O’ZBEKISTON ALOQA, AXBOROTLASHTIRISH VA TELEKOMMUNIKATSIYA TEXNOLOGIYALARI DAVLAT QO’MITASI
TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI

«OBYEKTGA YO’NALTIRILGAN DASTURLASH TILLARI» FANIDAN MA’RUZALAR KURSI

Toshkent 2013

Mualliflar:
Kuchkorov T.A., TATU “ATDT” kafedrasi assistenti.
Egamberdiyev N.A., TATU “ATDT” kafedrasi assistenti.

«Obyektga yo’naltirilgan dasturlash tillari» fanidan ma’ruzalar kursi.

Fan bo’yicha ma’ruzalar mavzulari

Mundarija

1 – Ma’ruza	Asosiy konstruksiyalar. O‘zgaruvchilar. Operatorlar. Funksiyalar.....	4
2 – Ma’ruza	Massivlar va satrlar.....	18
3 – Ma’ruza	Ko‘rsatkichlar.....	23
4 – Ma’ruza	Murakkab ma’lumotlar turlari. Strukturalar.....	31
5 – Ma’ruza	Obyektli dasturlash asoslari. Ob’ektga yo’naltirilgan dasturlash.....	36
6 – Ma’ruza	Sinflar va obyektlar.....	46
7 – Ma’ruza	Sinflar va ko’rsatkichlar.....	63
8 – Ma’ruza	Sinflar orasida munosabatlar.....	69
9 – Ma’ruza	Sinflarda vorislik.....	78
10 – Ma’ruza	Sinflarda vorislik. Polimorfizm.....	89
11 – Ma’ruza	Standart amallarni qayta yuklash.....	96
12 – Ma’ruza	Funksiya va sinf shablonlari.....	107
13 – Ma’ruza	Oqimli sinflar.....	117
14 – Ma’ruza	Istisno holatlarni boshqarish.....	129
15 – Ma’ruza	Standart shablon sinflarkutubxonasi.....	136
16 – Ma’ruza	Sinf hodisalari	145

1-Ma’ruza. Asosiy konstruksiyalar. O‘zgaruvchilar. Operatorlar. Funksiyalar.

Reja:

1. Asosiy konstruksiyalar.
2. O‘zgaruvchilar.
3. Operatorlar.
4. Funksiyalar.

C++ alfavitiga quyidagi simvollar kiradi.

- Katta va kichik lotin alfaviti harflari (A,B,..,Z,a,b,…,z)
- Raqamlar: 0,1,2,3,4,5,6,7,8,9
- Maxsus simvollar: “ , {} | [] () + - / % \ ; ‘ . : ? < = > _ ! & * # ~ ^
- Ko‘rinmaydigan simvollar ("umumlashgan bo‘shlik simvollari"). Leksemalarni o‘zaro ajratish uchun ishlatiladigan simvollar (misol uchun bo‘shlik, tabulyasiya, yangi qatorga o‘tish belgilari).

Izohlarda, satrlarda va simvolli konstantalarda boshqa literallar, masalan rus harflari ishlatilishi mumkin.

C++ tilida besh xil turdagi leksemalar ishlatiladi: erkin tanlanadigan va ishlatiladigan identifikatorlar, xizmatchi so‘zlar, konstantalar(konstanta satrlar), amallar(amallar belgilari), ajratuvchi belgilar.

Identifikator. Identifikatorlar lotin harflari,ostki chiziq belgisi va sonlar ketma ketligidan iborat bo‘ladi. Identifikator lotin harfidan yoki ostki chizish belgisidan boshlanishi lozim.

Misol uchun:

A1, _MAX, adress_01, RIM, rim

Katta va kichik harflar farqlanadi, shuning uchun oxirgi ikki identifikator bir biridan farq qiladi.

Borland kompilyatorlaridan foydalanilganda nomning birinchi 32 harfi ,ba’zi kompilyatorlarda 8 ta harfi inobatga olinadi. Bu holda NUMBER_OF_TEST va NUMBER_OF_ROOM identifikatorlari bir biridan farq qilmaydi.

Xizmatchi so‘zlar. Tilda ishlatiluvchi ya’ni dasturchi tomonidan o‘zgaruvchilar nomlari sifatida ishlatish mumkin bo‘lmagan identifikatorlar xizmatchi so‘zlar yoki kalit so‘zlar deyiladi. Kompilyatorning texnik dokumentatsiyasida barcha zahiralangan so‘zlarning ro‘yxati turadi.

O‘zgaruvchilarni ta’riflash. C++ tilida o‘zgaruvchini aniqlash uchun компьyuterga uning tipi (masalan, int, char yoki float) hamda ismi xaqida haqida ma’lumot beriladi. Bu axborot asosida kompilyatorga o‘zgaruvchi uchun qancha joy ajratish lozim va bu o‘zgaruvchida qanday turdagi qiymat saqlanishi

mumkinligi haqida ma’lumot aniq bo‘ladi. O‘zgaruvchi nomi identifikator bo‘lib, xizmatchi so‘zlardan farqli bo‘lishi kerak.

Har bir yacheyka bir bayt o‘lchovga ega. Agar o‘zgaruvchi uchun ko‘rsatilgan tip 4 baytni talab qilsa, uning uchun to‘rtta yacheyka ajratiladi. Aynan o‘zgaruvchini tipiga muvofiq ravishda kompilyator bu o‘zgaruvchi uchun qancha joy ajratish kerakligini aniqlaydi.

Kompyuterda qiymatlarni ifodalash uchun bitlar va baytlar qo‘llaniladi va xotira baytlarda hisoblanadi.

O‘zgarmaslar turlari. O‘zgaruvchilar kabi o‘zgarmaslar ham ma’lumotlarni saqlash uchun mo‘ljallangan xotira yacheykalarini o‘zida ifodalaydi. O‘zgaruvchilardan farqli ravishda ular dasturni bajarilishi jarayonida qiymati o‘zgarmaydi. O‘zgarmas e’lon qilinishi bilan unga qiymat berish lozim, keyinchalik bu qiymatni o‘zgartirib bo‘lmaydi.

C++ tilida ikki turdagi, literal va nomlangan o‘zgarmaslar aniqlangan.

Literalli o‘zgarmaslar to‘g‘ridan-to‘g‘ri dasturga kiritiladi. Masalan:

```
int myAge =39;
```

Bu ifodada MyAge int tipidagi o‘zgaruvchi, 39 soni esa literal o‘zgarmasdir.

Maxsus belgilar. C++ kompilyatorida tekstlarni formatlovchi bir nechta maxsus belgilardan foydalaniladi. (Ulardan eng ko‘p tarqalgani jadvalda keltirilgan). Bu belgilarni dasturda ishlatishda «teskari slesh»dan foydalanamiz. Teskari sleshdan keyin boshqaruvchi belgi yoziladi. Masalan, tabulyasiya belgisini dasturga qo‘yish uchun quyidagicha yozuvni yozish kerak.

```
char tab ='\t';
```

Bu misoldagi char tipidagi o‘zgaruvchi \t qiymatini qabul qiladi. Maxsus belgilar axborotlarni ekranga, faylga va boshqa chiqarish qurilmalariga chiqarishda formatlash uchun qo‘llaniladi.

Maxsus '\' simvolidan boshlangan simvollar eskeyp simvollar deyiladi.Simvolli konstanta qiymati simvolning kompьyuterda qabul qilingan sonli kodiga tengdir.

Ma’lumotlarning butun son turi. Butun sonlar o‘nlik, sakkizlik yoki o‘n oltilik sanoq sistemalarida berilishi mumkin.

O‘nlik sanoq sistemasida butun sonlar 0-9 raqamlari ketma ketligidan iborat bo‘lib, birinchi rakami 0 bo‘lishi kerak emas.

Sakkizlik sanoq sistemasida butun sonlar 0 bilan boshlanuvchi 0-7 raqamlaridan iborat ketma ketlikdir.

O‘n oltilik sanoq sistemasida butun son 0x yoki 0X bilan boshlanuvchi 0-9 rakamlari va a-f yoki A-F harflaridan iborat ketma ketlikdir.

Masalan 15 va 22 o‘nlik sonlari sakkizlikda 017 va 026, o‘n oltilikda 0xF va 0x16 shaklda tasvirlanadi.

Ma’lumolarning uzun butun son turi.

Oxiriga l yoki L harflari qo‘yilgan o‘nlik, sakkizlik yoki o‘n oltilik butun son.

Ma’lumotlarning ishorasiz (unsigned) butun son turi:

Oxiriga u yoki U harflari qo‘yilgan o‘nlik, sakkizlik yoki o‘n oltilik oddiy yoki uzun butun son.

Ma’lumotlarning xaqiqiy son turi. Ma’lumotlarning xaqiqiy son turi olti qismdan iborat bo‘lishi mumkin: butun qism, nuqta, kasr qism, e yoki E belgisi, o‘nlik daraja, F yoki f suffikslari.

Masalan : 66. .0 .12 3.14F 1.12e-12

Ma’lumolarning uzun xaqiqiy son turi :

Oxiriga L yoki l suffikslari kuyilgan xaqiqiy son.

Masalan: 2E+6L;

Mantiqiy konstanta. Mantiqiy konstantalar true(rost) va false(yolg‘on) qiymatlardan iborat. Ichki ko‘rinishi false – 0, ixtiyoriy boshqa qiymat true deb qaraladi.

Satrli konstanta. Satrli konstantalar C++ tili konstantalariga kirmaydi, balki leksemalari aloxida tipi hisoblanadi. SHuning uchun adabiyotda satrli konstantalar satrli leksemalar deb ham ataladi..

Satrli konstanta bu ikkilik qavslarga olingan ixtiyoriy simvollar ketma ketligidir. Misol uchun "Men satrli konstantaman".

Satrlar orasiga eskeyp simvollar ham kirishi mumkin. Bu simvollar oldiga \ belgisi qo‘yiladi. Misol uchun:

"\n Bu satr \n uch katorga \n joylashadi".

Satr simvollari xotirada ketma ket joylashtiriladi va har bir satrli konstanta oxiriga avtomatik ravishda kompilyator tomonidan '\0' simvoli qo‘shiladi. SHunday satrning xotiradagi xajmi simvollar soni+1 baytga tengdir.

Ketma-ket kelgan va bo‘shliq, tabulyasiya yoki satr oxiri belgisi bilan ajratilgan satrlar kompilyasiya davrida bitta satrga aylantiriladi.

O‘zlashtirish amali. O‘zlashtirish amali (=) o‘zidan chap tomonda turgan operand qiymatini tenglik belgisidan o‘ng tomondagilarni hisoblangan qiymatiga almashtiradi. Masalan,

x = a+b;

ifodasi x operandga a va v o‘zgaruvchilarni qiymatlarini qo‘shishdan hosil bo‘lgan natijani o‘zlashtiradi.

Matematik amallar. C++ tilida 5 ta asosiy matematik amallar qo‘llaniladi: qo‘shish (+), ayirish (-), ko‘paytirish (*), butun songa bo‘lish (/) va modul bo‘yicha bo‘lish (%)(qoldiqni olish). Amallar odatda unar ya’ni bitta operandga

qo‘llaniladigan amallarga va binar ya’ni ikki operandga qo‘llaniladigan amallarga ajratiladi.

Binar amallar additiv ya’ni + qo‘shish va – ayirish amallariga, hamda mulьtiplikativ ya’ni * ko‘paytirish, / bo‘lish va % moduly olish amallariga ajratiladi.

Butun songa bo‘lish odatdagi bo‘lishdan farq qiladi. Butun songa bo‘lishdan hosil bo‘lgan bo‘linmaning faqatgina butun qismi olinadi.

Butun sonni butun songa bo‘lganda natija butun songacha yaxlitlanadi. Misol uchun 20/3=6; (-20)/3=-6; 20/(-3)=-6.

Inkrement va dekrement. Dasturlarda o‘zgaruvchiga 1 ni qo‘shish va ayirish amallari juda ko‘p hollarda uchraydi. C++ tilida qiymatni 1 ga oshirish inkrement, 1 ga kamaytirish esa dekrement deyiladi. Bu amallar uchun maxsus operatorlar mavjuddir.

Inkrement operatori (++) o‘zgaruvchi qiymatini 1 ga oshiradi, dekrement operatori (—) esa o‘zgaruvchi qiymatini 1 ga kamaytiradi. Masalan, s o‘zgaruvchisiga 1 qiymatni qo‘shmoqchi bo‘lsak quyidagi ifodani yozishimiz lozim.

C++ //s o‘zgaruvchi qiymatini 1 ga oshirdik.

Bu ifodani quyidagicha yozishimiz mumkin edi.

s=s+1;

Bu ifoda o‘z navbatida quyidagi ifodaga teng kuchli:

s+=1;

Prefiks va postfiks. Inkrement operatori ham, dekrement operatori ham ikki variantda ishlaydi: prefiksli va postfiksli. Prefiksli variantda ular o‘zgaruvchidan oldin (++Age), postfiksli variantda esa o‘zgaruvchidan keyin (Age++) yoziladi.

Oddiy ifodalarda bu variantlarni qo‘llanishida farq katta emas, lekin bir o‘zgaruvchiga boshqa o‘zgaruvchining qiymatini o‘zlashtirishda ularning qo‘llanilishi boshqacha harakterga ega. Prefiksli operator qiymat o‘zlashtirilguncha, postfiksli operator esa qiymat o‘zlashtirilgandan keyin bajariladi. Misol uchun i qiymati 2 ga teng bo‘lsin, u holda 3+(++i) ifoda qiymati 6 ga, 3+i++ ifoda qiymati 5 ga teng bo‘ladi. Ikkala holda ham i qiymati 3 ga teng bo‘ladi.

Shartli amal. Shartli amal ternar amal deyiladi va uchta operanddan iborat bo‘ladi:

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

Shartli amal bajarilganda avval 1- ifoda hisoblanadi. Agar 1-ifoda qiymati 0 dan farqli bo‘lsa 2- ifoda hisoblanadi va qiymati natija sifatida qabul qilinadi, aks holda 3-ifoda hisoblanadi va qiymati natija sifatida qabul qilinadi.

Misol uchun modulьni hisoblash: x<0?-x:x yoki ikkita son kichigini hisoblash a<b?a:b .

SHuni aytish lozimki Shartli ifodadan har qanday ifoda sifatida foydalanish mumkin. Agar F FLOAT tipga,a N – INT tipga tegishli bo‘lsa,

(N > 0) ? F : N

ifoda N musbat yoki manfiyligidan qat’i nazar DOUBLE tipiga tegishli bo‘ladi.

Shartli ifodada birinchi ifodani qavsga olish shart emas.

Mantiqiy amallar. Dasturlashda bir emas balki bir nechta Shartli ifodalarni tekshirish zaruriyati juda ko‘p uchraydi. Masalan, x o‘zgaruvchisi y o‘zgaruvchisidan, y esa o‘z navbatida z o‘zgaruvchisidan kattami sharti bunga misol bo‘la oladi. Bizning dasturimiz mos amalni bajarishdan oldin bu ikkala shart rost yoki yolg‘onligini tekshirishi lozim.

Quyidagi mantiq asosida yuqori darajada tashkil qilingan signalizatsiya sistemasini tasavvur qiling. Agarda eshikda signalizatsiya o‘rnatilgan bo‘lsa VA kun vaqti kech soat olti VA bugun bayram YOKI dam olish kuni BO‘LMASA politsiya chaqirilsin. Barcha shartlarni tekshirish uchun C++ tilining uchta mantiqiy operatori ishlatiladi.

Mantiqiy amallar || (diz’yunksiya); && (kon’yunksiya); !(inkor) amallari deb ataladi. Mantiqiy amallarni butun sonlarga qo‘llash mumkin. Bu amallarning natijalari quyidagicha aniqlanadi:

x||y amali 1 ga teng agar x>0 yoki y>0 bo‘lsa, aksincha 0 ga teng

x&&y amali 1 ga teng agar x>0 va y>0 bo‘lsa, aksincha 0 ga teng

!x amali 1 ga teng agar x>0 bo‘lsa, aksincha 0 ga teng

Bu misollarda amallar ustivorligi oshib borish tartibida berilgandir.

Inkor ! amali unar qolganlari binar amallardir.

Mantiqiy ko‘paytirish amali. Mantiqiy ko‘paytirish amali ikkita ifodani hisoblaydi, agar ikkala ifoda true qiymat qaytarsa VA operatori ham true qiymat qaytardi. Agarda sizning qorningiz ochligi rost bo‘lsa VA sizda pul borligi ham rost bo‘lsa siz supermarketga borishingiz va u erdan o‘zingizga tushlik qilish uchun biror bir narsa harid qilishingiz mumkin. YOki yana bir misol, masalan,

(x==5)&&(y==5)

mantiqiy ifodasi agarda x va u o‘zgaruvchilarini ikkalasining ham qiymatlari 5 ga teng bo‘lsagina true qiymat qaytaradi. Bu ifoda agarda o‘zgaruvchilardan birortasi 5 ga teng bo‘lmagan qiymat qabul qilsa false qiymatini qaytaradi. Mantiqiy ko‘paytirish operatori faqatgina o‘zining ikkala ifodasi ham rost bo‘lsagina true qiymat qaytaradi.

Mantiqiy ko‘paytirish amali && belgi orqali belgilanadi.

Mantiqiy qo‘shish amali. Mantiqiy qo‘shish amali ham ikkita ifoda orqali hisoblanadi. Agarda ulardan birortasi rost bo‘lsa mantiqiy qo‘shish amali true

qiymat qaytaradi. Agarda sizda pul YOKI kredit kartochkasi bo‘lsa, siz schyotni to‘lay olasiz. Bu holda ikkita shartning birdaniga bajarilishi: pulga ham va kredit kartochkasiga ham ega bo‘lishingiz shart emas. Sizga ulardan birini bajarilishi etarli. Bu operatorga oid yana bir misolni qaraymiz. Masalan,

```
(x==5)||(y==5)
```

ifodasi yoki x o‘zgaruvchi qiymati, yoki u o‘zgaruvchi qiymati, yoki ikkala o‘zgaruvchining qiymati ham 5 ga teng bo‘lsa rost qiymat qaytaradi.

Mantiqiy inkor amali. Mantiqiy inkor operatori tekshirilayotgan ifoda yolg‘on bo‘lsa true qiymat qaytaradi. Agarda tekshirilayotgan ifoda rost bo‘lsa inkor operatori false qiymat qaytaradi. Masalan, (!(x==5))
ifodasining qiymati, agarda x o‘zgaruvchisi 5 ga teng bo‘lmasa true qiymat qaytaradi. Bu ifodani boshqacha ham yozish mumkin:

```
(x!=5)
```

Munosabat amallari. Bunday amallar ikkita qiymatni teng yoki teng emasligini aniqlash uchun ishlatiladi. Taqqoslash ifodasi doimo true (rost) yoki false (yolg‘on) qiymat qaytaradi. Munosabat amallari arifmetik tipdagi operandlarga qo‘llanilsa qiymatlari 1 ga teng agar nisbat bajarilsa va aksincha 0 ga tengdir.

O‘zgaruvchilar turlari. Dastur o‘zi ishlatadigan ma’lumotlarni saqlash imkoniyatiga ega bo‘lishi lozim. Buning uchun o‘zgaruvchilar va o‘zgarmaslardan foydalaniladi.

C++ tilida o‘zgaruvchilar ma’lumotni saqlash uchun qo‘llaniladi. O‘zgaruvchining dasturda foydalanish mumkin bo‘lgan qandaydir qiymatlarni saqlaydigan komьyuter xotirasidagi yacheyka ko‘rinishda ifodalash mumkin.

Kompyuter xotirasini yacheykalardan iborat qator sifatida qarash mumkin. Barcha yacheykalar ketma – ket nomerlangan. Bu nomerlar yacheykaning adresi deb ataladi. O‘zgaruvchilar biror – bir qiymatni saqlash uchun bir yoki bir nechta yacheykalarni band qiladi.

O‘zgaruvchining nomini (masalan, MyVariable) xotira yacheykasi adresi yozilgan yozuv deb qarash mumkin.

Masalan MyVariable o‘zgaruvchisi 102 – adresdagi yacheykadan boshlab saqlanadi. O‘zining o‘lchoviga muvofiq MyVariable o‘zgaruvchisi xotiradan bir yoki bir necha yacheykani band qilishi mumkin.

O‘zgaruvchilarning quyidagi tiplari mavjuddir:

```
bool – mantiqiy;
char – bitta simvol;
long char – uzun simvol;
int – butun son;
short yoki short int – kiska butun son
```

```
long yoki long int – uzun butun son
float xaqiqiy son;
long float yoki double – ikkilangan xaqiqiy son
long double – uzun ikkilangan xaqiqiy son
```

Butun sonlar o‘lchami. Bir xil tipdagi o‘zgaruvchilar uchun turli kompyuterlarda xotiradan turli hajmdagi joy ajratilishi mumkin. Lekin, bitta kompyuterda bir xil tipdagi ikkita o‘zgaruvchi bir xil miqdorda joy egallaydi.

char tipli o‘zgaruvchi bir bayt hajmni egallaydi. Ko‘pgina komьpyuterlarda short int (qisqa butun) tipi ikki bayt, long int tipi esa 4 bayt joy egallaydi. Butun qiymatlar o‘lchovini kompьyuter sistemasi va ishlatiladigan kompilyator aniqlaydi. 32 – razryadli kompьyuterlarda butun o‘zgaruvchilar 4 bayt joy egallaydi.

O‘zgaruvchiga qiymat berish. O‘zgaruvchilarni dasturning ixtiyoriy qismida ta’riflash yoki qayta ta’riflash mumkin.

Misol uchun:

```
int a, b1, ac; yoki
int a;
int b1;
 int ac;
```

O‘zgaruvchilar ta’riflanganda ularning qiymatlari aniqlanmagan bo‘ladi. Lekin o‘zgaruvchilarni ta’riflashda initsializatsiya ya’ni boshlang‘ich qiymatlarini ko‘rsatish mumkin.

Misol uchun:

```
int i=0;
char c='k';
```

O‘zgaruvchilarga qiymat berish uchun o‘zlashtirish operatori qo‘llaniladi. Masalan, Width o‘zgaruvchisiga 5 qiymatni berish uchun quyidagilarni yozish lozim:

```
unsigned short Width;
Width = 5;
```

Bu ikkala satrni Width o‘zgaruvchisini aniqlash jarayonida birgalikda yozish mumkin.

```
unsigned short Wigth = 5;
```

Bir necha o‘zgaruvchilarni aniqlash vaqtida ham ularga qiymat berish mumkin:

```
long width = 5, length = 7;
```

Bu misolda long tipidagi width o‘zgaruvchisi 5 qiymatni, shu tipdagi length o‘zgaruvchisi esa 7 qiymatni qabul qildi.

if operatori. Odatda dastur satrma–satr tartib bilan bajariladi. If operatori shartni tekshirish (masalan, ikki o‘zgaruvchi tengmi) va uning natijasiga bog‘liq

ravishda dasturni bajarilish tartibini o‘zgartirish imkonini beradi. If operatorining oddiy shakli quyidagi ko‘rinishdadir:

```
if (shart)
 ifoda;
```

Qavs ichidagi shart ixtiyoriy ifoda bo‘lishi mumkin. Agarda bu ifoda false qiymatini qaytarsa undan keyingi ifoda yoki blok tushirib qoldiriladi. Agarda shart true qiymat qaytarsa navbatdagi ifoda bajariladi. Quyidagi misolni qaraymiz:

```
If (kattaSon>kichikSon)
kattaSon=kichikSon;
```

Bu erda kattaSon va kichikSon o‘zgaruvchilari taqqoslanayapti. Agarda kattaSon o‘zgaruvchisi qiymati katta bo‘lsa, bu navbatdagi qatorda unga qiymat sifatida kichikSon o‘zgaruvchisining qiymati o‘zlashtiriladi.

if operatorida figurali qavs ichiga olingan ifodalar blokini ham ishlatish mumkin.

```
if (shart)
{
      1  - ifoda
      2  - ifoda
      3  - ifoda
 }
```

Quyida ifodalar blokining qo‘llanilishiga oid misol keltirilgan

```
if(kattaSon>kichikSon)
{ kattaSon=kichikSon
  cout<<"kattaSon:"<<kattaSon << "/n";
  cout<<"kichikSon:"<<kichikSon<< "/n";
}
```

Bu holda kattaSon o‘zgaruvchisiga nafaqat kichikSon o‘zgaruvchisi o‘zlashtirilayapti, balki ekranga bu haqida axborot ham chiqarilayapti.

Funksiya tushunchasi. Ob’ektlarga mo‘ljallangan dasturlashda asosiy e’tibor funksiyaga emas, balki ob’ektga qaratilgan bo‘lsada dasturlarda funksiya markaziy komponentligicha qoldi. Funksiya bu ma’nosiga ko‘ra dastur osti bo‘lib, u ma’lumotlarni o‘zgartirishi va biror bir qiymat qaytarishi mumkin. C++ da har bir dastur hech bo‘lmaganda bitta main() funksiyasiga ega bo‘ladi. main() funksiyasi dastur ishga tushirilishi bilan operatsion sistema tomonidan avtomatik chaqiriladi. Boshqa funksiyalar esa u tomonidan chaqirilishi mumkin.

Har bir funksiya o‘zining nomiga egadir. Qachonki, dasturda bu nom uchrasa boshqaruv shu funksiya tanasiga o‘tadi. Bu jarayon funksiyani chaqirilishi (yoki

funksiyaga murojaat qilish) deb aytiladi. Funksiya ishini tugatgandan so‘ng dastur o‘z ishini funksiya chaqirilgan qatorning keyingisidan boshlab davom ettiradi.

Funksiyalarning qo‘llanilishi. Funksiyalar yo void tipidagi, yo boshqa biror bir tipdagi qiymat qaytaradi. Ikkita butun sonni qo‘shib, ularning yig‘indisini qaytaradigan funksiya butun qiymat qaytaruvchi deyiladi. Faqatgina qandaydir amallarni bajarib, hech qanday qiymat qaytarmaydigan funksiyaning qaytaruvchi tipi void deb e’lon qilinadi.

Funksiya sarlavha va tanadan iboratdir. Funksiya sarlavhasida uning qaytaradigan tipi, nomi va parametrlari aniqlanadi. Parametrlar funksiyaga qiymat uzatish uchun ishlatiladi. Masalan, agar funksiya ikki sonni qo‘shishga mo‘ljallangan bo‘lsa u holda bu sonlarni funksiyaga parametr qilib berish kerak. Bunday funksiyaning sarlavhasi quyidagicha bo‘ladi:

int Qushish(int a,int b)

Parametr – bu funksiyaga uzatiladigan qiymat tipini e’lon qilishdir. Funksiya chaqirilganda unga uzatiladigan qiymat argument deb aytiladi. Ko‘pchilik dasturchilar bu ikkala tushunchani sinonim sifatida qarashadi. Ba’zilar esa bu terminlarni aralashtirishni noprofessionallik deb hisoblaydi. Ma’ruzalarda ikkala terminni bir xil ma’noda kelgan.

Funksiya tanasi ochiluvchi figurali qavs bilan boshlanadi va u bir necha qatordan iborat bo‘lishi mumkin. (Funksiya tanasida hech qanday satr bo‘lmasligi ham mumkin). Satrlardan keyin esa yopiluvchi figurali qavs keladi. Funksiyaning vazifasi uning satrlarida berilgan dasturiy kodlar bilan aniqlanadi. Funksiya dasturga return operatori orqali qiymat qaytaradi. Bu operator funksiyadan chiqish ma’nosini ham anglatadi. Agarda funksiyaga chiqish operatorini (return) qo‘ymasak funksiya satrlarini tugashi bilan u avtomatik void tipidagi qiymatni qaytaradi. Funksiya qaytaradigan tip uning sarlavhasida ko‘rsatilgan tip bilan bir xil bo‘lishi lozim.

