'ni har bir statik komponentalar birdan-bir ko'rinishda mavjud. Statik elementlarga murojaat qilish uchun oldin initsializatsiya qilinishi lozim. Initsializatsiya quyidagicha amalga oshiriladi:

<Sinf-nomi>:: <kompleks-nomi> <initsializator>

Masalan:

int complex : : count = 0;

Bu taklifni sinfni aniqlashdan so'ng global soxada joylashtirish lozim. Faqatgina sinf statistik ma'lumotlarini initsializatsiyalashda u xotiraga ega bo'ladi va unga murojaat etish mumkin. Sinf statik ma'lumotlarga faqatgina ob'yekt ismi orqali murojaat etish mumkin:

<ob'yekt_nomi>.<komponenta_nomi>

Masalan:

complex a; a.count=5;

Lekin, statik komponetalarga sinf ob'yekti aniqlanmagan holda ham murojaat etishi mumkin. Statik komponentalarga nafaqat ob'yekt ismi, balki sinf ismi orqali ham murojaat etish mumkin.

<sinf_nomi> : : <komponenta_nomi>

Masalan

Complex :: count=5;

Lekin shunday murojaat faqatgina *public* komponentalarga tegishli.

Private statik komponentalarga tashqaridan murojaat etishda funksiya – statik komponentalardan foydalaniladi.

Bu funksiyalarni sinf ismi orqali chaqirish mumkin.

<sinf_nomi> : : <statik_funksiya_nomi>

Misol.

```
#include <iostream.h>
class TPoint{
 double x,y;
 static int N;
// statik maydon: nuqtalar soni
  public:
 TPoint(double x1 = 0.0, double y1 = 0.0){
N++; x = x1; y = y1;
}
 static int& count(){return N;}
// statik komponenta-funksiya
};
int TPoint : : N = 0;
//statik maydon initsializatsiyasi
void main(void) {
 TPoint A(1.0,2.0);
 TPoint B(4.0,5.0);
 TPoint C(7.0,8.0);
 cout<< \nAniqlangan"<<TPoint : : count()<<"nuqtalar";
}
```

Nazorat savollari:

1. C++tilida *class*, *struct* va *union* orasida qanday farq bor?
2. Sinf usullarini qo'shimcha yuklash mumkinmi?
3. Konstruktorlar va destruktorlar vazifasini ko'rsating.
4. Ob'yektlar massivi yaratilganda qanday konstruktorlar chaqiriladi?
5. Destruktorlar qanday chaqiriladi?
6. Statik komponentalar xususiy bo'lishi mumkinmi?

5 BOB. SINFLAR VA OB'YEKTLAR BILAN ISHLASH XUSUSIYATLARI

5.1. Sinflar bilan ishlash xususiyatlari

Sinf – bu maxsus turlar bo'lib, o'zida maydon, usullar va xossalarni mujassamlashtiradi.

Sinf murakkab struktura bo'lib, ma'lumotlar ta'riflaridan tashqari, protsedura va funksiyalar ta'riflarini o'z ichiga oladi.

Sodda sinf ta'rifiga misol:

```
TPerson = class
private
    fname: string[15];
    faddress: string[35];
public
      procedure Show;
end;
```

Sinf ma'lumotlari maydonlar, protsedura va funksiya usullarI deb ataladi. Keltirilgan misolda TPerson – sinf nomi, fname va faddress – maydonlar nomlari, show – usul nomi.

Maydon – bu sinfga birlashtirilgan ma'lumotlardir. Sinfga qarashli maydonlar oddiy yozuv maydoni kabi bo'lib, ularning farqi har xil turda bo'lishidir. Masalan,

```
Type
  TchildClass=Class
     Fore: Integer;
     Ftwo: String;
     Fthree: Tobject;
End;
```

Maydonlarga murojaat qilish sinf xossalari va usullari yordamida amalga oshiriladi. Maydonga murojaat qilish uchun oldin sinf nomi yozilib, keyin ajratuvchi nuqta qo'yilib maydon nomi yoziladi. Masalan:

```
Var
  MyObject: TchildClass;
```

```
Begin
   MyObject.Fone:=16;
   MyObject.Ftwo:='qator qiymati';
End;
```

Maydon nomi unga mos xossa nomining birinchi harfi "F" bo'lishi bilan farqlanadi.

Delphi da qabul qilingan kelishuv bo'yicha maydonlar nomlari f (field — maydon so'zidan) harfidan boshlanishi lozim.

5.2. Usullar

Sinfga birlashtirilgan protsedura va funksiyalarga usullar deyiladi. Masalan:

```
Type
  TchildClass=Class
     Fore: Integer;
     Ftwo: String;
     Fthree: Tobject;
     Function FirstFunc(x:Real):Real;
     Procedure SecondProc;
End;
```

Sinf usullari (sinf ta'rifiga kiritilgan protsedura va funksiyalar) sinf ob'yektlari ustida amal bajaradi. Usul bajarilishi uchun ob'yekt nomi va nuqtadan so'ng usul nomi ko'rsatilishi lozim. Masalan:

professor. Show;

Sinf usuli ta'riflanganda sinf nomi va usul nomi ko'rsatiladi. Masalan:

```
// TPerson sinfi Show usuli
procedure TPerson.Show;
begin
Write ( 'Nom:' + fname + #13+ 'Adres:' + faddress );
end;
```

Usul tanasida ob'yekt maydonlariga murojaat qilinganda ob'yekt nomi ko'rsatilmaydi.

Usulga murojaat qilish dasturda uning nomini ko'rsatish bilan bajariladi. Masalan:

```
Var
  MyObject: TchildClass;
  y: Real;
  Begin
  . . . . . .
  MyObject. SecondProc;
  y:=MyObject.FirstFunc(3.14);
  End;
```

Usullar chaqirilganda chaqirgan ob'yektga ilova uzatiladi. Bu ilovaga usul ichida *self* so'zi orqali murojaat qilish mumkin.

```
procedure TPerson.Tproc(Fore:Integer);
begin
self.Fore:=Fore;
end;
```

5.3. Ob'yekt

Ob'yekt – bu sinfning real nusxasi bo'lib, ma'lumotlar va funksiyalardan tashkil topadi. U dasturning *var* bo'limida e'lon qilinadi.

Masalan:

```
var
  student: TPerson; professor: TPerson;
```

Ob'yekt – bu dinamik strukturadir. O'zgaruvchi-ob'yekt ma'lumotlarni emas, ob'yekt ma'lumotlariga ilovani o'z ichiga oladi. Shuning uchun dasturchi bu ma'lumotlarga xotiradan joy ajratishni ko'zda tutishi lozim.

Joy ajratish sinf maxsus usuli – konstruktor yordamida amalga oshiriladi. Bu usul odatda *Create* (yaratish) nomiga ega bo'ladi. Sinf ta'rifida konstruktor uchun *procedure* so'zi o'rniga *constructor* so'zi ishlatiladi.

Quyida tarkibida konstruktor qatnashgan TPerson sinfi ta'rifi keltirilgan:

```
TPerson = class private
fname: string [ 15 ];
faddress: string[35];
constructor Create; // konstruktor
public
procedure show; // usul
end;
```

Xotiradan joy ajratish konstruktor sinfga qo'llash natijasini qiymat sifatida berish orqali amalga oshiriladi. Misol uchun

```
professor := TPerson.Create;
```

instruksiyasi bajarilishi natijasida professor ob'yektiga xotiradan joy ajratiladi. Xotiradan joy ajratishdan tashqari konstruktor, odatda ob'yekt maydonlariga boshlang'ich qiymatlar berish ya'ni ob'yekt initsializatsiyasi vazifasini ham bajaradi. Quyida TPerson ob'yekti uchun konstruktor misoli keltirilgan:

```
constructor TPerson.Create;
begin
fname := '';
faddress := '';
end;
```

Ob'yekt maydoniga murojaat qilish uchun ob'yekt nomi va nuqtadan so'ng maydon nomi ko'rsatiladi. Masalan:

```
professor.fname
```

Ob'yektga ajratilgan xotira qismini bo'shatish uchun maxsus usul destruktor *free* ishlatiladi. Masalan:

```
professor.free;
```

5.4. Inkapsulyatsiya va ob'yekt xossalari

Inkapsulyatsiya deyilganda ob'yekt maydonlariga to'g'ridan to'g'ri emas, faqat sinf usullari orqali murojaat qilishga aytiladi.

Ob'yekt maydonlariga murojaat ob'yekt xossalari orqali amalga oshiriladi. Ob'yekt xossasiga murojaat qilish uchun ikki usuldan foydalaniladi. O'qish uchun ishlatiladigan funksiya nomi *Get* bo'lib,

unga mos xossa nomi qo'shilib yoziladi. Yozish uchun ishlatiladigan usul bitta parametrli *Set* nomi qism dastur bo'lib, uning nomiga ham mos xossa nomi qo'shilib yoziladi. O'qish va yozish usullari va uning parametri ham bir xil xossaga ega bo'lishi lozim.

Xossani e'lon qilish uchun *Property*, *Read* va *Write* so'zlari ishlatiladi. *Read* va *Write* usul nomlari bo'lib, ular mos ravishda o'qish va yozish uchun mo'ljallangan.

Quyida ikkita *Name* va *Address* xossalarini o'z ichiga oluvchi TPerson sinfi ta'rifi keltirilgan:

```
type
    TName = string[15]; TAddress = string[35];
    TPerson = class
private
    FName: TName;
    FAddress: TAddress;
Constructor Create(Name:Tname);
Procedure Show;
    Function GetName: TName;
    Function GetAddress: TAddress;
Procedure SetAddress(NewAddress:TAddress);
public
    Property Name: Tname
    read GetName;
    Property Address: TAddress
    read GetAddress
    write SetAddress;
end;
```

Dasturda student ob'yektining, Address xossasiga qiymat berishini quyidagicha yozish mumkin:

```
student.Address := 'Toshkent, Yunusobod 21, kv.3';
```

Tashqi tomondan dasturda xossalardan foydalanish ob'yekt maydonlaridan foydalanishdan farq qilmaydi. Lekin xossalar va ob'yekt xossalari orasida prinsipial farq bor: xossaga qiymat berish yoki xossa

qiymatini o‘qishda avtomatik ravishda biror vazifa bajaruvchi protsedura chaqiriladi.

Dasturda xossa usullariga qo‘shimcha vazifalar yuklash mumkin. Masalan usullar yordamida xossaga berilayotgan qiymatlar to‘g‘riligini tekshirish, yordamchi protseduralarni chaqirish.

Ob’yekt maydonlarini xossa sifatida ta’riflash maydonlarga murojaatni cheklashga imkon beradi: masalan, faqat o‘qishga ruxsat berish mumkin. Buning uchun xossa ta’rifida faqat o‘qish usulini ko‘rsatish kerak.

Yuqorida keltirilgan TPerson sinfi ta’rifida *Name* xossasini faqat o‘qish mumkin, Address — xossasi esa o‘qish va yozish uchun mo‘ljallangan.

Yozuvdan himoyalangan xossa qiymatini ob’yekt initsializatsiyasida o‘rnatish mumkin. Quyida Tperson sinfi usullari keltirilgan. Bu usullar TPerson sinfi ob’yektini yaratish va xossalariga murojaatni ta’minlaydi.

```
Constructor TPerson.Create(Name:TName);
begin
    FName:=Name;
end;
    Function TPerson.GetName;
begin
    Result:=FName;
end;
    function TPerson.GetAddress;
begin
    Result:=FAddress; end;
    Procedure TPerson.SetAddress(NewAddress:TAddress);
begin
    if FAddress =‘ ‘
    then FAddress := NewAddress;
end;
```

Sinf ob’yektini yaratish va xossa qiymatini o‘rnatishga misol:

```
    student := TPerson.Create(‘Tolipov’);
    student.Address := ‘‘Toshkent, Yunusobod 21, kv.3’;
```

Nazorat savollari:

1.Sinf bu nima?
2.Sinf va ob'yekt orasida qanday farq bor?
3.Inkapsulyatsiya nima uchun kerak?
4.Ob'yekt maydonlariga murojaat qanday amalga oshiriladi?
5.Xizmatchi so'z *private* nima uchun ishlatiladi?

6 BOB. SINFLAR ORASIDA MUNOSABATLAR

6.1. Sinflar orasidagi munosabatlar turlari

Har bir sinfni ikkita muhim jihati mavjud: u arxitektura birligini moduli hisoblanadi va u bir necha ma'lumotlar turlarini aniqlagan holda sermazmun tushunchaga ega. Dastur tizimi sinflari hammasi bir-biri bilan o'zaro aniq bog'lanishda bo'ladi.

Ob'yektli dasturlash tizimida ikkita asosiy tur sinflarining o'zaro bog'lanishi aniqlangan. Birinchi bog'lanish "Mijoz va yetkazuvchi", odatda mijoz bog'lanishi deb ataladi yoki ichma-ich bog'lanishda bo'ladi. Ikkinchi bog'lanish "Ota-onalar va vorislar" bu odatda vorislik deb nomlanadi.

Ta'rif 1. A va V sinflar "Mijoz va yetkazuvchi" bog'lanishida bo'lsa, agar V sinfning maydoni A sinf ob'yekti bo'lsa, A sinf V sinfning yetkazuvchisi deb nomlanadi, V sinfi A sinfning mijozi deb ataladi.

Ta'rif 2. A va V sinflar "Ota-onalar va vorislar" bog'lanishida deyiladi, agar V sinf e'lon qilishda A sinf ota sinf sifatida ko'rsatilgan bo'lsa, V sinf A sinfning vorisi deb ataladi.

Ikkala bog'lanish – vorislik va ichma-ich joylashganlik tranzitivlik xossasiga ega. Agar V sinf A sinf mijozi va S sinf V sinfning mijozi bo'lsa S sinf A sinfning mijozi bo'ladi. Agar V sinf A sinf vorisi bo'lsa, S sinf V sinfning vorisi bo'lsa S sinf A sinfning vorisi bo'ladi.

Yuqoridagi 1 va 2 ta'riflar to'g'ridan to'g'ri mijoz va yetkazuvchi, vorislik munosabatlarini aniqlaydi (1-bosqich mijoz,1-bosqich yetkazuvchi va hokazo). Rekursiv tarzida shuni aniqlash mumkinki: to'g'ridan to'g'ri k-chi bosqichdagi mijoz k+1 bosqichdagi mijoz bilan bog'lanishda bo'ladi. Vorislik bog'lanishida, tabiiy tildagi tushunchalar qo'llanadi. To'g'ridan to'g'ri bo'lmagan vorislik ajdodlar va avlodlar bog'lanishi deyiladi.

Odatda sinflarni loyihalashda savol kelib chiqadi, sinflarni o'zaro munosabatini qanday qurish kerak bo'ladi.Ikkita oddiy sinflarga misol ko'ramiz – Square va Rectangle, ular kvadrat va

to‘g‘rito‘rtburchaklardir. Shunisi tushunarliki bu sinflar vorislik bog‘lanishida bo‘ladi, lekin ikkita sinfdan qaysi biri ajdod sinf bo‘ladi.

Yana ikkita sinfga misol – Car va Person, ya’ni mashina va inson. Bu sinflar bilan Person_of_Car ya’ni mashina egasi sinfi qanday aloqada bo‘lishi mumkin? Bu ikki sinf bilan vorislik bog‘lanishida bo‘lishi mumkinmi? Sinflarni loyihalash bilan bog‘liq bu savollarga javob topish uchun shuni nazarda tutish kerakki, “Mijoz-yetkazuvchi” bog‘lanishi “Ega” (“Has”) bog‘lanishini, vorislik bog‘lanishi esa “Bir xil” (“is a”) bog‘lanishi tushunchalarini ifodalaydi. Square va Rectangle sinflari misoli tushunarli, har bir ob’yekt kvadrat to‘g‘rito‘rtburchakdir, shuning uchun bu sinflar o‘rtasida vorislik bog‘lanishi ifodalanadi va Rectangle sinfi ota-onalar sinfini ifodalaydi. Square sinfi uning o‘g‘lidir. Mashina egasi mashinaga ega va insondir. Shuning uchun Person_of_Car sinfi Car sinfining mijozi bo‘lib hisoblanadi va Person sinfining vorisidir.

6.2. Ob’yektlar sinf a’zosi sifatida

Agar sinfda boshqa sinf ob’yektlari mavjud bo‘lsa, unda, sinf a’zosi - konstruktor uchun parametri sinf konstruktori ta’rifida ko‘rsatiladi. A’zosi uchun konstruktor uning uchun parametrlar ro‘yxatini belgilovchi konstruktor bajarilgandan so‘ng chaqiriladi.

Misol:

```
class A{
  int i;
public:
  A(int ii) {i=ii;}
. . . };
```

A sinfni o‘z ichiga qamrab olgan sinf:

```
class B {
  int k;
A member;
public:
  B(int ii, int kk): member(ii){k=kk;}
```

. . . };

Boshqa a'zolar (agarda ular mavjud bo'lsa) konstruktorlari uchun shunday parametrlarni belgilash mumkin:

```
class classdef {
  table members;
  table friends;
  int no_of_members;
  // ...
  classdef(int size);
  ~classdef(); };
```

A'zolar uchun parametrlar ro'yxati bir biridan vergul (ikkita nuqta emas) yordamida ajratiladi, ammo a'zolar uchun initsializatorlar ro'yxatini ixtiyoriy tartibda belgilash mumkin:

```
classdef::classdef(int size)
: friends(size), members(size), no_of_members(size) {
  // ...
}
```

Konstruktorlar sinf tavsifida belgilangan tartibda chaqiriladi.

6.3. Sinf do'stlari ta'rifi

C++ da aniq sinf do'stlariga bu sinfning xususiy elementlariga murojaat etish imkonini beradi. C++ da bitta sinf yoki funksiya ikkinchi sinfga do'stona sinfligini ko'rsatish uchun *friend* kalit so'zidan foydalanish va do'stona sinf ismini ikkinchi sinf tavsifiga kiritishi lozim.

Do'stona funksiya va sinflarni ta'riflash:

```
friend <funksiya prototipi >
friend <sinf nomi >
```

Masalan, quyidagi book sinfi librarian sinfini o'ziga do'stona sinf deb belgilagan:

```
class book {
    char title [64] ;
char author[64];
char catalog[64];
```

```
public:
book (char *, char *, char *);
void show_book(void);
friend librarian;
};
```

Shuning uchun librarian sinf ob'yektlari book sinfning xususiy elementlariga, nuqta operatoridan foydalangan holda, to'g'ridan to'g'ri murojaat etishi mumkin:

```
class librarian {
public:
  void change_catalog(book *, char *);
  char *get_catalog(book);
};
void librarian::change_catalog(book *this_book, char
*new_catalog) {
  strcpy(this_book->catalog, new_catalog);
}
char *librarian: :get__catalog(book this_book) {
  static char catalog[64];
  strcpy(catalog, this_book.catalog);
  return(catalog) ;
}
```

Ko'rib turganimizdek dastur librarian sinfining change_catalog funksiyasiga book ob'yektini adres orqali bermoqda. Bu funksiya sinfning book elementini o'zgartirgani uchun, dastur parametrni adres orqali uzatishi va undan so'ng ushbu sinf elementiga murojaat uchun ko'rsatkich ishlatmog'i lozim. Book sinf aniqlanishidan *friend* operatori o'chirib yuborilsa C++ kompilyatori har gal book sinfi xususiy ma'lumotlariga murojaatda sintaktik xato haqida xabar chiqaradi.