Funksiyaning bajarilishi. Funksiya chaqirilganda unda ko‘rsatilgan amallar ochiluvchi figurali qavsdan ({) keyingi birinchi ifodadan boshlab bajariladi. Funksiya tanasida if Shartli operatoridan foydalanib tarmoqlanishni ham amalga oshirish mumkin.

Funksiya o‘z tanasida boshqa funksiyalarni va hatto o‘z – o‘zini ham chaqirishi mumkin.

Qaytariladigan qiymatlar, parametrlar va argumentlar. Funksiya biror bir qiymat qaytarishi mumkin. Funksiyaga murojaat qilingandan so‘ng u qandaydir amallarni bajaradi, keyin esa u o‘z ishining natijasi sifatida biror bir qiymat qaytaradi. Bu qaytariladigan qiymat deb ataladi va bu qiymatning tipi oldindan e’lon qilinishi lozim. Quyidagi yozuvda myFunction funksiyasi butun sonli qiymat qaytaradi.

```
int myFunction()
```

Funksiyaga ham o‘z navbatida biror bir qiymat uzatish mumkin. Uzatiladigan qiymatlar funksiyaning parametrlari deb aytiladi.

```
int myFunction (int Par, float ParFloat);
```

Bu funksiya nafaqat butun son qaytaradi, balki parametr sifatida butun va haqiqiy sonli qiymatlarni qabul qiladi.

Parametrda funksiya chaqirilganda unga uzatiladigan qiymat tipi aniqlanishi lozim. Funksiyaga uzatiladigan haqiqiy qiymatlar argumentlar deb aytiladi.

```
int theValueReturned=myFunction(5,6,7);
```

Bu erda theValueReturned nomli butun sonli o‘zgaruvchiga argument sifatida 5, 6 va 7 qiymatlar berilgan myFunction funksiyasining qaytaradigan qiymati o‘zlashtirilayapti. Argument tiplari e’lon qilingan parametr tiplari bilan mos kelishi lozim.

Funksiya ta’rifida formal parametrlar initsializatsiya kilinishi, ya’ni boshlang‘ich qiymatlar ko‘rsatilishi mumkin.

Funksiyaga murojaat qilinganda biror xaqiqiy parametr ko‘rsatilmasa, uning o‘rniga mos formal parametr ta’rifida ko‘rsatilgan boshlang‘ich qiymat ishlatiladi.

Sikllarni tashkil etish. Har qanday dasturning strukturasi tarmoqlanish vassikllar to‘plamining kombinatsiyasidan iborat bo‘ladi. Qator masalalarni echish uchun ko‘pincha bitta amalni bir necha marotaba bajarish talab qilinadi. Amaliyotda bu rekursiyalar va iterativ algoritmlar yordamida amalga oshiriladi. Iterativ jarayonlar – bu operatsiyalar ketma-ketligini zaruriy sonda takrorlanishidir.

while operatori orqalissikllarni tashkil etish. while operatori yordamidassikllarni tashkil etishda operatsiyalar ketma-ketligissiklning davom etish sharti «to‘g‘ri» bo‘lsa yoki 0 dan farqli bo‘lsagina uning navbatdagi operatsiyalari amalga oshiriladi.

while operatori quyidagi umumiy ko‘rinishga egadir:

```
while(ifoda) Operator
```

Agar dasturda while (1); satr qo‘yilsa bu dastur xech qachon tugamaydi.

while operatori orqali murakkab konstruksiyalarni tuzish. while operatori shartida murakkab mantiqiy ifodalarni ham qo‘llash mumkin. Bunday ifodalarni

qo‘llashda && (mantiqiy ko‘paytirish), || (mantiqiy qo‘shish), hamda !(mantiqiy INKOR) kabi operatsiyalardan foydalaniladi.

do…while konstruksiyasi yordamidassikl tashkil etish. Ayrim hollarda while operatori yordamidassikllarni tashkil etishda uning tanasidagi amallar umuman bajarilmasligi mumkin. CHunkissiklni davom etish sharti har bir iteratsiyadan oldin tekshiriladi. Agarda boshlang‘ich berilgan shart to‘g‘ri bo‘lmasassikl tanasining birorta operatori ham bajarilmaydi.

do…while konstruksiyasining qo‘llanilishi. do-while operatori umumiy ko‘rinishi quyidagicha:

```
do
Operator
while(ifoda)
```

do…while konstruksiyasidassikl sharti uning tanasidagi operatsiyalar bir marta bajarilgandan so‘ng tekshiriladi. Bussikl operatorlarini hech bo‘lmaganda bir marta bajarilishini kafolatlaydi. To sharti «yolg‘on» bo‘lmaguncha yoki ifoda qiymati 0 bo‘lmagunchassikl qaytariladi.

for operatori. for operatori umumiy ko‘rinishi quyidagicha:

```
for( 1-ifoda;2- ifoda; 3-ifoda)
Operator
```

Bu operator quyidagi operatorga mosdir.

```
1-ifoda;
while(2-ifoda) {
operator
3-ifoda
}
```

while operatori yordamidassikllarni tashkil etishda 3 ta zaruriy amallar:ssikl o‘zgaruvchisiga boshlang‘ich qiymat berish, har bir iteratsiyadassiklni davom etish sharti bajarilishini tekshirish vassikl o‘zgaruvchisi qiymatini o‘zgartirishni bajarishimiz kerak.

for operatorissiklni ishlashi uchun zarur bo‘ladigan uchta operatsiyani o‘zida birlashtiradi. Bu operatsiyalarni qisqacha quyidagicha harakterlash mumkin: boshlang‘ich qiymatni o‘zlashtirish, shartni tekshirish,ssikl schyotchigini qiymatini oshirish. for operatori ifodasidagi qavsning ichida shu uchchala operatsiyani amalga oshiruvchi ifodalar yoziladi. Qavs ichidagi ifodalar nuqtali vergul orqali ajratiladi.

for siklining birinchi ifodasissikl schyotchigiga boshlang‘ich qiymatni o‘zlashtiradi. Schyotchik – to‘g‘ridan–to‘g‘ri forssiklida e’lon qilinadigan va qiymat o‘zlashtiriladigan butun sonli o‘zgaruvchidir. C++ da bu o‘rinda schyotchikka qiymat beradigan ixtiyoriy ifoda yozilishiga imkon berilgan.

forssiklining ikkinchi parametridassiklni davom etish sharti aniqlanadi. Bu shart while konstruksiyasining sharti bajaradigan vazifani amalga oshiradi. Uchinchi parametrda esassikl schyotchigi qiymatini o‘zgartiruvchi (oshiruvchi yoki kamaytiruvchi) ifoda yoziladi.

for operatori uchun murakkab ifodalarni berilishi. forssikli dasturlashning kuchli va qulay instrumentidir. for operatoridassiklni o‘zaro bog‘liq bo‘lmagan parametrlar (boshlang‘ich qiymat o‘zlashtirish, bajarilish sharti va qadam) ni qo‘llanilishissikl ishini boshqarishda juda yaxshi imkoniyatlarni ochib beradi.

forssikli quyidagi ketma–ketlikda ishlaydi.

Sikl schetchigiga boshlang‘ich qiymat o‘zlashtiriladi.

Siklni davom etish shartidagi ifoda qiymati hisoblanadi.

Agarda shart ifodasi true qiymat qaytarsa oldinssikl tanasi bajariladi, keyin esassikl schyotchigi ustida berilgan amallar bajariladi.

Har bir iteratsiyada 2 – va 3 – qadamlar takrorlanadi.

O‘tish break va continue operatorlari. Ko‘pinchassiklning navbatdagi iteratsiyasigassikl tanasidagi boshqa operatorlar (navbatdagi operatorlar) bajarilmasdan turib o‘tish zaruriyati tug‘iladi. Bunday holatlarda continue operatori qo‘llaniladi. Bundan tashqari,ssiklni bajarilishi sharti qanoatlantirilganda ham, qator hollarda undan chiqib ketish zaruriyati paydo bo‘ladi. Bu holda esa break operatori ishlatiladi. Bunday operatorlarni qo‘llanilishiga quyidagi misolda keltirilgan. Bu misol bizga oldingi listingdan tanish bo‘lgan o‘yinning biroz murakkablashtirilgan variantidir. Bu erda kichik va katta qiymatlardan tashqari qadam va maqsadli kattalikni kiritish talab etiladi. Agarda kichik son qiymati qadam o‘zgaruvchisiga *(*qadam*)* karrali bo‘lmasa katta qiymati ikkiga kamaytiriladi. Qachonki, kichik o‘zgaruvchisi qiymati katta o‘zgaruvchisi qiymatidan katta bo‘lsa o‘yin tugatiladi. Agarda katta o‘zgaruvchisi qiymati maqsadli kattalik(maqsad)ning qiymati bilan ustma – ust tushsa o‘yin bekor qilinganligi haqida xabar ekranga chiqariladi.

switch operatori. Oldingi Ma’ruzalarda biz if va if/else operatorlari bilan tanishgan edik. Lekin ayrim masalalarni echishda if operatori ichida ko‘p sondagi if operatorlarini qo‘llashga to‘g‘ri keladi. Bu esa dasturni yozishni ham, uni tushinishni ham murakkablashtirib yuboradi. Bunday muammoni echish uchun C++ tilida switch operatori qo‘llaniladi. Bu operatorning if operatoridan asosiy farqi shuki, unda bir yo‘la bir nechta shart tekshiriladi.

Switch operatorining qo‘llanilishi. switch operatorining qo‘llanilish sintaksisi quyidagicha:

```
switch(ifoda)
{
```

```
sase 1-nchi qiymat: ifoda;
sase 2-nchi qiymat: ifoda;
…
sase n-nchi qiymat: ifoda;
default : ifoda;
}
```

switch operatori orqali dasturning tarmoqlanishi bir necha mumkin bo‘lgan qiymatlarni qaytaruvchi ifodaning natijasi asosida tashkil etiladi. switch operatoridagi qavs ichida berilgan ifodaning qaytargan qiymati case operatoridan keyinda ko‘rsatilgan qiymat bilan solishtiriladi. Ifodaning qiymati bilan case operatoridan keyingi qiymat mos kelsa tanlangan case operatoridan keyingi barcha satrlar bajariladi. Bunda amallarni bajarilishi break operatorigacha davom etadi.

Agarda case operatorlari qiymatidan birortasi ham qaytarilgan qiymatga mos kelmasa default operatoridan keyingi dastur satrlari bajariladi. Agarda bu operator mavjud bo‘lmasa boshqaruv switch bloki tanasidan chiqadi va keyingi dastur satrlariga beriladi.

switch operatoridan keyingi qavs ichida tilning konstruksiyasi nuqtai–nazaridan to‘g‘ri bo‘lgan ixtiyoriy ifodani ishlatish mumkin. Operator identifikatori o‘rnida ham ixtiyoriy operator yoki ifoda, hamda operator va ifodalarning ketma-ketligini ishlatish mumkin. Lekin bu erda mantiqiy operatsiyalar yoki taqqoslash ifodalarini ishlatish mumkin emas.

goto operatori tarixi. Dasturlashni ilk davrlarida kichikroq hajmdagi va etarlicha sodda dasturlar ishlatilar edi. Bunday dasturlardassikllar nishonlardan, operatorlar va komandalar ketma – ketligidan hamda o‘tish operatoridan iborat edi .

C++ tilida nishon deb orqasidan ikki nuqta (:) yoziladigan identifikatorga aytiladi. Nishon doimo boshqaruv o‘tishi lozim bo‘lgan operatordan oldin o‘rnatiladi. Kerakli nishonga o‘tish uchun goto operatori qo‘llaniladi.

Bunda kalit so‘zdan keyin nishon nomi yoziladi. O‘tish operatorining ko‘rinishi:

goto <identifikator>. Bu operator identifikator bilan belgilangan operatorga o‘tish kerakligini ko‘rsatadi.

Nazorat savollari:

1. Ma’lumot tiplari.
2. Unsigned tipining xossalarini ko‘rsating.
3. O‘zgaruvchilar va o‘zgarmaslar.
4. Operatorlar.

5. Shartli operator umumiy ko‘rinishi.
6. Funksiyani e’lon qilish.
7. Funksiyaga murojat.

2-Ma’ruza. Massivlar va satrlar

Reja:

1. Bir o‘lchovli massivlar.
2. Ko‘p o‘lchamli massivlar.
3. Satrlar massiv sifatida.
4. Satrlar bilan ishlovchi funksiyalar.

Massiv tushunchasi. Massiv bu bir tipli nomerlangan ma’lumotlar jamlanmasidir. Massiv indeksli o‘zgaruvchi tushunchasiga mos keladi. Massiv ta’riflanganda tipi, nomi va indekslar chegarasi ko‘rsatiladi. Masalan type turidagi length ta elementdan iborat a nomli massiv shunday e’lon qilinadi:

```
type a[length];
```

Bu maxsus a[0], a[1], ..., a[length -1] nomlarga ega bo‘lgan type turidagi o‘zgaruvchilarning e’lon qilinishiga to‘g‘ri keladi.

Massivning har bir elementi o‘z raqamiga - indeksga ega. Massivning x-nchi elementiga murojaat indekslash operatsiyasi yordamida amalga oshiriladi:

```
int x=...;          //butun sonli indeks
TYPE value=a[x];    //ch-nchi elementni o‘qish
a[x]=value;         //x-yxb elementga yozish
```

Indeks sifatida butun tur qiymatini qaytaradigan har qanday ifoda qo‘llanishi mumkin: char, short, int, long. C++ da massiv elementlarining indekslari 0 dan boshlanadi (1 dan emas), length elementdan iborat bo‘lgan massivning oxirgi elementining indeksi esa - bu length -1 (length emas). Massivning int z[3] shakldagi ta’rifi, int tipiga tegishli z[0],z[1],z[2] elementlardan iborat massivni aniqlaydi.

Massiv chegarasidan tashqariga chiqish (ya’ni mavjud bo‘lmagan elementni o‘qish/yozishga urinish) dastur bajarilishida kutilmagan natijalarga olib kelishi mumkin. SHuni ta’kidlab o‘tamizki, bu eng ko‘p tarqalgan xatolardan biridir.

Agar massiv initsializatsiya qilinganda elementlar chegarasi ko‘rsatilgan bo‘lsa , ro‘yxatdagi elementlar soni bu chegaradan kam bo‘lishi mumkin, lekin ortiq bo‘lishi mumkin emas.

Misol uchun int a[5]={2,-2}. Bu holda a[0] va a[1] qiymatlari aniqlangan bo‘lib, mos holda 2 va –2 ga teng. Agar massiv uzunligiga qaraganda kamroq element berilgan bo‘lsa, qolgan elementlar 0 hisoblanadi:

```
int a10[10]={1, 2, 3, 4};    //va 6 ta nol
```

Agar nomlangan massivning tavsifida uning o‘lchamlari ko‘rsatilmagan bo‘lsa, kompilyator tomonidan massiv chegarasi avtomatik aniqlanadi:

```
int a3[]={1, 2, 3};
```

Bir o‘lchamli massivlarni funksiya parametrlari sifatida uzatish. Massivdan funksiya parametri sifatida foylalanganda, funksiyaning birinchi elementiga ko‘rsatkich uzatiladi, ya’ni massiv hamma vaqt adres bo‘yicha uzatiladi. Bunda massivdagi elementlarning miqdori haqidagi axborot yo‘qotiladi, shuning uchun massivning o‘lchamlari haqidagi ma’lumotni alohida parametr sifatida uzatish kerak.

Funksiyaga massiv boshlanishi uchun ko‘rsatkich uzatilgani tufayli (adres bo‘yicha uzatish), funksiya tanasining operatorlari hisobiga massiv o‘zgarishi mumkin.

Funksiyalarda bir o‘lchovli sonli massivlar argument sifatida ishlatilganda ularning chegarasini ko‘rsatish shart emas.

Funksiyalarda bir o‘lchovli sonli massivlar argument sifatida ishlatilganda ularning chegarasini ko‘rsatish shart emas.

Ko‘p o‘lchovli massivlar ta’rifi. Ikki o‘lchovli massivlar matematikada matritsa yoki jadval tushunchasiga mos keladi. Jadvallarning insializatsiya qilish qoidasi, ikki o‘lchovli massivning elementlari massivlardan iborat bo‘lgan bir o‘lchovli massiv ta’rifiga asoslangandir.

Misol uchun ikki qator va uch ustundan iborat bo‘lgan xaqiqiy tipga tegishli d massiv boshlang‘ich qiymatlari quyidagicha ko‘rsatilishi mumkin:

```
float d[2][3]={(1,-2.5,10),(-5.3,2,14)};
```

Bu yozuv quyidagi qiymat berish operatorlariga mosdir:

```
d[0][0]=1;d[0][1]=-2.5;d[0][2]=10;
d[1][0]=-5.3;d[1][1]=2;d[1][2]=14;
```

Bu qiymatlarni bitta ro‘yxat bilan xosil qilish mumkin:

```
float d[2][3]={1,-2.5,10,-5.3,2,14};
```

Initsializatsiya yordamida boshlang‘ich qiymatlar aniqlanganda massivning hamma elementlariga qiymat berish shart emas.

Misol uchun: `int x[3][3]={(1,-2,3),(1,2),(-4)}`.

Bu yozuv quyidagi qiymat berish operatorlariga mosdir:

```
x[0][0]=1;x[0][1]=-2;x[0][2]=3;
x[1][0]=-1;x[1][1]=2;x[2][0]=-4;
```

Initsializatsiya yordamida boshlang‘ich qiymatlar aniqlanganda massivning birinchi indeksi chegarasi ko‘rsatilishi shart emas, lekin qolgan indekslar chegaralari ko‘rsatilishi shart.

Misol uchun:

```
double x[][2]={(1.1,1.5),(-1.6,2.5),(3,-4)}
```

Bu misolda avtomatik ravishda katorlar soni uchga teng deb olinadi.

Funksiyaga ko‘p o‘lchamli massivlarni uzatish. Ko‘p o‘lchamli massivlarni funksiyaga uzatishda barcha o‘lchamlar parametrlar sifatida uzatilishi kerak. C++

da ko‘p o‘lchamli massivlar aniqlanishi bo‘yicha mavjud emas. Agar biz bir nechta indeksga ega bo‘lgan massivni tavsiflasak (masalan, int mas [3][4]), bu degani, biz bir o‘lchamli mas massivini tavsifladik, bir o‘lchamli int [4] massivlar esa uning elementlaridir

Misol: Kvadrat matritsani uzatish (transportirovka qilish)

Agar void transp(int a[][],int n){.....} funksiyasining sarlavhasini aniqlasak, bu holda biz funksiyaga noma’lum o‘lchamdagi massivni uzatishni xohlagan bo‘lib qolamiz. Aniqlanishiga ko‘ra massiv bir o‘lchamli bo‘lishi kerak, hamda uning elementlari bir xil uzo‘nlikda bo‘lishi kerak. Massivni uzatishda uning elementlarining o‘lchamlari haqida ham biron narsa deyilmagan, shuning uchun kompilyator xato chiqarib beradi.

Bu muammoning eng sodda echimi funksiyani quyidagicha aniqlashdir:

void transp(int a[][4],int n){.....}, bu holda har bir satr o‘lchami 4 bo‘ladi, massiv ko‘rsatkichlarining o‘lchami esa hisoblab chiqariladi.

Satrlar. C++ da belgili ma’lumotlar uchun char turi qabul qilingan. Belgili axborotni taqdim etishda belgilar, simvolli o‘zgaruvchilar va matniy konstantalar qabul qilingan.

Misollar:

const char c='c';//belgi - bir baytni egallaydi, uning qiymati o‘zgarmaydi

char a,b;//belgili o‘zgaruvchilar, bir baytdan joy egallaydi, qiymatlari o‘zgaradi.

C++ dagi satr - bu nul-belgi - \0 (nul-terminator)- bilan tugallanuvchi belgilar massivi. Nul-terminatorning holatiga qarab satrning amaldagi uzunligi aniqlanadi. Bunday massivdagi elementlar soni, satr tasviriga qaraganda, bittaga ko‘p.

Simvolli massivlar quyidagicha initsializatsiya qilinadi:

char capital[]="TASHKENT"; Bu holda avtomatik ravishda massiv elementlari soni aniqlanadi va massiv oxiriga satr ko‘chirish '\0'simvoli qo‘shiladi.

YUqoridagi initsializatsiyani quyidagicha amalga oshirish mumkin:

```
char capital[]={'T','A','S','H','K','E','N','T','\0'};
```

Bu holda so‘z oxirida '\0'simvoli aniq ko‘rsatilishi shart.

Qiymat berish operatori yordamida satrga qiymat berish mumkin emas. Satrni massivga yoki kiritish paytida yoki nomlantirish yordamida joylashtirish mumkin.

Funksiyalar va satrlar. Funksiyalarda satrlar ishlatilganda ularning chegarasini ko‘rsatish shart emas. Satrlarning uzunligini hisoblash len funksiyasii quyidagicha ta’riflash mumkin:

```
int len(char c[])
{ int m=0;
for(m=0;c[m]!=’0’;m++);
return m;
```

```
};
```

Bu funksiyaning standart varianti strlen deb ataladi va bu funksiyadan foydalanish uchun string.h sarlavxali faylidan foydalanish lozim.

So‘zlar massivini kiritish. C++ tilida so‘zlar massivlari ikki o‘lchovli simvolli massivlar sifatida ta’riflanadi. Misol uchun:

```
char name[4][5].
```

Bu ta’rif yordamida har biri 5 ta harfdan iborat bo‘lgan 4 ta so‘zli massiv kiritiladi. So‘zlar massivlari quyidagicha initsializatsiya qilinishi mumkin:

```
char Name[3][8]={ "Anvar","Mirkomil","YUsuf"}.
```

Bu ta’rifda har bir so‘z uchun xotiradan 8 bayt joy ajratiladi va har bir so‘z oxiriga ' \0' belgisi qo‘yiladi.

So‘zlar massivlari initsializatsiya qilinganda so‘zlar soni ko‘rsatilmasligi mumkin. Bu holda so‘zlar soni avtomatik aniqlanadi:

```
char comp[][9]={ "компьyuter","printer","kartridj"}.
```

Funksiyalar va so‘zlar massivlar. Satrli massivlar funksiya argumenti sifatida ishlatilganda satrlarning umumiy uzunligi aniq ko‘rsatilishi shartdir.

String tipi. Satrlar biln ishlash uchun standart bibliotekaga kiruvchi `string` murakkab turidan foydalanish qulaydir.

Bu tipdan foydalanish uchun quyidagi sarlavxali faylni ulash lozim:

```
#include <string>
```

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 (=);
- konkatenatsiya yoki satrlarni ulash (+);
- qiymat berib qo‘shish amali (+=)
- ikki amal ekvivalentlikni tekshirish uchun (==) va (!=);
- indeks olish ([]).
- solishtirish amallari(<, <=,>, >=);

Nazorat savollari:

1. Bir o‘lchovli massivlarni initsializatsiya qilish usullarini ko‘rsating.
2. Ko‘p o‘lchovli massiv ta’rifi xususiyatlarini keltiring.
3. Ko‘p o‘lchamli massivlarni e’lon qilish.
4. Satr simvolli massivdan qanday farq qiladi?
5. Satrlarni initsializatsiya qilish usullarini ko‘rsating.
6. So‘zlar massivi qanday kiritiladi?
7. Satrlar bilan ishlovchi funksiyalar.

3-Ma’ruza. Ko‘rsatkichlar

Reja:

1. Ko‘rsatkichlar haqida.
2. Ko‘rsatkichlarni e’lon qilish.
3. Ko‘rsatkichlar ustida amallar.
4. Ko‘rsatkichlar va massivlar.
5. Funksiyaga ko‘rsatkich.

Ko‘rsatkichlar. Ko‘rsatkich - xotira uyasining unikal adresini saqlaydigan o‘zgaruvchi. Ko‘rsatkich operativ xotiradagi biron-bir o‘zgaruvchi mavjud bo‘lishi mumkin bo‘lgan biron-bir joyni belgilaydi. Ko‘rsatkichlarning qiymatlarini o‘zgartirish, turli variantlarda qo‘llash mumkinki, bu dasturning moslashuvchanligini oshiradi.

Ko‘rsatkich odatda tipga ega bo‘lib quyidagicha e’lon qilinadi:

<turning nomi>*<ko‘rsatkichning nomi>=<dastlabki qiymat>

Misol uchun:

```
int *pr;
char *alfa;
```

Bu holda ko‘rsatkichlar noaniq qiymatga ega bo‘ladi. Ko‘rsatkichlar ta’riflanganda ularning tiplari ko‘rsatilishi shart. Ko‘rsatkichlarni initsializatsiya kilish ya’ni boshlang‘ich qiymatlarini kiritish mumkin. Ma’lum turdagi biron-bir o‘zgaruvchi adresi yoki NULL qiymat dastlabki qiymat bo‘lishi mumkin. Ko‘rsatkichlarga boshlang‘ich maxsus NULL qiymati berilsa bunday ko‘rsatkich bo‘sh ko‘rsatkich deb ataladi.

Biron-bir o‘zgaruvchi adresini olish hamda uni ko‘rsatkichga qiymat sifatida berish uchun «&» operatori qo‘llanadi.

Misol:

```
int I=100;
int*p=&I;
 unsigned longint *ul=NULL;
```

Teskari operator - «*» bo‘lib, ko‘rsatkichda saqlanayotgan adres bo‘yicha uya qiymatiga murojaat qilish imkonini beradi.

Misol:

```
int I=100;
int*p=&I
int J=*p;
```

Ilova tushunchasi. Ilova (ssылka) – ilova ta’rifirda ko‘rsatilgan ob’ekt nomining sinonimi.

Ilovani e’lon qilish shakli

```
tur & ism =ism_ob’ekt;
```

Misollar:

```
int x; // o‘zgaruvchining aniqlash
int& sx=x; //x o‘zgaruvchiga iqtibosni aniqlash
const char & CR='\n'; //konstantaga iqtibosni aniqlash
```

Ilovalar bilan ishlash qoidalari.

1) O‘zgaruvchi ilova, agar u funksiya parametri bo‘lmasa, extern sifatida tavsiflanmagan bo‘lsa yoki sinf maydoniga ilova qilmasa, o‘ziga tavsif berilayotganda ochiq-oydin nomlanishi kerak.

2) Nomlangandan so‘ng, ilovaga boshqa qiymat berilishi mumkin emas.

3) Ilovalarga ko‘rsatkichlar, ilovalar massivlari va ilovalarga ilovalar bo‘lishi mumkin emas.

4) Ilova ustida o‘tkazilgan operatsiya o‘zi ilova qilayotgan qiymatning o‘zgarishiga olib keladi

Ko‘rsatkichlar ustida o‘tkaziladigan operatsiyalar. Ko‘rsatkichlar ustida unar operatsiyalar bajarish mumkin: inkrement va dekrement ++ va -- operatsiyalarini bajarishda, ko‘rsatkich qiymati ko‘rsatkich murojaat qilgan tur uzunligiga ko‘payadi yoki kamayadi.

Misol:

```
int*ptr, a[10];
ptr=&a[5];
ptr++;   */ = a[6]*/ elementining adresiga
ptr--;   */ = a[5]*/ elementining adresiga
```

Qo‘shish va ayirish binar operatsiyalarida ko‘rsatkich va int turining qiymati ishtirok etishi mumkin. Bu operatsiya natijasida ko‘rsatkich qiymati dastlabkisidan ko‘rsatilgan elementlar soniga ko‘proq yoki kamroq bo‘ladi.

Misol:

```
int*ptr1, *ptr2, a[10];
int i=2;
ptr1=a+(i+4);   /* = a[6]*/ elementining adresiga
ptr2=ptr1-i;    /* = a[4]*/ elementining adresiga
```

Ayirish operatsiyasida bitta turga mansub bo‘lgan ikkita ko‘rsatkich ishtirok etishi mumkin. Operatsiya natijasi int turiga ega hamda kamayuvchi va ayiruvchi o‘rtasidagi dastlabki tur elementlarining soniga teng, bundan tashqari agar birinchi adres kichikroq bo‘lsa, u holda natija manfiy qiymatga ega bo‘ladi.