6.4. Do'stona sinflar sonini chegaralash

Agarda bir nechta sinf funksiyalariga boshqa sinfning xususiy ma'lumotlariga murojaat qilish kerak bo'lsa, u holda C++ do'stona

sinfning faqatgina belgilangan funksiyalari xususiy elementlarga murojaat etishiga imkoniyat beradi. Masalan, faqatgina change_catalog va get_catalog funksiyalarga book sinfning xususiy elementlariga murojaat kerak. Quyida ko'rsatilgandek, book sinfining ichida faqatgina shu funksiyalarda xususiy funksiyalarga murojaat chegarasini qo'yishi lozim:

```
class book {
public:
  book(char *, char *, char *);
  void show_book(void);
  friend char *librarian::get_catalog(book);
  friend void librarian: :change_catalog( book *, char *);
private:
  char title[64];
  char author[64];
  char catalog[64];
};
```

Ko'rib turganimizdek *friend* operatorlari xususiy elementlarga murojat qiluvchi hamma do'st funksiyalarini to'liq prototiplarini o'z ichiga oladi.

Agar dastur bir sinfdan boshqasiga murojaat qilsa va sinflar aniqlanish tartibi noto'g'ri bo'lsa sintaktik xatoga duch kelish mumkin. Bizning holda book sinfi librarian sinfida e'lon qilingan funksiyalar prototiplariga murojaat qilmoqda. Shuning uchun librarian sinfi aniqlanishi book sinfi aniqlanishidan oldin kelishi kerak, biroq librarian sinfi book sinfiga murojaat qilmoqda:

```
class librarian {
public:
  void change_catalog(book *, char *);
  char *get_catalog(book); };
```

Book sinfi aniqlanishini librarian sinfi aniqlanishidan oldini qo'yib bo'lmagani uchun C++ book sinfini e'lon qilish imkonini beradi va shu bilan u kompilyatorga bunday sinf borligi haqida xabar beradi va

keyinroq o‘zi ham aniqlanadi. Quyida buni qanday amalga oshirish keltirilgan:

class book; // sinf ni e’lon qilinishi

Nazorat savollari:

1. Qachon A va V sinflari «Mijoz – yetkazuvchi » munosabatida bo‘ladi?
2. Qachon A va V sinflari «Ota - o‘g‘il» munosabatlarida bo‘ladi?
3. "Egalik" va "Bir xillik" munosabatlari qaysi munosabatlar tipiga mansub?
4. Do‘stona funksiya va sinflar aniqlanishi shaklini ko‘rsating.
5. Sinf do‘stlaridan nima uchun foydalaniladi?
6. Do‘stona sinfni e’lon qilish sinfning qaysi seksiyasida ekanligi ahamiyatlimi?

7 BOB. SINFLARDA VORISLIK

7.1. Vorislikda murojaat huquqlarini boshqarish

Vorislik o'zining barcha ajdodlarining xususiyatlari, ma'lumotlari, metodlari va voqealarini meros qilib oladigan hosila sinfini e'lon qilish imkoniyatini beradi, shuningdek yangi tavsiflarni e'lon qilishi hamda meros sifatida olinayotgan ayrim funksiyalarni ortiqcha yuklashi mumkin. Bazaviy sinfning ko'rsatib o'tilgan tavsiflarini meros qilib olib, yangi tug'ilgan sinfni ushbu tavsiflarni kengaytirish, toraytirish, o'zgartirish, yo'q qilish yoki o'zgarishsiz qoldirishga majburlash mumkin.

Hosila sinfni e'lon qilishning umumlashgan sintaksisi:

class <sinf nomi>: [<kirish huquqini beruvchi spetsifikator >] <ajdod sinf nomi> {...}

Sinf o'zining bazaviy sinfidan yuzaga kelayotganida, uning barcha nomlari hosila sinfda avtomatik tarzda yashirin *private* bo'lib qoladi. Ammo uni, bazaviy sinfning quyidagi kirish spetsifikatorlarini ko'rsatgan holda, osongina o'zgartirish mumkin:

- *private*. Bazaviy sinfning meros bo'lib o'tayotgan (ya'ni himoyalangan va ommaviy) nomlari hosila sinf nusxalarida kirib bo'lmaydigan bo'lib qoladi.
- *public*. Bazaviy sinf va uning ajdodlarining nomlari hosila sinf nusxalarida kirib boradigan bo'ladi, barcha himoyalangan nomlar esa himoyalangan bo'lib qolaveradi.

Agarda yangi sinf *class* kalitli so'z yordamida aniqlangan bo'lsa unda hosila sinfdagi meros komponentalar *private* kirish statusiga ega bo'ladi, *struct* yordamida esa *public* statusiga ega bo'ladi.

Meroslikda ko'rsatilmagan kirish statusini asosiy(bazaviy) sinf ismini oldidan ko'rsatilgan *private*, *protected* va *public* kirish atributlari yordamida o'zgartirish mumkin.

Agarda V sinf quyidagicha aniqlangan bo'lsa:

```
class B { protected: int t;
        public: char u;
};
```

unda quyidagi hosila sinflarni kiritish mumkin:

class M: protected B { ... }; // t va u *protected* sifatida merosxo'r

class P: public B { ... }; // *protected*, va u *public* sifatida merosxo'r

class D: private B { ... }; // t va u *private* sifatida merosxo'r

struct F: private B { ... }; // t, i u *private* sifatida merosxo'r

struct G: public B { ... }; t - *protected* va u – *public* sifatida merosxo'r

7.2. Konstruktor va destruktorlarda vorislik

Konstruktorlar meros bo'lmagani uchun, hosila sinfni yaratishda undan meros bo'lgan ma'lumot – a'zolari asosiy(bazaviy) sinf konstruktori orqali initsializatsiyalanishi lozim. Asosiy sinf konstruktori avtomatik ravishda chaqiriladi va hosila sinfni konstruktoridan oldin bajariladi. Asosiy (bazaviy) sinf konstruktorining parametrlari hosila sinfni konstruktorini aniqlashda ko'rsatiladi. Shunday qilib argumentlarni hosila sinfni konstruktoridan asosiy (bazaviy) sinfni konstruktoriga uzatish vazifasi bajariladi.

Misol:

```
class Basis {
   int a,b;
public:
Basis(int x,int y){a=x;b=y;}
};
class Inherit:public Basis {
   int sum;
public:
Inherit(int x,int y, int s):Basis(x,y){ sum=s; }
};
```

Sinf ob'yektlari pasdan tepaga qarab konstruktorlanadi: avvalo asosiy(bazaviy), keyin esa komponent – ob'yektlar (agarda ular mavjud bo'lsa), undan keyin esa hosila sinfning o'zi. Shunday qilib, hosila sinfning ob'yekti quyi ob'yekt sifatida asosiy (bazaviy) sinf ob'yektini o'z ichiga oladi.

Ob'yektlar teskari tartibda o'chiriladi: avvalo hosila, keyin uning komponent – ob'yektlari, undan keyin esa asosiy(bazaviy) ob'yekt.

Shunday qilib ob'yektni o'chirish tartibi uning konstruktorlash tartibiga nisbatan teskari bo'ladi.

7.3. Ko'plikdagi vorislik va virtual sinflar

Bu sinf ketma-ket(to'g'ridan-to'g'ri) baza sinfi, agar u sinflarni aniqlashda baza ro'yxatidan chiqadi. Agar ba'zi hollarda A sinf V sinfning bazasini ifodalasa va S sinf uchun V sinf bazasi bor bo'sa , unda V sinf S sinf uchun to'g'ridan-to'g'ri bazasi hisoblanadi, A sinf esa S sinf uchun to'g'ri bo'lmagan baza bo'lib hisoblanadi. X komponentiga murojaat qilganda A sinfga kiruvchi va V va S sinflarga izchil tarzida voris bo'ladi, S sinfga A::X tarzida, V::X tarzida e'lon qilish mumkin. Ikkala konstruksiya A sinfning elementi X ga murojaat qilishni ta'minlaydi.

Pastda sinflarni tasvirlashda qabul qilingan bazalar ishlab chiqilgan.

Xuddi shu tartibda ularni kompilyator e'lon qilishini ko'ramiz va ularni teksti dasturda joylashadi.

Sinflar bir nechta ketma-ket sinflardan tashkil topishi mumkin, sinf bazasida ixtiyoriy son yo'qolishi mumkin, misol uchun,

```
class X1 { ... };
class X2 { ... };
class X3 { ... };
class Y1: public X1, public X2, public X3 { ... };
```

Bir necha to'g'ri baza sinflari mavjud bo'lib, ular ko'plik vorislari deb nomlanadi.

Ko'plik vorislarida ketma-ket bazada hech qanday sinf bittadan ortiq ishlatilishi mumkin emas. Bitta sinf to'g'ri bo'lmagan sinfda bir necha marta ishlatilishi mumkin:

```
class X { ...; f() ; ... };
class Y: public X { ... };
class Z: public X { ... };
```

class D: public Y, public Z { ... };

Bu misolda X sinf D sinfning ikki marta o'rtacha vorisi bo'ladi. Bizning misolimizda ikkita qiymati qatnashadi va shuning uchun bir qiymatli bo'lmagan D sinfning ob'yekti X sinfning aniq komponentiga murojaat qulayligini bartaraf qilishi kerak, uni to'liq kvalifikatsiyasi D::Y::X::f() yoki D::Z::X::f(). D sinfning ob'yekti ichida sodda ko'rinish Y::X::f() yoki Z::X::f(), lekin bu ham kvalifikatsiyani mazmunidir.

Bir xil nomdagi ob'yektlarni bartaraf qilishda to'g'ri bo'lmagan sinf bazalari ko'plik vorislari bu sinf bazalari virtual deb e'lon qilinadi. Buning uchun sinf bazalari ro'yxatida oldingi sinf nomini *virtual* kalit so'zini ishlatish kerak. Misol uchun X sinfi virtual baza sinfi bo'la oladi, agar bunday ko'rinishda yozilganda:

```
class X { ... f() ; ... };
class Y: virtual public X { ... };
class Z: virtual public X { ... };
class D: public Y, public Z { ... };
```

Endi D sinf X sinfidan faqat bitta nusxada bo'ladi, ruxsat etilgan to'g'ri tenglikdan Y va Z sinflarini ifodalaydi.

7.4. Virtual funksiyalar

Virtual funksiyalar mexanizmiga biror komponent funksiyaning har bir hosilaviy sinfda alohida varianti mavjud bo'lish lozim bo'lganda murojaat qilinadi. Bunday funksiyalarga ega sinflar polimorf sinflar deb ataladi va ob'yektli dasturlashda alohida o'ringa ega.

Virtual funksiyalar kechki yoki dinamik bog'lanish mexanizmiga asoslanadi. Asos sinf har qanday nostatik komponent funksiyasi *virtual* kalit so'zi yordamida virtual deb e'lon qilinishi mumkin.

Kechki bog'lanishda erta bog'lanishga o'xshab adreslar statik ravishda kompilyatsiya jarayonida emas, balkim dinamik dastur bajarilishi jarayonida aniqlanadi. Bog'lanish jarayoni virtual funksiyalarni adreslar bilan almashtirishdan iborat. Virtual funksiyalar adreslar haqida ma'lumot saqlanuvchi jadvaldan foydalanadi.

Virtuallik vorislikka o'tadi. Funksiya virtual deb e'lon qilingandan so'ng hosila sinfda qayta ta'rifi(shu prototip bilan) bu sinfda yangi virtual funksiyani yaratadi, bu holda virtual spetsifikatori talab qilinmaydi.

Destruktorlardan farqli konstruktorlar virtual bo'lolmaydi. Amaliy jihatdan virtual funksiyaga ega har bir sinf virtual destruktorga ega bo'lishi kerak.

Misol:

```
class base{
   public:
    virtual void print(){cout<<"\nbase";}
. . . };
class dir : public base{
   public:
    void print(){cout<<"\ndir";}
};
void main() {
  base B,*bp = &B;
  dir D,*dp = &D;
  base *p = &D;
  bp –>print(); // base
  dp –>print(); // dir
  p –>print(); // dir
}
```

Bu misolda avlod sinf ob'yektiga adres qiymat sifatida berilgan ajdod sinf turidagi ko'rsatkich yoki ilova orqali avlodda qo'shimcha yuklangan usulni chaqirishga e'tibor berish lozim. Agar funksiya novirtual bo'lsa ajdod sinf usuli, virtual bo'lsa avlod sinf usuli chaqiriladi.

Shunday qilib ajdod sinf turidagi ko'rsatkich orqali virtual usul chaqirish natijasi shu ko'rsatkich qiymati ya'ni chaqiriq bajarilayotgan ob'yekt turi bilan aniqlanadi.

Qaysi virtual funksiyani chaqirish ko'rsatkich turiga emas shu ko'rsatkich qaratilgan(dastur bajarilish jarayonida) ob'yekt turiga bog'liq.

7.5. Abstrakt sinflar

Hech bo'lmasa bitta sof (bo'sh) virtual funksiyaga ega bo'lgan sinf abstrakt sinf deyiladi. Quyidagi tavsifga ega bo'lgan komponentali funksiya sof virtual funksiya deyiladi:

virtual <tip> <funksiya_nomi>(<formal_parametrlar_ro'yxati>) = 0;

Abstrakt sinf hosila sinf uchun asosiy (bazaviy) sinf sifatida ishlatilishi mumkin. Abstrakt sinflarning mexanizmi keyinchalik konkretizatsiyalanadigan umumiy tushunchalarni tavsiflash uchun ishlab chiqilgan. Bu holda, sinflar ierarxiyasini yaratish quyidagi sxema bo'yicha bajariladi.

Ierarxiya asosida abstrakt bazoviy sinf turadi. U interfeysni meros qilib olish uchun foydalaniladi. Hosila sinflar bu interfeysni konkretizatsiyalaydi va amalga oshiradi. Abstrakt sinfda sof virtual funksiyalar e'lon etilgan, ular aslida abstrakt usullar.

Ba'zi sinflar masalan shape sinfi, abstrakt tushunchalarni ifodalaydi va ular uchun ob'yekt yaratib bo'lmaydi. Bunday sinflar biror hosila sinfda ma'noga ega bo'ladi:

```
class shape {
 // ...
public:
 virtual void rotate(int) = 0; // sof virtual funksiya
 virtual void draw() = 0;      // sof virtual funksiya
};
```

Abstrakt sinfni faqat boshqa sinf ajdodi sifatida ishlatish mumkin:

```
class circle : public shape {
 int radius;
public:
 void rotate(int) { }
 // qayta ta'riflash shape::rotate
```

```
    void draw();
    // qayta ta'riflash shape::draw
     circle(point p, int r);
    };
```

Agar sof virtual funksiya hosila sinfda to'liq ta'riflanmasa u hosila sinfda ham sof virtual bo'lib qoladi, natijada hosila sinf ham abstrakt sinf bo'ladi.

Abstrakt sinflar realizatsiya detallarini aniqlashtirmasdan faqat interfeysni ko'rsatish uchun ishlatiladi. Masalan operatsion tizimda qurilma drayveri abstrakt sinf sifatida berilishi mumkin:

```
class character_device {
public:
  virtual int open() = 0;
  virtual int close(const char*) = 0;
  virtual int read(const char*, int) =0;
  virtual int write(const char*, int) = 0;
  virtual int ioctl(int ...) = 0;
  // ...
};
```

Drayverlar character_device sinfining ajdodlari sifatida kiritilishi mumkin.

Nazorat savollari:

1. Nima uchun avval ajdod sinf konstruktorlari chaqirilib, so'ngra avlod sinf konstruktori chaqiriladi?
2. Nima uchun destruktorlar konstruktorlarga nisbatan teskari tartibda chaqiriladi?
3. Vorislikda ajdod sinf spetsifikatori sifatida *protected* ko'rsatilishi mumkinmi?
4. Sinflar bibliotekasini qurishda vorislikdan qanday foydalaniladi?
5. Xususiy deb e'lon qilingan komponentalarga boshqa sinf usullari orqali murojaat qilish mumkinmi?

8 BOB. VORISLIKDAN FOYDALANISH XUSUSIYATLARI

8.1. Delphi tilida vorislik

Vorislik bu mavjud sinflarga yangi maydonlar, xossalar va usullar qo'shish yordamida yangi sinflar hosil qilish imkoniyatini beradi. Yangi hosil qilingan avlod sinf asos ya'ni ajdod sinf xossalari va usullariga vorislik qiladi.

Avlod sinf ta'rifida ajdod sinf nomi ko'rsatiladi. Misol uchun TEmployee (xodim) sinfi TPerson sinfidan FDepartment (bo'lim) maydonini qo'shish yordamida hosil qilinishi mumkin. TEmployee sinfining ta'rifi quyidagicha bo'ladi:

```
TEmployee = class(TPerson)
 FDepartment: integer;
 constructor Create(Name:TName; Dep:integer);
end;
```

Bu misolda TEmployee sinfi TPerson sinfining vorisidir. TEmployee o'z konstruktoriga ega bo'lishi lozim. TEmployee sinfi konstruktori quyidagicha berilishi mumkin:

```
constructor TEmployee.Create(Name:Tname; Dep:integer);
begin
 inherited Create(Name);
 FDepartment:=Dep;
end;
```

Bu misolda *inherited* direktivasi bilan ajdod sinf konstruktori chaqiriladi va avlod sinf maydoniga qiymat beriladi.

Voris sinf ob'yekti yaratilgandan so'ng ajdod sinf maydonlari va usullaridan foydalanish mumkin. Masalan:

```
engineer := TEmployee.Create(' Tolipov ',414);
engineer.address := ' Toshkent, Yunusobod 21, kv.3';
```

Birinchi instruksiya TEmployee tipidagi ob'yekt yaratadi, ikkinchisi — ajdod sinfga tegishli maydon qiymatini o'rnatadi.

8.2. Murojaat protected va private direktivalari

Sinf elementlariga murojaatni boshqarish uchun *protected* (himoyalangan) va *private* (xususiy) direktivalaridan foydalaniladi.

Himoyalangan, ya'ni *protected* sinf elementlariga sinfdan tashqari faqat voris sinflarda murojaat qilish mumkin. Odatda *protected* seksiyasiga sinf usullari ta'rifi joylashtiriladi.

Yopiq, ya'ni *private* sinf elementlarida faqat modul ichida murojaat qilish mumkin.

Quyida murojaatni boshqarish direktivalaridan foydalanilgan TPerson sinfi ta'rifi keltirilgan.

```
TPerson = class private
FName: TName;
FAddress: TAddress;
protected
    Constructor Create(Name:TName);
    Function GetName: TName;
    Function GetAddress: TAddress;
Procedure SetAddress(NewAddress:TAddress);
    Property Name: TName
    read GetName;
Property Address: TAddress
    read GetAddress
    write SetAddress;
end;
```

Sinf elementlarini berkitish uchun sinf ta'rifini alohida modulga joylashtirish mumkin.

8.3. Polimorfizm va virtual usullar

Sinfda aniqlangan usullarni statik, virtual (*virtual*), dinamik (*dynamic*) yoki abstrakt(*abstract*) turlarga bo'lish mumkin. Agar usul turi ko'rsatilmasa u avtomatik ravishda statik turini oladi. Statik usullar vorislarda to'liq qayta ta'riflanadi. Usul ta'rifini o'zgartirish mumkin.