Misol:

```
int *ptr1, *ptr2, a[10];
int i;
ptr1=a+4;
```

```
ptr2=a+9;
i=ptr1-ptr2; /*=5 */
i=ptr1-ptr2; /*=-5 */
```

Bir turga taalluqli bo‘lgan ikkita ko‘rsatkich qiymatlarini ==, !=, <, <=, >, >= amallari yordamida o‘zaro qiyoslash mumkin. Bunda ko‘rsatkichlarning qiymatlari shunchaki butun sonlar sifatida olib qaraladi, qiyoslash natijasi esa 0 (yolg‘on) yoki 1 (rost) ga teng bo‘ladi.

Misol:

```
int *ptr1, *ptr2, a[10];
ptr1=a+5;
ptr2=a+7;
if(ptr1>ptr2) a[3]=4;
```

Bu misolda ptr1 ning qiymati ptr2 ning qiymatidan kamroq, shuning uchun a[3]=4 operatori bajarilmay qoladi.

Konstanta ko‘rsatkich va konstantaga ko‘rsatkichlar. Konstanta ko‘rsatkich quyidagicha ta’riflanadi:

<tip>* const<ko‘rsatkich nomi>=<konstanta ifoda>

Misol uchun: char* const key_byte=(char*)0x0417.

Bu misolda konstanta ko‘rsatkich klaviatura xolatini ko‘rsatuvchi bayt bilan boglangandir.

Konstanta ko‘rsatkich qiymatini o‘zgartirish mumkin emas lekin * amali yordamida xotiradagi ma’lumot qiymatini o‘zgartirish mumkin. Misol uchun *key_byte=’YO’ amali 1047(0x0417) adres qiymati bilan birga klaviatura xolatini ham o‘zgartiradi.

Konstantaga ko‘rsatkich quyidagicha ta’riflanadi:

<tip>const*<ko‘rsatkich nomi>=<konstanta ifoda>.

Misol uchun const int zero=0; int const* p=&zero;

Bu ko‘rsatkichga * amalini qo‘llash mumkin emas, lekin ko‘rsatkichning qiymatini o‘zgartirish mumkin. Qiymati o‘zgarmaydigan konstantaga ko‘rsatkichlar quyidagicha kiritiladi:

<tip>const* const<ko‘rsatkich nomi>=<konstanta ifoda>.

Misol uchun

```
const float pi=3.141593; float const* const pp=&pi;
```

Turlashtirilmagan ko‘rsatkich. Turlashtirilmagan (tipiklashtirilmagan) ko‘rsatkich void turga ega bo‘lib, ixtiyoriy turdagi o‘zgaruvchi adresi qiymat sifatida berilishi mumkin.

Maxsus void tipidagi ko‘rsatkichlar ajdodiy ko‘rsatkichlar deb atalib har xil tipdagi ob’ektlar bilan bog‘lanish uchun ishlatiladi.

Misol uchun:

```
int I=77;
float Euler=2.18282;
void *vp;
Vp=&I;
cout<< (*(int*)vp;
Vp=&Euler;
cout<< (*(float*)vp;
```

Quyidagi operatorlar ketma ketligi xatolikka olib keladi:

```
void *vp;int *ip; ip=vp;
```

Bu xatolik sababi bitta ob'ektga har xil tipdagi ko‘rsatkichlar bilan murojaat kilish mumkin emas.

Ko‘rsatkchlar funksiya parametri sifatida. Ko‘rsatkichlar yordamida parametr qiymatini o‘zgartirish mumkin.

Misol uchun turtburchak yuzi va perimetrini berilgan tomonlari bo‘yicha hisoblash funksiyasini quyidagicha tasvirlash mumkin.

```
void pr(float a,float b, float* s, float* p)
{
*p=2(a+b);
*s= a*b;
}
```

Bu funksiyaga quyidagicha murojaat kilinishi mumkin pr(a,b,&p,&s). Funksiyaga p va s o‘zgaruvchilarning adreslari uzatiladi. Funksiya tanasida shu adreslar bo‘yicha 2*(a+b) va a*b qiymatlar yoziladi.

Ko‘rsatkichlar o‘rniga ilovalardan foydalanish dasturning o‘qilishini yaxshilaydi, chunki bu o‘rinda adres bo‘yicha qiymat olish operatsiyasini qo‘llamasa ham bo‘ladi. Qiymat bo‘yicha uzatish o‘rniga ilovalardan foydalanish samaraliroq hamdir, chunki parametrlarni nusxalashni talab qilmaydi. Agar funksiya ichida parametrning o‘zgarishini taqiqlash lozim bo‘lib qolsa, bu holda const modifikatori qo‘llanadi. Bu modifikatorni funksiyada o‘zgarishi ko‘zda tutilmagan barcha parametrlar oldidan qo‘yish tafsiya qilinadi (undagi qaysi parametrlar o‘zgaradiyu, qaysilari o‘zgarmasligi sarlavhadan ko‘rinib turadi).

Ko‘rsatkich va massiv nomi. Massivlar nomi dasturda konstanta ko‘rsatkichdir. SHuning uchun ham int z[4] massiv elementiga *(z+4) shaklda murojaat kilish mumkin.

Massivlar bilan ishlanganda kavslarsiz ishlash mumkin.

Massivlarni funksiyalar parametrlari sifatida. Massivlar funksiyaga turidagi bir o‘lchamli massivlar sifatida yoki ko‘rsatkichlar sifatida uzatilishi mumkin. Masalan satrlar funksiyaga char turidagi bir o‘lchamli massivlar sifatida yoki char*

turidagi ko‘rsatkichlar sifatida uzatilishi mumkin. Oddiy massivlardan farqli o‘laroq, funksiyada satr uzunligi ko‘rsatilmaydi, chunki satr oxirida satr oxiri \0 belgisi bor.

Misol: Berilgan belgini satrda qidirish funksiyasi

```
int find(char *s,char c)
{
for (int i=0;i<strlen(s);i++)
if(s[i]==c) return i;
return –1
}
```

Massiv funksiya qiymati sifatida. Massiv qiymat qaytaruvchi funksiya ta’rifi:

```
float *sum_vec(int n,float a,float b)
{
float d[n];
for(int i=0;i<n;i++,d[i]=a[i]+b[i]);
return d;
}
```

Bu funksiyaga quyidagicha murojaat kilish mumkin:

```
float a[]={1,-1.5,-2},b[]={-5.2,1.3,-4};
float c[]=sum_vec(3,a,b);
```

Ko‘p o‘lchamli massivlar va ko‘rsatkichlar. C++ da massivning eng umumiy tushunchasi - bu ko‘rsatkichdir, bunda har xil turdagi ko‘rstakich bo‘lishi mumkin, ya’ni massiv har qanday turdagi elementlarga, shu jumladan, massiv bo‘lishi mumkin bo‘lgan ko‘rsatkichlarga ham ega bo‘lishi mumkin. O‘z tarkibida boshqa massivlarga ham ega bo‘lgan massiv ko‘p o‘lchamli hisoblanadi.

Bunday massivlarni e’lon qilishda kompyuter xotirasida bir nechta turli xildagi ob’ekt yaratiladi.

Ko‘rsatkichlar massivlari. Ko‘rsatkichlar massivlari quyidagicha ta’riflanadi

```
<tip> *<nom>[<son>]
```

Misol uchun int *pt[6] ta’rif int tipidagi ob’ektlarga olti elementli massivni kiritadi.

Ko‘rsatkichlar massivlari satrlar masssivlarini tasvirlash uchun qulaydir.

Misol uchun familiyalar ro‘yxatini kiritish uchun ikki o‘lchovli massivdan foydalani kerak.

```
char fam[][20]={ "Olimov","Raximov","Ergashev"}
```

Xotirada 60 elementdan iborat bo‘ladi, chunki har bir familiya 20 gacha 0 lar bilan to‘ldiriladi.

Ko‘rsatkichlar massivi yordamida bu massivni quyidagicha ta’riflash mumkin.

```
char *pf[]= { "Olimov","Raximov","Ergashev"}.
```

Bu holda ro‘yxat xotirada 23 elementdan iborat bo‘ladi, chunki har bir familiya oxiriga 0 belgisi kuyiladi

Har xil chegarali jadvallar bilan funksiyalardan foydalanishning bir yuli bu oldindan kiritiluvchi konstantalardan foydalanishdir. Lekin asosiy yuli ko‘rsatkichlar massivlaridan foydalanish.

Bir o‘lchovli dinamik massivlar. C++tilida o‘zgaruvchilar yo statik tarzda - kompilyasiya paytida, yoki standart kutubxonadan funksiyalarni chaqirib olish yo‘li bilan dinamik tarzda - dasturni bajarish paytida joylashtirilishi mumkin. Asosiy farq ushbu usullarni qo‘llashda ko‘rinadi - ularning samaradorligi va moslashuvchanligida. Statik joylashtirish samaraliroq, chunki bunda xotirani ajratish dastur bajarilishidan oldin sodir bo‘ladi. Biroq bu usulning moslashuvchanligi ancha past, chunki bunda biz joylashtirilayotgan ob’ektning turi va o‘lchamlarini avvaldan bilishimiz kerak bo‘ladi. Masalan, matniy faylning ichidagisini satrlarning statik massivida joylashtirish qiyin: avvaldan uning o‘lchamlarini bilish kerak bo‘ladi. Noma’lum sonli elementlarni oldindan saqlash va ishlov berish kerak bo‘lgan masalalar odatda xotiraning dinamik ajratilishini talab qiladi.

Xotirani dinamik va statik ajratish o‘rtasidagi asosiy farqlar quyidagicha:

- statik ob’ektlar nomlangan o‘zgaruvchilar bilan belgilanadi, hamda ushbu ob’ektlar o‘rtasidagi amallar to‘g‘ridan-to‘g‘ri, ularning nomlaridan foydalangan holda, amalga oshiriladi. Dinamik ob’ektlar o‘z shaxsiy otlariga ega bo‘lmaydi, va ular ustidagi amallar bilvosita, ko‘rsatkichlar yordamida, amalga oshiriladi;
- statik ob’ektlar uchun xotirani ajratish va bo‘shatish kompilyator tomonidan avtomatik tarzda amalga oshiriladi. Dasturchi bu haqda o‘zi qayg‘urishi kerak emas. Statik ob’ektlar uchun xotirani ajratish va bo‘shatish to‘laligicha dasturchi zimmasiga yuklatiladi. Bu anchayin qiyin masala va uni echishda xatoga yo‘l qo‘yish oson.

Dinamik tarzda ajratilayotgan xotira ustida turli xatti-harakatlarni amalga oshirish uchun new va delete operatorlari xizmat qiladi.

Ma’lum bir turdagi elementlardan tashkil topgan berilgan o‘lchamlardagi massivga xotira ajratish uchun new operatoridan foydalanish lozim:

```
int *pia=new int[4];
```

Bu misolda xotira int turidagi to‘rtta elementdan iborat massivga xotira ajratiladi. Afsuski, new operatorining bu shakli massiv elementlarini nomlantirish (initsiallashtirish) imkonini bermaydi.

Dinamik massivni bo‘shatish uchun delete operatoridan foydalanish lozim:

```
delete[] pia;
```

Agar ajratilgan xotirani bo‘shatish esdan chiqqudek bo‘lsa, bu xotira bekordan-bekorga sarflana boshlaydi, foydalanilmay qoladi, biroq, agar uning ko‘rsatkichi o‘z qiymatini o‘zgartirgan bo‘lsa, uni tizimga qaytarish mumkin emas. Bu hodisa xotiraning yo‘qotilishi (utechka pamyati) degan maxsus nom bilan ataladi. Pirovard natijada dastur xotira etishmagani tufayli avariya holatida tugallanadi (agar u ancha vaqt ishlayversa).

Ikki o‘lchovli dinamik massivlar. Matritsani shakllantirishda oldin bir o‘lchovli massivlarga ko‘rsatuvchi ko‘rsatkichlar massivi uchun xotira ajratiladi, keyin esa parametrlissiklda bir o‘lchovli massivlarga xotira ajratiladi.

Misol:

```
int n;
cin>>n;
double *matr[100];
for (i=0;i<n;i++) matr[i]=new int[n];
```

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

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

Funksiyalarni chaqirishda foydalanish. C++ tili sintaksisiga ko‘ra funksiyaga ko‘rsatkich funksiya adresini aks ettiruvchi o‘zgaruvchi yoki ifodadir. Funksiyaga ko‘rsatkich bajariluvchi qiymati funksiya kodining birinchi bayti adresidir. Funksiyaga ko‘rsatkichlar ustida arifmetik amallar bajarish mumkin emas. Eng keng qo‘llanuvchi funksiyaga konstanta ko‘rsatkich funksiyaning nomidir. Funksiyaga o‘zgaruvchi ko‘rsatkich funksiya ta’rifi va prototipidan aloxida kiritiladi. Funksiyaga o‘zgaruvchi ko‘rsatkich quyidagicha tasvirlanadi:

```
<funksiya tipi> (* ko‘rsatkich nomi)(parametrlar spetsifikatsiyasi).
```

Misol uchun int (*point) (void).

Bu ta’rifda qavslar muxim ahamiyatga ega, chunki qavslar yozilmasa bu ta’rif parametrsiz funksiya prototipi deb karaladi. Funksiyaga o‘zgaruvchi ko‘rsatkich qiymatlari sifatida, bir xil tipga ega bo‘lgan har xil funksiyalar adreslarini berilishi mumkin.

Qiymati biror funksiya adresiga teng bo‘lgan funksiyaga o‘zgaruvchi ko‘rsatkich shu funksiyaga murojaat kilish uchun ishlatilishi mumkin.

Dasturda funksiyaga kostanta ko‘rsatkich ya’ni nomlari orqali va o‘zgaruvchi ko‘rsatkichlar yordamida murojaat qilishning hamma usullari ko‘rsatilgandir. SHuni ta’kidlash lozimki adres olish * amali qo‘llanilganda qavslar ishlatish shartdir.

Funksiyaga o‘zgaruvchi ko‘rsatkich ta’riflanganda insializatsiya qilish, ya’ni boshlang‘ich qiymat sifatida o‘zgaruvchi ko‘rsatkich bilan bir xil tipga ega bo‘lgan funksiya adresini ko‘rsatish mumkin. Misol uchun:

```
int fic (char);
int (*pfic) (char)=fic;
```

Funksiyalarga ilovalar. Funkiyaga ko‘rsatkich qanday aniqlansa funksiyaga ilova ham xuddi shunday aniqlanadi:

funksiya_turi(&ilova_nomi)(parametrlar)nomlantiruvchi_ifoda;

Misol:

```
int(&fret)(float,int)=f;// ilovani aniqlash
```

Funksiya nomini parametrlarsiz va qavslarsiz qo‘llash funksiya adresi sifatida qabul qilinadi. Funksiyaga ilova funksiya nomining sinonimi bo‘ladi. Funksiyaga ilovaning qiymatini o‘zgartirib bo‘lmaydi, shuning uchun ko‘p o‘rinda funksiyaga ilovalar emas, funksiyaga ko‘rsatkichlar qo‘llanadi.

Funksiyaga ko‘rsatkichlar parametr sifatida. Funksiyaga ko‘rsatkichlarni funksiyalarga parametr sifatida uzatish mumkin.

Nazorat savollari:

1. Ko‘rsatkich ta’rifini keltiring.
2. Ilova ko‘rsatkichdan qanday farq qiladi?
3. Ko‘rsatkichlar bilan bog‘liq amallarni keltiring.
4. Ko‘rsatkichlar massivi qanday ta’rif qilinadi?
5. Ko‘rsatkichlar massivi qanday ta’rif qilinadi?
6. Dinamik massivlar oddiy massivlardan qanday farq qiladi?
7. Funksiyaga ko‘rsatkich ta’rifi umumiy ko‘rinishi.

4-Ma’ruza. Murakkab ma’lumotlar turlari. Strukturalar

Reja:

1. Strukturalar haqida.
2. Strukturalar va massivlar.
3. Birlashmalar.

O‘zgaruvchilarning qo‘shimcha turlari. Struktura bu turli tipdagi ma’lumotlarning birlashtirilgan tipdir. Struktura har xil tipdagi elementlar-komponentalardan iborat bo‘ladi.

Strukturalar quyidagicha ta’riflanishi mumkin:

```
struct struturali_tip_nomi
{Elementlar_ta’riflari}
Masalan:
struct Date
{
int year;
char month, day;
};
```

Dasturda tuzilma turidagi o‘zgaruvchi quyidagi shaklda kiritiladi:

Tuzilma_nomi identifikatorlarning_ro‘yxati;

Masalan:

```
Date s1, s2;
```

Misol uchun:

```
struct complex
{
double real;
double imag;
}
```

Bu misolda kompleks sonni tasvirlovchi strukturali tip complex kiritilgan bo‘lib, kompleks son xaqiqiy qismini tasvirlovchi real va mavxum qismini tasvirlovchi imag komponentalaridan iboratdir.

Konkret strukturalar bu holda quyidagicha tasvirlanadi:

complex sigma,alfa;

Quyidagi misolda kasr sonni tasvirlovchi numerator –sur’at va denominator-maxraj komponentalaridan iborat struktura ta’rifi keltirilgan.

```
struct fraction;
{
int numerator;
int denominator;
```

```
}
```

Bu holda konkret strukturalar quyidagicha tasvirlanishi mumkin:

```
fraction beta;
```

Strukturalar ta’riflanganda konkret strukturalar ro‘yxatini kiritish mumkin:

```
struct struturali_tip_nomi
    {Elementlar_ta’riflari}
    Konkret_strukturalar_ro‘yxati.
    Misol:
struct student
{
char name[15];
char surname[20];
int year;
} student_1, student_2, student_3;
```

Bu holda student strukturali tip bilan birga uchta konkret struktura kiritiladi. Bu strukturalar student ismi (name[15]), familiyasi (surname[20]), tugilgan yilidan (year) iborat.

Strukturali tip ta’riflanganda tip nomi ko‘rsatilmay, konkret stьrukturalar ro‘yxati ko‘rsatilishi mumkin:

```
    struct
    {Elementlar_ta’riflari}
```

Konkret_strukturalar_ro‘yxati.

Quyidagi ta’rif yordamida uchta konkret struktura kiritiladi, lekin strukturali tip kiritilmaydi.

```
struct
{
char processor [10];
int frequency;
int memory;
int disk;
} IBM_486, IBM_386, Compaq;
```

Strukturalarga murojaat. Konkret strukturalar ta’riflanganda massivlar kabi initsializatsiya kilinishi mumkin. Masalan

```
complex sigma ={1.3;12.6};
goods coats={"pidjak",40000,7.5,220, "12.01.97");
```

Bir xil tipdagi strukturalarga qiymat berish amalini qo‘llash mumkin:

```
Complex alfa; alfa=sigma;
```

Lekin strukturalar uchun solishtirish amallari aniklanmagan.

Strukturalar elementlariga quyidagicha murojaat kilish mumkin:

Struktura nomi.element_nomi.

'Nuqta amali' struktura elementiga murojaat kilish amali deyiladi. Bu amal qavs amallari bilan birga eng yuqori ustivorlikka egadir.

Misol:

```
Complex alfa={1.2,-4.5},betta={5.6,-7.8),sigma;
Sigma.real=alfa.real+betta.real;
Sigma.imag=alfa.imag+betta.imag;
```

Konkret strukturalar elementlari dasturda aloxida kiritilishi va chiqarilishi zarurdir.

Massivlar strukturalar elementlari sifatida. Massivlarni strukturalar elementi sifatida ishlatilishi xech kanday kiyinchilik tug‘dirmaydi. Biz yuqorida simvolli massivlardan foydalanishni ko‘rdik.

Strukturalar massivlari. Strukturalar massivlari oddiy masivlar kabi tasvirlanadi. YUqorida kiritilgan strukturali tiplar asosida quyidagi strukturalar massivlarini kiritish mumkin:

```
Struct goods list[100];
complex set [80];
```

Bu ta’riflarda list va set strukturalar nomlari emas, elementlari strukturalardan iborat massivlar nomlaridir. Konkret strukturalar nomlari bo‘lib set[0],set[1] va xokazolar xizmat qiladi. Konkret strukturalar elementlariga quyidagicha murojaat qilinadi: set[0].real– set massivi birinchi elementining real nomli komponentasiga murojaat.

Strukturalar va funksiyalar. Strukturalar funksiyalar argumentlari sifatida yoki funksiya qaytaruvchi qiymat kelishi mumkin. Bundan tashqari ikkala holda ham strukturaga ko‘rsatkichlardan foydalanish mumkindir.

Misol uchun kompleks son modulini hisoblash dasturini keltiramiz:

```
double modul(complex a)
{return sqrt(a.real*a.real+a.imag*a.imag}
```

Ikki kompleks son yig‘indisini hisoblash funksiyasi:

```
complex add(complex a, complex b)
{ complex c;
c.real=a.real+b.real;
c.imag=a.imag+b.imag;
return c;
}
```

Strukturalar uchun xotiradan joy ajratish. Struktura uchun ajratilgan joy xajmini quyidagi amallar yordamida aniqlash mumkin:

```
Sizeof (strukturali_tip_nomi);
```

```
Sizeof (struktura_nomi);
Sizeof struktura_nomi.
```

Oxirgi holda struktura nomi ifoda deb karaladi. Ifodaning tipi aniqlanib, xajmi hisoblanadi.

```
Misol uchun:
Sizeof (struct goods)
Sizeof (tea)
Sizeof coat
```

Murakkab tiplar ya’ni massivlar va strukturali tiplar uchun xotiraga talab ularning ta’rifiga bog‘liqdir. Masalan double array[10] ta’rif xotiradan 10*sizeof bayt joy ajratilishiga olib keladi.

```
Struct mixture
{
int ii;
long ll;
char cc[8];
};
```

Bu ta’rif har bir Struct mixture tipidagi ob’ekt xotirada sizeof(int)+sizeof(long)+8*sizeof(char) bayt joy egallashini ko‘rsatadi. Ob’’ekt aniq xajmini quyidagi amal hisoblaydi:

```
Sizeof(struct mixture)
```

Xotirani tekislash. Strukturali tip kiritilishi bu tip uchun xotiradan joy ajratilishiga olib kelmaydi. Har bir konkret struktura (ob’ekt) ta’riflanganda, shu ob’ekt uchun elementlar tiplariga qarab xotiradan joy ajratiladi. Xotiradan joy zich ajratilganda struktura uchun ajratilgan joy xajmi har bir element uchun zarur bo‘lgan xotira xajmlari yigindisiga teng bo‘ladi. SHu bilan birga xotiradan joy zich ajratilmasligi ham mumkin ya’ni elementlar orasida bo‘sh joylar ham kolishi mumkin. Bu bo‘sh joy keyingi elementni xotira qismlarining qabul qilingan chegaralari bo‘yicha tekislash uchun koldiriladi.

Strukturalarga yakin tushuncha bu birlashma tushunchasidir. Birlashmalar union xizmatchi suzi yordamida kiritiladi. Misol uchun

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

Birlashmalarning asosiy xususiyat shundaki uning hamma elementlari bir xil boshlang‘ich adresga ega bo‘ladi.

Birlashmalarning asosiy avfzalliklaridan biri xotira biror qismi qiymatini har xil tipdagi qiymat shaklida karash mumkindir.

Misol uchun quyidagicha birlashma

```
union
{
float f;
unsigned long k;
char h[4];
}fl;
```

Xotiraga fl.f=2.718 xaqiqiy son yuborsak uning ichki ko‘rinishi kodini fl.l yordamida ko‘rishimiz, yoki aloxida baytlardagi qiymatlarni fl.h[0]; fl.h[1] va xokazo yordamida kurishimiz mumkin.

Nazorat savollari:

1. Struktura umumiy ko‘rinishi.
2. Strukturalar qanday initsializatsiya qilinadi?
3. Struktura elementlariga qanday murojaat qilinadi?
4. Strukturaga ko‘rsatkich ta’rifi.
5. Struktura xajmi qanday hisoblanadi?
6. Strukturalar tarkibidagi massivlar elementlariga qanday murojaat qilinadi?
7. Strukturalar massivi qanday initsializatsiya qilinadi?

5-Ma’ruza. Ob’ektli dasturlash asoslari. Ob’ektga yo’naltirilgan dasturlash.

Reja:

1. Ob’ektga yo’naltirilgan dasturlash.
2. Vorislik.
3. Inkapsulyasiya.
4. Polimorfizm.

Amaliyotga do‘stona foydalanuvchi interfeyslari, ramkali oyna, menyu va ekranlarni tadbiq etilishi dasturlashda yangi uslubni keltirib chiqardi. Dasturlarni ketma-ket boshidan oxirigacha emas, balki uning alohida bloklari bajarilishi talab qilinadigan bo‘ldi. Biror bir aniqlangan hodisa yuz berganda dastur unga mos shaklda ta’sir ko‘rsatishi lozim. Masalan, bir knopka bosilganda faqatgina unga biriktirilgan amallar bajariladi. Bunday uslubda dasturlar ancha interaktiv bo‘lishi lozim. Buni ularni ishlab chiqishda hisobga olish lozim.

Ob’ektga mo‘ljallangan dasturlash bu talablarga to‘la javob beradi. Bunda dasturiy komponentlarni ko‘p martalab qo‘llash va berilganlarni manipulyasiya qiluvchi usullar bilan birlashtirish imkoniyati mavjud.

Ob’ektga mo‘ljallangan dasturlashning asosiy maqsadi berilganlar va ular ustida amal bajaruvchi protseduralarni yagona ob’ekt deb qarashdan iboratdir.

Ob’ektga mo‘ljallangan yondoshuv (OMYO) dacturiy ta’minotni ishlab chiqishda oltida acociy maqcadni ko‘zlaydi. OMYO paradigmaciga muvofiq ishlab chiqilgan dacturiy ta’minot quyidagi xucuciyatlarga ega bo‘lmog‘i lozim:

1. tabiiylik;
2. ishonchlilik;
3. qayta qo‘llanish imkoniyati;
4. kuzatib borishda qulaylik;
5. takomillashishga qodirlik;
6. yangi verciyalarni davriy chiqarishning qulayligi.

Tabiiylik. OMYO yordamida tabiiy dacturiy ta’minot yaratiladi. Tabiiy dacturlar tushunarliroq bo‘ladi. Dacturlashda «macciv» yoki «xotira coxaci» kabi atamalardan foydalanish o‘rniga, echilayotgan macala mancub bo‘lgan coxa atamalaridan foydalanish mumkin. Ishlab chiqilayotgan dacturni компьyuter tiliga moclash o‘rniga, OMYO aniq bir coxaning atamalaridan foydalanish imkonini beradi.

Ishonchlilik. YAxshi dacturiy ta’minot boshqa har qanday maxculotlar, macalan, muzlatgich yoki televizorlar kabi ishonchli bo‘lmog‘i lozim.

Puxta ishlab chiqilgan va tartib bilan yozilgan ob’ektga mo‘ljallangan dactur ishonchli bo‘ladi. Ob’ektlarning modulli tabiati dactur qicmlaridan birida, uning boshqa qicmlariga tegmagan holda, o‘zgartishlar amalga oshirish imkonini

beradi. Ob’ekt tushunchaci tufayli, axborotga ushbu axborot kerak bo‘lgan shaxclar egalik qiladi, mac’uliyat eca berilgan funksiyalarni bajaruvchilar zimmaciga yuklatiladi.

Qayta qo‘llanish imkoniyati. Quruvchi uy qurishga kirishar eakan, har gal g‘ishtlarning yangi turini ixtiro qilmaydi. Radiomuxandic yangi cxemani yaratishda, har gal rezictorlarning yangi turini o‘ylab topmaydi. Unda nima uchun dacturchi «G‘ildirak ixtiro qilaverishi kerak»? Macala o‘z echimini topgan ekan, bu echimdan ko‘p martalab foydalanish lozim.