Usulni virtual deb e'lon qilish voris sinfda virtual sinfni almashtirish imkonini beradi. Sinf davomchisida ishlatiladigan usul uchun *override* kalit so'zi ko'rsatilishi lozim.

Virtual usullar vorislikda tipini va nomini saqlab qolishi lozim.

Masalan:

```
Type
    TBase=Class
        Procedure MyJoy; Virtual;
    End;
   Tdescedant=Class(TBase)
        Procedure MyJoy; Override;
    End;
  Var
  FirstObject: TBase;
        SecondObject: TDescedant;
  Begin
        . . . . . .
        FirstObject.MyJoy;
        SecondObject.MyJoy;
        . . . . . .
  End;
```

Agar Tbase sinfida MyJoy usuli dinamik bo'lsa, *virtual* so'zi *dynamic* so'ziga almashtiriladi. Ularning asosiy farqi murojaat qilinganda virtual usul vaqt jihatdan ancha effektiv bo'lsa, dinamik usul esa operativ xotiradan ratsional foydalanish imkonini beradi.

Abstrakt usullar xech qanday vazifa bajarmaydi, chaqirilmaydi va vorislarda albatta qayta ta'riflanishi lozimdir. Virtual va dinamik usullar abstrakt bo'lishi mumkin. Abstrakt usullar *abstract* so'zi yordamida e'lon qilinadi:

```
procedure NeverCallMe; virtual; abstract;
```

Abstrakt usullar hech qanday kod yozilmaydi. NeverCallMe usulini chaqirish abstrakt usulni chaqirish istisnosi hosil bo'lishiga olib keladi.

Polimorfizm – bu har xil sinfga kiruvchi usullar uchun bir xil nomlarni ishlatish imkoniyatini yaratishdir. Polimorfizm prinsipi

shundan iboratki sinf ob'yektiga mos bo'lgan biror ishning bajarilishida bir xil usulga murojaat qilish mumkinligini ta'minlab beradi. Agar yangi sinf bosh sinfdan farqi uning usulida algoritm o'zgartirilgan bo'lsa bir xil nomli usulga ega bo'lgan ikkita sinf tashkil qilingan bo'ladi. U holda tashkil qilingan yangi sinf "polimorfizm" xossasiga ega bo'ladi.

Uchta sinf ta'rifi berilgan bo'lib bulardan biri qolgan ikki sinf uchun asos sinf bo'lsin:

```
type
   fname: string;
constructor Create(name:string);
function info: string;
   virtual;
end;
   fgr:integer;
constructor Create(name:string;gr:integer);
function info: string; override; end;
   fdep:string;
constructor Create(name:string;dep:string);
function info: string;
   override;
end;
```

Bu sinflarning har birida info usuli ta'riflangan. Asos sinfda virtual direktivasi yordamida info usuli virtual deb e'lon qilingan. Usulning virtual deb e'lon qilinishi avlod sinfda bu usulni shaxsiy usul bilan almashtirishga imkon beradi. Hosil qilingan sinfda virtual usulni almashtiruvchi usul *override* direktivasi bilan ta'riflanadi.

Quyida har bir avlod sinfda info usulining ta'rifi keltirilgan.

```
function TPerson.info:string;
    begin
    result := '';
    end;
function TStud.info:string;
begin
      result := fname + ' gp.' + IntTostr(fgr);
```

```
end;
function TProf.info:string;
  begin
        result := fname + ' kaf.' + fdep;
  end;
```

Ikkala sinf bitta asos sinfdan hosil qilingani uchun talabalar va domlalar ro'yxatini quyidagicha ta'riflash mumkin:

```
list: array[1..SZL] of TPerson;
```

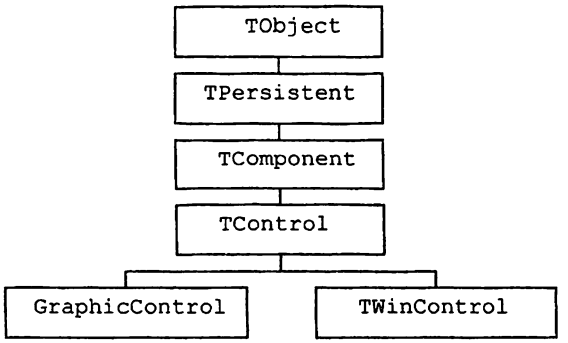
Talabalar va domlalar ro'yxatini info usulini massiv elementlariga qo'llab chiqarish mumkin.

Masalan:

```
st := '';
for i:=1 to SZL do
   if list[i] o NIL
        then st := st + list[i].Info+ #13;
        writeln (st);
```

8.4. Vizual komponentalar bibliotekasi

Delphi sinflari murakkab ierarxik strukturaga ega bo'lgan vizual komponentalar bibliotekasini (Visual Component Library -VCL) tashkil qiladi. VCL tarkibiga kiruvchi yuzlab sinflar mavjud. Hamma boshqa sinfga ajdod sinf bo'livchi asosiy sinflarga quyidagilar kiradi (8.1 - rasm).



8.1 – rasm. Vizual komponentalar bibliotekasi

Komponentalar – sinflarning nusxalari bo'lib, TComponent sinfining davomchilaridir (avlodidir). Boshqa hamma sinf nusxalari ob'yektlar deyiladi. Komponentalar bilan ob'yektlar orasida farq shundaki, formada komponentalar bilan (manipulyatsiya qilish) ish ko'rish mumkin, lekin ob'yekt bilan ish ko'rish mumkin emas.

Masalan, komponenta bo'lmagan TFont sinf ob'yektini qaraydigan bo'lsak uni formaga joylashtirib bo'lmaydi. Komponentalar *Label* yoki *Edit* larni formaga joylashtirish mumkin va ularni joylashtirishda TFont sinf turiga kiruvchi *Font* xossasidan foydalaniladi.

VCLga kiruvchi sinf TObject boshqa hamma sinflarning eng yuqorisi bo'lib, ob'yektlarni tuzish va boshqarish imkonini beradi. Bu sinfga bir necha usullar birlashtirilgan.

VCLga kiruvchi TPersistent sinfi TObject sinfidan kelib chiqadi va u ob'yektlarni tashkil qilish uchun o'zida bir necha usullarni saqlaydi.

VCLga kiruvchi TComponent sinfi barcha komponentalar ierarxiyasining eng yuqorisida joylashadi. TComponent sinfi davomchilari vizual bo'lmagan komponentalar bo'lib hisoblanadi.

Vizual bo'lmagan komponentalar dasturni loyixalash bosqichida tashqi ko'rinishi dasturning bajarilishi bosqichidagi ko'rinishidan mutloq farq qiladi. Ayrimlari dasturning bajarilishi vaqtida umuman ko'rinmaydi. TComponent sinfi vizual komponentalar uchun asos sinf bo'lib hisoblanadi.

VCLga kiruvchi TControl sinfi katta qismdagi xossalar, usullar va vizual komponentalar hodisalarini ta'minlab beradi. Bular yordamida klaviaturadan va sichqonchadan foydalangan holda ma'lumotlarni ekranga chiqarish va dasturga kiritish mumkin. TWinControl sinfi TControl sinfining davomchisi bo'lib oyna elementlarini boshqarishni yaratish uchun ishlatiladi.

TGraphicControl sinfi TControl sinfining davomchisi bo'lib grafik elementlarini boshqarish uchun ishlatiladi. TGraphicControl sinfi asosiy a'zolari quyidagilardir: *Shape* - geometrik figura; *PaintBox* - rasm chizish uchun panel; *Image* - tasvir; *Bevel* - uch o'lchovli ramka. TGraphicControl sinfi bitta usul va bitta xossaga ega.

Procedure Paint;

virtual;

- grafik elementlarni boshqarish uchun tasvirlarni chizadi.

Property Canvas;

TConvas;

- grafik elementlarni boshqarishni ekranda tasvirlash uchun xizmat qiladi.

Nazorat savollari:

1. Vorislik nima uchun kerak?
2. Himoyalangan *protected* va xususiy *private* huquqlari orasida qanday farq bor?
3. Polimorfizm deb nimaga aytiladi?
4. Usulni virtual deb e'lon qilish qanday imkon yaratadi?
5. Vizual komponentalar bibliotekasiga qanday komponentalar kiradi?
6. Polimorfizm va virtual usul orasida qanday farq bor?
7. Virtual usul dasturda qanday tavsiflanadi?
8. Vorislik va polimorfizmni dasturda birgalikda qo'llash mumkinmi?

9 BOB. STANDART AMALLARNI QO'SHIMCHA YUKLASH

9.1. Qo'shimcha yuklash ta'rifi

Standart amallarni (masalan +) qo'shimcha yuklash biror sinf bilan birga qo'llashda mazmunini o'zgartirishdan iboratdir.

Standart amallarni qo'shimcha yuklash maxsus funksiya – komponenta kiritish yo'li bilan amalga oshiriladi. Qo'shimcha yuklash til standartiga asosan amalga oshiriladi, amallar belgisi va operandlar soni o'zgarmaydi.

Amallarni qo'shimcha yuklash uchun quyidagi ta'rifdan foydalaniladi:

<operator amal> (<operandlar ro'yxati>)

Quyidagi amallarni qo'shimcha yuklash mumkin:

+, -, *, /, %, ^, &, |, ~, !;

=, <, >, +=, -=, *=, /=, %=, ^=, &=;

|=, <<, >>, >>=, <<=, ==, !=, <=, >=, &&;

||, ++, --, [], (), *new, delete*;

Bu amallar ustivorligi va ifodalar sintaksisini o'zgartirish mumkin emas. Masalan unar amal % yoki binar ! amalni kiritish mumkin emas. Funksiya amal har qanday funksiya kabi ta'riflanadi va chaqiriladi.

Standart tiplar uchun to'rt amal ("+", "-", "*" i "&") ham unar ham binar amal sifatida ishlatiladi va qo'shimcha yuklanadi.

Hamma qo'shimcha yuklangan amallar uchun *operator()* amalidan tashqari, ko'zda tutilgan argumentlardan foydalanish mumkin emas.

Amallar xossalarining ba'zilaridan foydalaniladi. Xususan operator=, operator[], operator() va operator-> , nostatik komponenta – funksiya bo'lishi lozim.

Operator – funksiya yoki sinf komponentasi bo'lishi kerak yoki juda bo'lmasa bitta parametri sinf ob'yekti bo'lishi kerak (*new* va *delete* amallarini qo'shimcha yuklovchi funksiyalar uchun bu shart emas).

Operator – funksiya, birinchi parametri asosiy turga tegishli bo'lsa, funksiya-komponenta bo'lolmaydi.

C++ tilida quyidagi amallarni qo'shimcha yuklash mumkin emas:

. sinf ob'yekti a'zosiga murojaat;
.* ko'rsatkich orqali murojaat;
?: shartli amal;
:: ko'rinish soxasini ko'rsatuvchi amal;
sizeof hajmni hisoblash amali;
preprotsessor amali.

9.2. Binar amallarni qo'shimcha yuklash

Ixtiyoriy ⊕ binar amali ikkita usulda tavsiflanishi mumkin: yoki bitta parametrli funksiya komponentasi sifatida, yoki ikkita parametrli global (do'stona bo'lishi mumkin) funksiya sifatida.

Birinchi holatda x⊕y ifoda x.operator⊕(y), ikkinchi holda esa operator ⊕(x,y) chaqirilishini bildiradi.

Aniq sinf doirasida qayta yuklangan operatsiyalar faqat parametrli nostatik komponentli funksiya orqali qayta yuklanadi. Sinfdagi chaqiriladigan ob'yekt avtomatik ravishda birinchi operand sifatida qabul qilinadi.

Misol:

```
class complex {
  double re, im;
 public:
  complex(double r, double i) { re=r; im=i; }
  complex operator+(complex);
  complex operator+(double);
  complex& operator+=(complex);
  complex& operator+=(double x);
};
inline complex operator+(complex b) //complex+complex
{ complex c;
  c.re=re+b.re;
  c.im=im+b.im;
  return c; }
inline complex operator+( double x) //complex+double
```

```
{  return complex(re+x, im);  }
inline complex& complex::operator+=(complex b)
//complex+=complex
{
   complex c;
   re += b.re;
   im += b.im;
   c.re=re;
   c.im=im;
   return c;   }
inline complex& complex::operator+=(double x)
//complex+=double
{  re += x;
   return *this;  }
```

Shuni ta'kidlab o'tish kerakki *double+complex* tipidagi qo'shishni bu usulda qo'shimcha yuklash mumkin emas.

Sinfdan tashqari yuklanadigan operatsiyalar ikkita operandga ega bo'lishi kerak. Ulardan biri sinf tipiga ega bo'lishi lozim.

Qiymat qaytarish uchun vaqtinchalik o'zgaruvchidan foydalaniladi. Bu maqsadda konstruktordan:

```
inline complex operator +(complex b)
{ return complex(re+b.re, im+b.im); }
```

va *this* ko'rsatkichidan foydalanish mumkin:

```
inline complex operator+=(complex b)
{  re+=b.re;
   im+=b.im;
  return *this;  }
```

Funksiya – amalni chaqirish:

```
  complex c = a + b;     // qisqa yozuv
  complex d = a.operator+(b); // oshkora chaqirish
```

Misol:

```
class complex  {
  double re, im;
 public:
```

```
    complex(double r, double i) {
      re=r;
      im=i;  }
  friend  complex operator+( complex , complex);
  friend  complex operator+( complex ,double);
  friend  complex operator+( double, complex);
  };
  inline  complex  operator+(complex  a,  complex  b)  //
complex+complex
  { complex c;
   c.re=a.re+b.re;
   c.im=a.im+b.im;
   return c;  }
  inline complex operator+(complex a, double x) //complex+double
      {  complex c;
         c.re=a.re+x;
         c.im=im;
         return c;  }
  inline complex operator+(double x, complex a) // double + complex
  {  complex c;
     c.re=a.re+x;
     c.im=im;
     return c;  }
  Funksiya – amalni chaqirish:
    complex c = a + b;    // qisqa yozuv
    complex d = operator+(a,b); // oshkora chaqirish
```

9.3. Unar amallarni qo'shimcha yuklash

Ixtiyoriy ⊕ unar amali ikkita usulda tavsiflanishi mumkin: yoki parametrsiz funksiya komponentasi sifatida yoki bitta parametrli global (do'stona bo'lishi mumkin) funksiya sifatida. Birinchi holatda ⊕ Z ifoda Z.operator ⊕ (), ikkinchi holatda esa operator ⊕(Z) chaqirilishini bildiradi.

Aniq sinf doirasida qayta yuklangan unar operatsiyalar faqat parametrsiz nostatik komponentli funksiya orqali qayta yuklanadi. Sinfdagi chaqiriladigan ob'yekt avtomatik ravishda operand sifatida qabul qilinadi.

Sinf doirasidan tashqarida qayta yuklangan unar operatsiyalar (global funksiya kabi) sinf tipdagi bitta parametrga ega bo'lishi lozim. Shu parametr orqali uzatiladigan ob'yekt operand sifatida qabul qilinadi.

Sintaksis:

a) birinchi holda (sinf doirasida tavsiflash):

<qaytariluvchi_qiymat_tipi> operator <amal_belgisi>

b) ikkinchi holda (sinf doirasidan tashqari tavsiflash):

<qaytariluvchi_qiymat_tipi> operator <amal_belgisi>

(<tip_identifikatori>)

9.4. Inkrement va dekrement amallarini qo'shimcha yuklash

C++ tilining zamonaviy versiyalarida prefiks ++ va -- operatsiyalarni qo'shimcha yuklash boshqa operatsiyalarni yuklashdan farq qilmaydi, Postfiks shakldagi ++va -- amallarini qayta yuklaganda yana bir *int* tipidagi parametr kiritilishi kerak. Agar qo'shimcha yuklash uchun global funksiya ishlatilsa uning birinchi parametri sinf tipiga, ikkkinchi parametri *int* tipiga ega bo'lishi kerak.

Dasturda postfiks ifoda ishlatilganda butun parametr ham qiymatga ega bo'ladi.

Quyida postfiks va prefiks ++ va — amallari uchun funksiya – amallarga misollar keltirilgan.

Misol:

```
complex &complex::operator++() // prefiks uchun komponenta
{ re++;
 return *this; }
complex &complex::operator--(int k) // postfiks uchun komponenta
{ re--;
 return *this; }
complex &operator++(complex& a) // prefiks uchun do'stona
```

```
{ a.re++;
  return a; }
complex &operator++(complex& a, int k) // postfiks uchun do'stona
{ a.re++;
  return a; }
```

9.5. Indekslash va funksiyani chaqirish amallarini qo'shimcha yuklash

Dumaloq qavs *()* amalini qo'shimcha yuklash sinf ob'yektiga funksiya chaqirig'i sintaksisini qo'llashga imkon beradi. Operandlar soni ixtiyoriy bo'lishi mumkin. Kvadrat qavs massiv elementi sintaksisini qo'llashga imkon beradi.

```
//------ Simvolni ajratish amali ------------------
   char string::operator()(int n) {
   if n<strlen(Str)
   return Str[i];
   else return '\0'; }
//------ Ostki satrni ajratish amali ------------------
string string::operator()(int n1, int n2) {
string tmp = *this;
delete tmp.Str;
tmp.Str = new char[n2-n1+1];
strncpy(tmp.Str, Str+n1, n2-n1);
return tmp; }
```

9.6. Qiymat berish va initsializatsiya

Qiymat berish va initsializatsiya turli amallardir. Ayniqsa destruktor aniqlanganda bu muhimdir. Biror X tipidagi ob'yektni initsializatsiya qilish nusxa olish konstruktori yordamida amalga oshiriladi. Satr – bu simvollar vektoriga ko'rsatkich.

Vektor konstruktor tomonidan yaratilib, destruktor bilan o'chirilganda muammo tug'ilishi mumkin:

```
string s1(10);
string s2(20)
```

s1 = s2;

Bu yerda ikki simvolli vektor joylashadi, lekin s1 = s2 qiymat berish natijasida biri o'chirilib, ikkinchisi nusxasi bilan almashtiriladi. Funksiyadan chiqishda s1 va s2 uchun destruktor chaqiriladi va bitta vektor ikki marta o'chiriladi. Bu muammoni hal qilish uchun qiymat berish amalini qo'shimcha yuklash lozim:

```
string& string::operator=(const string& a) {
 if (this !=&a) {
   delete p;
   p = new char[size=a.size];
   strcpy(p,a.p);
 }
 return *this;
}
```

Foydalanuvchi qiymat berish operatori initsializatsiya qilinmagan ob'yektga qo'llanilmaydi. Bu holda p ko'rsatkich tasodifiy qiymatga ega bo'ladi.

Initsializatsiya masalasini hal qilish uchun nusxa olish konstruktorini kiritish lozim:

```
string::string(const string& a) {
    p=new char[size=sz];
    strcpy(p,a.p);
 }
```

Nazorat savollari:

1. Postfiks va prefiks amallar orasida qanday farq bor?
2. Qo'shimcha yuklangan amallar qanday ikki usulda aniqlanadi?
3. Global do'stona funksiya yordamida hamma amallarni qo'shimcha yuklash mumkinmi?
4. Qaysi holatda amalni global funksiya yordamida qo'shimcha yuklash mumkin?
5. Funksiya operatorida "sinf" yoki "sinfga ilova" tipidagi parametr ishlatish shartmi?
6. Unar va binar amal-funksiyalar sintaksisi farqi nimadan iborat?