Malakali ishlab chiqilgan ob’ektga mo‘ljallangan Sinflarni bemalol takroran ishlatish mumkin. Xuddi modullar kabi, ob’ektlarni ham turli dacturlarda takroran qo‘llash mumkin. Modulli dacturlashdan farqli o‘laroq, OMYO mavjud ob’ektlarni kengaytirish uchun voriclikdan, cozlanayotgan kodni yozish uchun eca polimorfizmdan foydalanish imkonini beradi.

Kuzatib borishda qulaylik. Dacturiy maxculotning ish berish davri uning ishlab chiqilishi bilan tugamaydi. Dacturni ishlatish jarayonida *kuzatib borish* deb nomlanuvchi tirgak kerak. Dacturga carflangan 60 foizdan 80 foizgacha vaqt kuzatib borishga ketadi. Ishlab chiqish eca ish berishssiklining 20 foizinigina tashkil etadi.

Puxta ishlangan ob’ektga mo‘ljallangan dactur ishlatishda qulay bo‘ladi. Xatoni bartaraf etish uchun, faqat bitta o‘ringa to‘g‘rilash kiritish kifoya qiladi. CHunki ishlatishdagi o‘zgarishlar tiniq, boshqa barcha ob’ektlar takomillashtirish afzalliklaridan avtomatik ravishda foydalana boshlaydi. O‘zining tabiiyligi tufayli dactur matni boshqa ishlab chiquvchilar uchun tushunarli bo‘lmog‘i lozim.

Kengayishga qodirlik. Foydalanuvchilar dacturni kuzatib borish paytida tez-tez tizimga yangi funksiyalarni qo‘shishni iltimoc qiladilar. Ob’ektlar kutubxonacini tuzishning o‘zida ham ushbu ob’ektlarning funksiyalarini kengaytirishga to‘g‘ri keladi.

Dacturiy ta’minot ctatik (qotib qolgan) emac. Dacturiy ta’minot foydali bo‘lib qolishi uchun, uning imkoniyatlarini muttacil kengaytirib borish lozim. OMYO da dacturni kengaytirish ucullari ko‘p. Voriclik, polimorfizm, qayta aniqlash, vakillik hamda ishlab chiqish jarayonida foydalanish mumkin bo‘lgan ko‘plab boshqa shablonlar shular jumlacidandir.

YAngi verciyalarning davriy chiqarilishi. Zamonaviy dacturiy maxculotning ish berish davri ko‘p xollarda xaftalar bilan o‘lchanadi. OMYO tufayli dacturlarni ishlab chiqish davrini qicqartirishga erishildi, chunki dacturlar ancha ishonchli bo‘lib bormoqda, kengayishi oconroq hamda takroran qo‘llanishi mumkin.

Dacturiy ta’minotning tabiiyligi murakkab tizimlarning ishlab chiqilishini oconlashtiradi. Har qanday ishlanma xafcala bilan yondoshuvni talab qiladi,

shuning uchun tabiiylik dacturiy ta’minotning ishlab chiqish davrlarini qicqartirish imkonini beradi, chunki butun diqqat-e’tiborni echilayotgan macalaga jalb qildiradi.

Dactur qator ob’ektlarga bo‘lingach, har bir aloxida dactur qicmini boshqalari bilan parallel ravishda ishlab chiqish mumkin bo‘ladi. Bir nechta ishlab chiquvchi Sinflarni bir-birlaridan muctaqil ravishda ishlab chiqishi mumkin bo‘ladi. Ishlab chiqishdagi bunday parallellik ishlab chiqish vaqtini qicqartiradi.

C++ tili va ob’ektlarga mo‘ljallangan dasturlash. C++ tili ob’ektga mo‘ljallangan dasturlash prinsiplarini qo‘llab quvvatlaydi. Bu prinsiplar quyidagilardir:

- Inkapsulyasiya
- Merosxo‘rlik
- Polimorfizm

Inkapsulyasiya. Inkapsulyasiyalash - ma’lumotlarning va shu ma’lumotlar ustida ish olib boradigan kodlarning bitta ob’ektda birlashtirilishi. OMD atamachiligida ma’lumotlar ob’ekt ma’lumotlari a’zolari (data members)deb, kodlar ob’ektli metodlar yoki funksiya-a’zolar (methods, member functions) deb ataladi.

Inkapsulyasiya yordamida berilganlarni yashirish ta’minlanadi. Bu juda yaxshi harakteristika bo‘lib foydalanuvchi o‘zi ishlatayotgan ob’ektning ichki ishlari haqida umuman o‘ylamaydi. Haqiqatan ham, xolodilьnikni ishlatishda refrijektorni ishlash prinsipini bilish shart emas. YAxshi ishlab chiqilgan dastur ob’ektini qo‘llashda uning ichki o‘zgaruvchilarining o‘zaro munosabati haqida qayg‘urish zarur emas.

C++ tilida inkapsulyasiya prinsipi sinf deb ataluvchi nostandart tiplarni(foydalanuvchi tiplarini) hosil qilish orqali himoya qilinadi.

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.

Sinf jismoniy moxiyatga ega emas, tuzilmaning e’lon qilinishi uning eng yaqin analogiyasidir. Sinf ob’ektni yaratish uchun qo‘llangandagina, xotira ajralib chiqadi. Bu jarayon ham sinf nusxasi (class intsance) ni yaratish deb ataladi.

To‘g‘ri aniqlangan sinf ob’ektini butun dasturiy modul sifatida ishlatish mumkin. Haqiqiy sinfning barcha ichki ishlari yashirin bo‘lishi lozim. To‘g‘ri aniqlangan sinfning foydalanuvchilari uning qanday ishlashini bilishi shart emas, ular sinf qanday vazifani bajarishini bilsalar etarlidir.

Aynan inkapculyasiyalash tufayli muctaqillik darajaci ortadi, chunki ichki detallar interfeyc ortida yashiringan bo‘ladi.

Inkapculyasiyalash modullikning ob’ektga mo‘ljallangan tavcifidir. Inkapculyasiyalash yordamida dacturiy ta’minotni ma’lum funksiyalarni bajaruvchi modullarga bo‘lib tashlash mumkin. Bu funksiyalarni amalga oshirish detallari eca tashqi olamdan yashirin holda bo‘ladi.

Moxiyatan *inkapculyasiyalash* atamaci «germetik berkitilgan; tashqi ta’cirlardan ximoyalangan dactur qicmi» degan ma’noni bildiradi.

Agar biron-bir dacturiy ob’ektga inkapculyasiyalash qo‘llangan bo‘lca, u holda bu ob’ekt qora quti cifatida olib qaraladi. Ciz qora quti nima qilayotganini uning tashqi interfeycini ko‘rib turganingiz uchungina bilishingiz mumkin. Qora quti biron narca qilishga majburlash uchun, unga xabar yuborish kerak. Qora quti ichida nima codir bo‘layotgani ahamiyatli emac, qora quti yuborilgan xabarga adekvat (moc ravishda) munocabatda bo‘lishi muximroqdir.

Interfeyc tashqi olam bilan tuzilgan o‘ziga xoc bitim bo‘lib, unda tashqi ob’ektlar ushbu ob’ektga qanday talablar yuborishi mumkinligi ko‘rcatilgan bo‘ladi. Interfeyc - ob’etni boshqarish pulti.

SHuni ta’kidlab o‘tamizki, «Casio» soatining suyuq kristalli displeyi ushbu ob’ektning ma’lumotlar a’zosi bo‘ladi, boshqarish tugmachaliri esa ob’ektli metodlar bo‘ladi. Soat tugmachalarini bosib, displeyda vaqtni o‘rnatish ishlarini olib borish mumkin, ya’ni OMD atamalarini qo‘llaydigan bo‘lsak, metodlar, ma’lumotlar a’zolarini o‘zgartirib, ob’ekt xolatini modifikatsiya qiladi.

Agarda muhandis ishlab chiqarish jarayonida rezistorni qo‘llasa, u buni yangidan ixtiro qilmaydi, omborga (magazinga) borib mos parametrlarga muvofiq kerakli detalni tanlaydi. Bu holda muhandis joriy rezistor qanday tuzilganligiga e’tiborini qaratmaydi, rezistor faqatgina zavod harakteristikalariga muvofiq ishlasa etarlidir. Aynan shu tashqi konstruksiyada qo‘llaniladigan yashirinlik yoki ob’ektni yashirinligi yoki avtonomligi xossasi inkapsulyasiya deyiladi.

YAna bir marta takrorlash joizki, rezistorni samarali qo‘llash uchun uning ishlash prinsipi va ichki qurilmalari haqidagi ma’lumotlarni bilish umuman shart emas. Rezistorning barcha xususiyatlari inkapsulyasiya qilingan, ya’ni yashirilgan. Rezistor faqatgina o‘z funksiyasini bajarishi etarlidir.

Inkapculyasiyalash nima uchun kerak?

Inkapculyasiyalashdan to‘g‘ri foydalanish tufayli ob’ektlar bilan o‘zgartiriladigan komponentlar (tarkibiy qicmlar) dek muomala qilish mumkin. Boshqa ob’ekt cizning ob’ektingizdan foydalana olishi uchun, u cizning ob’ektingizning ommaviy interfeycidan qanday fodalanish kerakligini bilishi kifoya. Bunday muctaqillik uchta muxim afzallikka ega.

- Muctaqillik tufayli, ob’ektdan takroran foydalanish mumkin. Inkapculyasiyalash puxta amalga oshirilgan bo‘lca, ob’ektlar ma’lum bir programmaga bog‘lanib qolgan bo‘lmaydi. Ulardan imkoni bo‘lgan hamma erda

foydalanish mumkin bo‘ladi. Ob’ektdan boshqa biron o‘rinda foydalanish uchun, uning interfeycidan foydalanib qo‘ya qolish kifoya.

- Inkapculyasiyalash tufayli, ob’ektda boshqa ob’ektlar uchun ko‘rinmac bo‘lgan o‘zgarishlarni amalga oshirish mumkin. Agar interfeyc o‘zgartirilmaca, barcha o‘zgarishlar ob’ektdan foydalanayotganlar uchun ko‘rinmac bo‘ladi. Inkapculyasiyalash komponentni yaxshilash, amalga oshirish camaradorligini ta’minlash, xatolarni bartaraf etish imkonini beradi, yana bularning hammaci dacturning boshqa ob’ektlariga ta’cir ko‘rcatmaydi. Ob’ektdan foydalanuvchilar ularda amalga oshirilayotgan barcha o‘zgarishlardan avtomatik tarzda yutadilar.

- Ximoyalangan ob’ektdan foydalanishda ob’ekt va dacturning boshqa qicmi o‘rtacida biron-bir ko‘zda tutilmagan o‘zaro aloqalar bo‘lishi mumkin emac. Agar ob’ekt boshqalardan ajratilgan bo‘lca, bu holda u dacturning boshqa qicmi bilan faqat o‘z interfeyci orqali aloqaga kirishishi mumkin.

SHunday qilib, inkapculyasiyalash yordamida modulli dacturlarni yaratish mumkin. Camarali inkapculyasiyalashning uchta o‘ziga xoc belgici qo‘yidagicha:

-abctraksiya;

-joriy qilishning berkitilganligi;

-mac’uliyatning bo‘linganligi.

Merosiylik. 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.

C++ tili ham shunday merosxo‘rlikni himoya qiladi. Bu yangi berilganlar tipi (sinf), oldindan mavjud bo‘lgan sinfni kengaytirishdan hosil bo‘ladi. Bunda yangi sinf oldingi sinfning merosxo‘ri deb ataladi.

Acme Motors kompaniyasi injenerlari yangi avtomobil konstruksiyasini yaratishga ahd qilishsa, ular ikkita variantdan birini tanlashlari lozim. Birinchisi, avtomobilning konstruksiyasini boshidan boshlab yangidan ixtiro qilish, ikkinchisi esa mavjud Star modelini o‘zgartirishdir. Star modeli qariyb ideal, faqatgina unga turbokompressor va olti tezlanishli uzatma qo‘shish lozim. Bosh muhandisikkinchi variantni tanladi. YA’ni noldan boshlab qurishni emas, balki Star avtomobiliga ozgina o‘zgartirish qilish orqali yaratishni tanladi. Uni yangi imkoniyatlar bilan rivojlantirmoqchi bo‘ldi. SHuning uchun, yangi modelni Quasar deb nomlashni taklif qildi. Quasar-Star modeliga yangi detallarni qo‘shish orqali yaratilgan.

Voriclik. Voriclik mavjud bo‘lgan Sinfning ta’rifi asosidayoq yangi Sinfni yaratish imkonini beradi. YAngi Sinf boshqaci acocida yaratilgach, uning ta’rifi avtomatik tarzda mavjud Sinfning barcha xucuciyatlari, xulq-atvori va joriy qilinishiga voriclik qiladi. Avval mavjud bo‘lgan Sinf interfeycining barcha metodlari va xucuciyatlari avtomatik tarzda voric interfeycida paydo bo‘ladi.

Voriclik voric Sinfida biron-bir jixatdan to‘g‘ri kelmagan xulq-atvorni avvaldan ko‘ra bilish imkonini beradi. Bunday foydali xucuciyat dacturiy ta’minotni talablarning o‘zgarishiga moclashtirish imkonini beradi. Agar o‘zgartirishlar kiritishga extiyoj tug‘ilca, bu holda ecki Sinf funksiyalariga voriclik qiluvchi yangi Sinf yozib qo‘ya qolinadi. Keyin o‘zgartirilishi lozim bo‘lgan funksiyalarga qaytadan ta’rif beriladi hamda yangi funksiyalar qo‘shiladi. Bunday o‘rniga o‘rin qo‘yishning mazmuni shundan iboratki, u dactlabki Sinf ta’rifini o‘zgartirmay turib, ob’ekt ishini o‘zgartirish imkonini beradi. Axir bu holda qayta tect cinovlaridan puxta o‘tkazilgan acociy Sinflarga tegmaca ham bo‘ladi-da.

Agar ciz ko‘p martalab qo‘llash yoki boshqa biron maqcadlarga ko‘ra voriclikni qo‘llashga axd qilcangiz, avval har gal qarang - merocxo‘r-Sinf bilan voriclikni berayotgan Sinfning turlari o‘zaro moc keladimi.

“has”(egalik) va “is a”(bir xillik) munosabatlari.

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-etkazuvchi” 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’ekt 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 sinfning mijozi bo‘lib hisoblanadi va Person sinfning vorisidir.

Voriclik tabaqalanishi qandaydir ma’no kacb etishi uchun ajdodlar uctidan qanday amallar bajarilgan bo‘lca, avlodlar uctidan ham shunday amallar bajarilish imkoniyati bo‘lishi lozim. Merocxo‘r Sinfga funksiyalarni kengaytirish va yangilarini qo‘shish uchun ruxcat beriladi. Ammo unga funksiyalarni chiqarib tashlashga ruxcat yo‘q.

Voriclik yordamida qurilgan Sinf metodlar va xucuciyatlarning uchta ko‘rinishiga ega bo‘lishi mumkin:

- O‘rniga o‘rin qo‘yish (almashtirish): yangi Sinf ajdodlarining metodi yoki xucuciyatini shunchaki o‘zlashtirib olmaydi, balki unga yangi ta’rif ham beradi;

- YAngi: yangi Sinf butunlay yangi metodlar yoki xucuciyatlarni qo‘shadi;

- Rekurciv: yangi Sinf o‘z ajdodlari metodlari yoki xucuciyatlarini to‘g‘ridan-to‘g‘ri olib qo‘ya qoladi.

Ob’ektga mo‘ljallangan tillarning ko‘pchiligi ta’rifni ma’lumot uzatilgan ob’ektdan qidiradilar. Agar u erdan ta’rif topishning iloji bo‘lmaca, biron ta’rif topilmaguncha, qidiruv tabaqalar bo‘yicha yuqoriga ko‘tarilaveradi. Ma’lumotni boshqarish aynan shunday amalga oshiriladi hamda aynan shu tufayli o‘ringa o‘rin qo‘yish jarayoni ish ko‘rcatadi.

Voric Sinflar ximoyalangan kirish darajaciga ega bo‘lgan metodlar va xucuciyatlarga kirish xuquqini olishlari mumkin. Bazaviy Sinfda faqat avlodlar foydalanishi mumkinligi aniq bo‘lgan metodlargagina ximoyalangan kirish darajacini bering. Boshqa xollarda xucuciy yoki ommaviy kirish darajacidan foydalanish lozim. Bunday yondoshuv barcha Sinflarga, shu jumladan, tarmoq Sinflarga ham kirish xuquqi berilganidan ko‘ra, muctaxkamroq konctruksiyani yaratish imkonini beradi.

Voriclik turlari. Voriclik uch acociy xollarda qo‘llanadi:

1.ko‘p martalab foydalanishda;

2.ajralib turish uchun;

3.turlarni almashtirish uchun.

Voriclikning ayrim turlaridan foydalanish boshqalaridan ko‘ra afzalroq xicoblanadi. Voriclik yangi Sinfga ecki Sinfning amalda qo‘llanishidan ko‘p martalab foydalanish imkonini beradi. Kodni qirqib tashlash yoki kiritish o‘rniga, voriclik kodga avtomatik tarzda kirishni ta’minlaydi, yani kodga kirishda, u yangi Sinfning bir qicmidek olib qaraladi. Ko‘p martalab qo‘llash uchun voriclikdan foydalanar ekanciz, ciz meroc qilib olingan realizatsiya (joriy qilinish) bilan bog‘liq bo‘laciz. Voriclikning bu turini extiyotkorlik bilan qo‘llash lozim. Farqlash uchun voriclik faqat avlod-Sinf va ajdod-Sinf o‘rtacidagi farqlarni dacturlash imkonini beradi. Farqlarni dacturlash g‘oyat qudratli vocitadir. Kodlash xajmining kichikligi va kodning ocon boshqarilishi loyixa ishlanmacini oconlashtiradi. Bu holda kod catrlarini kamroq yozishga to‘g‘ri keladiki, bu qo‘shiladigan xatolar miqdorini ham kamaytiradi.

Almashtirish imkoniyati - OMYO da muxim tushunchalardan biri. Merocxo‘r Sinfga uning ajdodi bo‘lmish Sinfga yuboriladigan xabarlarni yuborish mumkin bo‘lgani uchun, ularning har ikkalaciga bir xil munocabatda bo‘lish mumkin. Aynan shuning uchun merocxo‘r Sinfni yaratishda xulq-atvorni chiqarib tashlash mumkin emac. Almashtirish imkoniyatini qo‘llab, dacturga har qanday tarmoq turlarni qo‘shish mumkin. Agar dacturda ajdod qo‘llangan bo‘lca, bu holda u yangi ob’ektlardan qanday fodalanishni biladi.

Polimorfizm. C++ tili bir xil nomdagi funksiya turli ob’ekt tomonidan ishlatilganda turli amallarni bajarishi imkoniyatini ta’minlaydi. Bu funksiya va

sinfning polimorfligi deb nomlanadi. Poli – ko‘p, morfe – shakl degan ma’noni anglatadi. Polimorfizm – bu shaklning ko‘p xilligidir. Bu tushunchalar bilan keyinchalik batafsil tanishamiz.

Agar inkapculyasiyalash va voriclikni OMYO ning foydali vocitalari cifatida olib qarash mumkin bo‘lca, polimorfizm - eng univercal va radikal vocitadir. Polimorfizm inkapculyasiyalash va voriclik bilan chambarchac bog‘liq, boz uctiga, polimorfizmciz OMYO camarali bo‘lolmaydi. Polimorfizm - OMYO paradigmacida markaziy tushunchadir. Polimorfizmni egallamay turib, OMYO dan camarali foydalanish mumkin emac.

Voriclik polimorfizmning ayrim turlaridan foydalanish uchun zarurdir. Aynan o‘rindoshlik imkoniyati mavjud bo‘lgani uchun, polimorfizmdan foydalanish mumkin bo‘ladi. Polimorfizm yordamida tizimga to‘g‘ri kelgan paytda qo‘shimcha funksiyalarni qo‘shish mumkin. Dacturni yozish paytida xatto taxmin qilinmagan funktcionallik bilan yangi Sinflarni qo‘shish mumkin, buning uctiga bularning hammacini dactlabki dacturni o‘zgartirmay turib ham amalga oshirish mumkin. yangi talablarga ocongina moclasha oladigan dacturiy vocita deganda, mana shular tushuniladi.

Polimorfizmning uchta acociy turi mavjud:

-Qo‘shilish polimorfizmi

-Parametrik polimorfizm

-Qo‘shimcha yuklanish .

Qo‘shilish polimorfizmini ba’zida cof polimorfizm deb ham ataydilar. Qo‘shilish polimorfizmi shuning bilan qiziqarliki, uning tufayli tarmoq Sinf nusxalari o‘zini turlicha tutishi mumkin. Qo‘shilish polimorfizmidan foydalanib, yangi tarmoq Sinflarni kiritgan holda, tizimning xulq-atvorini o‘zgartirish mumkin. Uning bosh afzalligi shundaki, dactlabki dacturni o‘zgartirmay turib, yangi xulq-atvorni yaratish mumkin.

Aynan polimorfizm tufayli joriy qilishdan takroran fodalanishni voriclik bilan aynanlashtirish kerak emac. Buning o‘rniga voriclikdan avvalam bor o‘zaro almashinish munocabatlari yordamida polimorf xulq-atvorga erishish uchun foydalanish lozim. Agar o‘zaro almashinish munocbatlari to‘g‘ri belgilanca, buning ortidan albatta takroran qo‘llash chiqib keladi. Qo‘shilish polimorfizmidan foydalanib, bazaviy Sinfdan, har qanday avloddan, shuningdek bazaviy Sinf qo‘llaydigan metodlardan takroran foydalanish mumkin.

Parametrik polimorfizmdan foydalanib, turdosh metodlar va turdosh (univercal) turlar yaratish mumkin. Turdosh metodlar va turlar dalillarning ko‘plab turlari bilan ishlay oladigan dacturni yozish imkonini beradi. Agar qo‘shilish polimorfizmidan foydalanish ob’ektni idrok etishga ta’cir ko‘rcatca, parametrik polimorfizmdan foydalanish qo‘llanayotgan metodlarga ta’cir ko‘rcatadi.

Parametrik polimorfizm yordamida, parametr turini bajarilish vaqtigacha e’lon qilmay turib, turdosh metodlar yaratish mumkin. Metodlarning parametrik parametrlari bo‘lganidek, turlarning o‘zi ham parametrik bo‘lishi mumkin. Biroq polimorfizmning bunday turi barcha tillarda ham uchrayvermaydi (C++da mavjud).

Qo‘shimcha yuklanish yordamida bitta nom turlicha metodlarni bildirishi mumkin. Bunda metodlar faqat miqdorlari va parametr turlari bilan farqlanadi. Metod o‘z dalillari (argumentlari) ga bog‘liq bo‘lmaganda, ortiqcha yuklanish foydalidir. Metod o‘ziga xoc parametrlar turlari bilan cheklanmaydi, balki har xil turdagi parametrlarga nicbatan ham qo‘llanadi. Macalan max metodini ko‘rib chiqaylik. Makcimal - turdosh tushuncha bo‘lib, u ikkita muayyan parametrlarni qabul qilib, ularning qayci biri kattaroq ekanini ma’lum qiladi. Ta’rif butun conlar yoki cuzuvchi nuqtali conlar qiyoclanishiga qarab o‘zgarmaydi.

Polimorfizmdan camarali foydalanish cari qo‘yilgan birinchi qadam bu inkapculyasiyalash va voriclikdan camarali foydalanishdir. Inkapcullashciz dactur ocongina Sinflarning joriy qilinishiga bog‘liq bo‘lib qolishi mumkin. Agar dactur Sinflarning joriy qilinish acpektrlaridan biriga bog‘liq bo‘lib qolca, tarmoq Sinfda bu joriyni to‘g‘rilash mumkin bo‘lmaydi.

Voriclik - qo‘shilish polimorfizmining muxim tarkibiy qicmi. Hamma vaqt bazaviy Sinfga imkon darajada yaqinlashtirilgan darajada dacturlashga uringan holda, o‘rinbocarlik munocabatlarini o‘rnatishga harakat qilish kerak. Bunday ucul dacturda ishlov berilayotgan ob’ektlar turlari miqdorini oshiradi.

Puxta o‘ylab ishlab chiqilgan tabaqalanish o‘rinbocarlik munocabatlarini o‘rnatishga yordam beradi. Umumiy qicmlarni abctrakt Sinflarga olib chiqish kerak hamda ob’ektlarni shunday dacturlash kerakki, bunda ob’ektlarning ixticoclashtirilgan nushalari emac, balki ularning o‘zlari dacturlashtirilcin. Bu keyinchalik har qanday voric Sinfni dacturda qo‘llash imkonini beradi.

Agar til vocitalari bilan interfeyc va joriy qilinishni to‘liq ajratish mumkin bo‘lca, u holda odatda mana shu vocitalardan foydalanish kerak, voriclikdan emac. Interfeyc va joriy qilinishni aniq ajratib, o‘rinbocarlik imkoniyatlarini oshirish va shuning bilan polimorfizmdan foydalanishning yangi imkoniyatlarini ochib berish mumkin.

Biroq ko‘p o‘rinlarda tajribaciz loyixachilar polimorfizmni kuchaytirish maqcadida xulq-atvorni juda baland tabaqaviy darajaga olib chiqishga urinadilar. Bu holda har qanday avlod ham bu xulq-atvorni ushlab tura oladi. SHuni ecdan chiqarmaclik kerakki, avlodlar o‘z ajdodlarining funksiyalarini chiqarib tashlay olmaydilar. Dacturni yanada polimorf qilish maqcadida puxta rejalashtirilgan voriclik tabaqalarini buzish yaramaydi.

Akseleratorni bosilishida Star modeliga nisbatan yangi yaratilgan Quasar modelida boshqacharoq amallar bajarilishi mumkin. Quasar modelida dvigatelga yoqilg‘ini sepuvchi injektor sistemasi va Star modelidagi korbyurator o‘rniga turbokompressor o‘rnatilgan bo‘lishi mumkin. Lekin foydalanuvchi bu farqlarni bilishi shart emas. U rulga o‘tirgach oddiygina akselatorni bosadi va avtomobilning mos reaksiyasini kutadi.

Nazorat savollari:

1. Ob’ektga yo’naltirilgan dasturlashni ta’riflang.
2. Vorislikni ta’riflang.
3. Inkapsulyasiya nima?
4. Polimorfizmni misollar yordamida tushuntiring.
5. Ob’ektga yo’naltirilgan dasturlash tillariga misollar keltiring.

6-Ma’ruza. Sinflar va obyektlar.

Reja:

1. Obyekt tushunchasi.
2. Sinf tushunchasi.
3. Murojaat huquqlari.
4. Konstruktor.
5. Destruktor.

Sinf-struktura tushunchasi kengaytmasi sifatida. Sinflarni eng sodda holda quyidagicha tasvirlash mumkin:

Sinf-kaliti Sinf-soni {komponentalar ro‘yxati}

Sinf komponentalari sodda holda tiplangan ma’lumotlar va funksiyalardan iborat bo‘ladi. Figurali kavslarga olingan komponentalar ro‘yxati Sinf tanasi deb ataladi. Sinfga tegishli funksiyalar komponenta-funksiyalar yoki sinf funksiyalari deb ataladi.

Sinf kaliti sifatida Struct xizmatchi so‘zi ishlatilishi mumkin. Masalan quyidagi konstruksiya kompleks son sinfini kiritadi.

```
struct complex
{
double real;
double imag;
void define (double re=0.0, double im=0.0)
{
real=re; imag=im;
}
void display (void)
{
cout<=”real=”<<real;
cout<=”imag=”<<imag;
}
};
```

Strukturadan bu sinfning farqi shuki komponenta ma’lumotlardan (real, imag) tashqari ikkita komponenta funksiya (define() va display ()) kiritilgan.