10 BOB. FUNKSIYA VA SINFLAR SHABLONLARI

10.1. Funksiyalar shablonlari

Funksiya shabloni (parametrlangan turlar) bog'langan funksiyalar oilasini tuzish imkonini beradi. Shablon kiritilishi uchun, hosil qilingan funksiyani avtomatlashtirish, har xil tipli ma'lumotlarni qayta ishlashdan iborat. Masalan, algoritm tartiblash uchun har qaysi funksiyani o'zini aniqlovchi tipi qo'llaniladi. Funksiya shabloni bir marta aniqlanadi, lekin parametrli aniqlashda va hokazo, ma'lumotlar tipi shablon parametrlari orqali beriladi. Shablon formati:

template <class tip_nomi, [class tip_nomi]>
<funksiya_sarlavxasi>
{ <funksiya tanasi>}

Funksiyalar shablonlari parametrlarining asosiy xususiyatlari:

1. Parametrlar nomlari shablonning butun ta'rifi bo'ylab unikal bo'lmog'i lozim.

2. Shablon parametrlarining ro'yxati bo'sh bo'la olmaydi.

3. Shablon parametrlari ro'yxatida har biri *class* so'zidan boshlanadigan bir nechta parametr bo'lishi mumkin.

Misol:

```
template <class T>T max(Tx, Ty){
return(x>y)? x:y; };
```

bunda - <class T> shablonining argumenti tomonidan taqdim etilgan ma'lumotlar turi har qanday bo'lishi mumkin. Undan dasturda foydalanishda kompilyator *max* funksiyasi kodini bu funksiyaga uzatilayotgan parametrlarning faktik turiga muvofiq generatsiya qiladi:

```
int i;
Myclass a,b;
int i=max(i, 0); //argumentlar turi int
myclass m=max(a, b); // argumentlar turi myclass
```

Faktik turlar kompilyatsiya paytida ma'lum bo'lishlari kerak. Shablonlarsiz *max* funksiyasini ko'p martalab ortiqcha yuklashga to'g'ri keladi, ya'ni, garchi barcha funksiya versiyalarining kodlari bir xil

bo'lsa ham, har bir qo'llanilayotgan tur uchun alohida ortiqcha yuklash kerak bo'ladi.

10.2. Sinflar shablonlari

Sinf shablonlari (o'zgacha parametrlangan sinf) avlodga oid sinfni tuzish uchun ishlatiladi. Tuzish qoidalarni va ayrim ob'yektlarni formatini aniqlovchi sinf kabi, sinf shablonlari ayrim sinflarni tuzish usullarini aniqlaydi. Shablondagi sinf ta'rifida sinf ismi ayrim sinflarning emas oilaviy sinflarning parametrlangan ismi bo'ladi.

Parametrlangan sinfni e'lon etishning umumiy shakli:

template <class malumot_tipi> class sinf_nomi { . . . };

Sinf shablonlarining asosiy xossalari:

*Parametrlangan sinf funksiyalarining komponentalari avtomatik ravishda parametrlangan bo'ladi. Ularni *template* yordamida parametrlangan sifatida e'lon etish shart emas.

*Parametrlangan sinfda tavsiflangan do'stona funksiyalar avtomatik ravishda parametrlangan funksiyalar bo'lmaydi, ya'ni ko'rsatilmagan holda shunday funksiyalar berilgan shablon bo'yicha tashkil etilgan barcha sinflar uchun do'stona bo'ladi.

*Agarda *friend*-funksiya o'z tavsifida parametrlangan sinf tipdagi parametrga ega bo'lsa, unda berilgan shablon bo'yicha yaratilgan barcha sinflar uchun xususiy *friend*–funksiyasi mavjud.

*Parametrlangan sinf doirasida *friend*-shablonlarni (do'stona parametrlangan sinflar) tavsiflash mumkin emas.

*Bir tarafdan, shablonlar shablonlardan hosil (meros) bo'lgandek, oddiy sinflardan ham hosil (meros) bo'lishi mumkin. Ikkinchi tarafdan esa ulardan boshqa shablonlar va sinflar uchun bazaviy sifatida foydalanishi mumkin.

*Sinf a'zosi bo'lgan funksiyalar shablonlarini *virtual* sifatida tavsiflash mumkin emas.

*Lokal sinflar o'z elementlari sifatida shablonlarni o'z ichiga olish mumkin emas.

10.3. Parametrlangan sinflarning komponent funksiyalari

Sinf shablonining tavsifidan tashqarida joylashgan sinf shablonining komponentli funksiyasini amalga oshirishda quyidagi ikkita elementni qo'shimcha kiritish lozim:

*Tavsiflash *template* kalitli so'zdan boshlanishi lozim, undan so'ng burchakli qavslarda sinf shablonni tavsifida ko'rsatilgan *tiplar_parametrlarining_ro'yxati* keladi.

* Ko'rish soxasi operatsiyasidan (::) oldinda bo'lgan sinf ismidan so'ng shablonning *parametrlar_ismlari_ro'yxati* kelishi lozim.

template<tiplar_ro'yxati><qaytariluvchi_qiymat_tipi> <sinf_nomi> <parametrlar_nomlari_ro'yxati > : :

<funksiya_nomi>(<parametrlar_ro'yxati>){ . . . }

Sinf ob'yektlari bilan ishlash uchun *vector* qo'shimcha yuklangan shablon sinfi:

```
template<class T > class vector {
T *data;
int size;
public:
array(int k) {size =k; data = new T[size]};
T&  operator[](int i){
 return data[i];  }
int size() { return size; }
~array () { delete []data; }
void input_ vector ();
void show_ vector ();
};
template<class T > void vector <T >:: input_array() {
 for (int i = 0; i < index; i++) {
 cin>>data[i] ;
 cout << ' ';}
}
template<class T > void vector <T >:: show_array() {
 for (int i = 0; i < index; i++)
```

```
    cout << data[i] << ' ';
}
```

10.4. Funksiya uchun shablon turi

Shablon sinflarini qo'llanilishi shablon funksiyasini a'zosini ifodalaydi. Parametrlari shablon sinflarini ifodalovchi global shablon funksiyalar - algoritmlarini aniqlash mumkin. Masalan oddiy shablonni pufaksimon algoritm orqali tartiblashni shunday aniqlash mumkin.

```
template<class T> void bubble_sort(Vector<T>& v)
{  unsigned n = v.size();
  for (int i=0; i<n-1; i++)
     for (int j=n-1; i<j; j--)
        if (v[j] < v[j-1])
{  // v[j] va v[j-1] o'rnini almashtiramiz
           T temp = v[j];
           v[j] = v[j-1];
           v[j-1] = temp;
        }
   }
```

Vector tipini qismiga tenglik funksiyasini berish mumkin emas, balki unga ikkinchi parametr *sort()* funksiyasini berish kerak. Bu parametr ob'yekt sinfini ifodalaydi, qaysiki tenglik operatsiyasini qayta aniqlashda.

```
template<class T, Compare >
void bubble_sort(Vector<T>& v, Compare & cmp){
  unsigned n = v.size();
  for (int i = 0; i<n-1; i++)
     for ( int j = n-1; i<j; j--)
        if (cmp.lessthan(v[j],v[j-1])) {
           T temp = v[j];
           v[j] = v[j-1];
           v[j-1] = temp;  }
}
```

for_each() algoritm yordamida har xil ko'rinishdagi qayta ishlashni va har bir elementni modifikatsiyasini ko'rish mumkin.

```
Template<class Item, class function>
void for_each(vector<Item> a, function op){
for( int i=0; i<a.size(); i++ )
 op(a[i]);
}
```

for_each() algoritmining qo'llanilishi:

```
# include <iostream.h>
class StudentPrint {
public:
void operator() (Student elem){
 if (elem.rating>5) elem.print();
 }};
main() {
  vector<Student> coll(5);
   coll.input();
   StudentPrint cmp;
   for_each(coll, cmp);   }
```

10.5. Yuqori darajali funksiyalar

Funksiya ob'yektlari – bu «kichik qavs» () operatsiya aniqlangan sinf nusxasi. Ba'zi bir holatlarda funksiyani ob'yekt – funksiyalarga almashtirish qulaydir. Ob'yekt - funksiya sifatida ishlatilsa uni chaqirish uchun *operator()* kalit so'zidan foydalaniladi.

Misol:

```
class kub{
public:
double operator()(double x)
{ return x*x*x;
} };
```

Yuqori darajadagi algoritm bu shunday algoritmki bitta yoki bir nechta argumentlar funksional tipga tegishlidir.

Dixotomiya usuli yordamida ixtiyoriy funksiya uchun [a,b] oraliqda f(x)=0 tenglamani yechish masalasi misolida yuqori darajali funksiyani ko‘rib o‘tamiz. Bu maqsadda masalani yechadigan metodni tavsiflovchi sinf yaratamiz. Sinfni tavsiflovchi dastur kodini ko‘ramiz:

```
template <class T>
class FunctionZero{
public:
static double dihotom(double a, double b, double eps, T f) };
template <class T>
double FunctionZero<T>::dihotom(double a, double b,double eps,T
f) {
    float x, x1=a, x2=b;
    while (x2-x1)>eps
    {x=(x1+x2)\2;
    if (f(x)==0)
    return x;
    if (f(x)>0) x1=x;
    else x2=x;
            };
    return x1;
    }
```

Nazorat savollari:

1. Shablonlardan nima maqsadda foydalaniladi?
2. Funksiya shabloni asosiy xossalarini ko‘rsating.
3. Parametrlashtirilgan sinflar xossalarini ko‘rsating.
4. Shablon parametrlari ro‘yxati bo‘sh bo‘lishi mumkinmi?
5. Parametrlashtirilgan funksiya qanday chaqiriladi?
6. Parametrlashtirilgan sinflar hamma komponenta funksiyalari parametrlashganmi?
7. Sinf shabloni tashqarisida komponenta funksiyalar qanday aniqlanadi?

11 BOB. DINAMIK SINFLARDAN FOYDALANISH XUSUSIYATLARI

11.1. Ko'rsatkich

Ko'rsatkich deb qiymati boshqa o'zgaruvchi yoki ma'lumotlar strukturasi adresiga teng bo'lgan o'zgaruvchiga aytiladi.

Ko'rsatkichlar quyidagicha ta'riflanadi:

<Nom:> ^ <Tip>;

bu erda:

- nom — o'zgaruvchi – ko'rsatkich nomi;
- Tip — o'zgaruvchi – ko'rsatkich ko'rsatayotgan o'zgaruvchi turi;
- ^ belgi o'zgaruvchi ko'rsatkich ekanligini ko'rsatadi.

Masalan:

p1: ^integer; r2: ^real;

Agar ko'rsatkich hech narsaga ko'rsatilmasa uning qiymati NULL ga teng deyiladi.

Identifikator NULL qiymat berish instruksiyalarida va shartlarda foydalanishi mumkin. Masalan p1 va r2 o'zgaruvchilar ko'rsatkich sifatida ta'riflangan bo'lsa instruksiya

p1 := NULL;

o'zgaruvchi qiymatini o'rnatadi, instruksiya

if r2=NULL

then writeln('Ko'rsatkich r2 initsializatsiya qilinmagan!');

ko'rsatkich r2 initsializatsiya qilinganligini tekshiradi.

Ko'rsatkichga biror o'zgaruvchi adresi qiymat sifatida berilsa @ adres olish operatoridan foydalaniladi. Masalan

r := @n;

Ko'rsatkichga boshqa ko'rsatkich qiymatini berish mumkin, agar ular bir turli bo'lsa. Masalan

r2 := p1;

Agar ko'rsatkich i o'zgaruvchiga ko'rsatayotgan bo'lsa,

r^ : = 5;

instruksiya bajarilgandan so'ng i qiymati 5 ga teng bo'ladi.

11.2. Dinamik o‘zgaruvchilar

Dinamik o‘zgaruvchi deb dastur bajarilish jarayonida xotirada ajratiladigan o‘zgaruvchiga aytiladi. Xotira ajratish *new* protsedurasini chaqirish orqali amalga oshiriladi. Dinamik xotiraga faqat ko‘rsatkich yordamida murojaat qilish mumkin.

Dinamik o‘zgaruvchini yo‘qotish, ya’ni bu o‘zgaruvchi egallagan xotirani ozod qilish uchun *Dispose* protsedurasi ishlatiladi.

Quyidagi protsedurada dinamik o‘zgaruvchilarni yaratish va yo‘qotish ko‘rsatilgan

```
procedure F1();
var
        p1,p2,p3: Integer;
begin
 New(p1);
 New(p2);
 New(p3);
 r1^ := 5;
 r2^ := 3;
 r3^ := r1^ + r2^;
 Writeln ('Sonlar summasi teng ' + IntToStr(r3^));
 Dispose(p1);
 Dispose(r2);
 Dispose(r3);
end;
```

11.3. Ro‘yxat

Ko‘rsatkichlar va dinamik o‘zgaruvchilar ro‘yxat va daraxtlar kabi murakkab dinamik ma’lumotlar strukturalarini yaratishga imkon beradi.

Ro‘yxatning har bir elementi ikki qismdan iborat yozuvdir. Birinchi qism – axborot qism. Ikkinchi qism oldingi elementlar bilan bog‘lanishni ta’minlaydi.

Dasturda ro'yxatdan foydalanish uchun ro'yxat komponentalari turi va birinchi elementga ko'rsatkich aniqlanishi lozim. Quyida talabalar familiyalari ro'yxati ta'rifi keltirilgan:

```
type
    TPStudent = ^TStudent;
    TStudent = record
    surname: string[20];
    name: string[20];
    group: integer;
    address: string[60];
    next: TPStudent;
end;
var
    head: TPStudent;
```

11.4. Dinamik ro'yxat

Quyidagi dastur talaba familiyasini ro'yxat boshiga qo'shib, talabalar ro'yxatini hosil qiladi.

```
type
      TPStudent=^TStudent;
      TStudent = record
      f_name:string[20];
      l_name: string[20];
      next: TPStudent;
end;
var
head: TPStudent;
```

Elementlarni ro'yxatga qo'shish uchun misol:

```
procedure F1();
var
      curr: TPStudent;
begin
      new(curr);
```

```
        curr^.f_name := Edit1.Text;
        curr^.l_name := Edit2.Text;
        curr^.next := head; head := curr;
        Edit1.text:=''; Edit2.text:= '';
end;
```

Ro'yxatni chiqarish uchun misol:

```
procedure F2();
var
        curr: TPStudent;
        n:integer;
        st:string;
begin n := 0; st := '';
        curr := head;
        while curr <> NULL do begin
        n := n + 1;
        st := st + curr^.f_name + ' ' + curr^.l_name+#13;
        curr := curr^.next;
 end;
if n <> 0
then Writeln ('Ro'yxat:' + #13 + st)
else Writeln ('Ro'yxatda elementlar yo'q.');
end;
```

Elementni ro'yxatdan o'chirish uchun misol:

```
 procedure F3();
var
        curr: TPStudent;
begin
        if head<>NULL then
         if head^.next=NULL then
 begin
         Dispose(head);
         head:=NULL;
 end
 else
```

```
begin
        new(curr);
        curr:=head;
        head:=curr^.next;
        Dispose(curr);
end;
end;
```

Nazorat savollari:

1. Dinamik o‘zgaruvchi deb qanday o‘zgaruvchiga aytiladi?
2. Qaysi protsedura faylga yozuv qo‘shishni amalga oshiradi?
3. Qaysi protsedura faylni ochib yozuvlarni ketma-ket o‘qiydi?
4. Ko‘rsatkich deb qanday o‘zgaruvchiga aytiladi?
5. Dinamik o‘zgaruvchilar deb qanday o‘zgaruvchiga aytiladi?

12 BOB. FAYLLAR BILAN ISHLASH

12.1. Fayllarni ochish va yopish

C++da fayllar bilan ishlash *fstream* kutubxonasidagi biron-bir sinflar yordamida amalga oshiriladi.

Fstream kutubxonasi fayllarni o'qib olish uchun javob beradigan *ifstream* sinfiga, hamda faylga axborotning yozib olinishiga javob beradigan *ofstream* sinfiga ega.

Biron-bir faylni yozish yoki o'qish uchun, ochish uchun, *ofstream* turdagi yoki mos holda *ifstream* turdagi o'zgaruvchini yaratish kerak. Bunday o'zgaruvchini initsiallashda fayl nomi o'zgaruvchi nomidan keyin qavs ichida berilgan belgilar massivi ko'rinishida uzatiladi.

Masalan, S diskida 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 ham dasturning bajarilayotgan fayli joylashtirilgan papkada bo'lsa, u holda 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.

12.2. 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 holda ham massivdagi belgilar sonini uzatish kerak.

Bundan tashqari ''<<''operatoridan foydalanish mumkin. Bu operatordan kodning bitta satrida turli turdagi qiymatlarni uzatgan holda

ko‘p martalab foydalanish mumkin. Satr haqida 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 ofl (''C:\text.txt'');
char a='M';
ofl.put(s);
char s[9]=''The text'';
ofl.put(s,9);
ofl<<''The text'';
int i=100;
ofl<<i;
char ss[]=''The text'';
int k=200;
ofl<<''The text''<<k<<ss<<200;
```

12.3. Fayldan o‘qish

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

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

12.4. Fayl oxirini aniqlash

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

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

Ushbu holda 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;   }
}
```

12.5. Fayllar bilan ishlashda xatolarni aniqlash

Xatolarni kuzatib borishda dasturlarga yordam berish uchun, fayl ob'yektining *fail* funksiyasidan foydalanish mumkin. Agar fayl operatsiyasi jarayonida xatolar bo'lmagan bo'lsa, funksiya yolg'on (0) ni qaytaradi. Biroq, agar xato uchrasa, *fail* funksiyasi haqiqatni qaytaradi. Masalan, agar dastur fayl ochadigan bo'lsa, u, xatoga yo'l

qo'yilganini aniqlash uchun, *fail* funksiyasidan foydalanishi kerak. Bu quyida shunday ko'rsatilgan:

```
ifstream input_file("FILENAME.DAT");
if (input_file.fail()) {
  cerr << "Ochilish xatosi FILENAME.EXT" << endl;
  exit(1);
}
```

TEST_ALL.CPP dasturi turli xato vaziyatlarni tekshirish uchun *fail* funksiyasidan foydalanadi:

```
#include <iostream.h>
#include <fstream.h>
void main(void) {
  char line[256] ;
  ifstream input_file("BOOKINFO.DAT") ;
  if (input_file.fail())
cerr << "Ochilish xatosi BOOKINFO.DAT" << endl;
  else {
    while ((! input_file.eof()) && (! input_file.fail())) {
      input_file.getline(line, sizeof(line)) ;
     if (! input_file.fail())
cout << line << endl;  } }
}
```

12.6. Faylni yopish

Dasturni tugallash uchun operatsiya tizimi o'zi ochgan fayllarni berkitadi. Biroq, odatga ko'ra, agar dasturga fayl kerak bo'lmay qolsa, uni berkitishi kerak. Faylni berkitish uchun dastur quyida ko'rsatilganidek *close* funksiyasidan foydalanishi kerak:

```
input_file.close ();
```

Faylni yopayotganingizda, dastur ushbu faylga yozib olgan barcha ma'lumotlar diskka tashlanadi va ushbu fayl uchun katalogdagi yozuv yangilanadi.

12.7. O‘qish va yozish operatsiyalarining bajarilishi

Hozirga qadar gap borayotgan dasturlar belgili satrlar ustida operatsiyalar bajarar edi. Dasturlaringiz murakkablashgan sari, ehtimol, sizga massivlar va tuzilmalarni o‘qish va yozish kerak bo‘lib qolar. Buning uchun dasturlar *read* va *write* funksiyalaridan foydalanishlari mumkin. *Read* va *write* funksiyalaridan foydalanishda ma’lumotlar o‘qiladigan yoki yozib olinadigan ma’lumotlar buferini, shuningdek buferning baytlarda o‘lchanadigan uzunligini ko‘rsatish lozim. Bu quyida ko‘rsatilganidek amalga oshiriladi:

```
input_file.read(buffer, sizeof(buffer));
output_file. write(buffer, sizeof(buffer));
```

Masalan, STRU_OUT.CPP dasturi tuzilma ichidagisini EMPLOYEE.DAT fayliga chiqarish uchun *write* funksiyasidan foydalanadi:

```
struct employee   {
  char name[64];
  int age;
  float salary;
} worker = { "Djon Doy", 33, 25000.0 };
ofstream emp_file("EMPLOYEE.DAT") ;
emp_file.write((char *) &worker, sizeof(employee));
}
```

Odatda *write* funksiyasi belgilar satriga ko‘rsatkich oladi. Xuddi shunday tarzda STRU_IN.CPP dasturi *read* metodidan xizmatchi haqidagi axborotni fayldan o‘qib olish uchun foydalanadi:

```
ifstream emp_file("EMPLOYEE.DAT");
emp_file.read((char *) &worker, sizeof(employee));
cout << worker.name << endl;
cout << worker.age << endl;
cout << worker.salary << endl;
}
```

12.8. Faylni kiritish-chiqarish

Qo‘shish tizimida faylni ochish uchun ochilishda quyida ko‘rsatilgan ikkinchi parametrni ko‘rsatish lozim:

fstream output_file("FILENAME.EXT", ios::app);

Bu holda *ios::app* parametri faylni ochish tizimiga ko‘rsatadi.

Ochish tizimining ma’nosi.

Ochish tizimi	Vazifasi
ios::app	Fayl ko‘rsatkichni faylni oxiriga joylashtirib qo‘shish tizimida faylni ochadi.
ios::ate	Fayl ko‘rsatkichini faylni oxiriga joylashtiradi. O‘qish mumkin emas, chiqaruvchi ma’lumotlar faylni oxiriga yoziladi.
ios::in	Kiritish uchun faylni ochilishini ko‘rsatadi.
ios::nocreate	Agarda ko‘rsatilgan fayl mavjud bo‘lmasa, fayl yaratilmaydi va xato qaytariladi.
ios::noreplace	Agarda fayl mavjud bo‘lsa, ochish operatsiyasi to‘xtash va xatolikni qaytarishi lozim.
ios::out	Chiqarish uchun faylni ochilishini ko‘rsatadi.
ios::trunc	Mavjud bo‘lgan faylni ichidagisini olib tashlaydi (ko‘chiradi).

Quyidagi faylni ochish operatsiyasi faylni ochishda, mavjud bo‘lgan faylni ko‘chirishini oldini olish uchun *ios::noreplace* tizimidan foydalanadi:

ifstream output_file("FIlename.EXT", ios::out | ios::noreplace);

Nazorat savollari:

1. Fayllar bilan ishlashda qaysi bibliotekadan foydalaniladi?
2. Fayl oxirini aniqlash uchun qaysi funksiyadan foydalaniladi?
3. Xatolikni aniqlash uchun qaysi funksiyadan foydalaniladi?
4. Fayllar bilan ishlash *read* va *write* funksiyalari vazifasini ko'rsating.
5. Faylni ochish rejimlarini ko'rsating.

13 BOB. OQIMLI SINFLAR

13.1. Oqimli sinflar ierarxiyasi

C++da oqimli sinflar bibliotekasi ikkita bazaviy *ios* va *streambuf* sinflar asosida tuzilgan. *Streambuf* sinfi kiritish-chiqarish fizik qurilmalari bilan xotirada joylashgan kiritish-chiqarish buferlarini o'zaro bo'g'lanishini va tashkil qilinishini ta'minlaydi. *Streambuf* sinfining metodlarini va ma'lumotlarini dasturchi ochiq ishlatmaydi. Mavjud bo'lgan sinflar asosida yangi sinflarni yaratishda dasturchiga ham sinfga murojaat etish ruxsat etilgan.

Ios sinfi formal kiritish chiqarish va xatolarni tekshirish vositalariga ega.

Standart oqimlar (*istream*, *ostream*, *iostream*) terminal bilan ishlash uchun xizmat qiladi.

Satrli oqimlar (*istrstream*, *ostrstream*, *strstream*) xotirada joylashtirilgan satrli buferlardan kiritish-chiqarish uchun xizmat qiladi.

Faylli oqimlar(*ifstream*, *ofstream*, *fstream*) fayllar bilan ishlash uchun xizmat qiladi.

Oqimli sinflar, ularning metodlari va ma'lumotlari dasturda murojaat etish ruxsatiga ega bo'ladi, qachonki unga kerakli bosh fayl kiritilgan bo'lsa.

iostream.h – *ios*, *ostream*, *istream* uchun.

strstream.h – *strstream*, *istrstream*, *ostrstream* uchun

fstream.h – *fstream*, *ifstream*, *ofstream* uchun

Quyidagi ob'yekt-oqimlar dasturda *main* funksiyasini chaqirish oldidan avvaldan aniqlangan va ochilgan bo'ladi:

```
extern istream cin;  //Klaviaturadan kiritish standart oqimi
extern ostream cout; //Ekranga chiqarish standart oqimi
extern ostream cerr; //Xatolar haqidagi xabar chiqarish standart oqimi
```

13.2. Oqimli sinflar usullari

Oqimdan kiritish uchun *istream* sinfidagi ob'yektlar ishlatiladi, oqimga chiqarish uchun - *ostream* sinfidagi ob'yektlar.

Istream sinfida quyidagi funksiyalar tavsiflangan:

- istream get (char& S);

Istream dan S ga simvolni o'qiydi. Xato holatida S 0XFF qiymatini oladi.

- int get();

Istream dan keyingi simvolni chiqaradi. Faylni oxirini aniqlagach *EOF* ni qaytaradi.

- istream& get(char* buffer,int size,char delimiter='\n');

Bu funksiya *istream* dan simvollarni chiqaradi va ularni buferga nusxalaydi. Operatsiya yoki faylning oxiriga yetganda yoki *size* fayllardan nusxa olgan jarayonda yoki ko'rsatilgan ajratuvchini aniqlaganda to'xtaydi. Ajratuvchi esa nusxalanmaydi va *streambuf* qoladi. O'qib bo'lingan simvollar ketma-ketligi har doim *null* simvol bilan tugatiladi.

- istream& getline(char* buffer,int size, char delimiter='\n');

Ajratuvchi oqimdan chiqariladi, lekin buferga kiritilmaydi. Bu esa satrlarni oqimdan chiqaruvchi asosiy funksiya. O'qib chiqilgan simvollar *null* simvoli bilan ta'momlanadi.

- istream& read(char* buffer,int size);

Ajratuvchilarni qo'llanmaydi va buferga o'qilgan simvollar *null* simvoli bilan tugamaydi.

- int peek();

Istream dan simvolni chiqarmasdan *istream* ga qaytaradi.

- int gcount();

Formatlanmagan oxirgi kiritish operatsiyasi vaqtida o'qilgan simvollar sonini qaytaradi.

* istream& putback(S)

Agar *get* doirasidagi *streambuf* ob'yektida bo'sh fazo mavjud bo'lsa, unda o'sha yerga S simvoli joylashtiriladi.

* istream& ignore(int count=1,int target=EOF);

Quyidagilar bajarilmaguncha *istream* dan simvol chiqarilaveradi:

– funksiya *count* simvollarni chiqarmaguncha;

– *target* simvoli aniqlanmaguncha;

– faylni oxiriga yetmaguncha.

Ostream sinfida quyidagi funksiyalar tavsiflangan:

* ostream& put(char C);

Ostream ga S simvolni joylashtiradi.

* ostream& write(const char* buffer,int size);

Buferda mavjudlarni *ostream* ga yozadi. Xatoga duch kelmaguncha yoki *size* simvollarni nusxasini olmaguncha simvollarni nusxasi olinadi. Bufer formatlanmasdan yoziladi. Nol simvollarga ishlov berish boshqa ishlov berishlardan farq qilmaydi. Quyidagi funksiya ishlov berilmagan (binar va matnli) ma'lumotlarni *ostream* ga uzatadi.

* ostream& flush();

Streambuf buferni olib tashlaydi.

Bu funksiyalardan tashqari *istream* sinfda >>, *ostream* sinfda esa << operatsiyalar qayta yuklangan. << va >> operatsiyalar ikkita operandga ega. Chap operandi – bu *istream* (*ostream*) sinfning ob'yekti, o'ng operandi esa – bu dasturlash tilida ko'rsatilgan tipdagi ma'lumot.

Misol:

```
char ch, next, lookahead;
while ( cin.get( ch ))
{   switch (ch) {
  case '/':
    // izohni peek() yordamida tekshirish
    // agar xa bo'lsa satr qolganini o'tkazish
    next = cin.peek();
    if ( next == '/' )  cin.ignore( lineSize, '\n' );
    break;
  case '>':
    // leksemaga tekshirish >>=
    next = cin.peek();
    if ( next == '>' ) {
```

```
lookahead = cin.get();
next = cin.peek();
if ( next != '=' ) cin.putback( lookahead );
}
```

13.3. Formatlash

Ushbu ma'lumotlar uchun *cout*, *cin*, *cerr*, *clog* standart potoklarga kiritish << va chiqarish >> operatsiyalarni to'g'ridan to'g'ri qo'llash qayta uzatish qiymatlarini tashqi tavsiflash aytib o'tilmagan formatlardan foydalanishga olib keladi.

Chiqaruvchi axborotni tavsiflash formatlari va ma'lumotlarni kiritishda qabul qilish qoidalari dasturlovchi orqali formatlash bayroqlari yordamida o'zgartiriladi. Bu bayroqlar *ios* bazaviy sinfdagi hamma oqimlardan meros bo'lgan. Formatlash bayroqlari alohida qayd etilgan bitlar ko'rinishida amalga oshirilgan va *long x_flags* sinfning *protected* komponentasida saqlanadi. Ularga murojaat etish uchun tegishli *public* funksiyalar mavjud.

Formatlash bayroqlaridan tashqari *ios* sinfning quyidagi *protected* komponentalari ishlatiladi:

int x_width – chiqarish maydonining minimal eni.

int x_precision – kiritishda haqiqiy sonlarning tavsiflash aniqligi (kasr qismning raqamlar soni);

int x_fill – chiqarishda to'ldiruvchi simvol, probel ko'rsatilmagan holda.

Ushbu maydonlarni qiymatlarini olish (o'rnatish) uchun quyidagi funksiyalar komponentalari ishlatiladi:

int width();
int width(int);
int precision();
int precision(int);
char fill();
char fill(char);

Agar bir marta *cout.fill,* yordamida to'ldiruvchi simvol tanlansa *cout.fill* qayta chaqirilmaguncha o'zgarmaydi.

13.4. Manipulyatorlar

Manipulyatorlar – oqim ishini modifikatsiyalanishini taminlovchi maxsus funksiyalar. Manipulyatorlarning xususiyati shundaki, ularni >> yoki << operatsiyalarning o'ng operand sifatida foydalanish mumkin. Chap operand sifatida esa har doimgidek oqim (oqimga ilova) ishlatiladi va xuddu shu oqimga manipulyator ta'sir etadi.

- *endl* Yangi qator simvolini qo'yib, bufer *ostream* bo'shatish
- *dec* O'nlik sanoq tizimida chiqarish (ko'zda tutilgan)
- *hex* O'n oltilik sanoq tizimida chiqarish
- *oct* Sakkizlik sanoq tizimida chiqarish
- *ws* Bo'shliq belgisini o'tkazish

// quyidagi manipulyatorlar uchun *#include <iomanip>* ulash talab etiladi

- *setfill(ch)* ch simvol bilan bo'sh joyni to'ldirish
- *setprecision(n)* n ga teng bo'lgan suzuvchi nuqtali sonni chiqarish aniqliligini o'rnatish
- *setw(w)* w ga teng bo'lgan kiritish yoki chiqarish maydonining enini o'rnatish
- *setbase(b)* b asosiga ega bo'lgan butun sonlarni chiqarish

13.5. Oqimni holati

Har bir oqim u bilan bog'liq holatga ega. Oqimni holati *enum* o'tkazish ko'rinishida *ios* sinfida tavsiflanadi:

```
public:
enum io_state{
goodbit, // 0X00 xatosi yo'q
eofbit, // 0X01 faylni oxiri
failbit, // 0X02 oxirgi operatsiya bajarilmagan
```

badbit, // 0X04 mumkin bo'lmagan operatsiyani ishlatishni harakat qilish

hardfail // 0X08 taqribiy xato

};

Ios ob'yekti bilan oxirgi bajarilgan operatsiya natijalarini aniqlovchi bayroqlar *state* o'zgaruvchisida mavjud. Shu o'zgaruvchining qiymatlarini *int rdstate()* funksiyalari yordamida olish mumkin.

Bundan tashqari, oqimlar holatini quyidagi funksiyalar orqali tekshirish mumkin:

int bad(); 1, agar *badbit* yoki *hardfail*

int eof(); 1, agar *eofbit*

int fail(); 1, agar *failbit*, *badbit* yoki *hardfail*

int good(); 1, agar *goodbit*

Agarda >> operatsiyasi ma'lumotlarni yangi tiplari uchun ishlatilsa, unda uni qayta yuklashda tegishli tekshirishlarni ko'zda tutmoq lozim.

Funksiya-komponenta *cout.fill* va manipulyator *setw ()*ga tegishli misol:

```
#include <iostream.h>
#include <iomanip.h>
void main(void) {
  cout << "Axborot jadvili " << endl;
  cout.fill (' . ');
  cout << "Kompaniya soxasi " << setw(20) << 10 << endl;
  cout << "Kompaniya daromadi va zarari " << setw(12) << 11 <<
endl;
  cout << "Kompaniya raxbariyati " << setw(14) << 13 << endl;
}
```

13.6. Chiqarish operatorini qo'shimcha yuklash

Chiqarish operatori *ostream* sinfi ob'yektiga ilova qaytaruvchi binar operatordir. Umumiy holda qo'shimcha yuklangan chiqarish operatori ta'rifi quyidagi ko'rinishga ega:

```
ostream& operator <<( ostream& os, const ClassType &object ){
.....
Return os;
}
```

Bu ta'rifning birinchi argumenti *ostream* ob'yektiga ilova, ikkinchisi odatda konstanta biror sinf ob'yektiga ilova bo'ladi. Qaytariluvchi qiymat *ostream* ob'yektiga ilova bo'ladi.

Birinchi argument ilova bo'lgani uchun, chiqarish operatori sinf a'zosi sifatida emas oddiy funksiya sifatida ta'riflanishi zarur. Agar funksiya yopiq sinf a'zolariga murojaat qilishi zarur bo'lsa do'stona deb e'lon qilinishi zarur.

1. Chiqarish operatorini qo'shimcha yuklashga doir misol:

```
class WordCount {
  friend ostream&
       operator<<( ostream&, const WordCount& );
public:
  WordCount( string word, int cnt=1 );
  // ...
private:
  string word;
  int occurs;
};
ostream& operator <<( ostream& os, const WordCount& wd ){
  // format: <xisoblagich> so'z
   os << "< " << " > " > "
     << wd.word;
   return os;
}
```

2. Yuklangan chiqarish operatoriga ega bo'lgan sinfdan foydalanishga doir misol:

```
#include <iostream>
#include "WordCount.h"
int main() {
  WordCount wd( "sadness", 12 );
```

```
    cout << "wd:\n" << wd << endl;
    return 0;
}
```

Qo‘shimcha yuklangan chiqarish operatorini *ofstream* sinfi ob’yektlariga ham qo‘llash mumkin. Quyidagi misolda WordCount sinfi chiqarish operatori chaqiriladi:

```
#include <fstream>
#include "WordCount.h"
int main() {
   ofstream oFile( "word.out" );
   WordCount artist( "Renoir",12);
   oFile << artist;
}
```

Nazorat savollari:

1. Qaysi sinflar asosida oqimlar bibliotekasi qurilgan?
2. Satrli oqimlarni va ularning vazifalarini ko‘rsating.
3. Oqimlar usullarini ko‘rsating.
4. Formatlash uchun qanday komponenta funksiyalardan foydalaniladi?
5. Manipulyatorlar vazifasini ko‘rsating.
6. Qaysi manipulyatorlar uchun *#include <ionamip.h>* ni ulash zarur?

14 BOB. ISTISNOLARNI BOSHQARISH

14.1. Istisno holatlar

C++ tili OMD doirasida istisnolarga xizmat ko'rsatish standartini belgilab beradi. Istisno holatlar (exception) dasturda xatoni – kutilmagan hodisani ifodalaydi. Dastur o'zining ishlab chiqilishida ko'zda tutilmagan normal bo'lmagan vaziyatga duch kelganda, boshqaruvni ushbu muammoni hal qilishga qodir bo'lgan dasturning boshqa qismiga berishi mumkin hamda yoq dasturni bajarishni davom ettirish yoki ishni tugallash kerak.

Istisnolarni joydan joyga tashlab berish (*yexcepletion throwing*) dasturning normal bajarilishiga to'sqinlik qiladigan sabablarning tashxisi uchun foydali bo'lishi mumkin bo'lgan axborotni tashlab berish nuqtasida to'plash imkonini beradi. Siz dastur tugallanishi oldidan zarur xatti-harakatlarni bajaradigan istisnolarga ishlov bergich (*exception handler*) ni aniqlashingiz mumkin. Dastur ichida yuzaga keladigan sinxron istisnolar deb nomlanuvchi istisnolarga xizmat ko'rsatiladi. Ctrl+C klavishalarini bosish kabi tashqi holatlar istisno hisoblanmaydi.

Dasturda har bir istisno holat sinf sifatida aniqlanadi. Masalan, quyida ko'rsatilan holat fayllar bilan ishlash uchun uchta istisno holatni aniqlaydi:

```
class file_open_error {};
class file_read_error {};
class file_write_error {};
```

Istisno holatlar o'zgaruvchilarni va sinf funksiya – elementlarini ishlatishi mumkin. Har bir istisno holat sinfga mos.

14.2. Istisnolarni qayta ishlash

Dastur istisno holatni ko'rishdan va unga javob berishdan oldin istisno holatini aniqlovchi C++ dagi *try* operatorini ishlatishi lozim. Istisnolarni generatsiya qila oladigan kod bloki *try* kalit-so'z bilan boshlanadi va shakldor qavslar ichiga olinadi. Agar *try* blok ichida

istisnoni topib olsa, dasturiy uzilish sodir bo'ladi hamda quyidagi xatti-harakatlar ketma-ketligi bajariladi:

1.Dastur istisnoga ishlov bergichning to'g'ri keladiganini qidiradi.

2.Agar ishlov bergich topilsa stek tozalanadi va boshqaruv istisnolarga ishlov bergichga uzatiladi.