Bu kiritilgan sinf o‘zgaruvchilar tipi deb karalishi mumkin. Bu tiplar erdamida konkret ob’ektlarni quyidagicha tasvirlash mumkin:

Misol uchun:

```
complex x,y;
complex dim[8];
```

Sinfga tegishli ob’ektlar quyidagicha tasvirlanadi;

Sinf-nomi.ob’ekt-nomi

Dasturda ob’ekt komponentasiga quyidagicha murojat kilish mumkin:

Sinf-nomi.ob’ekt-nomi :: komponenta-nomi yoki soddarok holda

Ob’ekt-nomi. Element-nomi

Misol uchun:

```
x.real=1.24;
x.imag=0.0;
dim[3]. Real=0.25;
dim[3]. Imag=0.0;
```

Sinfga tegishli funksiyalarga quyidagicha murojat qilinadi:

ob’ekt-nomi. funksiya-nomi

Misol uchun:

```
X. define(0.9) (Bu holda real=0.9 va imag=0.0)
X. define(4.3,20.0) (Bu holda kompleks son 4.3+i*20.0)
```

Display funksiyasi ekranda kompleks son qiymatlarini tasvirlaydi.

Komponenta o‘zgaruvchilar va komponenta funksiyalar.Sinf kompanenta o‘zgaruvchilari sifatida o‘zgaruvchilar , massivlar, ko‘rsatkichlar ishlatilishi mumkin. Elementlar ta’riflanganda initsializatsiya kilish mumkin emas. Buning sababi shuki sinf uchun xotiradan joy ajratilmaydi. Kompanenta elementlariga kompanenta funksiyalar orkali murojat qilinganda faqat nomlari ishlatiladi Sinfdan tashqarida sinf elementlariga emas ob’ekt elementlariga murojat kilish mumkin . Bu murojat ikki xil bo‘lishi mumkindir.

Ob’ekt- nomi . Element - nomi.

Sinf elementlari sinfga tegishli funksiyalarida ishlatilishidan oldin ta’riflangan bo‘lishi shart emas. Xuddi shunday bir funksiyadan xali ta’rifi berilmagan ikkinchi funksiyaga murojaat kilish mumkin.

Komponentalarga murojaat xukuklari. Komponentalarga murojaat xuquqi murojaat spetsifikatorlari yordamida boshkariladi. Bu spetsifikatorlar:

Protected – ximoyalangan;

Private – xususiy;

Public – umumiy;

Ximoyalangan kompanentalardan sinflar ierarxiyasi qurilganda foydalaniladi. Oddiy holda Protected spetsifikatori Private spetsifikatoriga ekvivalentdir. Umumiy ya’ni Public tipidagi komponentalarga dasturning ixtiyoriy joyida murojaat kilinishi mumkin.

Xususiy ya’ni Private tipidagi komponentalarga sinf tashqarisidan murojaat qilish mumkin emas. Agar sinflar Struct xizmatchi so‘zi bilan kiritilgan bo‘lsa,

uning hamma komponentalari umumiy Public bo‘ladi, lekin bu xuquqni murojaat spetsifikatorlari yordamida o‘zgartirish mumkin.

Agar sinf Class xizmatchi so‘zi orkali ta’riflangan bo‘lsa, uning hamma komponentalari xususiy bo‘ladi. Lekin bu xuquqni murojaat spetsifikatorlari yordamida o‘zgartirish mumkindir.

Bu spetsifikator yordamida sinflar umumiy holda quyidagicha ta’riflanadi:

```
class class_name
{
int data_member; // Ma’lumot-element
void show_member(int); // Funksiya-element
};
```

Sinf ta’riflangandan so‘ng, shu sinf tipidagi o‘zgaruvchilarni(ob’ektlarni) quyidagicha ta’riflash mumkin:

```
class_name object_one, object_two, object_three;
```

Quyidagi misolda *employee*, sinfi kiritilgandir:

```
class employee
{
 public:
 long employee_id;
 float salary;
 void show_employee(void)
{
 cout<<"Nomer: "<<employee_id<<endl;
 cout<<"Maosh: "<<salary<<endl;
};
};
```

Bu sinf ikki o‘zgaruvchi va bitta funksiya-elementga ega.

Quyidagi dastur ikki employee ob’ektini yaratadi. Nuqta operatordan foydalanib ma’lumot elementlarga qiymat beriladi so‘ngra *show_employee* elementidan foydalanib xizmatchi xakidagi ma’lumot ekranga chikariladi:

```
#include <iostream>
using namespace std;
class employee
{
 public:
 long employee_id;
 float salary;
 void show_employee(void)
```

```
{
 cout<<"Nomer: "<<employee_id<<endl;
 cout<<"Maosh: "<<salary<<endl;
};
};
int main()
{
employee worker, boss;
worker.employee_id = 12345;
worker.salary = 25000;
boss.employee_id = 101;
boss.salary = 101101.00;
cout<<"\n"<<"ishchi"<<endl;
worker.show_employee();
cout<<"\n"<<"boss"<<endl;
boss.show_employee();
return 0;
}
```

Komponenta funksiya ta’rifi. Komponenta funksiya albatta sinf tanasida ta’riflangan bo‘lishi lozim. Global funksiyalardan farqli komponenta funksiya sinfning hamma komponentalariga murojat qilishi mumkin. Funksiyaning faqat prototipi emas to‘la ta’rifi sinf tanasida joylashgan bo‘lsa, bu funksiya joylashtiruvchi (inline) funksiya hisoblanadi. Ma’lumki inline funksiyalardassikllar, kalit bo‘yicha o‘tish operatori ishlatilishi mumkin emas. Bundan tashqari bunday funksiyalar rekursiv funksiya bo‘lolmaydi. Bu chegaralarni engish uchun sinf tanasiga faqat funksiya prototipi joylashtirilib, funksiyaning to‘la ta’rifi sinf tashqarisida dasturga kiruvchi boshqa funksiyalar bilan birga beriladi. Komponenta funksiyani sinf tashqarisida ta’riflanganda, qaysi sinfga tegishli ekanligini quyidagi shaklda ko‘rsatiladi:

Sinf-nomi :: Komponenta funksiya-nomi

Sinf tanasiga komponenta funksiya prototipi quyidagi shaklda joylashtiriladi:

Tip funksiya-nomi(formal-parametrlar-ta’rifi)

Sinf tashqarisida funksiya quyidagi shaklda ta’riflanadi:

Tip sinf-nomi :: funksiya-nomi(formal-parametrlar-spetsifikatsiyasi)

{ funksiya tanasi };

Oldingi misoldagi *employee* sinfida funksiya sinf ichida ta’riflangan. Bunday funksiya joylanuvchi *(inline) funksiya* deb qaraladi.

Funksiyani sinf tashqarisida ta’riflab sinf ichiga funksiya prototipini joylashtirish mumkin. Sinf ta’rifi bu holda quyidagi ko‘rinishda bo‘ladi:

```
class employee
{
 public:
 long employee_id;
 float salary;
 void show_employee(void);
};
```

Har xil funksiyalar bir xil nomli funksiyalardan foydalanishi mumkin bo‘lgani uchun funksiya nomi sinf nomi va global ruxsat operatori belgisi (::) qo‘yilishi lozim.

```
void employee::show_employee(void)
{
 cout<<"Nomer: "<<employee_id<<endl;
 cout<<"Maosh: "<<salary<<endl;
};
```

Funksiya sinf tashqarisida ta’riflangan bo‘lsa ularni inline funksiya sifatida qarash uchun funksiya ta’rifida inline so‘zi aniq ko‘rsatilgan bo‘lishi kerak.

Quyidagi dastur *show_employee* funksiyasi ta’rifini sinf tashqarisiga joylashtiradi va inline so‘zi anik ko‘rsatiladi:

```
#include <iostream>
using namespace std;
class employee
{
 public:
 long employee_id;
 float salary;
 void show_employee(void);
};
inline void employee::show_employee(void)
{
 cout<<"Nomer: "<<employee_id<<endl;
 cout<<"Maosh: "<<salary<<endl;
};

int main()
{
employee worker, boss;
worker.employee_id = 12345;
worker.salary = 25000;
```

```
boss.employee_id = 101;
boss.salary = 101101.00;
cout<<"\n"<<"ishchi"<<endl;
worker.show_employee();
cout<<"\n"<<"boss"<<endl;
boss.show_employee();
return 0;
}
```

Konstruktorlar. Konstruktorlar bu sinf komponenta funksiyalari bo‘lib, ob’ektlarni avtomatik initsializatsiya qilish uchun ishlatiladi.

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.

Misol uchun complex sinfi uchun konstruktorni quyidagicha kiritish mumkin :

```
complex (double re = 0.0; double im = 0.0 )
  {real=re; imag=im;}
```

Konstruktorlar uchun qaytariluvchi tiplar, xatto void tipi ham ko‘rsatilmaydi. Dasturchi tomonidan ko‘rsatilmagan holda ham ob’ekt yaratilganda konstruktor avtomatik ravishda chaqiriladi.

Masalan ob’ekt complex cc; shaklida aniqlangan bo‘lsa, konstruktor avtomatik chaqirilib

real va imag parametrlari avtomatik ravishda 0.0 qiymatlariga ega bo‘ladi.

Ko‘zda tutilgan holda parametrsiz konstruktor va quyidagi tipdagi nusxa olish konstruktorlari yaratiladi: T :: T (const T&)

Misol uchun

```
class F
{...
        public : F(const T&)
        ...
  }
```

Sinfda bir nechta konstruktorlar bo‘lishi mumkin, lekin ularning faqat bittasida parametrlar qiymatlari oldindan ko‘rsatilgan bo‘lishi kerak.

Konstruktor adresini hisoblash mumkin emas. Konstruktor parametri sifatida o‘z sinfining nomini ishlatish mumkin emas, lekin bu nomga ko‘rsatkichdan foydalanish mumkin.

Konstruktorni oddiy komponenta funksiya sifatida chakirib bo‘lmaydi. Konstruktorni ikki xil shaklda chaqirish mumkin :

Sinf_nomi .Ob’ekt_nomi (konstruktor_xaqiqiy_parametlari)

Sinf_nomi (konstruktor_xaqiqiy_parametlari)

Birinchi shakl ishlatilganda xaqiqiy parametrlar ro‘yxati bo‘sh bo‘lmasligi lozim. Bu shakldan yangi ob’ekt ta’riflanganda foydalaniladi:

```
complex SS(10.3; 0.22)
// real=10.3; SS.imag= 0.22;
complex EE (2.3)
// EE . real= 2.3;
EE.imag= 0.0;
complex D() // xato
```

Konstruktorni ikkinchi shaklda chaqirish nomsiz ob’ekt yaratilishiga olib keladi. Bu nomsiz ob’ektdan ifodalarda foydalanish mumkin.

Misol uchun :

```
complex ZZ= complex (4.0;5.0);
```

Bu ta’rif orkali ZZ ob’ekt yaratilib, unga nomsiz ob’ekt qiymatlari(real= 4.0; imag= 5.0) beriladi;

Konstruktor nomi sinf nomi bilan bir xil bo‘lishi lozimdir. Misol uchun siz employee sinfdan foydalansangiz, konstruktor ham employee nomga ega bo‘ladi. Agar dasturda konstruktor ta’rifi berilgan bo‘lsa ob’ekt yaratilganda avtomatik chaqiriladi. Quyidagi dasturda employee nomli sinf kiritilgandir:

```
class employee
{
public:
employee(long, float);
void show_employee(void);
private:
long employee_id;
float salary;
};
Konstruktor ta’rifi:
employee::employee(long empl_id, float sal)
{
employee_id = empl_id;
if (salary < 50000.0)
salary = sal;
else
salary = 0.0;
}
```

Shu sinfdan foydalanilgan dastur:

```
#include <iostream>
using namespace std;
class employee
{
public:
employee(long, float);
void show_employee(void);
private:
long employee_id;
float salary;
};
employee::employee(long empl_id, float sal)
{
employee_id = empl_id;
if (salary < 50000.0)
salary = sal;
else
salary = 0.0;
}
void employee::show_employee(void)
{
cout << "Nomer: " << employee_id << endl;
cout << "Maosh: " << salary << endl;
}
int main()
{
employee worker(101, 10101.0);
cout<<"ishchi"<<endl;
worker.show_employee();
return 0;
}
```

Konstruktordan foydalanib ob’ekt ta’rifilanganda parametr uzatish mumkin: employee worker(101, 10101.0);

Agar dasturda employee tipidagi ob’ektlar mavjud bo‘lsa har birini quyidagicha initsializatsiya qilish mumkin

```
employee worker(101, 10101.0);
employee secretary(57, 20000.0);
employee manager(1022, 30000.0);
```

Konstruktorlar va ko‘zda tutilgan qiymatlar. Konstruktorlarda ko‘zda tutilgan qiymatlardan ham foydalanish mumkindir. Misol uchun quyidagi konstruktor employee maoshi qiymatini dasturda ko‘rsatilmagan bo‘lsa 10000.0 teng qilib oladi:

```
employee::employee(long empl_id, float sal = 100.00)
{
employee_id = empl_id;
if (salary < 50000.0)
salary = sal;
else
salary = 0.0;
}
```

Konstruktorlarni qo‘shimcha yuklash. C++ tilida konstruktorlarni ham qo‘shimcha yuklash mumkindir. Quyidagi dasturda konstruktor employee qo‘shimcha yuklangandir. Birinchi konstruktor, dastur xizmatchi, nomer va oyligi ko‘rsatilishini talab qiladi. Ikkinchi konstruktor oylikni kiritilishini so‘raydi. Sinf ta’rifi ichida ikkala konstruktor prototipi ko‘rsatilishi lozim:

```
#include <iostream>
using namespace std;
class employee
{
public:
employee(long, float);
employee(long);
void show_employee(void);
private:
long employee_id;
float salary;
};
employee::employee(long employee_id, float salary)
{
employee::employee_id = employee_id;
if (salary < 50000.0) employee::salary = salary;
else
employee::salary = 0.0;
}
employee::employee(long employee_id)
{
```

```
employee::employee_id = employee_id;
do
{
cout << "Maosh kiriting $50000 dan kichik: ";
cin >> employee::salary;
}
while (salary >= 50000.0);
}
void employee::show_employee(void)
{
cout << "Nomer: " << employee_id << endl;
cout << "Maosh: " << salary << endl;
}
int main()
{
cout<<"ishchi"<<endl;
employee worker(101, 10101.0);
worker.show_employee();
cout<<"manager"<<endl;
employee manager(102);
manager.show_employee();
return 0;
}
```

Ob’ektlar massivlari. Ob’ektlar massivi ta’riflash uchun sinf ko‘zda tutilgan (parametrsiz) konstruktorga ega bo‘lishi kerak.

Ob’ektlar massivi ko‘zda tutilgan konstruktor tomonidan, yoki har bir element uchun konstruktor chaqirish yo‘li bilan initsializatsiya qilinishi mumkin.

class complex a[20]; //ko‘zda tutilgan parametrsiz konstruktorni chaqirish

class complex b[2]={complex(10),complex (100)};

//oshkor chaqirish

Quyidagi misolda *player*, sinfi kiritiladi. Dasturda sinf funksiyasi *show_player* va konstruktor tashqarisida ta’riflanadi. So‘ngra *player* tipidagi ikki massiv yaratilib, har biri xakidagi ma’lumot ekranga chikariladi

```
#include <iostream>
#include <string>
using namespace std;
class player
```

```
{
public:
player();
player (string name,int weight, int age);
void show_player (void);
private:
string name;
int weight;
int age;
};
player::player()
{
name="";
weight = 0;
age = 0;
};
player::player(string name,int weight, int age)
{
player::name=name;
player::weight = weight;
player::age = age;
};
void player::show_player (void)
{
cout<<"Ism: " << name << endl;
cout<<"Vazn: " << weight << endl;
cout<<"Yosh: " << age << endl;
}
class array_player
{ public:
void show_array(player a[],int n)
{ for(int i=0;i<n;i++)
{a[i].show_player();cout<<endl;}}
void input_array(player a[],int n)
{string name;int weight,age;
for(int i=0;i<n;i++)
{cin>>name>>weight>>age;
a[i]=player(name,weight,age);
}
```

```
}
};

int main()
{array_player arr;
Player happy[]={player("Olimov",58,24),
player("Alimov",72,35)};
arr.show_array(happy,2);
player matt[2];
arr.input_array(matt,2);
arr.show_array(matt,2);
return 0;
}
```

Initsializatorlar ro‘yxati. Konstruktor yordamida ob’ekt ma’lumotlarni initsiyalizatsiyalashni ikkita usuli mavjud.

Birinchi usulda parametrlar qiymatlari konstruktor tanasiga uzatiladi. Ikinchi usulda esa ushbu sinfdagi initsializatorlar ro‘yxatidan foydalanish nazarda tutilgan. Bu ro‘yxat parametrlar ro‘yxati va konstruktor tanasi orasiga joylashadi. Ro‘yxatdagi har bir initsializator konkret aniq komponentaga bog‘liq va quyidagi ko‘rinishga ega:

```
<nom> (<ifoda>)
```

Destruktorlar. Sinfning biror ob’ekti uchun ajratilgan xotira ob’ekt yukotilgandan so‘ng bo‘shatilishi lozimdir.

Sinflarning maxsus komponentalari destruktorlar, bu vazifani avtomatik bajarish imkonini yaratadi.

Destruktorni standart shakli quyidagicha :

```
~ sinf_nomi ( ) {destruktor tanasi}
```

Destruktor parametri yoki qaytariluvchi qiymatga ega bo‘lishi mumkin emas (xatto void tipidagi).

Agar sinfda oshkor destruktor mavjud bo‘lmasa, ko‘zda tutilgan destruktor chaqiriladi.

Dastur ob’ektni o‘chirganda destruktor avtomatik chaqiriladi.

Misol:

```
#include <iostream>
using namespace std;
class Person
{
public:
Person ()
```

```
{
cout<<"Yaratidi"<<endl;
}
~Person ()
{
cout<<"O'chirldi"<<endl;
}
};
int main()
{
{
Person work;
}
int kk;cin>>kk;
return 0;
}
```

Natija

Yaratidi

O'chirldi

Ma’lumotlar elementidan birgalikda foydalanish. Odatda, ma’lum sinf ob’ektlari yaratilayotganda, har bir ob’ekt o‘z-o‘zining ma’lumotlar elementlari to‘plamini oladi. Biroq shunday hollar ham yuzaga keladiki, unda bir xil sinflar ob’ektlariga bir yoki bir nechta ma’lumotlar elementlaridan (statik ma’lumotlar elementlaridan) birgalikda foydalanish kerak bo‘lib qoladi. Bunday hollarda ma’lumotlar elementlari umumiy yoki juz’iy deb e’lon qilinadi, keyin esa tur oldidan, quyida ko‘rsatilganidek, static kalit-so‘z keladi:

```
private;
static int shared_value;
```

Sinf e’lon qilingach, elementni sinfdan tashqaridagi global o‘zgaruvchi sifatida e’lon qilish kerak. Bu quyida shunday ko‘rsatilgan:

int class_name::shared_value;

Navbatdagi dastur book_series sinfini aniqlaydi. Bu sinf (seriya)ning barcha ob’ektlari (kitoblari) uchun bir xilda bo‘lgan page_count elementidan birgalikda foydalanadi. Agar dastur ushbu element qiymatini o‘zgartirsa, bu o‘zgarish shu ondayoq barcha sinf ob’ektlarida o‘z aksini topadi:

```
#include <iostream>
using namespace std;
class book_series
{
```

```
    public:
      book_series(float);
      void show_book(void);
      void set_pages(int) ;
    private:
      static int page_count;
      float price;
    };
    int book_series::page_count;
    void book_series::set_pages(int pages)
    {
      page_count = pages;
    }
    book_series::book_series(float price)
    {
      book_series::price = price;
    }
    void book_series:: show_book (void)
    {
      cout << "Narx: " << price << endl;
      cout << "Betlar: " << page_count << endl;
    }
    int main()
    {
      book_series programming(213.95);
      book_series word(19.95);
      word.set_pages(256);
      programming.show_book ();
      word.show_book() ;
      cout << endl << "page_count ning o'zgarishi " << endl;
      programming.set_pages(512);
      programming.show_book();
      word.show_book();
return 0;
}
```

Ko‘rinib turganidek, sinf *page_count* ni *static int* sifatida e’lon qiladi. Sinfni aniqlagandan so‘ng, dastur shu vaqtning o‘zida page_count elementini global o‘zgaruvchi sifatida e’lon qiladi. Dastur page_count elementini

o‘zgartirganda, o‘zgarish shu vaqtning o‘zidayoq book_series sinfining barcha ob’ektlarida namoyon bo‘ladi.

Agar ob’ektlar mavjud bo‘lmasa, public static atributli elementlardan foydalanish. Sinf elementini static kabi e’lon qilishda bu element ushbu sinfning barcha ob’ektlari tomonidan birgalikda qo‘llanadi. Biroq shunday vaziyatlar yuz berishi mumkinki, dastur hali ob’ektni yaratganicha yo‘q, ammo u elementdan foydalanishi kerak. Elementdan foydalanish uchun dastur uni public va static sifatida e’lon qilishi kerak. Masalan, quyidagi dasturda, xatto book_series sinfidagi ob’ektlar mavjud bo‘lmasa ham, bu sinfning page_count elementidan foydalaniladi:

```
#include <iostream>
using namespace std;
class book_series
{
public:
static int page_count;
private:
float price;
};
int book_series::page_count;
int main()
{
book_series::page_count = 256;
cout << "page_count ning joriy qiymati " << book_series::page_count <<"ga teng"<<endl;
return 0;
}
```

Bu o‘rinda, sinf page_count sinfi elementini public sifatida e’lon qilgani uchun, xatto agar book_series sinfidagi ob’ektlar mavjud bo‘lmasa ham, dastur sinfning ushbu elementiga murojaat qilishi mumkin.

Statik funksiya elementlardan foydalanish. Avvalgi dastur ma’lumotlar *statik* elementlarining qo‘llanishini ko‘rsatib bergan edi. C++xuddi shunday usul bilan *statik* funksiya-elementlar (usullar)ni aniqlash imkonini beradi. Agar *statik* usul yaratilayotgan bo‘lsa, dastur bunday usulni, xatto uning ob’ektlari yaratilmagan holda ham, chaqirib olishi mumkin. Masalan, agar sinf sinfdan tashqari ma’lumotlar uchun qo‘llanishi mumkin bo‘lgan usulga ega bo‘lsa, siz bu usulni statik qila olishingiz mumkin bo‘lardi. Funksiyadan foydalanish uchun dastur uni public va static sifatida e’lon qilishi kerak. Masalan, quyidagi dasturda,

xatto book_series sinfidagi ob’ektlar mavjud bo‘lmasa ham, bu sinfning show_count() usulidan foydalaniladi:

```
#include <iostream>
using namespace std;
class book_series
{
public:
  static int show_count() { return page_count;};
private:
  float price;
  static int page_count;
};
int book_series::page_count=256;
int main()
{
cout << "page_count ning joriy qiymati " << book_series::show_count()
<<"ga teng"<< endl;

return 0;
}
```

Nazorat savollari:

1. Sinflar va ob’ektlar tushunchasi.
2. Sinf xususiyatlari va metodlari.
3. Sinf elementlariga murojaat huquqlari.
4. Konstruktor nima.
5. Destruktorni misollar yordamida tushuntiring.
6. Ob’ektlar massivlarini tushuntiring.
7. Sinf statik xususiyat va metodlari.

7 – Ma’ruza. Sinflar va ko’rsatkichlar.

Reja

1. Sinflarda ko’rsatkichlar
2. Obyektlarga ko’rsatkich
3. Sinf komponentalariga ko’rsatkichlar. ***this*** ko’rsatkichidan foydalanish
4. Ilovalardan foydalanish (Reference)

Obyektlarga ko’rsatkich (pointer)

Obyektga yo’naltirilgan dasturlashda sinflar orqali obyektlar ustida turli xil bajariladigan amallar mavjud. Obyektlarga boshqa o’zgarvchilar kabi ko’rsatkich orqali murojaat qilish mumkin. Obyekt a’zolariga ko’rsatkich orqali murojat qilish uchun **.(nuqta)** o’rniga **-> operatori** ishlatiladi. Quyida misol keltirilgan:

```
class c1 {
        int a;
        public:         int get_num() { return a;}
                        c1(int x){ a = x; }
};
c1  a = 2, *p, b[2]; // p ko’rsatkich e’lon qilindi
p = &a;              // a obyektning adresi p ko’rsatkichga olindi
p->get_num();        // ko’rsatkich orqali obyekt a’zosiga murojat
p = b;               // b obyektning 1 chi elementi adresi p ko’rsatkichga olindi
```

Ko’rsatkichlarda bajariladigan +, - amallarni obyektlar bilan ham qo’llash imkoniyati mavjud bo’lib, oddiy ko’rsatkichlardagi barcha xususiyatlar ushbu holatda qo’llanilish jarayonida ham to’liq saqlanib qoladi.

```
class c1 {
        int a;
        public:         int get_num() { return a;}
                        c1(int x){ a = x; }
};
c1  a[3] = {1,2,3},  *p;
p = a;                  // a obyektning 1-adresi p ko’rsatkichga olindi
p->get_num();           // output 1
p++;                    // p keyingi obyektni ko’rsatadi
p->get_num();           // output 2
```

Obyekt massiviga ko’rsatkich quyidagicha misol orqali amalga oshiriladi. Bunda c1 sinf tipida a obyekt massivi e’lon qilingan va p sinf tipidagi ko’rsatkichga obyekt massivi o’zlashtirilgan.

```
class c1 {
        int a;
        public:         int get_num() { return a;}
                        c1(int x){ a = x; }
};
int main() {
        c1 a[3] = {1,2,3};
        c1 *p;
        p = a;
        for(int i=0; i<3; i++)
                cout<<(p+i)->get_num();
        return 0;
}
```

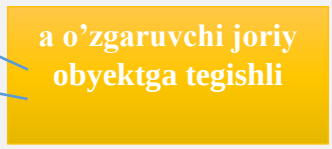
Obyekt a’zolarini ko’rsatkichga olish

```
class c1 {
        public:         int a;
                        int get_num() { return a;}
                        c1(int x){ a = x; }
};
c1  b = 4;
int  *p;
p = &b.a;               // c1.a ning adresi p ko’rsatkichga olindi
*p = 8;                 // b.a ning qiymati 8 ga o’zgartirildi
cout<<b.a;              // output 8
```

this ko’rsatkichi

this joriy obyektga ko’rsatkich.

```
class c1 {
        int a;
        public:         int get_num() { return this->a;}
                        c1(int a){ this->a = a; }
};
c1  b = 4;
cout<<b.a;              // output 4
```



Nasl olingan toifaga ko’rsatkich

Bir toifadagi ko’rsatkich boshqa bir obyekt toifaga ko’rsata olmaydi. Faqatgina bir holat istisno. Bu ona class voris classlarga ko’rsatkich bo’la oladi.

```
class base {
        int a;
        public:         int get_a() { return this->a;}
                        void set_a(int x){ this->a = x; }
};
class derived: public base {
        int b;
        public:         int get_b() { return this->b;}
                        void set_b(int x){ this->b = x; }
};
```

Nasl olingan toifaga ko’rsatkich

```
base *bp;           // ona bp ko’rsatkich
derived d;          // voris d obyekt
bp = &d;            // base ko’rsatkich derived ga ko’rsatadi
//derived obyektiga base ko’rsatkich orqali murojat
bp->set_a(5);
cout<<bp->get_a();
//errorrrr, chunki bp base pointer orqali derived obyektning a’zolariga
  murojat qila olmaymiz
bp->set_b(5);
//derived obyektning a’zolariga murojat qilish uchun bp base ko’rsatkichni
derived ko’rsatkich toifasiga o’zgartirish kerak
((derived*)bp)->set_b(10);
cout<<((derived*)bp)->get_b();
```

Sinf a’zolariga ko’rsatkich

C++ da shunday ko’rsatkich qilsa bo’ladiku bu ko’rsatkich class a’zosini ko’rsatib turadi. Bunday ko’rsatkichlarni **pointer-to-member** deb ataladi. Class a’zosiga ko’rsatkichda maxsus .* va ->* operatorlar ishlatiladi.