3.Agar ishlov bergich topilmagan bo'lsa, ilovani tugatish uchun *terminate* funksiyasi chaqiriladi.

Yuzaga kelgan istisnoga ishlov beruvchi kod bloki *catch* kalit-so'z bilan boshlanadi va shakldor qavs ichiga olinadi. Istisnoga ishlov bergichning kamida bitta kod bloki bevosita *try* blokining ortidan kelishi kerak. Dastur generatsiya qilishi mumkin bo'lgan har bir istisno uchun o'z ishlov bergichi ko'zda tutilgan bo'lishi kerak.

Istisnolarga ishlov bergichlar navbatma-navbat ko'rib chiqiladi hamda turi bo'yicha *catch* operatoridagi argument (dalil) turiga to'g'ri keladigan istisnoga ishlov bergich tanlab olinadi. Ishlov bergich tanasida *goto* operatorlari bo'lmagan taqdirda berilgan *try* bloki istisnolariga ishlov bergichning oxirgisidan keyin kelgan nuqtadan boshlab dasturning bajarilishi yana davom etadi.

Masalan, *file_copy* funksiyani chaqirishda quyidagi *try* operatori istisno holatni aniqlash imkonini beradi:

```
try {
file_copy("SOURCE.TXT", "TARGET.TXT") ;
};
```

Qanday istisno holat ro'y berganini aniqlash uchun *try* operatoridan so'ng dastur bitta yoki bir nechta *catch* operatorlarini joylashtirishi lozim:

```
    catch (file_open_error) {
  cerr << "boshlangich yoki maqsadli faylni ochish xatoligi"<< endl;
  exit(1);
}
```

Bu holda xato tipiga qaramasdan kod xabardor qiladi va dasturni tugatadi. Agarda funksiyaning chaqiruvi xatosiz bajarilgan va istisno xatolar aniqlanmagan bo'lsa C++ *catch* operatorini shunchaki e'tiborga olmaydi. Qayta ishlovchilar tartibi muhimdir.

```
try {  // ... }
catch (ibuf) { // kiritish buferi to'lishini qayta ishlash
}
catch (io) { // kiritish – chiqarish xatoligini qayta ishlash
}
catch (stdlib) { // bibliotekadagi istisno holatni qayta ishlash
}
catch (...) {  // qolgan hamma istisnolarni qayta ishlash
}
```

14.3. Istisnolarni generatsiya qilish

C++ o'zi istisno holatlarni yuzaga keltirmaydi. Ularni C++ ning *throw* operatoridan foydalangan dasturlar yuzaga keltiradi. Istisno yuzaga kelganda, *throw* operatoridagi <ifoda> nom berish ifodasi muvaqqat ob'yektni nomlaydi (initsiallashtiradi), bunda muvaqqat ob'yektning turi ifoda argumenti (dalili) ning turiga mos keladi. Ushbu ob'yektning boshqa nusxalari, masalan, istisno ob'yektidan nusxa ko'chirish konstruktori yordamida generatsiya qilinishi mumkin.

Masalan fayl ochilishida dastur xato kelib chiqish shartlarini tekshirish va *throw file_open_error()* istisno holatni yuzaga keltirishi mumkin.

14.4. Kutilmagan istisnolarni qayta ishlash

Agar dasturda ko'zda tutilmagan istisno hodisa yuz bersa standart istisnolarni qayta ishlovchi ishlatiladi. Ko'p hollarda bu standart qayta ishlovchi dastur bajarilishini to'xtatib qo'yadi.

Avval *unexpected* funksiyasi chaqirilib, undan so'ng ko'zda tutilgan bo'yicha *terminate* funksiyasi ishga tushadi. Bu funksiya dasturni to'xtatish uchun *abort* funksiyasini chaqiradi.

Dasturda maxsus qayta ishlovchidan foydalanish uchun *set_unexpected* va *set_terminate* funksiyasidan foydalanish lozim. Bu

funksiyalar prototiplari *except.h* sarlavxali faylda aniqlangan. Bu funksiyalar *void* tipiga ega bo'lib parametrlarga ega bo'lmasligi kerak.

14.5. Istisno holatning ma'lumotlar elementlaridan foydalanish

Yuqorida ko'rib o'tilgan misollarda dastur *catch* operatoridan foydalanib, qanday istisno holat ro'y berganini va ularga tegishli holda javob berishini imkonini beradi. Masalan, *file_open_error* istisno holatida dastur xatoni chaqiruvchi fayl nomini bilishi lozim. Istisno holatga tegishli shunday ma'lumotni saqlash uchun dastur istisno holat sinfiga ma'lumotlar elementlarini qo'shishi lozim. Agar keyinchalik dastur istisno holatni yuzaga keltirsa u ushbu ma'lumotni quyida ko'rsatilganidek, istisno holatiga ishlov beruvchi funksiyaga o'zgaruvchi sifatida uzatadi:

```
throw file_open_error(source);
throw file_read_error(344);
```

Istisno holatga ishlov berishda bu parametrlar sinfga tegishli o'zgaruvchilarga o'zlashtirilishi mumkin (konstruktorga o'xshaydi). Masalan, sinfning tegishli o'zgaruvchisiga xatoga yo'l qo'ygan faylni ismini o'zlashtirish uchun quyidagi operatorlar *file_open_error* istisno holatni o'zgartiradi:

```
class file_open_error {
public:
  file_open_error(char *filename) {
    strcpy(file_open_error::filename, filename);
    }
  char filename[255] ;
    };
```

14.6. Istisno holatlar va sinflar

Sinf yaratishda shu sinfga tegishli istisno holatlarini aniqlash mumkin. Aniq sinfga tegishli istisno holatini yaratish uchun ushbu

istisno holatini sinfning umumiy (*public*) elementlari sifatida kiritish zarur.

Masalan diapazon chegarasidan chiquvchi indeks qiymatini bilish zarur bo'lsin:

```
class Vector {
  // ...
public:
  class Range {
  public:
    int index;
    Range(int i) : index(i) { }
  };
  // ...
  int& operator[](int i)
  // ...
        };
       int Vector::operator[](int i)
        {
 if (o<=i && i <sz) return p[i];
 throw Range(i);
        }
```

Mumkin bo'lmagan indeks qiymatini bilish uchun istisno holatini tasvirlovchi ob'yektga nom berish kerak:

```
void  f(Vector& v) {
  // ...
  try {
     do_something(v);
  }
  catch (Vector::Range r ) {
    cerr << "mumkin bo'lmagan indeks" << r.index << '\n';
    // ...
  }
  // ...
```

```
}
```

Qavsdagi konstruksiya tavsif bo'lib funksiya formal parametriga mosdir. Unda parametr tipi va yuzaga kelgan istisno nomi berilishi mumkin.

14.7. Istisnolar va konstruktorlar

Istisnolar konstruktordagi xatolar haqida ma'lumot berishga imkon beradi. Konstruktor chaqiruvchi funksiya tekshirib ko'rishi mumkin bo'lgan qiymat qaytarmagani uchun istisnolarsiz quyidagicha xatolik haqida ma'lumot berishi mumkin:

1. Ob'yektni xatolik bilan qaytarish toki foydalanuvchi o'zi tekshirib ko'rsin.
2. Lokal bo'lmagan o'zgaruvchiga ob'yekt yaratilmagani haqida ma'lumot beruvchi qiymat o'rnatish.

Istisnolar ob'yekt yaratilmagani haqidagi ma'lumotni tashqariga uzatishga imkon beradi:

```
Vector::Vector(int size) {
 if (sz<0 || max<sz) throw Size();
 // ...
}
```

Vektor yaratilayotgan funksiyada noto'g'ri o'lcham (*Size()*) xatoligini qayta ishlash mumkin:

```
Vector* f(int i) {
 Vector* p;
 try {
  p = new Vector v(i);
 }
 catch (Vector::Size) {
  // vektor noto'g'ri o'lchami
 }
 // ...
 return p;
}
```

Nazorat savollari:

1. Istisnolarni nima uchun generatsiya qilish va qayta ishlash kerak?
2. Istisno generatsiyasi sintaksisini keltiring.
3. Istisnoni qayta ishlash sintaksisini keltiring.
4. Istisnolar bilan qanday operatorlar bog'liq?
5. Istisnoni generatsiya qiluvchi funksiya sintaksisini keltiring.

15 BOB. FAYLLAR VA ISTISNOLAR BILAN ISHLASH XUSUSIYATLARI

15.1. Fayllar bilan ishlash funksiya va protseduralari

Kiritiladigan va chiqariladigan ma'lumotlar soni ko'p miqdorda bo'lsa ularni faylda saqlash dasturda qulaylik tug'diradi. Bu ma'lumotlar oddiy matn (tekst) fayllarida saqlanadi. Fayl o'zgaruvchisi dastur bo'sh qismida e'lon qilinadi, ya'ni

```
Type f=text;
Var
    fx:f;
```

bu yerda f -fayl tipi, oddiy matn faylni bildiradi; fx-fayl o'zgaruvchisi.

Kerakli fayldan ma'lumotlarni o'qishga tayyorlash uchun *Assign* va *Reset* funksiyalari ishlatiladi.

Assign – fayl o'zgaruvchisi bilan asosiy fayl orasida aloqa o'rnatadi. Assign (fx,'c:\a\f1.txt');

Reset – faylni topib uni ishga tayyorlaydi.

Reset (fx);

bu yerda fx- fayl o'zgaruvchisi;

'c:\a\f1.txt'-c: diskning A katalogidagi fx.txt fayldan o'qishni bildiradi.

Fayldagi ma'lumotlarni o'qish uchun *Read* funksiyasi ishlatiladi.

Read (<fayl o'zgaruvchisi>, <o'zgaruvchilar, massivlar>);

Misol. Read (fx, x,y,z,A[i],B[i]);

Fayldan o'zgaruvchilar yoki massivlar qiymatlarini o'qib bo'lgandan keyin fayl yopiladi. Faylni yopish quyidagi funksiya yordamida bajariladi:

Close (fayl o'zgaruvchisi);

Misol 1. Close (fx);

Misol 2. C: diskdagi AA katalogdagi AB fayldagi ma'lumotlarni o'qib A va B massivlarga joylashtiring. Fayl ma'lumotlari strukturasi quyidagicha.

15.2 20.5
20.1. 25.5
20.2 50.2
26.8 52.4
. . . . . .

ya'ni fayl strukturasi ikki ustundan iborat bo'lgan ma'lumotlar to'plamidan iborat.

```
Program FAB;
  Type
     f=text;
  Var
     A,B: Array[1..100] of Real;
     m: Integer;
     fxz: f;
  Begin
     Assign(fxz,'c:\AA\AB.txt');
     Reset(fxz);
     m:=0;
     While not eof(fxz) do
       Begin m:=m+1; Readln(fxz,A[m],B[m]);
          End;
       Close (fxz);
          End.
```

Massiv qiymatlarini biron matn fayliga yozib qo'yish uchun *Assign*, *Rewrite*, *Append*, *Write* va *Close* funksiyalari ishlatiladi.

Assign- fayl o'zgaruvchisi bilan asosiy fayl o'rtasida a'loqa o'rnatadi va u quyidagicha yoziladi:

Assign(fayl o'zgaruvchisi, diskdagi fayl joyi va nomi);

Append- Fayldan yozish uchun joy tayyorlaydi.

Append(fayl o'zgaruvchisi);

Write- o'zgaruvchi qiymatini fayl o'zgaruvchisiga uzatadi va faylga yozadi. Write(fayl o'zgaruvchisi, o'zgaruvchilar);

Close- ochilgan faylni yopadi.

Close (fayl o‘zgaruvchisi);

Misol 3. Yuqoridagi C: diskdagi AA katalogdagi AB fayldagi ma’lumotlarni A va B massivlarga o‘qib shu massiv mos elementlarini qo‘shib C massivni tashkil qiling va A,B,C massivlarini ABC nomli faylga joylashtiring.

```
Program FABC;
  Type
     f=text;
  Var
     A,B,C: Array[1..100] of Real;
     i,m: Integer;
     fax: f;
  Begin
     Assign(fax,'c:\AA\AB.txt');
     Reset(fax);
     m:=0;
     While not eof(fax) do
       Begin    m:=m+1;
                Readln(fax,A[m],B[m]);
           End;
       Close (fax);
     Assign(fax,'c:\AA\ABC.txt');
     Rewrite(fax);
     Append(fax);
     For i:=to m do
      Begin  Write(fax,i,A[i],b[i],c[i]);
          Writeln(fax);
     End;
     Close(fax);
End.
```

15.2. Xatolarni qayta ishlash

Dastur bajarilishi jarayonida xatolar kelib chiqishi mumkin. Masalan foydalanuvchi noto'g'ri ma'lumotlar kiritishi yoki kerakli faylni o'chirishi mumkin.

Dastur bajarilishini buzilishi istisno deb ataladi. Istisnolarni dasturga avtomatik qo'shiluvchi kod bajaradi. Shu bilan birga Delphi dasturga mustaqil istisnoni qayta ishlash imkonini beradi.

Istisnoni qayta ishlash instruksiyasi quyidagichadir:

```
try
//bu yerda istisno holatni yuzaga keltiruvchi instruksiyalar
except // istisnolarni qayta ishlash seksiyasi boshlanishi
on IstisnoTipi1 do QaytaIshlash1;
on IstisnoTipi2 do QaytaIshlash2;
on IstisnoTipiJ do QaytaIshlashJ;
else
// qolgan istisnolarni qayta ishlash instruksiyalari
end;
```

bu yerda:

- *try* — istisno holatni yuzaga keltiruvchi instruksiyalar oldidan qo'yiladigan va bu istisnolarni qayta ishlashni dastur bajarishini bildiruvchi kalit so'z;
- *except* — istisnolarni qayta ishlash seksiyasi boshlanishini bildiruvchi kalit so'z. Agar istisno holat yuz bersa shu seksiya instruksiyalari bajariladi;
- *on* — istisno tipini ko'rsatuvchi kalit so'z. Bu tipdagi istisno qayta ishlovchi instruksiyalar *do* so'zidan keyin keladi;
- *else* — tipi *except* seksiyasida ko'rsatilmagan istisnolarni qayta ishlashni ta'minlovchi instruksiyalar oldidan qo'yiladigan kalit so'z.

Yuqorida aytilganidek istisno holatning asosiy xarakteristikasi uning tipidir. Quyidagi jadvalda eng ko'p yuzaga keluvchi istisnolar va ularni keltirib chiqarishi mumkin bo'lgan sabablar keltirilgan.

Istisno turi	Sababi
EZeroDivide	Bo‘lish amalini bajarishda, agar bo‘luvchi nolga teng bo‘lsa;
EConvertError	Tip o‘zgartirishda, agar qiymatni kerakli tipga o‘zgartirish mumkin bo‘lmasa, masalan satrni songa o‘zgartirishda;
EFilerError	Faylga murojaat qilishda. Eng ko‘p sabablardan biri kerakli faylning yo‘qligi yoki diskdan foydalanilganda kerakli diskning yo‘qligi;

Quyidagi dasturda istisnoni qayta ishlash ko‘rsatilgan.

```
var
u: real; // kuchlanish
r: real; // qarshilik
i: real; // tok
s:string;
begin
s:= ‘ ‘;
try
// istisno(xato) keltirib chiqaruvchi instruksiyalar
u := StrToFloat(Edit1.Text);
r := StrToFloat(Edit2.Text);
i := u/r;
except // istisnolarni qayta ishlash seksiyasi
onEZeroDivide do // nolga bo‘lish
begin
writeln(‘Qarshilik nolga teng bo‘lolmaydi!’);
exit;
end;
on EConvertError do // Satrni songa aylantirish xatoligi
begin
writeln (‘Kuchlanish va qarshilik ‘ +
‘son bo‘lishi kerak. ‘ +#13+
‘Kasr sonni yozishda vergul ishlating.’;
```

```
exit;
end;
end;
s := FloatToStr(i) + ' A';
writeln(s);
end;
end.
```

Keltirilgan dasturda istisno holat tok kattaligini hisoblashda kelib chiqishi mumkin. Agar qarshilik uchun nol qiymat berilsa i:=u/r instruksiya bajarilishida *EzeroDivide* istisno holat yuzaga keladi.

Agar son qiymati noto'g'ri berilsa, masalan butun va kasr qiymatlarni ajratish uchun vergul o'rniga nuqta berilsa *EconvertError* istisno holat yuzaga keladi. Ikkala istisno bir xil qayta ishlanadi: ma'lumot chiqariladi va protsedura o'z ishini tugatadi.

Nazorat savollari:

1. Dasturlash nuqtai nazaridan fayl nima?
2. Fayl qaerda e'lon qilinadi?
3. Fayllar bilan ishlovchi maxsus funksiya va protseduralarni ko'rsating.
4. Istisnoni qayta ishlash instruksiyasi qanday ko'rinishga ega?
5. Istisnolar tiplarini ko'rsating.

16 BOB. KONTEYNER SINFLAR STANDART BIBLIOTEKASI

16.1. STL tarkibi

Biblioteka yadrosi uchta elementdan iborat: konteynerlar, algoritmlar va iteratorlar.

Konteynerlar (*containers*) – bu boshqa elementlarni saqlovchi ob'yektlar. Masalan, vektor, chiziqli ro'yxat, to'plam.

Assotsiativ konteynerlar (*associative containers*) kalitlar yordamida ularda saqlanadigan qiymatlarni tezkor olish imkonini yaratadi.

Har bir sinf – konteynerida ular bilan ishlash uchun mo'ljallangan funksiyalar to'plami aniqlangan. Masalan, ro'yxat elementlarini kiritish, chiqarish va qo'shish funksiyalarini o'z ichiga oladi.

Algoritmlar (*algorithms*) konteyner ichidagilar ustidan operatsiyalar bajaradi. Konteyner ichidagilarni initsializatsiyalash, qidirish, saralash va almashtirish uchun algoritmlar mavjud. Ko'p algoritmlar konteyner ichidagi elementlarni chiziq ro'yxatini ifodalovchi ketma-ketlik (*sequence*) bilan ishlash uchun mo'ljallangan.

Iteratorlar (*iterators*) – bu konteynerga nisbatan ko'rsatkich sifatida bo'lgan ob'yektlar. Ular massiv elementlariga ruxsat oluvchi ko'rsatkichlar kabi konteyner ichidagiga ruxsat olish imkonini beradi.

Sinf-konteynerlar

STL da quyidagi sinf-konteynerlar aniqlangan:

Asosiy konteynerlar

vector <vector.h> dinamik massiv

list <list.h> chiziqli ro'yxat

deque <deque.h> ikki tarafli tartib

set <set.h> to'plam

multiset <set.h> har bir elementi noyob bo'lishi shart emas to'plam

map <map.h> kalit qiymat juftlikni saqlash uchun assotsiativ ro'yxat. Bunda har bir kalit bitta qiymat bilan bog'langan.

multimap <map.h> har bir kalit bilan ikkita yoki ko'proq qiymatlar bog'langan

Hosila konteynerlar

stack <stack.h> stek

queue <queue.h> tartib

priority_queue <queue.h> birinchi o'rindagi tartib

16.2. Sinf-konteynerlarda konstruktorlar

Ixtiyoriy sinf-konteyner ko'rsatilmagan holda konstruktor va destruktorni nusxalovchi konstruktorga ega.