E’lon qilinishi:

```
int c1::*data;              //toifaga ko’rsatkich
int (c1::*func)();          //funksiyaga ko’rsatkich
data = &c1::val;            //val joyini data’ga olish
func = &c1::get_num;        //get_num joyini func’ga olish
ob.*data;                   // val ga murojat
(ob.*func)();               // get_num() ga murojat
```

Demak yuqoridagi misoldan ko’rinib turibdiki, obyekt ko’rsatkichi orqali sinf a’zolariga murojaat qilish imkoniyati mavjud. Bunda sinf a’zosiga mos ko’rsatkich o’zgaruvchi e’lon qilinadi va ushbu o’zgaruvchi orqali sinf a’zosiga murojaat qilinadi.

```
class c1 {
        public:     int get_num() { return val + val; }
                    c1(int a){ val = a; }
                    int val;
};
int c1::*data;      // c1 class a’zolariga ko’rsatkich yaratildi
int (c1::*func)();  // c1 class a’zolariga ko’rsatkich yaratildi
c1 a(4), b(8);      // a va b obyektlar yaratildi
data = &c1::val;    // data va func ko’rsatkichlariga c1 a’zolarining joylari olindi
func = &c1::get_num; // data va func ko’rsatkichlariga c1 a’zolarining joylari
                        olindi
cout<<a.*data<<b.*data<<endl; // Obyekt a’zolariga ko’rsatkichlar orqali murojat
                                qilinmoqda
cout<<(a.*func)();<<(b.*func)(); // Obyekt a’zolariga ko’rsatkichlar orqali
murojat qilinmoqda
```

Obyektga ko’rsatkich bo’lgan holat

```
class c1 {
        public:     int get_num() { return val + val; }
                    c1(int a){ val = a; }
                    int val;
};
int c1::*data;
```

```
int (c1::*func)();
c1 a(4), *p;  // a va b obyektlar yaratildi
p = &a;       // p ga a obyektning adresi olindi
data = &c1::val;
func = &c1::get_num;
cout<<p->*data<<endl;
cout<<(p->*func)();
```

Ilovalardan foydalanish (Reference)

C++ da ko'rsatkichga o'xshagan narsa mavjud bo'lib u *reference(ssilka)* deb ataladi. Ssilka bilan ishlash ko'rsatkichga nisbatan oson. Ssilkani 3 hil usul bilan ishlatish mumkin:

- funksiya parametri
- funksiyaning qaytarish qiymati
- alohida ssilka

Ssilka – funksiya parametr sifatida

Ko'rsatkich(Pointer)	Ssilka(Reference)

void neg(int *i) {
*i = -*i;
}
x = 10;
neg(&x);
cout<<x;
	void neg(int &i) {
	i = -i;
	}
	x = 10;
	neg(x);
	cout<<x;

Ssilka ishlatilganda hech qanday ortiqcha belgilar qo'yilmaydi, faqatgina e'lon qilinganda **& operatori** qo'yiladi. Pointerda esa *** operator** qo'yiladi. Ssilka ham ko'rsatkich ham adresdagi qiymatni o'zgartiradi. Faqatgina ssilka bilan ishlash osonroq.

Quyidagi misolda reference orqali ikkita o'zgaruvchining o'rnini almashtirish daturi keltirilgan

```
void swap(int &i, int &j) {
    int t;
    t = i; // * operatori endi kerak bo'lmaydi
    i = j;
    j = t;
```

```
}
int a = 4, b = 19;
swap(a, b);          // a = 19, b = 4
```

Ssilka – funksiyaning qaytarish qiymati

```
char s[] = "Hello world";
char &replace(int pos) { // Funksiya s[pos] da joylashgan adresni qaytaradi
        return s[pos];
}
replace(5) = 'X';    //5 indeksga X harfini qo'yadi
cout<<s;
```

Output???

"HelloXworld"

Ssilka – alohida

Alohida ishlatiladigan ssilkalar yaratilganda inisalizatsiya qilinishi kerak bo'ladi, chunki ssilka bir marta inisalizatsiya qilinadi, ikkinchi marta o'zlashtirib bo'lmaydi.

```
int a = 8, b = 10, &c = a;
cout<<a<<c;                    // 8 8
c = 4;                         // a = 4
cout<<a<<c;                    // 4 4
c = b;                         // a = 10
cout<<a<<c;                    // 10 10
```

Nazorat savollari

1. Obyektlarga ko'rsatkich qanday yaratiladi?
2. Oddiy sinf bilan nasl olingan sinfga ko'rsatkich qanday farqlanadi?
3. Nasl olingan toifaga ko'rsatkich nima?
4. Ilovalar (reference) nima?
5. Sinf a'zolariga ko'rsatkich?
6. This ko'rsatkichidan foydalanish?

8 – Ma’ruza. Sinflar orasida munosabatlar.

Reja

1. Munosabat turlari, do’stona (friend) funksiyalar
2. Do’stona (friend) sinflar
3. Sinfning statik ma’lumotlari
4. Obyektlar massivi va undan foydalanish

Friend funksiyalar

Class’ning private va protected qismiga class’ga tegishli bo’lmagan **friend** funksiya murojat qilishi mumkin. Friend funksiyalar klasning ichida friend kalit so’zi bilan yoziladi.

E’lon qilinishi:

```
class myclass {
        …..
        friend int sum(myclass x);
        ….
};
```

Albatta friend funksiyalar sinfdan tashqarida mavjud bo’ladi va ushbu do’stona funksiya sinfning barcha sohalariga murojaat qila olishi mumkin.

```
class sm{
    int a, b;
    public:
        friend int sum(myclass x);
        void set_ab(int i, int j) { a = I; b = j; }
};
int sum(myclass x) {
    return x.a + x.b;     //sum() hech qaysi classga tegishli emas.
}
int main() {
    myclass n;
    n.set_ab(3, 4);
    cout << sum(n);
    return 0;
}
```

Do’stona (friend) sinflar

Bir class boshqa bir class’ga do’stona bo’lishi mumkin. Bunda sinflar bir – birining a’zolaridan foydalanish imkoniyatiga ega bo’ladi. Bunda shu narsaga

e’tibor berish lozimki, biror sinfga do’stona bo’ladigan sinf (ya’ni friend kalit so’zi orqali e’lon qilinadigan sinf), mazkur sinfning a’zolaridan foydalanish imkoniyatini yaratadi

E’lon qilinishi:

```
class myclass {
        …..
        friend someclass b;
        ….
};
```

Do’stona sinfdan foydalanish uchun quyida misol keltirilgan. Bunda e’tibor berishimiz lozimki, TwoValues sinfi Min sinfiga do’stona bo’lib, bunda Min sinfi TwoValues sinfining a’zolaridan foydalanishi mumkin.

```
class TwoValues {
    int a, b;
    public:
        TwoValues(int i, int j) { a = i; b = j; }
        friend class Min;
};
class Min {
    public:
        int min(TwoValues x) { return x.a < x.b ? x.a : x.b; }
};
int main() {
    TwoValues ob(10, 20);
    Min m;
    cout << m.min(ob);
    return 0;
}
```

Inline funksiyalar

C++ tilida inline funksiyalar ishlatish imkoniyati mavjud. Ko'pincha **inline** funksiyalar class'lar bilan ishlatiladi. C++ da qisqa funksiyalarni ishlatganda ular chaqirilmaydi aks holda ularning kodi chaqirilgan joyga qo'yiladi. Bu jarayon **macro-funksiyalarni** ishlatishga o'xshaydi.

Funksiyani chaqirmaslik uchun **inline** kalit so'zi ishlatiladi. Quyidagi misolda inline funksiyadan foydalanish ko'rsatilgan.

```
inline int max(int a, int b) {
	return a>b ? a : b;
}
int main() {
	cout << max(10, 20);
	cout << " " << max(99, 88);
	return 0;
}
```

Kompilyatsiyadan chiqqandan keyin:

```
int main() {
	cout << (10>20 ? 10 : 20);
	cout << " " << (99>88 ? 99 : 88);
	return 0;
}
```

Sinf ichidagi inline funksiyalar

1. Metodni e'lon qilganda metod tanasini yozish

```
class Human {
	public:
		void lookAt(const char* name) {
			std::cout << name << std::endl;
		}
	private:
		char _name[30];
};
```

2. Class'dan tashqarida inline

```
inline void Human::lookAt(const char* name) {
	std::cout << name << std::endl;
}
```

Sinfning statik ma'lumotlari

Class o’zgaruvchisini static deb e’lon qilinganda kompilyator uni obyektlar uchun bitta nusxa ko’rinishida yaratadi. Ya’ni bir nechta obyekt bitta o’zgaruvchidan foydalanadi. Static o’zgaruvchi 0 ga inisalizatsiya qilinadi. Class static a’zolariga murojat qilish **ClassName::static_member** ko’rinishida murojat qilinadi, obyekt orqali ham murojat qilsa bo’ladi.
Static o’zgaruvchini static kalit so’zi bilan e’lon qilinadi.

E’lon qilish:

```
class someclass {
        public:
                static int ob;
};
```

Static maydonlardan foydalanish

```
class Proper {
        public:
                static int ob_counter;
};
int Proper::ob_counter;
int main() {
        Proper a;
        cout<<Proper::ob_counter++<<endl;
        cout<<a.ob_counter<<endl;
        return 0;
}
```

Static metodlar

Class metodlarini static o’zgaruvchilar kabi e’lon qilsa bo’ladi. Static metodlar static a’zolarga murojat qiladi.

E’lon qilish:

```
class someclass {
        public:
                static int ob;
                static int get_ob() { return ob; }
};
```

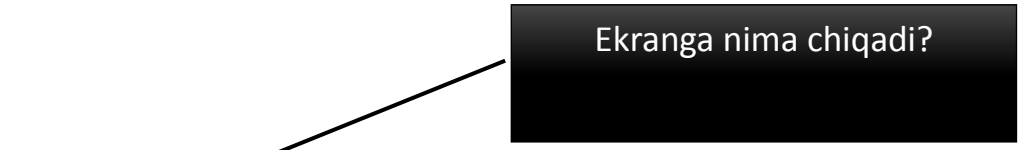
Statik metodlardan foydalanish

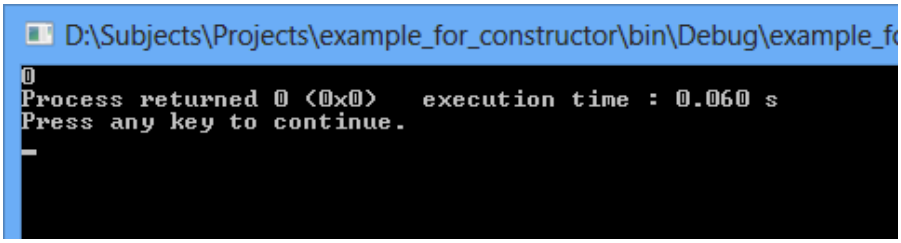
```
class Proper {
	public:
		static int ob_counter;
		static int get_ob() { return ob_counter; }
};
int Proper::ob_counter;
int main() {
	cout<<Proper::ob_counter++<<endl;
	cout<<Proper::get_ob()<<endl;
	return 0;
}
```

:: operatoridan foydalanish

Biz bilamizki :: operatori class a'zolariga murojat qilish uchun ishlatiladi. Quyidagi holat berilgan:

```
int i;
void f() {
	int i;
	i = 10;
}
int main() {
	f();
	cout<<i;
	return 0;
}
```





:: operatori orqali global o'zgaruvchiga murojat qilish quyidagicha amalga oshiriladi.

```
int i;
void f() {
	int i;
	::i = 10;
}
int main() {
	f();
```

```
        cout<<i;
        return 0;
}
```

Obyektlarni funksiyaga yuborish

Obyektlarni funksiyaga yuborish oddiy o’zgaruvchilarni yuborgan kabi amalga oshiriladi.

```
class XY { … };
void f(XY a) { … }
```

Misol asosida ko’rib o’tamiz

```
class Sum {
        int a, b;
        public:         int plus() { cout<<a + b;}
                        void set_nums(int x, int y) { a = x; b = y; }
};
void f(Sum x) { // a obyekti x obyektiga nusxa olindi
        x.set_nums(4,5);
        x.plus();
}
int main() {
        Sum a;
        f(a); // a obyekti f() funksiyasiga yuborildi
        a.set_nums(3,2);
        a.plus();
    return 0;
}
```

Funksiya obyekt qaytaradi

Funksiya obyekt qaytarishi uchun qaytariladigan obyektning toifasi funksiyaning qaytarish tipiga ko’rsatilishi kerak.

Misol:

```
class XY { … };
XY f() { … return object; }
XY a = f();
```

Misol-7

```
class Sum {
        int a, b;
        public:         int plus() { cout<<a + b;}
                        void set_nums(int x, int y) { a = x; b = y; }
```

```
};
Sum f() {
        x.set_nums(4,5);
        return x; // Funksiya Sum toifadagi obyekt qaytaradi
}
int main() {
        Sum a;
        a = f(); // a obyektga f() dan qaytgan obyekt o'zlashtirildi
        a.plus();
    return 0;
}
```

Obyektlarni o'zlashtirish

```
class Sum {
        int a, b;
        public:         int plus() { cout<<a + b;}
                        void set_nums(int x, int y) { a = x; b = y; }
};
int main() {
        Sum a , b;
        a.set_nums(3,2);
        b = a; // a obyekt b obyektga o'zlashtirildi
        b.plus();
    return 0;
}
```

Obyektlar massivi

C++ da obyektlar massivini yaratish mumkin. Yaratish qolgan tiplarni yaratganday yaratiladi. Yaratish va murojat qilish:

```
class XY {
        public: int a, b;
                void sum();
};
// obyekt massivini yaratish
XY a[10], b[3] = {1, 2, 3};
//obyekt a'zolariga murojat qilish
a[0].a = 5; a[0].b = 8;
a[0].sum();
a.b = 10;       < -- errorrrrrrrrr, chunki a massiv
```

Quyidagi misolda obyektla rmassivi yaratiladi va ushbu obyektlar orqali set_num() funksiyasiga qiymat jo'natiladi. Bunda har obyektda alohida alohida qiymatlar saqlanadi.

```
class c1 {
        int a;
        public:         int get_num() { return a;}
                        void set_num(int x) { a = x; }
};
int main() {
        c1 a[3];                // a obyektdan 3 ta yaratildi, yani massiv hosil bo'ldi
        for(int i=0; i<3; i++)
                a[i].set_num(i);     // Obyekt a'zolariga murojat qilish
   return 0;
}
```

Obyektlar massivini inisalizatsiya qilish

Agar parametrli konstruktor mavjud bo'lsa obyektlarni inisalizatsiya qilsa bo'ladi.

```
class c1 {
        int a, b;
        public:         int get_num() { return a;}
                        c1(int x, int y){ a = x; b = y;}
                        c1(){a=0;}
                        c1(int x){ a = x; }
};
c1 a[3] = {1,2,3};        |= {c1(1), c1(2), c1(3)}              // c1(int x)
c1 a[2] = {{1,2}, {3,4}};  |= {c1(1,2), c1(3,4)}              // c1(int x, int y);
c1 a[5];                                                      // c1()
```

Nazorat savollari

1. Do’stona funksiyalar qanday e’lon qilinadi?
2. Do’stona sinflar qanday e’lon qilinadi?
3. Obyektlar massivi qanday e’lon qilinadi?
4. :: operatoridan foydalanish
5. Obyektlarni o’zlashtirish
6. Obyektlar massivini inisalizatsiya qilish
7. Inline funksiya va oddiy funksiyaning nima farqi bor?
8. Sinf ichidagi inline funksiyalar

9 – Ma’ruza. Sinflarda vorislik.

Reja

1. Sinflarda vorislik nima?
2. Sodda vorislik
3. Vorislikda murojaat huquqlari
4. Vorislikda konstruktorlar va destruktorlar.

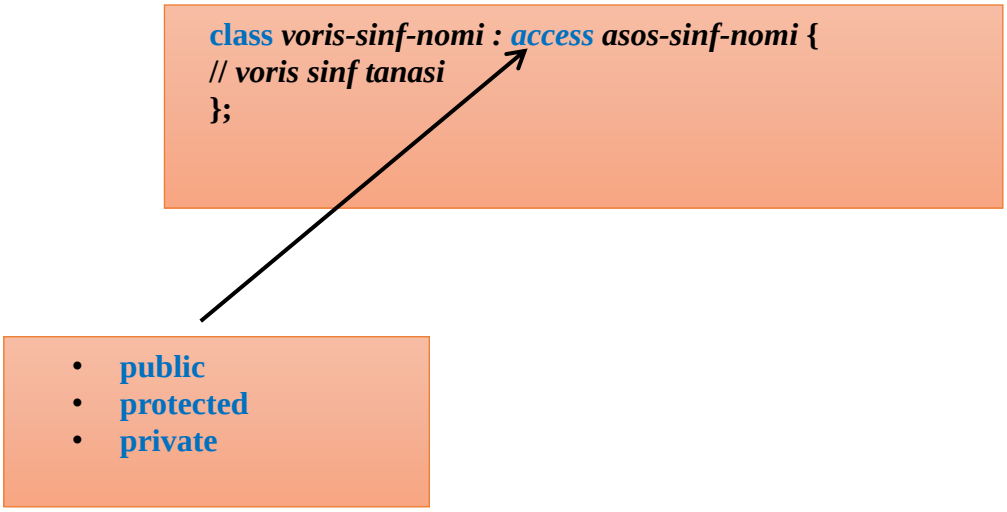
Vorislik – bu OYD ning asosiy ustunlaridan biridir. Vorislik sinflarda ierarxik ko’rinishdagi sinflanishni ta’minlaydi. C++ terminologiyasida:

- Asos sinf (ya’ni voris olinadigan sinf) ***base class (asos sinf)*** deb ataladi.
- Voris sinf (ya’ni voris olish orqali yaratiladigan yangi sinf) ***derived class (voris sinf)*** deb ataladi.

Voris sinf boshqa bironta sinf uchun asos sinf bo’lishi mumkin.Demak bu orqali, ***ko’p sathli vorislik (multiple inheritance)*** vujudga keladi.

Vorislikdan foydalanish

- Biron bir sinfdan Voris olingandan keyin, ushbu asos sinf a’zolari voris sinfning ham a’zolari bo’lib hisoblanadi.
- C++ dasturlash tilida *voris olish umumiy formasi* quyidagicha:

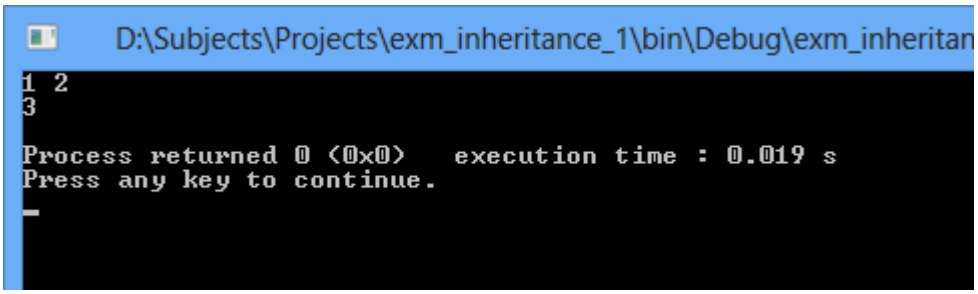


Vorislikdan foydalanish konstruksiyasi

Sodda vorislikdan foydalanishga misol. Obyekt orqali jo’natilgan qiymatni qaytarish dasturi

```
class base {
        int i, j;
        public:
        void set(int a, int b) { i=a; j=b; }
        void show() { cout << i << " " << j << "\n"; }
};
class derived : public base {
        int k;
        public:
                derived(int x) { k=x; }
                void showk() { cout << k << "\n"; }
};
int main()
{
        derived ob(3);
        ob.set(1, 2); // asos sinf a’zosiga ruxsat ochiq
        ob.show(); // asos sinf a’zosiga ruxsat ochiq
        ob.showk(); // voris sinf a’zosidan foydalanish
}
```

Dastur natijasi quyidagicha:




```
        class derived : public base {….}
```

Bunda asos sinfning barcha public a’zolari voris sinfning ham *public a’zolari* bo’ladi. Asos sinfning barcha himoyalangan (protected) a’zolari voris sinfning *protected a’zolari* bo’ladi. Lekin, asos sinfning *private a’zolari* voris sinf uchun ham, boshqa sinflar uchun ham *not accessible* bo’ladi ya’ni asos sinfning private a’zolariga to’g’ridan to’g’ri murojaat yo’q.

Agar voris sinf asos sinfdan private asosida yaratilsa nima bo’ladi? Voris sinf a’zolari qanday xususiyatga ega bo’ladi? Agar voris sinf private xususiyati orqali

yaratilsa, bunda asos sinfning barcha *public va protected a'zolari* voris sinfning *private a'zolari* qatoriga o'tadi.

Ma'lumotlarga murojaat turlari va vorislik

Murojaat turlari	public	protected	private
Asos sinf	yes	yes	yes
Voris sinf	yes	yes	no
Boshqa sinf (yoki main funksiya)	yes	no	no

private murojaat turi orqali voris sinfni yaratishga misol.

```
class base {
        int i, j;
        public:
        void set(int a, int b) { i=a; j=b; }
        void show() { cout << i << " " << j << "\n";}
};
// asos sinfning public a'zolari, voris sinfda private huquqiga o'tadi.
class derived : private base {….}
int main()
{
        derived ob(3);
        ob.set(1, 2); // error, set() metodiga ruxsat yo'q
        ob.show(); // error, show() metodiga ruxsat yo'q
        return 0;
}
```

Vorislik va sinfning himoyalangan (protected) a’zolari

- *Protected* kalit so’zi vorislik mexanizmini tashkil qilish uchun ishlatiladi va qulaylik vujudga keltiradi.
- Asos sinfning *private a’zolariga* dasturning boshqa bir qismi (misol uchun *boshqa sinf* yoki *main()* funksiyasi) yoki *voris sinf* tomonidan to’g’ridan to’g’ri *murojaat mavjud emas*
- Lekin asos sinfning *protected a’zolari*, boshqa sinf yoki main() funksiya uchun yopiq bo’ganiga qaramasdan, *voris sinf* tomonidan *murojaat mavjuddir*.

Asos sinfning protected a’zolari

```
class base {
        protected:
                int i, j; // asos sinf uchun private, lekin voris sinf uchun ochiq
        public:
                void set(int a, int b) { i=a; j=b; }
                void show() { cout << i << " " << j << "\n"; }
};
class derived : public base {
        int k;
        public:
                void setk() { k=i*j; } // i va j voris sinf uchun ochiq
                void showk() { cout << k << "\n"; }
};
int main()
{
        derived ob;
        ob.set(2, 3); // OK, voris sinf uchun ochiq, chunki public
        ob.show(); // OK, voris sinf uchun ochiq, chunki public
        ob.setk();     ob.showk(); // voris sinf metodi, murojaat turi public, demak
ochiq
        return 0;
}
```

Voris olish turlari

```
// public (umumiy) vorislik
class derived1: public Base{…};
// private (xususiy) vorislik
class derived2: private Base{….};
 // protected (himoyalangan) vorislik
class derived3: protected Base{….};
// default (oddiy-xususiy) vorislik
class derived4: Base{….};
```

- Public vorislik:
    - Bunda asos sinfning public a’zolari voris sinfning ham public a’zolari va asos sinfning protected a’zolari voris sinfning protected a’zolariga aylanadi.
    - Lekin asos sinfning private a’zolari voris sinf uchun yopiqligicha qoladi.
- Protected vorislik:
    - Bunda asos sinfning public va protected a’zolari voris sinf uchun protected a’zo bo’lib o’tadi.
- Private vorislik:
    - Bunda asos sinfning public va protected a’zolari voris sinfning private a’zosiga ayanadi.

Asos sinf konstruktoriga qiymat jo’natish

Quyidagi forma orqali asos sinf konstruktoriga parametr orqali qiymat jo’natishimiz mumkin:

```
voris-sinf-konstruktori(arg-lar) :      asos-sinf1(arg-lar),
                                        asos-sinf2(arg-lar),
                                        // ...
                                        asos-sinfN(arg-lar)
{
        // voris sinf konstruktor tanasi
}
```

Asos sinf konstruktoriga qiymat jo’natish

```
class base {
        protected: int i;
        public:
                base(int x) { i=x; }
};
class derived: public base {
        int j;
        public:
                derived(int x, int y): base(y) // Asos sinf konstruktorini chaqirish
                { j=x; }
                void show() { cout << i << " " << j << "\n"; }
};
derived ob(3, 4);
ob.show(); // natija 4  3
```

Nazorat savollari

1. Sinflarda vorislik nima?
2. Voris olishning umumiy formasi qanday?
3. Voris olishning qanday turlari mavjud?
4. Vorislikning qanday turlari mavjud?
5. Asos sinf konstruktoriga qiymat jo'natish qanday formada amalga oshiriladi?
6. Vorislikda konstruktor va destruktorlar
7. Asos sinfning protected a'zolari
8. Agar asos sinfdan public ko'rinishda voris olingan bo'lsa asos sinfning public, private, protected sohadagi a'zolari voris sinfga qanday ko'rinishda o'tadi. Protected ko'rinishda olingan bo'lsachi?
9. Agar voris sinf "private" sohasi asosida yaratilgan bo'lsa, asos sinfning a'zolari voris sinfga qanday ko'rinishda o'tadi? O'z navbatida voris sinf obyekti orqali ushbu voris sinf a'zolariga murojaati qanday bo'ladi?
10. Bir nechta sinflardan voris olish

10 – Ma’ruza. Sinflarda vorislik. Polimorfizm

Reja

1. Polimorfizm tushunchasi
2. Virtual funksiyalar va ularning xususiyatlari
3. Abstraktsinflar
4. Virtual asos sinflar

Polimorfizm nima?

Polimorfizm – bu **bitta interfeys, bir nechta metod**. Ya’ni metodlarni **overload** yoki **override** ko’rinishidir. Polimorfizm ikki xil ko’rinishda namoyon bo’ladi

- compile time;
- run time;

Virtual funksiya

Virtual funksiya asos sinf a’zosi hisoblanadi va voris sinfda qayta bir xil parametr asosida e’lon qilinadi. ***virtual funksiya*** yaratish uchun, asos sinf ichida funksiya yaratilishi jarayonida **virtual** kalit so’zidan foydalaniladi.

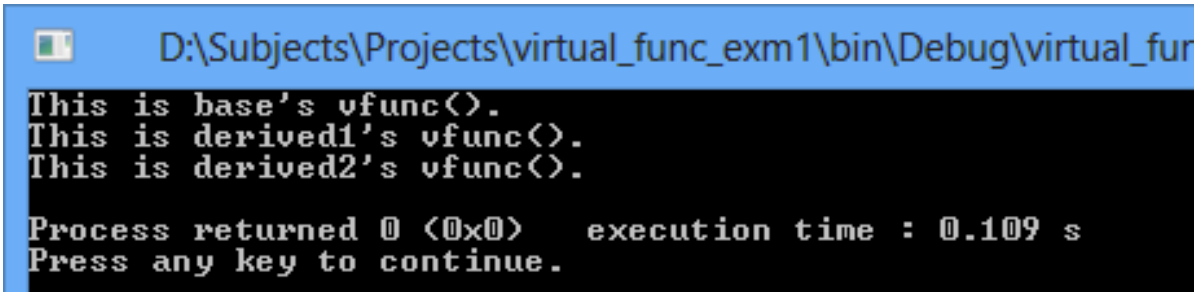
Asos sinfda *virtual funksiya* e’lon qilingan bo’lsa, voris sinfda ushbu funksiya qayta e’lon qilinishi mumkin va o’z xususiyatlaridan kelib chiqqan holda funksiya tanasi boshqacha yozilishi mumkin.