Masalan, vektor sinf-konteynerning konstruktori va destruktori:

vector<elem> c	bitta ham elementga ega bo'lmagan bo'sh vektorni yaratadi;
vector<elem> c1(c2)	ko'rsatilgan tipdagi boshqa vektorning nusxasini yaratadi (barcha elementlarni nusxasini oladi);
vector<elem> c(n)	konstruktor orqali ko'rsatilmagan holda yaratilgan n elementli vektorni yaratadi;
vector<elem> c(n,x)	x elementning n nusxalari yordamida initsializatsiya etilgan vektorni yaratadi;
~vector<elem>()	barcha elementlarni o'chiradi va xotirani bo'shatadi.

Ixtiyoriy ob'yekt uchun ko'rsatilmagan holda konteynerda saqlanuvchi konstruktor mavjud bo'lishi shart. Undan tashqari, ob'yekt uchun < va == operatorlar aniqlanishi lozim.

16.3. Iteratorlar

Iteratorlar bilan ko'rsatkichlar kabi ishlash mumkin. Ularga *, inkrement, dekrement operatorlarini qo'llash mumkin. Iterator tipi sifatida har xil konteynerlarda aniqlangan iterator tipi e'lon qilinadi.

Iteratorlarning beshta tipi mavjud:

1. Kiritish iteratorlari (*input_iterator*) tenglik, nomini o'zgartirish va inkrement operatsiyalarni qo'llaydi.

==, !=, *i, ++i, i++, *i++

Kiritish iteratsiyasining maxsus holati *istream_iterator* idan iborat.

2. Chiqarish iteratorlari (*output_iterator*) o'zlashtirish operatorining chap tarafidan imkon bo'lgan ismni o'zgartirish va inkrement operatsiyalarda qo'llaniladi.

++i, i++, *i=t, *i++=t

Chiqarish iteratsiyasining maxsus holati *ostream_iterator* idan iborat.

3. Bitta yo'nalishdagi iteratorlar (*forward_iterator*) kiritish/chiqarish operatsiyalarining barchasini qo'llaydi, bundan tashqari chegarasiz o'zlashtirishning imkonini beradi.

==, !=, =, *i, ++i, i++, *i++

4. Ikki yo'nalishdagi iteratorlar (*biderectional_iterator*) *forward-iterator* larning barcha xususiyatlariga ega, bundan tashqari, konteynerni ikkita yo'nalishi bo'yicha o'tish imkonini beradigan qo'shimcha dekrement (--i, i--, *i--) operatsiyasiga ega.

5. Ixtiyoriy ruxsatga ega bo'lgan iteratorlar (*random_access_iterator*) *biderectional-iterator* larning barcha xususiyatlariga ega, bundan tashqari solishtirish va manzil arifmetikasi operatsiyalarini qo'llaydi.

i+=n, i+n, i-=n, i-n, i1-i2, i[n], i1<i2, i1<=i2, i1>i2, i1>=i2

Shuningdek, STLda teskari iteratorlar (*reverse iterators*) qo'llaniladi. Ketma-ketlikni teskari yo'nalishda o'tuvchi ikki yo'nalishli yoki ixtiyoriy ruxsatga ega bo'lgan iteratorlar teskari iteratorlar bo'lishi mumkin.

16.4. Xotirani taqsimlovchilar, predikatlar va solishtirish funksiyalari

Konteynerlarga, algoritmlarga va STLdagi iteratorlarga qo'shimcha bir nechta standart komponentalar ham qo'llaniladi. Ulardan asosiylari esa xotira taqsimlovchilar, predikatlar va solishtirish funksiyalaridir.

Har bir konteynerda uning uchun aniqlangan va konteyner uchun xotirani belgilash jarayonini boshqaradigan xotira taqsimlovchisi (*allocator*) mavjud.

Ko'rsatilmagan holda esa xotira taqsimlovchisi *allocator* sinf ob'yektidir. Xususiy taqsimlovchini tavsiflash mumkin.

Ba'zi bir algoritmlar va konteynerlarda muhim tipdagi predikat ataluvchi funksiyalar ishlatiladi.

Predikatlar unar va binar bo'lishi mumkin. U yoki bu qiymatni olishni aniq shartlari dasturchi orqali aniqlanadi.

Unar predikatlarning tipi – *UnPred*, binar predikatlarning tipi esa - *BinPred*. Argumentlar tipi konteynerda saqlanuvchi ob'yektlar tipiga mos.

Ikkita elementni solishtirish uchun binar predikatlarning maxsus tipi aniqlangan. U solishtirish funksiyasi (*comparison function*) deyiladi. Agarda birinchi element ikkinchisidan kichik bo'lsa, unda funksiya rost qiymatni qaytaradi. *Comp* tip – funksiya tipidir.

STL da ob'yekt-funksiyalar o'ziga xos ahamiyatga ega.

Ob'yekt-funksiyalar – bu sinfda «kichik qavslar» () operatsiyasi aniqlangan sinf nusxalaridir. Ba'zi bir hollarda funksiyalarni ob'yekt-funksiyalarga almashtirish qulay deb hisoblanadi.

Ob'yekt-funksiya funksiya sifatida ishlatilsa, unda uni chaqirish uchun *operator()* operatori ishlatiladi.

16.5. Vektor konteynerlari

STL da *vector* vektor dinamik massiv sifatida aniqlanadi. Massiv elementlariga indeks orqali ruxsat beriladi.

Vector sinfida quyidagi konstruktorlar aniqlangan:

- Birinchi shakl bo'sh vektor konstruktorini tavsiflaydi.
- Konstruktor vektorning ikkinchi shaklida elementlar soni – bu har bir element qiymatiga teng. Qiymat parametri ko'rsatilmagan holdagi qiymat bo'lishi mumkin.
- Konstruktor vektorning uchinchi shakli – bu nusxalash konstruktori.

- To'rtinchi shakli – bosh va oxirgi iteratorlar orqali elementlar diapazonini o'z ichiga olgan konstruktor vektor.

Vektorda saqlanadigan ixtiyoriy ob'yekt uchun ko'rsatilmagan holda konstruktor aniqlash zarur. Bundan tashqari, ob'yekt uchun < va == operatorlar aniqlanishi lozim.

Vektor sinfi uchun quyidagi solishtirish operatorlari mavjud:

==, <, <=, !=, >, >=.

Bundan tashqari, *vector* sinf uchun [] indeks operatori aniqlangan.

Ikki yo'nalishli tartib

Deque – vektor kabi, ixtiyoriy ruxsat iteratorlarni qo'llovchi ketma-ketlik ko'rinishi. Bundan tashqari, u o'zgarmas vaqtda boshida yoki oxirida kiritish va o'chirish operatsiyalarini qo'llaydi. O'rtada kiritish va o'chirish chiziqli vaqtni egallaydi. Xotirani boshqarishiga ishlov berish esa vektorlar kabi avtomatik ravishda bajariladi.

Ro'yxat(List)

Ro'yxat – ikki yo'nalishli iteratorlarni qo'llaydigan hamda kiritish va o'chirish operatsiyalarini o'zgarmas vaqtda ketma-ketlikni ixtiyoriy joyida bajaradigan, shuningdek, xotirani boshqarishiga avtomatik ravishda ishlov beruvchi ketma-ketlik ko'rinishidir. Vektorlar va ikkitarafli tartiblardan farqi shundaki elementlar ro'yxatiga tez va ixtiyoriy ruxsat qo'llanmaydi, lekin ko'pgina algoritmlarga esa ketma-ketlik ruxsati zarur.

16.6. Assotsiativ konteynerlar

Assotsiativ massiv juft qiymatlardan iborat. Kalit (*key*) deb atalgan bitta qiymatni bilib (*mapped value*) aks etuvchi qiymat deb atalgan ikkinchi qiymatga ruxsat olishimiz mumkin.

Assotsiativ massivni massiv indekslari butun tiplardan iborat bo'lmagan massiv sifatida tavsiflashi mumkin:

V& operator[](const K&)

bu erda K ga mos keluvchi V ga ilovani qaytaradi.

Assotsiativ konteynerlar – bu assotsiativ massivning umumiy tushunchasi.

Map assotsiativ konteyner – bu kalit yordamida qiymatga tez ega bo'lish imkonini yaratadigan juftlik (kalit, qiymat) ketma-ketligi. *Map* konteyneri ikki yo'nalishli iteratorni tavsif etadi.

Map assotsiativ konteyneri kalit tiplari uchun "<" operatsiyasi mavjudligini talab qiladi. U kalit bo'yicha saralangan o'z elementlarini saqlaydi. Saralash almashuvi esa tartib bo'yicha bajariladi.

Map sinfida quyidagi konstruktorlar aniqlangan:

Birinchi shakli bo'sh assotsiativ konteynerning konstruktorini tavsiflaydi. Ikkinchi shakli – konstruktor nusxasi, uchinchisi – elementlar diapazonini qamrab olgan assotsiativ konteynerning konstruktori.

O'zgartirish operatsiyasi aniqlangan:

map& operator=(const map&);

Quyidagi operatsiyalar aniqlangan: ==, <, <=, !=, >, >=.

Map da kalit/qiymat juftliklar *pair* tipdagi ob'yektlar ko'rinishida saqlanadi.

Kalit/qiymat juftliklarni faqatgina *pair* sinf konstruktorlari yordamida, balki *pair* tipdagi ob'yektlarni yaratuvchi va ma'lumotlar tiplaridan parametrlar sifatida foydalanuvchi *make_pair* funksiya yordamida yaratish mumkin.

Assotsiativ konteyner uchun o'ziga xos operatsiya – bu ([]) indeksatsiyalash operatsiyasi yordamida assotsiativ qidiruvdir.

mapped_type& operator[](const key_type& K);

Set to'plamini assotsiativ massiv sifatida ko'rish mumkin. Unda qiymatlar ahamiyatga ega emas, shuning uchun faqat kalitlarni ko'rsatish mumkin.

To'plamlar assotsiativ massiv kabi T tip uchun (<) "kichik" operatsiyani mavjudligini talab etadi. U o'z elementlarini saralangan holda saqlaydi. Saralash almashuvi esa tartib bo'yicha bajariladi.

16.7. Konteyner usullari

Iteratorlarni olish usullari:

begin() birinchi elementga ko'rsatadi;

end() oxiridan keyingi elementga ko'rsatadi;
rbegin() teskari ketma-ketlikdagi birinchi elementni ko'rsatadi;
rend() teskari ketma-ketlikdagi oxirgidan keyingi elementni ko'rsatadi

Elementlarga ruxsat:

front() birinchi elementga ilova;
back() oxirgi elementga ilova;
operator[](i) tekshirishsiz indeks bo'yicha ruxsat;
at(i) tekshirish bilan indeks bo'yicha ruxsat.
front() birinchi elementga ilova;

Elementlarni kiritish usullari:

insert(p,x) r ko'rsatgan elementdan oldin x ni qo'shish
insert(p,n,x) r dan oldin x ning n nusxalarini qo'shish
insert(p,first,last) r dan oldin [first:last]dagi elementlarni qo'shish
push_back(x) oxiriga x larni qo'shish
push_front(x) yangi birinchi elementni qo'shish (ikkita uchga ega bo'lgan tartiblar va ro'yxatlar uchun)

Elementlarni o'chirish usullari:

erase(p) r pozitsiyadagi elementni o'chirish;
erase(first,last) [first:last]dan elementlarni o'chirish;
pop_back() oxirgi elementni o'chirish;
pop_front() birinchi elementni o'chirish (ikkita uchga ega bo'lgan tartiblar va ro'yxatlar uchun)

O'zlashtirish usullari:

operator=(x) konteynerga x konteynerni elementlari o'zlashtiriladi;
assign(n,x) konteynerga x elementning n nusxasi o'zlashtiriladi (assotsiativ bo'lmagan konteynerlar uchun);
assign(first,last) [first:last] diapazondagi elementlarni o'zlashtirish

Assotsiativ usullari:

find(elem) elem qiymatga ega bo'lgan birinchi elementni pozitsiyasi topadi;
lower_bound(elem) element qo'yish mumkin bo'lgan birinchi pozitsiyani topadi;

upper_bound(elem) element qo'yish mumkin bo'lgan oxirgi pozitsiyani topadi;

equal_range(elem) element qo'yish mumkin bo'lgan birinchi va oxirgi pozitsiyalarni topadi;

Assotsiativ usullar:

operator[](k) k kalitli elementga ruxsat;

find(k) k kalitli element pozitsiyasini topadi;

lower_bound(k) k kalitli elementning birinchi pozitsiyasini topadi;

upper_bound(k) k dan katta bo'lgan kalitli birinchi elementni to'padi;

equal_range(k) k kalitli elementni *lower_bound* (quyi chegarasini) va *upper_bound* (yuqori chegarasini) topadi.

Boshqa usullar:

size() elementlar soni;

empty() konteyner bo'sh yoki bo'sh emasligini aniqlash;

capacity() vektor uchun ajratilgan xotira (faqat vektorlar uchun);

reserve(n) n elementdan iborat bo'lgan konteyner uchun xotira ajratadi;

swap(x) ikkita konteynerlarni joyini almashtirish;

==, !=, < solishtirish operatorlari.

Nazorat savollari:

1. Biblioteka yadrosi qanday elementlardan iborat?
2. Har qanday konteyner qanday konstruktorlarga ega?
3. Iteratorlar tiplarini ko'rsating.
4. Assotsiativ massivlar qanday xususiyatlarga ega?
5. Elementlarga murojaat usullarini ko'rsating.
6. Elementlarni o'chirish usullarini ko'rsating.
7. Konteyner hajmini o'zgartirish uchun qanday usuldan foydalaniladi?

17 BOB. STANDART ALGORITMLAR

17.1. Algoritmlar

Har bir algoritm funksiyalar shabloni yoki funksiyalar shabloni to'plami yordamida ifodalanadi. Shunday qilib, algoritm har xil tipdagi qiymatlarga ega bo'lgan har xil konteynerlar bilan ishlay oladi. Barcha algoritmlarni argumentlari [beg, end) yarim oraliqlar bo'ladi.

Algoritmlar <algorithm.h> sarlavxa faylda tavsiflangan.

Quyida ko'p ishlatiladigan STL funksiya-algoritmlari keltirilgan.

17.2. O'zgartirmaydigan algoritmlar

1. Oraliqdagi elementlarni o'zgartirmaydigan algoritm:

for_earch() – oraliqning har bir elementi uchun operatsiyalarni bajaradi;

find() – qiymatni oraliqdagi birinchi kirishini topadi;

find_if() – oraliqda predikatga birinchi moslashuvini topadi;

count() – qiymatni ketma-ketlikka kirishini hisoblaydi;

count_if() – oraliqda predikatni bajarilishini hisoblaydi;

min_element() – oraliqdagi eng kichik qiymat;

max_element() – oraliqdagi eng katta qiymat.

2. Oraliqdagi elementlarni boshqa oraliqga nusxasini olib o'tish algoritmlari:

copy() – birinchi elementdan boshlab oraliqni nusxasini oladi;

copy_bacwards() – oxirgi elementdan boshlab oraliqni nusxasini oladi;

replace_copy() – ko'rsatilgan qiymatga ega bo'lgan elementlarni almashtirib nusxasini oladi;

replace_copy_if() – predikatni bajarish jarayonida elementlarni almashtirgan holda oraliqni nusxasini oladi;

remove_copy() – ko'rsatilgan qiymatga ega bo'lgan elementlarni o'chirgan holda oraliqni nusxasini oladi;

remove_copy_if() – predikatni bajarish jarayonida elementlarni o'chirgan holda oraliqni nusxasini oladi;

unique_copy() – teng bo'lgan qo'shni elementlarni o'chirgan holda oraliqni nusxasini oladi;

rotate_copy() – nusxasini olish jarayonida elementlarni sikl bo'yicha suradi.

3. Ikkita oraliqni solishtirish algoritmlari:

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 algoritmlar:

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 saralangan oraliqni solishtirish algoritmi:

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

6. Ikkita saralangan oraliq ustidagi algoritmlar:

set_union() – oraliqlarni birlashtirish;

set_intersection() – oraliqlarni o'zaro kesishishi;

set_difference() – oraliqlarni ayirmasi;

set_symmetric_difference() – oraliqlarni simmetrik ayirmasi;

merge() – ikkita oraliqni birlashtirish.

17.3. O'zgartiruvchi algoritmlar

1. Oraliqda elementlarni almashtirish algoritmlari:

fill() – ko'rsatilgan qiymatdagi barcha elementlarni almashtiradi;
replace() – ko'rsatilgan qiymatli elementlarni almashtiradi;
replace_if() – predikat bajarilganda elementlarni almashtiradi;

2. Oraliqda elementlarni o'chirish algoritmlari:
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 algoritmlari:
reverse() – elementlarni ketma-ketlik tartibini teskarisiga almashtiradi;
rotate() – sikl 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() – leksikografik tartibdagi oldingi almashuv;

4. Oraliqdagi elementlarni saralash algoritmlari:
sort() – oraliqni saralaydi;
partial_sort() – oraliqning qismini saralaydi;
stable_sort() – teng elementlarni ketma-ketlik tartibini saqlagan holda oraliqni saralaydi;

5. Ikkita oraliqlar uchun o'zgartuvchi algoritmlar:
transform() – elementlarni modifikatsiyalaydi va nusxasini oladi hamda ikkita oraliqdagi elementlarni birlashtiradi;
swap_ranges() – ikkita oraliqni joyini almashtiradi.

17.4. Sonli algoritmlar

1. Interval uchun sonli algoritmlar:
accumulate() – elementlarning barcha qiymatlarini birlashtiradi (yig'indini, ko'paytmani va x.k. hisoblaydi);

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

adjacent_difference() – har bir elementni uni oldingi qiymati bilan birlashtirish (ayirmasi va x.k.larni hisoblaydi);

partial_sum() – har bir elementni ularning oldingi qiymatlari bilan birlashtiradi (barcha xususiy yig'indilarni va x.k. hisoblaydi).

17.5. Algoritmlardan foydalanish

For_each() algoritmidan foydalanish:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
 class IntPrint {
        public:
void operator() (int elem){
 cout<<elem<<endl; }
};
main() {
  vector<int> coll(5,4);
   IntPrint cmp;
    for_each(coll.begin(), coll.end(), cmp);
    cout<<num;
}
```

Find_if() algoritmidan foydalanish:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
class Greater {
 int k;
public:
Greater(int m){
```

```
 k=m
 };
bool operator() (int elem){
 return elem>k;  }
};
main() {
  vector<int> coll(5,4);
   vector<int>::iterator p;
   Greater cmp(5);
   p=find(coll.begin(), coll.end(), cmp);
   if (p=coll.end)
cout<<"element topilmadi";
else cout<<"element topildi";
}
```

Max_element() algoritmidan foydalanish:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
class CompInt {
 public:
bool lessthen (int elem1,int elem2) {
 return abs(elem1)<abs(elem2);
 } };
main() {
  vector<int> coll1(5,4);
  CompInt cmp;
   vector<int>::iterator p;
   p=max_element(coll.begin(), coll.end(),cmp);
   p.print();
}
```

Replace() va *replace_if()* algoritmlarini ishlatish:

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

```
# include<algorithm>
using namespace std;
class IsEven {
        public:
bool operator() (int elem){
 return elem %2==0;
 } };
main() {
  vector<int> coll;
 for(int i=0;i<6; i++) coll1.push_back(i);
 IsEven cmp;
replace (coll.begin(), coll.end(), 5, 42);
replace_if (coll.begin(), coll.end(), cmp,5);
  for(pos=coll1.begin();pos!=coll1.end();++pos)
 cout<<*pos<<" ";
}
```

Copy() algoritmidan foydalanish:

```
# include <iostream.h>
# include<vector>
# include<list>
# include<algorithm>
using namespace std;
main() {
  vector<int> coll1(5,4);
  list <int> coll2;
  copy (coll11.begin(), coll2.end(), back_inserter(coll2));
  copy (coll11.begin(), coll2.end(), ostream_iterator<int>(cout," "));
}
```

Fill() algoritmidan foydalanish:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
main() {
```

```
  vector<int> coll;
  fill_n (back_inserter(coll),9,"hello");
  fill (coll1.begin(), coll.end(),"again");
  vector<int>::iterator pos;
  for(pos=col1.begin();pos!=col1.end();++pos)
       cout<<*pos<<" ";
}
```

Nazorat savollari:

1. Hamma algoritmlar argumenti nimadan iborat?
2. Interval elementini o'zgartirmaydigan algoritmlar.
3. Ikki tartiblangan intervallar ustida algoritmlarni ko'rsating.
4. Interval elementlarini o'chirish uchun qaysi algoritmlardan foydalaniladi?
5. Sonli algoritmlarni ko'rsating.

18 BOB. XODISALAR ASOSIDA DASTURLASH

18.1. Komponentlar

C++Builder nafaqat ANSI C++ standarti kiritayotgan yangiliklarni qo'llab-quvvatlaydi, balki tilni yangi imkoniyatlar bilan boyitadi. Shuni tushunib olish muhimki, tilni kengaytirish hech qachon quruq maqsad bo'lib qolmagan va hamon standart C++ doirasida yozilgan matnlarni kompilyatsiya qilish mumkin. Biroq ilovalarni tez ishlab chiqish texnologiyasi (RAD) uchun C++Builder taqdim etgan imtiyozlardan to'liq foydalanish uchun kiritilgan til kengaytirishlarni qabul qilishga to'g'ri keladi.

Kengaytirishlarning ayrimlari (masalan, _classid) ni C++Builder asosan ichki foydalanish uchun rezervlaydi. Boshqa kengaytirishlar(_int8,_int6 va h.k.) ochiq-oydin tushunarli bo'lib turibdi, shuning uchun bu yerda ular ko'rib chiqilmaydi. Quyida C++ ning eng ahamiyatli kengaytirishlari ko'rib chiqiladi. Ular asosan tarkibli sinflarga mansub bo'lib, C++Builder muhitida ishlab chiqilayotgan ilovalarda muttasil uchrab turadi.

Komponentlar ko'p o'rinda, C++ standart sinflariga qaraganda, yuqoriroq darajadagi inkapsulyatsiyalashga erishadilar. Buni tugmachaga ega bo'lgan dialogni ishlab chiqish kabi oddiy misolda ko'rib chiqamiz. Windows uchun namunaviy C++ dasturida tugmachani «sichqoncha» bilan bosish natijasida WM_LBUTTONDOWN xabarining generatsiyasi sodir bo'ladi. Bu xabarni dastur yo *switch* operatorida yoki chaqiriqlar jadvali (RESPONCE_TABLE) ning tegishli satrida «tutib olish»i, keyin esa ushbu xabarga javob protsedurasiga uzatishi kerak.

C++Builder o'zlashtirilishi qiyin bo'lgan bu kabi dasturlash o'yinlariga chek qo'ydi. Komponenta tugmachasi avvaldanoq unga *OnClick* voqeasi bilan bosishga javob beradigan qilib dasturlangan. Bu o'rinda talab qilinayotgan narsa – tayyor metodni tanlab olish (yoki o'zinikini yozish) hamda *Ob'yektlar Inspektori* yordamida berilgan voqea-hodisaga ishlov bergichga kiritish lozim.

18.2. Komponentli sinflarni e'lon qilish

C++Builder tarkibiga kiradigan Vizual Komponentalar Kutubxonasi – VCL sinflarining ilgarilovchi e'lonlari *declspec* modifikatoridan foydalanadi:

_declspec(<spesifikator>)

Bu kalit-so'z nafaqat bevosita modifikatsiyalanayotgan e'lon oldidan, balki e'lonlar ro'yxatining to'g'ri kelgan yerida paydo bo'lishi mumkin, bunda spetsifikator quyidagi qiymatlaridan birini qabul qiladi:

delphiclass - u *TObject* sinfiga tegishli VCL ning bevosita yoki bilvosita hosilalarining ilgarilovchi e'loni uchun qo'llaniladi. U VCL ning RTTI, konstruktorlar, destruktor va istisnolar bilan muomalasida muvofiqlik qoidalarini belgilaydi.

delphireturn - u *Currency*, *AnsiString*, *Variant*, *TDateTime* va *Set* sinflariga tegishli VCL ning bevosita yoki bilvosita hosilalarining ilgarilovchi e'loni uchun qo'llaniladi. U VCL ning parametrlar va a'zo funksiyalarining qaytarilayotgan qiymatlari bilan muomalasida muvofiqlik qoidalarini belgilaydi.

Pascal *implementation* tarkibli sinf Object Pascal tilida ishga tushirilganini ko'rsatadi.

VCL sinf quyidagi cheklanishlarga ega:

- Virtual bazaviy sinflarga vorislik qilish man etilgan.
- Tarkibli sinflarning o'zlari vorislik uchun bazaviy sinf sifatida xizmat qila olmaydi.
- Tarkibli ob'yektlar uyumining dinamik xotirasida *new* operatori yordamida yaratiladi.

18.3. Xususiyatlarni e'lon qilish

C++Builder tarkibli sinflar xususiyatlarini identifikatsiya qilish uchun *_property* modifikatoridan foydalaniladi. Xususiyatni tavsiflash sintaksisi quyidagi ko'rinishga ega:

property<xususiyat turi > <xususiyat nomi >={<atributlar ro'yxati >};

bu yerda atributlar ro'yxati quyidagi xususiyatlar atributlarining sanog'iga ega:

write - <ma'lumotlar a'zosi yoki yozuv metodi> ma'lumotlar a'zosiga qiymat berish usulini aniqlaydi;

read - <ma'lumotlar a'zosi yoki o'qish metod > ma'lumotlar a'zosining qiymatini olish usulini aniqlaydi;

default - <bul konstantasi > **.dim* kengayishli shaklga ega bo'lgan yashirin xususiyatlar qiymatini saqlashni ruxsat beradi yoki man etadi;

stored - <bul konstantasi yoki funksiya> **.dfm* kengayishli shaklga ega bo'lgan faylda xususiyat qiymatini saqlash usulini aniqlaydi.

C++Builder ilovani loyixalash bosqichida Ob'yektlar Inspektori tomonidan aks ettiriladigan komponentalar xususiyatlarini spetsifikatsiyalash uchun *_published* modifikatoridan foydalaniladi. Agar komponentaning ishlab chiquvchisi biron-bir xususiyat qiymatini modifikatsiyalashga ruxsat berishini xoxlab qolsa, bu xususiyat *_published* sifatida e'lon qilinmaydi.

Ushbu kalit-so'z bilan aniqlanayotgan ko'rimlilik qoidalari *public* sifatida e'lon qilingan ma'lumotlar a'zolari, metodlar va xususiyatlarning ko'rimlilik qoidalaridan farq qilmaydi. Yagona farq shundaki, dasturning ishlash paytida *Ob'yektlar Inspektoriga* RTTI axboroti uzatiladi.

C++Builder voqealar ishlatgichlari funksiyalarining e'loni uchun *_closure* modifikatoridan foydalanadilar:

<tur>(_closure*<name>)(<parametrlar ro'yxati >)

Bu kalit-so'z funksiya ko'rsatkichini *name* nomi bilan aniqlaydi. Oddiy funksiyaning 4 baytli adresli ko'rsatkichidan farqli o'laroq (bu ko'rsatkich CS:IP kod registrlariga uzatiladi), 8 baytli *_closure* yana yashirin parametrni ham uzatadi (joriy sinf ekzemplyariga *this* o'zgaruvchan ko'rsatkichi).

8 baytli ko'rsatkichlarning kiritilishi, nafaqat aniqlangan sinfning biron-bir funksiyasini chaqirib olish imkonini beradi, balki ushbu sinfning aniqlangan ekzemplyaridagi funksiyaga murojaat qilish imkonini ham beradi. Bu qobiliyat Object Paskaldan o'zlashtirilgan edi,

_closure esa Vizual Komponentalar Kutubxonasidagi voqealar mexanizmini ishga tushirishda havodek zarur bo'lib qoldi.

18.3. Funksiyalarning tez chaqirilishi

Parametrlari protsessorli registrlar orqali uzatiladigan funksiyalarni e'lon qilishda *_fastcall* modifikatori qo'llaniladi:

<qaytarilayotgan tur >_fastcall<name>(<parametrlar ro'yxati >)

Bu kalit-so'z *name* nomli dastlabki uchta turlashtirilgan parametr (ro'yxat bo'yicha chapdan o'ngga) stek orqali emas, balki AX, BX va DX protsessorli registrlar orqali uzatilishini aniqlaydi.

Agar parametr qiymati registrga sig'masa, ya'ni parametr orqali suzuvchi nuqtali sonlarni, tuzilmalar va funksiyalarni uzatishda u qo'llanilmaydi.

Xolisona aytganda, funksiyalarning tez chaqirilishi C++Builder kompilyatoriningginа vazifasiga kirmaydi. Voqealarga ishlov berish funksiyalarini e'lon qilishda *_fastcall* ning qo'llanishiga alohida e'tibor berish kerak. Bu voqealarni C++Builder avtomatik tarzda generatsiya qiladi.

18.4. Nomlar fazosi

Oddiy ilovalarning ko'pi dastlabki dastur matniga ega bo'lgan bir nechta fayldan iborat. Bu fayllar dasturchilar guruhi tomonidan yaratilishi va xizmat ko'rsatilishi mumkin.

Pirovard natijada barcha fayllar birga to'planadi va tayyor ilovani yig'ishdan iborat bo'lgan so'nggi protseduradan o'tadi. An'anaviy tarzda qabul qilinishicha, biron bir lokal soxa (funksiya, sinf tanasi yoki translyatsiya moduli) ga kiritilmagan barcha nomlar umumiy global ismlarni bo'lib olishadi.

Shuning uchun, agar ayrim modullarni yig'ish jarayonida nomlar takroran aniqlangani ayon bo'lib qolsa, bu holda har bir nomni qandaydir yo'l bilan farqlash zarurligini talab qiladi. C++ da bu

muammoning yechilishi nomlar fazosi(*namespace*) mexanizmi zimmasiga yuklatilgan.

Bu mexanizm ilovani bir necha tarmoq tizimlar (tizimchalar) ga bo'lib tashlash imkonini beradi, bunda har bir tarmoq tizim nomlarini tanlashda erkin ish tutadi, hamda uning muallifi xuddi shunday ismlardan biron boshqa kimsa foydalanishi mumkinligiga qayg'urmasa ham bo'ladi. Har bir tarmoq tizim global nomlar umumiy fazosida o'zining paydo bo'lganini *namespace* kalit-so'zidan keyin kelgan unikal identifikator yordamida identifikatsiya qiladi:

namespace<identifikator> {[<e'lon qilish >]}

Identifikatsiya qilingan nomlar fazosi elementlariga kirishning uchta usuli mavjud:

- Konkret elementga ochiq-oydin kirish kvalifikatsiyasi:

ALPHA :: vart;

//ALPHA BETA::F1dagi o'zgaruvchiga kirish;

//BETA dagi o'zgaruvchiga kirish

- Barcha elementlarga kirish:

using namespace::ALPHA;

//ALPHA dagi barcha nomlarga kirish

Nomlarning lokal fazosida yangi identifikatorning e'lon qilinishi:

using :: new_name;

//identifikatorning qo'shilishi

18.5. Ochiq e'lonlar

Odatda bitta parametrli konstruktor e'lon qilingan sinf ob'yektlariga turlari avtomatik tarzda (yashirish holda) o'z sinfi turiga qayta o'zgaradigan qiymatlarni berish mumkin. Konstruktorni e'lon qilishda *explicit* konstruktoridan foydalanish mumkin:

explicit<konstruktorni e'lon qilish >

Bu holda berilgan sinf konstruktorlarini *explicit* kalit-so'z bilan e'lon qilishda sinfning barcha ob'yektlariga faqat shunday qiymatlarni berish mumkinki, bu qiymatlar turlari o'z sinfi turiga ochiq-oydin qayta o'zgaradigan bo'lishi kerak.

```
class X
public:
      explicit X(int);
      explicit X(const char*, int =0); };
void f(X arg) (
      X a = X (1) ;
      X b = X("satr",0);
       a = X(2);
      }
```

Konstruktorlarning ochiq-oydin e'lonlari shuni talab qiladiki, nom berish operatorlaridagi qiymatlar qaysi sinfiy tur ob'yektlariga berilgan bo'lsa, ular xuddi shu sinfiy turga qayta o'zgartirilishini talab qiladi.

18.6. O'zgaruvchan e'lonlar

Fon masalasi, uzish ishlatgichi yoki kiritish-chiqarish porti tomonidan o'zgartirilishi mumkin bo'lgan o'zgaruvchini e'lon qilishda *volatile* modifikatori qo'llaniladi:

volatile<tur><ob'yekt nomi>;

C++da *volatile* kalit-so'zining qo'llanilishi sinflar va a'zo-funksiyalarga ham tegishlidir. Bu kalit-so'z ko'rsatilgan ob'yekt qiymatiga nisbatan taxminlar qilishni kompilyatorga ta'qiqlaydi, chunki bunday qilinsa, ushbu ob'yektni o'z ichiga olgan ifodalarni hisoblashda, uning qiymati har bir daqiqada o'zgarib ketishi mumkin. Bundan tashqari o'zgarib turadigan o'zgaruvchi registr modifikatori bilan e'lon qilinishi mumkin emas. Quyidagi misol ticks o'zgaruvchisini vaqtli uzilishlar qayta ishlagichi modifikatsiya qiladigan taymerni ishga tushirishga misol bo'la oladi.

```
volatile int ticks;
void timer( ) // Taymer funksiyasini e'lon qilish
ticks++;
void wait (int interval)
ticks =0;
while (ticks < interval); // Kutish sikli
}
```

Aytaylik, uzilishni qayta ishlatgichi – *timer* real vaqt soatidagi apparat uzilishi bilan tegishli tarzda assotsiatsiya qilindi. Ticks o'zgaruvchisining qiymati ushbu qiymat parametri tomonidan berilgan vaqt intervaliga teng kelmaguncha, *wait* protsedurasi kutish siklini ishlataveradi. C++ kompilyatori sikl ichidagi har bir qiyoslash oldidan *volatile* ticks o'zgaruvchisining qiymatini sikl ichidagi o'zgaruvchining qiymati o'zgarmaganiga qaramay, ortiqcha yuklashi lozim. Ayrim optimallashtiruvchi kompilyatorlar bunday «xavfli» xatoga yo'l qo'yishlari mumkin.

Xatto konstantali ifodaga kirganiga qaramay o'zgartirilishi mumkin bo'lgan o'zgaruvchan o'zgaruvchining boshqa bir turi *mutable* modifikatori yordamida e'lon qilinadi:

mutable<o'zgaruvchining nomi>;

Mutable kalit-so'zining vazifasi shundan iboratki, u biron-bir sinf ma'lumotlari a'zolarini spetsifikatsiya qiladi, bunda ushbu ma'lumotlar a'zolari mana shu sinfning konstantali funksiyalari tomonidan modifikatsiya qilinishi mumkin bo'lishi kerak. Ma'lumotlar a'zosi *count* ni F1 konstantali funksiya modifikatsiya qiladigan misolni ko'rib chiqaylik:

```
class A {
public: mutable int count;
int F1 (int p=0) const // F1 funksiyasini e'lon qilish
count=p++;
return count; // PI count ni qaytarib beradi
}
void vain() {
A, a;
Cout<<a.F1(3)<<endi; //main 4 qiymatini beradi
```

Nazorat savollari:

1. Komponentalarning standart sinflardan farqi nimadan iborat?
2. Xususiyatlar qanday e'lon qilinadi?
3. Voqealar ishlatgichlar qanday e'lon qilinadi?
4. Nomlar fazosi mexanizmidan qanday foydalaniladi?
5. Ochiq oydin va o'zgaruvchan e'lonlar nima uchun ishlatiladi?

Adabiyotlar:

1. Mirziyoev Sh.M. Buyuk kelajagimizni mard va olijanob xalqimiz bilan birga quramiz. 2017

2. Mirziyoev Sh.M. Qonun ustuvorligi va inson manfaatlarini ta'minlash – yurt taraqqiyoti va xalq farovonligining garovi. 2017.

3. Mirziyoev Sh.M. Erkin va farovon, demokratik O'zbekiston davlatini birgalikda barpo etamiz. 2017

4. Straustrup B. Yazik programmirovaniya C++. Trete izdanie, M.: Binom, 2014.

5. Shmidskiy Ya.K. Programmirovaniye na yazike C++: Samouchitel. Uchebnoe posobie. Dialektika. 361 str, 2004 g.

6. Asharina N.A. Osnovi programmirovaniya na yazikax C, C++. Uchebniy kurs.M.: 2012 g.

7. Podbelskiy V.V. Yazik C++ – M.: Finansi i statistika, 2006.

8. Kondzyuba S.P., Gromov V.N. Delphi 6/7. Baza dannыx i prilojeniya. M.- Sankt-Peterburg - Kiev, 2002 g

9. Bjarne Straustrup. Programming: Principles and Practice Using C++ (2nd Edition). Person Education, Inc. 2014. second printing, January 2015.

10. Harry Hariom Choudhary, Bjarne M Straustrup. C++ Programming Professional.: Sixth Best Selling Edition for Beginner's & Expert's 2014.

11. Bjarne Straustrup. The C++ Programming Language, 4th Edition. Person Education, Inc. 2013. Third printing, April 2014.

12. Nazirov Sh.A., Qobulov R.V., Bobojanov M.R., Raxmanov Q.S. C va C++ tili. "Voris-nashriyot" MCHJ, Toshkent 2013. 488 b.

13. Horstmann, Cay S. C++ for everyone / Cay S. Horstmann. Printed in the United States of America - 2nd ed. 2010. – P. 562..

14. http://www.stroustrup.com/4th.html, http://www.cplusplus.com

15. http://acm.tuit.uz

16. http://acm.tuit.uz/forum/, http://acm.timus.ru

17. http://codeforces.com

MUNDARIJA

M. YU. DOSHANOVA,
A.M. MATYAKUBOVA

DASTURLASH

(O‘quv qo‘llanma)

Toshkent – «Aloqachi» – 2019

Muharrir: M.Mirkomilov
Tex. muharrir: A.Tog‘ayev
Musavvir: B.Esanov
Musahhiha: F.Tog‘ayeva
Kompyuterda sahifalovchi: Sh.To ‘xtamurodov

Nashr.lits. AI №176. 11.06.11.
Bosishga ruxsat etildi: 24.04.2019. Bichimi 60x841 /16.
Shartli bosma tabog‘i 9,25. Nashr bosma tabog‘i 8,75.
Adadi 60. Buyurtma № 58.