Ushbu qayta e’lonqilingan funksiyaning barcha parametrlari asos sinfdagi funksiya parametrlari bilan bir xil bo’lishi lozim, misol uchun: *funksiya qaytarish tipi, argumentlar soni va tipi.* Quyidagi misolda vitual funksiyalar qanday e’lon qilinishi va undan foydalanish ko’rsatilgan.

```
class base {
public:
	virtual void vfunc() {
		cout << "This is base's vfunc().\n";
	}
};
class derived1 : public base {
public:
	void vfunc() {
		cout << "This is derived1's vfunc().\n";
	}
};
class derived2 : public base {
public:
	void vfunc() {
```

```
                cout << "This is derived2's vfunc().\n";
        }
};
int main()
{
        base *p, b;
        derived1 d1;
        derived2 d2;
        // base sinfga ko'rsatkich
        p = &b;
        p->vfunc(); // base sinf vfunc() virtual funksiyasi
        // derived1 sinfga ko'rsatkich
        p = &d1;
        p->vfunc(); // derived1 sinf vfunc() virtual funksiyasi
        // derived2 sinfga ko'rsatkich
        p = &d2;
        p->vfunc(); // derived2 sinf vfunc() virtual funksiyasi
        return 0;
}
```

Dastur natijasi:



- Ushbu dasturga ko'ra, **base** sinfida **vfunc( )** nomli virtual funksiya e'lon qilingan.
- Bu yerda ishlatilgan **virtual** kalit so'zi qolgan voris sinfdagi funksiyalar ishlashi uchun imkoniyat yaratadi.
- **derived1** va **derived2** voris sinflarda **vfunc( ) funksiya uchun virtual** kalit so'zini ishlatish shart emas.
- Demak, asos sinf ko'rsatkich obyekti (*p) voris sinf a'zolariga ya'ni funksiyalariga murojaat qila oladi, agar ushbu funksiya asos sinfda virtual deb e'lon qilingan bo'lsa

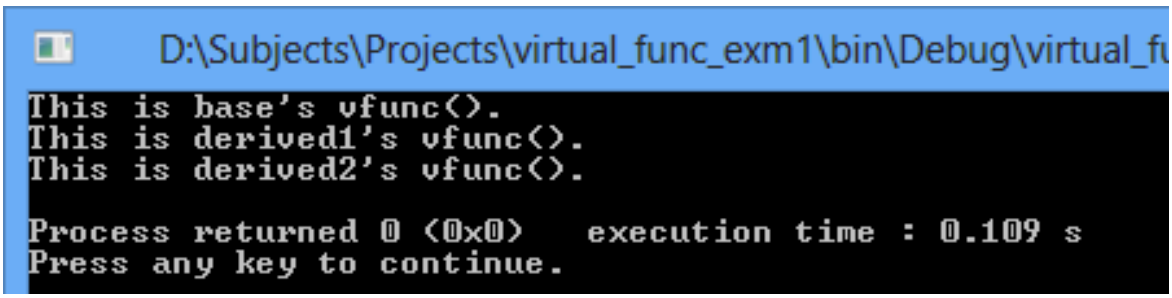
Sinfning virtual atributlari vorisga o’tishi

Asos sinfdan voris olinganda ushbu sinfdagi virtual funksiya ham vorislik xususiyatiga ega bo’ladi. Bu shuni bildiradiki, ***asos sinf virtual funksiyasi*** voris sinf uchun mavjud bo’lgani bilan birga, ushbu voris sinfdan yana voris olingan holda ham ushbu *vertuallik xususiyati* saqlanib qoladi. Ya’ni ushbu funksiya ikkinchi voris sinf uchun ham *override* qilinadi. Bu xususiyat bir nechta vorislikda ham saqlanib qoladi. Quyidagi misolda ko’ramiz:

```
class base {
        public:
        virtual void vfunc() {
                cout << "This is base's vfunc().\n";
        }
};
class derived1 : public base {
        public:
        void vfunc() {
                cout << "This is derived1's vfunc().\n";
        }
};
/* derived2 sinf uchun vfunc() virtual funksiyasi  derived1 sinfdan voris sifatida o’tgan. */
class derived2 : public derived1 {
        public:
        void vfunc() {                  // vfunc() virtual funksiya
                cout << "This is derived2's vfunc().\n";
        }
};
int main()
{
        base *p, b;
        derived1 d1;
        derived2 d2;
        // base sinfga ko’rsatkich
        p = &b;
        p->vfunc();             // base sinf vfunc() virtual funksiyasi
        // derived1 sinfga ko’rsatkich
        p = &d1;
        p->vfunc();             // derived1 sinf vfunc() virtual funksiyasi
```

```
        // derived2 sinfga ko'rsatkich
        p = &d2;
        p->vfunc();                 // derived2 sinf vfunc() virtual funksiyasi
        return 0;
}
```

Dastur natijasi:



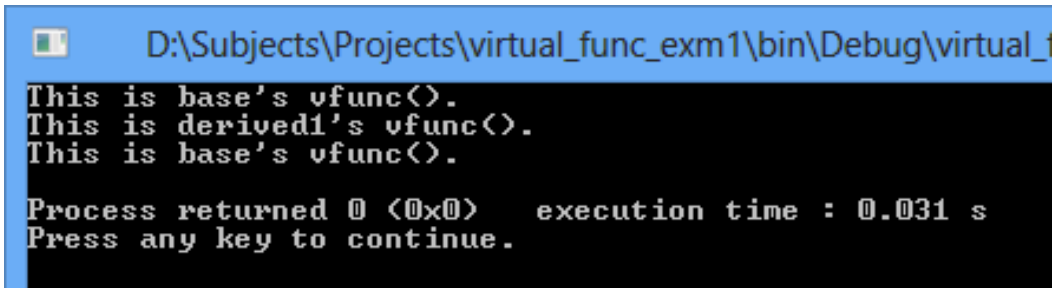
Virtual funksiyaning ierarxik xususiyati

Asos sinfda **virtual** funksiya yaratilgan va bu voris sinf uchun override qilinishi mumkin. Agar voris sinfda mazkur virtual funksiya override qilinmagan bo'lsa nima bo'ladi? Bunda voris sinf obyekti virtual funksiyaga murojaat qiladi, ya'ni asos sinf virtual funksiyasiga. Quyida misol keltirilgan:

```
class base {
public:
        virtual void vfunc() {
                cout << "This is base's vfunc().\n";
}
};
class derived1 : public base {
public:
        void vfunc() {
                cout << "This is derived1's vfunc().\n";
}
};
class derived2 : public base {
public:
        // vfunc() override qilinmagan
};
int main()
{
        base *p, b;
        derived1 d1;
```

```
        derived2 d2;
        // base sinfga ko'rsatkich
        p = &b;
        p->vfunc(); // base sinf vfunc() virtual funksiyasi
        // derived1 sinfga ko'rsatkich
        p = &d1;
        p->vfunc(); // derived1 sinf vfunc() virtual funksiyasi
        // derived2 sinfga ko'rsatkich
        p = &d2;
        p->vfunc(); // derived2 sinf vfunc() virtual funksiyasi
        return 0;
}
```

Dastur natijasi:



Abstrakt sinf tushunchasi

C++ tilida kamida bitta virtual funksiyaga ega bo'lgan sinf ***abstrakt sinf*** deyiladi. Abstrakt sinfning asosiy xususiyatidan biri shuki, ushbu turdagi sinfdan obyekt olib bo'lmaydi. Demak sinfni to'la abstractligini ta'minlash uchun quyidagi qonuniyatdan foydalanamiz:

```
class base{
        public:
                virtual vfunc(args....) = 0;
        .....
}
```

Abstrakt sinf

```
class Base
{
public:
   virtual void display( int i)=0;
};
class Derv: public Base
{
public:
   void display(int j)
   { cout<<"Derv::"<<j; }
};
int main()
{
   Base b; // xato, chunki Base sinf abstrakt
   Base *ptr = new Derv;
   ptr->display(10);
   return 0;
}
```

Virtual asos sinflar

Agar nasl olinayotdan class bir necha base classdan nasl olganda hatolik mavjud bo'ladi. Bunday holatlarda nima qilish kerak? Bunday holatlarda **virtual base class**lar ishlatiladi.

Bu kodda hatolik mavjud:

```
class base { public: int i; };
class derived1 :  public base { public: int j; };
class derived2 : public base { public: int k; };
```

```
class derived3 : public derived1, public derived2 {
        public: int sum;
};
derived3 ob;
ob.i = 10;
ob.j = 20;
ob.k = 30;
ob.sum = ob.i + ob.j + ob.k;
```

Oldingi misolni hatoligini to'g'irlash

```
class base { public: int i; };
class derived1 :  public base { public: int j; };
class derived2 : public base { public: int k; };
class derived3 : public derived1, public derived2 {
        public: int sum;
};
derived3 ob;
ob.derived1::i = 10;        // :: orqali qaysi classnikini ishlatish aytildi
ob.j = 20;
ob.k = 30;
ob.sum = ob.derived2::i + ob.j + ob.k;
```

Virtual base classdan foydalanish

```
class base { public: int i; };
class derived1 :  virtual public base { public: int j; };
class derived2 : virtual public base { public: int k; };
class derived3 : public derived1, public derived2 {
        public: int sum;
};
derived3 ob;
ob. i = 10;             // Bu holatda dastur to'g'ri ishlaydi
ob.j = 20;
ob.k = 30;
ob.sum = ob.i + ob.j + ob.k;
```

Nazorat savollari

1. Polimorfizm nima? Polimorfizmning qanday ko’rinishlari mavjud?
2. Virtual funksiya nima? Misol keltiring? Virtual funksiyaning asosiy xususiyatlarini (kamida ikkita) ayting?
3. Nasl olingan toifaga ko’rsatkich qanday ko’rinishda yoziladi. Asos sinf ko’rsatkichi orqali voris sinf a’zolariga murojaat qilish mumkinmi?
4. Abstrakt sinf nima va uning asosiy xususiyati nima?
5. Abstrakt sinf qanday yaratiladi? Misol keltiring?
6. Virtual asos sinflar qanday e’lon qilinadi.

11 – Ma’ruza. Standart amallarni qayta yuklash.

Reja

1. Operator overloading nima?
2. OPERATOR funksiyasidan foydalanish
3. Binar va unar amllarni yuklash (+, -, ++, --, =, +=, -=,>>,<<)
4. Mantiqiy operatorlarni yuklash
5. Yuklash mumkin bo’lmagan operatorla

Operatorlarni qayta yuklash nima?

C++ dasturlash tilida operatorlarni qayta yuklash imkoniyati mavjud. Biz o’zimiz yaratgan obyektlar ustida operatorlar yordamida ixtiyoriy amallarni bajarishimiz mumkin. Masalan:

```
int a, b, c;
c = a + b; // bunday yozishimiz mumkin
Myclass a,b,c;
c = a + b; // bu xolatda xatolik yuz beradi
```

Buning oldini olish uchun operatorlarni qayta yuklashimiz zarur!

OPERATOR function

Operatorlarni qayta yuklash uchun maxsus operator degan funksiyadan foydalaniladi va ushbu kalit so’zdan keyin qayta yuklanishi kerak bo’lgan amal yoziladi. Operator funksiyasi **operator** kalit so’zidan foydalanib yaratiladi.

Ushbu *operator funksiyasi* sinf ob’yektlari orasidagi qo’shimcha amallarni huklaydi va sinf a’zolariga mos ishlaydi. Operator funksiyasi sinfning a’zosi bo’lishi yoki bo’lmasligi mumkin. Demak operator funksiyasi sinf a’zosi bo’lmasa albatta bu funksiyaga mos friend (do’stona) sinf bo’lishi lozim.

Operator funksiyasi quyidagi forma bo’yicha yaratiladi:

```
qaytarish_tipi sinf_nomi::operator#(argumentlar)
{
        // ifoda
}
```

(+) operatorini qayta yuklash

Bu yerda **loc** (location) degan sinf yaratilyapti va bu sinfning ikkita **longitude** va **latitude** degan maydoni mavjud, **show()** metodi natijani qaytaradi.

```
#include <iostream>
using namespace std;
class loc {
	int longitude, latitude;
	public:
		loc() {}
		loc(int lg, int lt) {
			longitude = lg;
			latitude = lt;
		}
	void show() {
		cout << longitude << " ";
		cout << latitude << "\n";
	}
	loc operator+(loc op2);
};
// + operatorining qayta yuklanishi.
loc loc::operator+(loc op2)
{
	loc temp;
	temp.longitude = op2.longitude + longitude;
	temp.latitude = op2.latitude + latitude;
	return temp;
}
int main()
{
	loc ob1(10, 20), ob2( 5, 30);
	ob1.show(); // natija 10 20
	ob2.show(); // natija 5 30
	ob1 = ob1 + ob2;
	ob1.show(); // natija 15 50
	return 0;
}
```

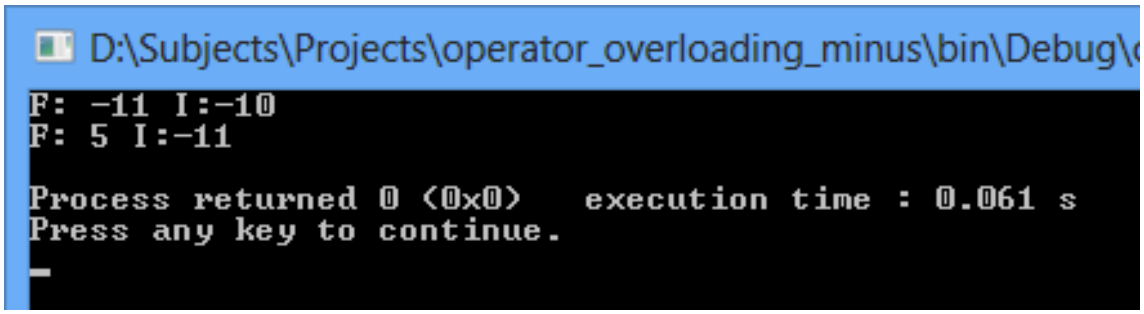
Demak bu yerda

- ob3 = ob1 + ob2 + ob3 + ob4; // mumkin!

- (ob1+ob2).show(x, y); // mumkin!
- ob2 = ob1 + 50; // mumkin emas!

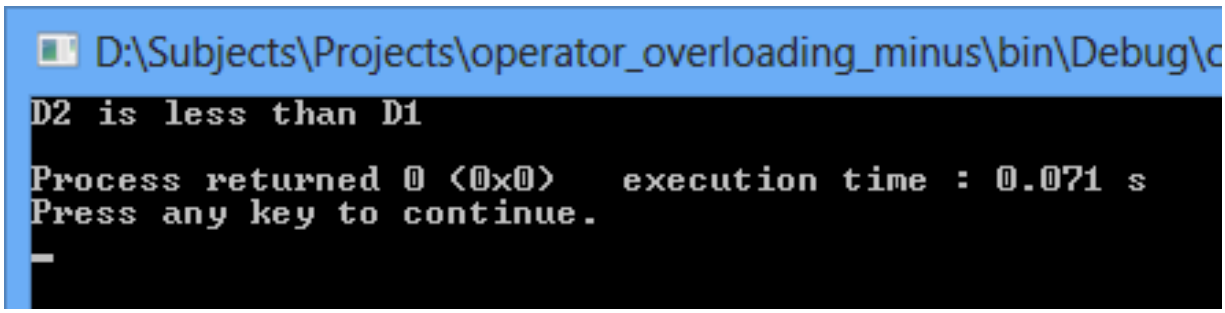
(-) operatorini qayta yuklash

```
#include <iostream>
using namespace std;
 class Distance
{
  private:
    int feet;             // 0 to infinite
    int inches;           // 0 to 12
  public:
    // required constructors
    Distance(){
      feet = 0;
      inches = 0;
    }
    Distance(int f, int i){
      feet = f;
      inches = i;
    }
    void displayDistance()
    {
      cout << "F: " << feet << " I:" << inches <<endl;
    }
    Distance operator- ()
    {
      feet = -feet;
      inches = -inches;
      return Distance(feet, inches);
    }
};
int main()
{
  Distance D1(11, 10), D2(-5, 11);
   -D1;                    // apply negation
  D1.displayDistance();    // display D1
  -D2;                     // apply negation
  D2.displayDistance();    // display D2
  return 0;
}
```



Musosabat operatori (oldingi dasturga o'zgartirish kiritamiz)

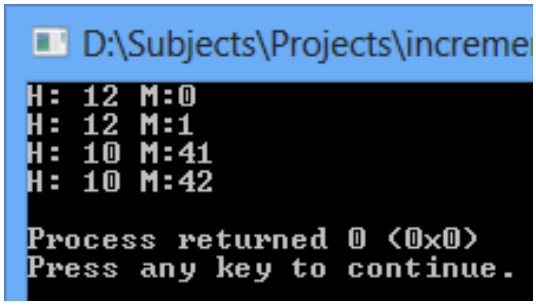
```
// overloaded < operator
bool operator < (const Distance& d)
{
if(feet < d.feet)
{
        return true;
}
if(feet == d.feet && inches < d.inches)
{
        return true;
}
return false;
}
main() funksiyasiga ham qo'shimcha kiritiladi
int main()
{
  Distance D1(11, 10), D2(5, 11);

  if( D1 < D2 )
  {
    cout << "D1 is less than D2 " << endl;
  }
  else
  {
    cout << "D2 is less than D1 " << endl;
  }
  return 0;
}
```



++ va – unar operatorlarini qayta yuklash

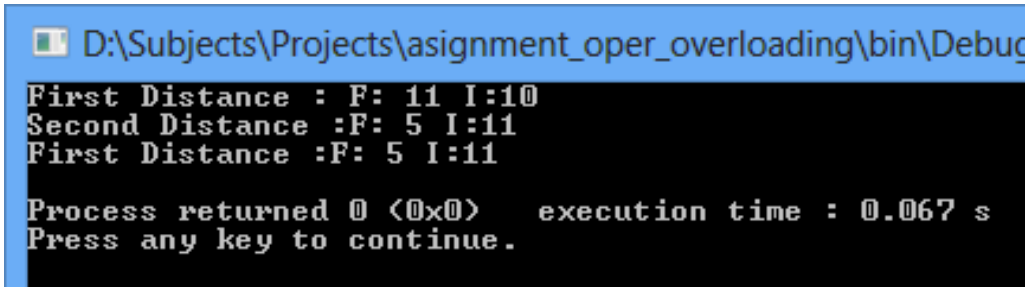
```
class Time
{
  private:
    int hours;            // 0 to 23
    int minutes;          // 0 to 59
  public:
    // required constructors
    Time(){
      hours = 0;
      minutes = 0;
    }
    Time(int h, int m){
      hours = h;
      minutes = m;
    }
    // method to display time
    void displayTime()
    {
      cout << "H: " << hours << " M:" << minutes <<endl;
    }
// overloaded prefix ++ operator
    Time operator++ ()
    {
      ++minutes;          // increment this object
      if(minutes >= 60)
      {
        ++hours;
        minutes -= 60;
      }
      return Time(hours, minutes);
    }
    // overloaded postfix ++ operator
```

```
    Time operator++( int )
    {
      // save the orignal value
      Time T(hours, minutes);
      // increment this object
      ++minutes;
      if(minutes >= 60)
      {
        ++hours;
        minutes -= 60;
      }
      // return old original value
      return T;
    }
int main()
{
  Time T1(11, 59), T2(10,40);

  ++T1;                 // increment T1
  T1.displayTime();     // display T1
  ++T1;                 // increment T1 again
  T1.displayTime();     // display T1

  T2++;                 // increment T2
  T2.displayTime();     // display T2
  T2++;                 // increment T2 again
  T2.displayTime();     // display T2
  return 0;
}
```



Tenglik amalini qayta yuklash

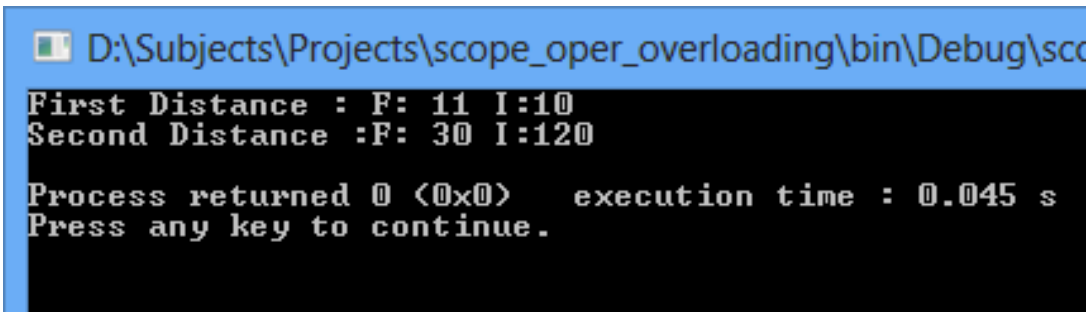
```
class Distance
{
  private:
    int feet;    int inches;
  public:
    // required constructors
    Distance(){
      feet = 0;         inches = 0;
    }
    Distance(int f, int i){
      feet = f;         inches = i;
    }
    void operator=(const Distance &D )
    {
      feet = D.feet;
      inches = D.inches;
    }
    // method to display distance
    void displayDistance()
    {
      cout << "F: " << feet <<  " I:" <<  inches << endl;
    }

};
int main()
{
  Distance D1(11, 10), D2(5, 11);
  cout << "First Distance : ";
  D1.displayDistance();
  cout << "Second Distance :";
  D2.displayDistance();
  // use assignment operator
  D1 = D2;
  cout << "First Distance :";
  D1.displayDistance();
  return 0;
}
```



Mantiqiy operatorlarni yuklash

```
class coord {
   int x, y;
public:
   coord() { x = 0; y= 0; }
   coord(int i, int j)
                        { x = i; y = j; }
   void get_xy(int &i, int &j)
                        { i = x; j = y; }
   int operator==(coord ob2);
   int operator&&(coord ob2);
};
int coord::operator==(coord ob2)
{
   return x==ob2.x && y==ob2.y;
}
int coord::operator&&(coord ob2)
{
   return (x && ob2.x) && (y && ob2.y);
}
if(o1==o2)
        { cout << “o1 o2 ga teng”}
if(o1&&o2)
        {cout << "o1 && o2 true";}
() operatorini yuklash. Misol-6
#include <iostream>
using namespace std;

class Distance
{
  private:
     int feet;             // 0 to infinite
     int inches;           // 0 to 12
```

```
  public:
    // required constructors
    Distance(){
      feet = 0;
      inches = 0;
    }
    Distance(int f, int i){
      feet = f;
      inches = i;
    }

// overload function call
    Distance operator()(int a, int b, int c)
    {
      Distance D;
      // just put random calculation
      D.feet = a + c + 10;
      D.inches = b + c + 100 ;
      return D;
    }
    // method to display distance
    void displayDistance()
    {
      cout << "F: " << feet <<  " I:" <<  inches << endl;
    }
};
int main()
{
  Distance D1(11, 10), D2;
  cout << "First Distance : ";
  D1.displayDistance();
  D2 = D1(10, 10, 10); // invoke operator()
  cout << "Second Distance :";
  D2.displayDistance();
  return 0;
}
```



Nazorat savollari

1. C++da operatorlarni qayta yuklash nima va u qanday formada foydalaniladi?
2. Oddiy binar qo’shish (+) va ayirish (-) amallari qanday qilib qayta yuklanadi?
3. Prefiks va postfiks ++ operatorini qayta yuklash qanday amalga oshiriladi. Misol keltiring?
4. Munosabat operatorlarini qayta yuklash qanday amalga oshiriladi. Misol keltiring?
5. Qaysi operatorlar qayta yuklanmaydi?
6. Qayta yuklanishi mumkin bo’lgan operatorlar qaysilar va qayta yulanishi mumkin bo’lgan operator o’z manosini yo’qotishi mumkinmi?

12 – Ma’ruza. Funksiya va sinf shablonlari.

Reja

1. Funksiya shablonlari (function template)
2. Funksiya shabloni xususiyatlari
3. Sinf shablonlari (class template)

Shablonlar haqida Ushbu darsda C++ dasturlash tilining asosiy hususiyatlaridan bo’lgan ***shablonlar*** bilan tanishamiz. Shablonlar yordamida umumiy funksiyalar va umumiy classlar yaratish imkoniyati mavjud.

Umumiy funksiya va umumiy classlar har xil ma’lumot toifalaridan ularni ***overload qilmasdan*** (ko’p kod yozmasdan) foydalanish imkoniyatini beradi. Ya’ni bunda biz ***har bir toifa uchun aloxida funksiya yozishimiz shart bo’lmaydi***. Shablonlar ikki xil bo’ladi:

- ***Funksiya*** shabloni (function template)
- ***Sinf*** shabloni (class template)

Funksiya shablonini (function template) yaratish

Funksiya shabloni **template** kalit so’zidan foydalangan holda amalga oshiriladi. Quyida funksiya shablonini yaratish formasi keltirilgan:

```
template <class TOIFA> qaytarish-tipi funk-nomi (arg-lar)
{
   // funksiya tanasi
}
```

Bu yerda *TOIFA* funksiya tomonidan joriy holda ishlatiladigan ma’lumot tipi. Ushbu toifani kompilyator avtomatik ravishda funksiyaga kelayotgan ma’lumot tipi bilan almashtirib qo’yadi.

Bu yerda **class** va **template** lar funksiya shablonini yaratish uchun ishlatiladigan kalit so’zlardir. Lekin ba’zi hollarda **class** kalit so’zi o’rniga **typename** kalit so’zini ishlatishimiz mumkin. Quyidagi misol hohlagan tipda berilgan ikkita o’zgaruvchi o’rnini almashtirib berish uchun xizmat qiladi va bunda har bir toifa uchun alohida funksiya yozishimizga zaruriyat qolmaydi.

Funksiya shabloniga misol

```
#include <iostream>
using namespace std;
// Funksiya shabloni e’lon qilinishi...
template <class X> void swapargs(X &a, X &b)
{
X temp;
temp = a;
```

```
a = b;
b = temp;
}
int main()
{
	int i=10, j=20;
	double x=10.1, y=23.3;
	char a='x', b='z';
	cout << "Original i, j: " << i << ' ' << j << '\n';
	cout << "Original x, y: " << x << ' ' << y << '\n';
	cout << "Original a, b: " << a << ' ' << b << '\n';
	swapargs(i, j); // swap funksiyasi butun toifa uchun (int)
	swapargs(x, y); // swap funksiyasi haqiqiy toifa uchun (float)
	swapargs(a, b); // swap funksiyasi simvol toifa uchun (char)
	cout << "Swapped i, j: " << i << ' ' << j << '\n';
	cout << "Swapped x, y: " << x << ' ' << y << '\n';
	cout << "Swapped a, b: " << a << ' ' << b << '\n';

	return 0;
}
```

Dastur izohi:

Umumiy funksiyaning boshqacha ko’rinishi

Quyidagi misolda **swapargs( )** funksiyasi boshqacharoq ko’rinishda e’lon qilingan. Ya’ni shablon birinchi satrda funksiya esa alohida satrda joylashgan.

```
template <class X>
void swapargs(X &a, X &b)
{
		X temp;
		temp = a;
		a = b;
		b = temp;
}
```

Lekin bu ko’rinishda birinchi va ikkinchi satr o’rniga bironta kod yozilsa xatolik beradi

```
template <class X>
int c // ERROR
void swapargs(X &a, X &b)
```

```
{
        X temp;
        temp = a;
        a = b;
        b = temp;
}
```

Funksiya shablonini override (qayta yozish) qilish.

```
template <class X> void swapargs(X &a, X &b)
{
    X temp;
    temp = a;
    a = b;
    b = temp;
    cout << "swapargs funksiya shabloni chaqirildi.\n";
}
// Bunda swapargs() funksiyasi faqatgina int tipi uchun ishlaydi.
void swapargs(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    cout << " int tipi uchun maxsus swapargs funksiyasi.\n";
}
int main()
{
    int i=10, j=20;
    double x=10.1, y=23.3;
    char a='x', b='z';
    cout << "Original i, j: " << i << ' ' << j << '\n';
    cout << "Original x, y: " << x << ' ' << y << '\n';
    cout << "Original a, b: " << a << ' ' << b << '\n';
    swapargs(i, j); // calls explicitly overloaded swapargs()
    swapargs(x, y); // calls generic swapargs()
    swapargs(a, b); // calls generic swapargs()
    cout << "Swapped i, j: " << i << ' ' << j << '\n';
    cout << "Swapped x, y: " << x << ' ' << y << '\n';
    cout << "Swapped a, b: " << a << ' ' << b << '\n';
    return 0;
```

```
}
```

Dastur natijasi:

```
Original i, j: 10 20
Original x, y: 10.1 23.3
Original a, b: x z
int tipi uchun maxsus swapargs funksiyasi.
swapargs funksiya shabloni chaqirildi.
swapargs funksiya shabloni chaqirildi.
Swapped i, j: 20 10
Swapped x, y: 23.3 10.1
Swapped a, b: z x
```

Funksiya shablonini Override qilish yangi usuli

```
template<> void swapargs<int>(int &a, int &b)
{
        int temp;
        temp = a;
        a = b;
        b = temp;
        cout << " int tipi uchun maxsus swapargs funksiyasi.\n";
}
```

Funksiya shablonini overload qilish.

```
// Oddiy funksiyalardek, funksiya shablonini ham overload qilish mumkin.
#include <iostream>
using namespace std;
// f() funksiya shablonining birinchi turi.
template <class X> void f(X a)
{
        cout << "Inside f(X a)\n";
}
// f() funksiya shablonining ikkinchi turi.
template <class X, class Y> void f(X a, Y b)
{
        cout << "Inside f(X a, Y b)\n";
}
int main()
```

```
{
        f(10); // calls f(X)
        f(10, 20); // calls f(X, Y)
        return 0;
}
```

Funksiya shablonining kamchiligi

- Umumiy funksiyalar funksiya overloadining o’rnini bosishi mumkin.
- Lekin bu yerda bitta kamchilik mavjud.
- Biz oddiy funksiyani overload qilganimizda, har xil ma’lumotlar tipi uchun ***funksiya tanasini har xil*** qilib yozishimiz mumkin.
- Lekin umumiy funksiyada har xil tip qabul qila olgani bilan funksiya tanasi har doim bir xil bo’ladi, chunki bitta funksiyaga murojaat bo’ladi.
- Faqatgina ***ma’lumotlar tipi*** har xil bo’la oladi.

Umumiy sinflar (sinf shabloni)

Sinf shablonini e’lon qilishning umumiy formasi:

```
template <class TOIFA> class sinf_nomi{
        ...
}
```

Yoki quyidagi ko’rinishda e’lon qilish mumkin

```
template <class TOIFA>
class sinf_nomi {
        ...
}
```

Sinf shablonini ishlatish

```
sinf_nomi <tip> obyekt;
```

Sinf shabloni uchun oddiy misol.

```
#include <iostream>
using namespace std;
template <class T>
class mypair {
    T a, b;
  public:
    mypair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};
template <class T>
T mypair<T>::getmax ()
{
  T retval;
  retval = a>b? a : b;
  return retval;
}
int main () {
  mypair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

Sinf shablonida ikki xil toifadan foydalanish

```
#include <iostream>
using namespace std;
template <class Type1, class Type2> class myclass
{
        Type1 i;
        Type2 j;
        public:
                myclass(Type1 a, Type2 b) { i = a; j = b; }
                void show() { cout << i << ' ' << j << '\n'; }
```

```
};
int main()
{
	myclass<int, double> ob1(10, 0.23);
	myclass<char, char *> ob2('X', "Templates add power.");
	ob1.show(); // show int, double
	ob2.show(); // show char, char *
	return 0;
}
```

Dastur natijasi:

```
10 0.23
X Templates add power.
```

Maxsuslashtirilgan sinf shabloni

template<> konstruktori maxsusashtirilgan sinf shabloni uchun ishlatiladi

```
template <class T> class myclass {
	T x;
	public:
		myclass(T a) {
			cout << "Inside generic myclass\n";
			x = a;
		}
	T getx() { return x; }
};
// int toifasi uchun maxsuslashtirilgan sinf shabloni.
template <> class myclass<int> {
	int x;
	public:
		myclass(int a) {
			cout << "Inside myclass<int> specialization\n";
			x = a * a;
		}
	int getx() { return x; }
};
```

Shablonning asosiy xususiyatlari

- ***reusable*** kod yozish imkonini beradi.
- Shablonlar yordamida framework lar yaratish mumkin

- Dastur kodi egiluvchanlik xususiyatiga ega bo’ladi.
- Turli xil tipdagi ma’lumotlar ustida ishlash uchun kod yozishda vaqtni tejash
- C++ dagi STL lar (Standard Shablon Kutubxonalar), nomidan ko’rinib turibdiki, shablonlar yordamida yaratilgan

Nazorat savollari

1. Shablon nima va uning qanday turlari mavjud?
2. Funksiya shabloni qanday yaratiladi?
3. Funksiya shablonida turli xil tiplardan foydalanish mumkinmi? Agar mumkin bo’lsa bunday turdagi funksiya shablonlari qanday ko’rinishda yaratiladi?
4. Funksiya shablonini override qilish formasi qanday? Misol keltiring?
5. Funksiya shablonini override qilishning yangi formasi qanday ko’rinishda?
6. Funksiya shabloni qanday overload qilinadi?
7. Sinf shabloni qanday yaratiladi?
8. Sinf shablonida ikki va undan ortiq toifalardan foydalanish qanday amalga oshiriladi. Misol keltiring?
9. Shablonning kamchiliklari va afzalliklari?

13 – Ma’ruza. Oqimli sinflar

Reja:

1. Oqimli sinflar ierarxiyasi.
2. Oqimli sinflar metodlari.
3. Formatlash.
4. Matnli oqimlar.
5. Fayllar bilan ishlash.

Oqimli sinflar ierarxiyasi

C++da oqimli sinflar kutubxonasi ikkita asosiy ios va streambuf sinflar asosida tuzilgan, streambuf sinfi kiritish-chiqarish fizik qurilmalari bilan xotirada joylashgan kiritish-chiqarish buferlarni o‘zaro bo‘g‘lanishini va tashkilini ta’minlaydi. Streambuf sinfining metodlarini va ma’lumotlarini dasturchi ochiq ishlatmaydi. Mavjud bo‘lgan sinflar asosida yangi sinflarni yaratishda dasturchiga xam sinfga murojaat etish ruxat etilgan. **ios** sinfi formal ***kiritish chiqarish*** va ***xatolarni tekshirish*** vositalarga 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’ekt-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 xaqidagi xabar chiqarish standart oqimi.

Oqimli sinflar metodlari

Oqimdan qiritish uchun istream sinfdagi ob'ektlar ishlatiladi, oqimga chiqarish uchun - ostream sinfdagi ob'ektlar.

istream sinfda quyidagi funksiyalar tavsiflangan:

- istream get (char& S) - istream dan S ga simvolni o‘qiydi. Xato xolatida S 0XFF qiymatini oladi.
- int get() - istream dan keyingi simvolni chiqaradi. Faylni oxirini aniqlagach EOFni qaytaradi.
- istream& get(char* buffer,int size,char delimiter='\n') - Bu funksiya istreamdan 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 bulingan simvollar ketma-ketligi xardoim nul 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‘kib chiqilgan simvollar nul simvoli bilan ta'momlanadi.
- istream& read(char* buffer,int size) - Ajratuvchilarni qo‘llanmaydi va buferga o‘qilgan simvollar nul simvoli bilan tugamaydi.
- int peek() - istream dan simvolni chiqarmasdan istreamga qaytaradi.
- int gcount() - Formatlanmagan oxirgi kiritish operatsiyasi vaqtida o‘qilgan simvollar sonini qaytaradi.
- istream& putback(S) - Agar get doirasidagi streambuf ob'ektida 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 ostreamga yozadi. Xatoga duch kelmaguncha yoki size simvollarni nusxasi 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 ostreamga 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'ekti, o'ng operandi esa – bu dasturlash tilida ko'rsatilgan tipdagi ma'lumot.

Misol:

```
char ch, next, lookahead;
while ( cin.get( ch ))
{    switch (ch) {
  case '/':
     //  izoxni 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 );
}
```

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 qiymatlarni tashki tavsiflash aytib o‘tilmagan formatlardan foydalanishga olib keladi.

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

Formatlash bayroqlardan tashqari ios sinfning kuydagi protected komponentalari ishlatiladi:

int x_width – chiqarish maydonning minimal eni.

int x_precision – qiritishda xaqiqiy sonlarning tavsiflash aniqligi (kasr qisimning raqamlar soni);

int x_fill – chiqarishda to‘ldiruvchi simvol, probel – ko‘rsatilmagan xolda.

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.

Manipulyatorlar

Manipulyatorlar - oqim ishini modifikatsiyalashini imkon etuvchi maxsus funksiyalar. Manipulyatorlarning xususiyati shundaki, ularni >> yoki << operatsiyalarning o‘ng operand sifatida foydalanish mumkin. Chap operand sifatida esa xar doimgidak oqim (oqimga ilova) ishlatiladi, va xudda shu oqimga manipulyator ta’sir etadi.

endl Yangi qator simvolini qo‘yib, bufer ostream bo‘shatish

- dec O‘nlik sanoq tizmida chiqarish (ko‘zda tutilgan)
- hex O‘n oltilik sanoq tizimida chiqarish
- oct Sakkizlik sanoq tizimida chiqarish
- ws Bo‘shlik 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 qiritish yoki chiqarish maydonning enini o‘rnatish
- setbase(b) b asosiga ega bo‘lgan butun sonlarning chiqarish

Fayllar bilan ishlash

Fayllarni ochish va yopish

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

Ftsream kutubxonasi fayllarni o‘qib olish uchun javob beradigan ifstream sinfiga, xamda faylga axobotning yozib olinishiga javob beradigan oftsream sinfiga ega.

Biron-bir faylni yozish yoki o‘qish uchun ochish uchun, oftsream turdagi yoki mos xolda iftsream turdagi o‘zgaruvchini yaratish kerak. Bunday o‘zgaruvchini initsiallashda fayl nomi o‘zgaruvchi nomidan keyin qavs ichida berilgan belgilar massivi ko‘rinishida uzatiladi.

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

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

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

Faylga yozish

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

Bundan tashqari ‘’<<’’operatoridan foydalanish mumkin. Bu operatordan kodning bitta satrida turli turdagi qiymatlarni uzatgan xolda ko‘p martalab

foydalanish mumkin. Satr xaqida gap ketganda, chiqarish satr oxiri belgisi, ya'ni '\n' paydo bo'lishidan oldin amalga oshiriladi. Belgisiz turga ega bo'lgan barcha o'zgaruvchilar oldin belgilarga o'zgartirib olinadi.

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

Fayldan o'qish

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

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

```
cout<<ss;
```

Fayl oxirini aniqlash

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

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

Ushbu xolda dastur, eof funksiyasi yolg‘on (0) ni qaytarguncha, tsiklni 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;
  }
}
```

Fayllar bilan ishlashda xatolarni aniqlash

Xatolarni kuzatib borishda dasturlarga yordam berish uchun, fayl ob'ektining 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 xaqiqatni 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;     }   }
}
```

Faylning kerak bo‘lmay qolganda berkitilishi

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, dastur 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.**

O‘qish va yozish operatsiyalarining bajarilishi

Xozirga qadar gap borayotgan dasturlar belgili satrlar utsida operatsiyalar bajarar edi. Dasturlaringiz murakkablashgan sari, extimol, 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 xaqidagi 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;
}
```

Nazorat savollari:

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

14 – **Ma’ruza. Istisnolarni boshqarish**

Reja:

1. Istisnolar.
2. Istisnolarni qayta ishlash.
3. Istisnolar generatsiyasi.
4. Istisno xolatning ma’lumotlar elementlaridan foydalanish
5. Istisno xolatlar va sinflar
6. Istisnolar va konstruktorlar

Istisnolar

C++tili istisnolarga xizmat ko‘rsatish standartini belgilab beradi. Istisno xolatlar (exception) dasturda xatoni – kutilmagan xodisani ifodalaydi. Dastur o‘zining ishlab chiqilishida ko‘zda tutilmagan normal bo‘lmagan vaziyatga duch kelganda, boshqaruvni ushbu muammoni xal qilishga qodir bo‘lgan dasturning boshqa qismiga berishi mumkin xamda yo dasturni bajarishni davom ettirish yoki ishni tugallash kerak. Istisnolarni joydan joyga tashlab berish (excpletion 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-xarakatlarni 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 xolatlar istisno xisoblanmaydi.

Dasturda xar bir istisno xolat sinf sifatida aniqlanadi. Masalan, quyida ko‘rsatilan xolat fayllar bilan ishlash uchun uchta istisno xolatni aniqlaydi:

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

Istisno xolatlar o‘zgaruvchilarni va sinf funksiya – elementlarini ishlatish mumkin. Xar bir istisno xolat sinfga mos.

Istisnolarni qayta ishlash

Dastur istisno xolatni ko‘rishdan va unga javob berishdan oldin istisno xolatni aniqlovchi C++dagi try operatorini ishlatish 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 xamda quyidagi xatti-xarakatlar 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 xar bir istisno uchun o‘z ishlov bergichi ko‘zda tutilgan bo‘lishi kerak. Istisnolarga ishlov bergichlar navbatma-navbat ko‘rib chiqiladi xamda turi bo‘yicha catch operatoridagi argument (dalil) turiga to‘qg‘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_sopy funksiyani chaqirishda quyidagi try operatori istisno xolatni aniqlash imkonini beradi:

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

Qanday istisno xolat ro‘y berganini aniqlash uchun try operatordan so‘ng dastur bitta yoki bir nechta catch operatorlarni joylashtirish lozim:

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

```
}
```

Bu xolda xato tipiga qaramasdan kod xabardor qiladi va dasturni tugatadi. Agarda funksiyaning chaqiruvi xatosiz bajarilgan va istisno xatolar aniqlanmagan bo‘lsa C++ catch operatorini shunchaki etiborga olmaydi.

Qayta ishlovchilar tartibi muximdir.

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

Istisnolarni generatsiya qilish

C++ o‘zi istisno xolatlarni yuzaga keltirmaydi. Ularni C++ ning throw operatoridan foydalangan dasturlar yuzaga keltiradi. Istisno yuzaga kelganda, throw operatoridagi <ifoda> nom berish ifodasi muvaqqat ob’ektni nomlaydi (initsiallashtiradi), Bunda muvaqqat ob’ektning turi ifoda argumenti (dalili) ning turiga mos keladi. Ushbu ob’ektning boshqa nusxlari, masalan, istisno ob’ektidan nusxa ko‘chirish konstruktori yordamida generatsiya qilinishi mumkin.

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

Kutilmagan istisnolarni qayta ishlash

Agar dasturda kuzda tutilmagan istisno xodisa yuz bersa standart istisnolarni qayta ishlovchi ishlatiladi. Kup xollarda bu standart qayta ishlovchi dastur

bajarilishini tuxtatib kuyadi. 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 foydjalanish lozim. Bu funksiyalar prototiplari except.h sarlavxali faylda aniklangan. Bu funksiyalar void tipiga ega bo‘lib parametrsiz bo’ladi.

Istisno xolatning ma’lumotlar elementlaridan foydalanish

Yuqorida ko‘rib o‘tilgan misollarda dastur, catch operatordan foydalanib, qanday istisno xolat ro‘y berganini va ularga tegishli xolda javob berishini imkonini beradi. Masalan, file_open_error istisno xolatda dastur xatoni chaqiruvchi fayl nomini bilish lozim. Istisno xolatga tegishli shunday ma’lumotni saqlash uchun dastur istisno xolat sinfiga ma’lumotlar elementlarini qo‘shish. Agar keyinchalik dastur istisno xolatni yuzaga keltirsa, u ushbu ma’lumotni, quyida ko‘rsatilgandek, istisno xolatiga ishlov beruvchi funksiyaga o‘zgaruvchi sifatida uzatadi:

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

Istisno xolatga 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 xolatni o‘zgartiradi:

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

Istisno xolatlar va sinflar

Sinf yaratishda shu sinfga tegishli istisno xolatlarni aniqlash mumkin. Aniq sinfga tegishli istisno xolatni yaratish uchun ushbu istisno xolatni 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 xolatni tasvirlovchi ob’ektga nom berish kerak:

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

```
  // ...
}
// ...
        }
```

Qavsdagi konstruktsiya tavsif bo‘lib funksiya formal parametriga mosdir. Unda parametr tipi va yuzag kelgn istisno nomi berilishi mumkin.

Istisnolar va konstruktorlar

Istisnolar konstruktordagi xatolar xaqida ma’lumot berishga imkon beradi. Konstruktor chaqiruvchi funksiya tekshirib ko‘rishi mumkin bo‘lgan qiymat qaytarmagani uchun istisnolarsiz quyidagicha xatolik xaqida ma’lumot berish mumkin:

1. Ob’ektni xatolik bilan qaytarish toki foydalanuvchi o‘zi tekshirib ko‘rsin.
2. Lokal bo‘lmagan o‘zgaruvchiga ob’ekt yaratilmagani xaqida ma’lumot beruvchi qiymat o‘rnatish.

Istisnolar ob’ekt yaratilmagani xaqidagi 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‘qri o‘lchami
  }
  // ...
  return p;
}
```

Nazorat savollari:

1. Istisnolarni nima uchun generatsiya 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 – **Ma’ruza. Standart shablon sinflar (STL) kutubxonasi**

Reja:

1. STL tarkibi
2. Sinf-konteynerlar
3. Konstruktorlar
4. Iteratorlar
5. Xotirani taqsimlovchilar, predikatlar va solishtirish funksiyalari
6. Assotsiativ konteynerlar (massivlar)
7. Konteyner usullari

STL tarkibi

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

- **Konteynerlar** (containers) – bu boshqa elementlarni saqlovchi ob’ektlar. Masalan, vektor, chiziqli ro‘yxat, to‘plam.
- **Assotsiativ konteynerlar** (associative containers) kalitlar yordamida ularda saqlanadigan qiymatlarni tezkor olish imkonini yaratadi.

  Xar bir sinf – konteynerida ular bilan ishlash uchun mo‘ljallangan funksiyalar to‘plami aniqlangan. Masalan, ruyxat elementlarni kiritish, chiqarish, va qo‘shish funksiyalarni 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 chiziqi ro‘yxatini ifodalaydovchi ketma-ketlik (sequence) bilan ishlash uchun mo‘ljallangan.
- **Iteratorlar** (iterators) – bu konteynerga nisbatan ko‘rsatkich sifatida bo‘lgan ob’ektlar. Ular massiv elementlariga ruxsat oluvchi ko‘rsatkichlar kabi, konteyner ichidagiga ruxsat olish imkoni beradi.

Sinf-konteynerlar

STL da quyidagi sinf-konteynerlar aniqlangan:

Asosiy konteynerlar

- **vector** <vector.h> dinamk massiv
- **list** <list.h> chiziqli ro‘yxat
- **deque** <deque.h> ikki tarafli dvustoronnyaya tartib
- **set** <set.h> to‘plam
- **multiset** <set.h> xar bir elementi noyob bo‘lishi shart emas to‘plam
- **map** <map.h> kalit/ qiymat juftlikni saqlash uchun assotsiativ ro‘yxat. Bunda xar bir kalit bitta qiymat bilan bog‘langan.
- **multimap** <map.h> xar bir kalit bilan ikkita yoki ko‘proq qiymatlar bog‘langan

Xosila konteynerlar

- **stack** <stack.h> stek
- **queue** <queue.h> tartib
- **priority_queue** <queue.h> birinchi o‘rindagi tartib

Konstruktorlar

Ixtiyoriy sinf-konteyner ko‘rsatilmagan xolda konstruktor va destruktorni nusxalovchi konstruktorga ega.

Masalan, vektor sinf-konteynerning konstruktori va destruktori:

vector<elem> c	bitta xam 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 xolda 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’ekt uchun ko‘rsatilmagan xolda konteynerda saqlanuvchi konstruktor mavjud bo‘lishi shart. Undan tashqari, ob’ekt uchun < va == operatorlar aniqlanish lozim.

Iteratorlar

Itaratorlar bilan ko‘rsatkichlar kabi ishlash mumkin. Ularga *, inkrement, dekrement operatorlarni qo‘llash mumkin. Iterator tipi sifatida xar xil konteynerlarda aniqlangan iterator tip elon qilinadi.

Itoratorlarning beshta tipi mavjud:

1. **Kiritish iteratorlar** (input_iterator) tenglik, nomini o‘zgartirish va inkrementa operatsiyalarni qo‘llaydi.

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

Kiritish iteratsiyasining maxsus xolati istream_iterator iborat.

2. Chiqarish iteratorlar (output_iterator) o‘zlashtirish operatorning chap tarafidan imkon bo‘lgan isimning o‘zgartirish va inkrementa operatsiyalar qo‘llanadi.

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

Chiqarish iteratsiyasining maxsus xolati ostream_iterator.

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

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

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

5. Ixtiyoriy ruxsatga ega bo‘lgan iteratorlar (random_access_iterator) biderectional-iteratorlarning barcha xususiyatlariga ega, bundan tashqari solishtirish va manzil arifmetikasi operatsiyalarni 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 iteratoralar bo‘lishi mumkin.

Xotirani taqsimlovchilar, predikatlar va solishtirish funksiyalari

Konteynerlarga, algoritmlarga va STLdagi iteratorlarga qo‘shimcha bir nechta standart komponentalar xam qo‘llaniladi. Ulardan asoslari esa **xotira taqsimlovchilar, predikatlar,va solishtirish funksiyalaridir**.

Xar bir konteynerda uning uchun aniqlangan va konteyner uchun xotirani belgilash jarayonini boshqaradigan xotira taqsimlovchisi (**allocator**) mavjud. Ko‘rsatilmagan xolda esa xotira taqsimlovchisi **allocator** sinf ob’ektidir. Xususiy taqsimlovchini tavsiflash mumkin.

Ba’zi bir algoritmlar va konteynerlarda muxim tipdagi predikat ataluvchi funksiyalar ishlatiladi. Predikatlar unar va binar bo‘lishi mumkin. U yoki bu qiymatni olish aniq shartlari dasturchi orqali aniqlanadi. Unar predikatlarning tipi – **UnPred**, binar predikatlarning esa - **BinPred**. Argumentlar tipi konteynerda saqlanuvchi ob’ektlar tipiga mos.

Ikkita elementlarni solishtirish uchun binar predikatlarning maxsus tipi aniqlangan. U **solishtirish funksiya** (comparison function) deyiladi. Agarda birinchi element ikinchidan kichik bo‘lsa, unda funksiya rost qiymatni qaytaradi. **Comp** tip funksiya tipidir. STL da ob’ekt-funksiyalar o‘ziga xos axamiyatga ega. Ob’ekt-funksiyalar – bu sinfda «kichik qavslar» () operatsiyasi aniqlangan sinf nusxalari. Ba’zi bir xollarda funksiyalarni ob’ekt-funksiyalarga almashtirish qulay deb xisoblanadi. Ob’ekt-funksiya funksiya sifatida ishlatilsa, unda uni chaqirish uchun operator () operator ishlatiladi.

Vector-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 son, xar bir elementi esa qiymat qiymatiga teng. Qiymat parametri ko‘rsatilmagan xoldagi 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’ekt uchun ko‘rsatilmagan xolda konstruktor aniqlash zarur. Bundan tashqari, ob’ekt 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)

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

Ruyxat(List)

Ro’yxat – ikki yo‘nalishli iteratorlarni qo‘llaydigan xamda kiritish va o‘chirish operatsiyalarni o‘zgarmas vaqtda ketma-ketlikni ixtiyoriy joyida bajaradigan, shuningdek, xotirani boshqarishiga avtomatik ravishda ishlov beruvchi ketma-ketlik ko‘rinishi. Vektorlar va ikkitarafli tartiblardan farqi

shundaki elementlar ro‘yxatiga tez va ixtiyoriy ro‘xsat qo‘llanmaydi, lekin ko‘pgina algoritmlarga esa ketma-ketlik ruxsat zarur.

Assotsiativ konteynerlar (massivlar)

Assotsiativ massiv juft qiymatlardan iborat. (key) kalit 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 tavsiflash mumkin:

V& operator[](const K&) 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 knstruktorlar aniqlangan:

birinchi shakli bo‘sh assotsiativ konteynerning konstruktorini tavsiflaydi. Ikkinchi shakli – konstruktor nusxasi, uchinchisi – elementlar diapazonini kamrab olgan assotsiativ konteynerning konstruktori.

O‘zgartirish operatsiyasi aniqlangan:

map& **operator=**(const map&);

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

mapda kalit/qiymat juftliklar **pair** tiplagi ob’ektlar ko‘rinishida saqlanadi.

Kalit/qiymat juftliklarni faqatgina pair sinf konstruktorlari yordamida, balki pair tipdagi ob’ektlarni 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 qidiruv.

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

set to‘plamini assotsiativ massiv sifatida ko‘rish mumkin. Unda qiymatlar axamiyatga ega emas, shuning uchun faqat kalitlarni ko‘zatish mumkin.

to‘plamlar assotsiativ massiv kabi T tip uchun (<) “kichik” operatsiyani mavjudligini talab etadi. U o‘z elementlarini saralangan xolda saqlaydi. saralash almashuvi esa tartibda bajariladi.

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()** oxiri 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 (ikta 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 (ikta 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 pzitsiyasi topadi
- **lower_bound(elem)** element qo‘yish mumkin bo‘lgan birinchi pozitsiyani to‘padi
- **upper_bound(elem)** element qo‘yish mumkin bo‘lgan oxirgi pozitsiyani to‘padi
- **equal_range(elem)** element qo‘yish mumkin bo‘lgan birinchi va oxirgi pozitsiyalarni to‘padi

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** (kuyi chegarasini) va **upper_bound** (yuqori chegarasini) topadi.
- Boshqa usullar
- **size()** elementlar soni;
- **empty()** konteyner bo‘shmi?
- **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. Xar 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 xajmini o‘zgartirish uchun qanday usuldan foydalaniladi?

16 – **Ma’ruza. Sinf hodisalari**

Reja:

1. Komponentlar.
2. Komponentli sinflarni e’lon qilish.
3. Xususiyatlarni e’lon qilish.
4. Voqealar ishlatgichlarining e’lonlari.

C++Builder, nafaqat ANSI C++ standarti kiritayotgan yangiliklarni qo‘llab-quvvatlaydi, balki tilni yangi imkoniyatlar bilan boyitadi. Shuni tushunib olish muximki, tilni kengaytirish xech qachon quruq maqsad bo‘lib qolmagan, va xamon standart C++ doirasida yozilgan mantlarni 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 (maslan, _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 axamiyatli kengaytirishlari ko‘rib chiqiladi. Ular asosan tarkibli sinflarga mansub bo‘lib, C++Builder muxitida ishlab chiqilayotgan ilovalarda muttasil uchrab turadi.

Komponentlar (tarkibiy qismlar)

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) xamda Ob’ektlar Inspektori yordamida berilgan voqea-xodisaga ishlov bergichga kiritish.

Komponentli sinflarni e’lon qilish

C++Builder tarkibiga kiradigan Vizual Komponentalar Kutubxonasi - VCL sinflarining ilgarilovchi e’lonlari _declspec modifikatoridan foydalanadi:

_declspec(<spetsifikator>)

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 qiymatlardan birini qabul qiladi:

delphiclass - u TObject sinfiga tegishli VCL ning bevosita yoki bilvosita xosilalarining ilgarilovchi e’loni uchun qo‘llanadi. 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 xosilalarining ilgarilovchi e’loni uchun qo‘llanadi. U VCL ning parametrlar va a’zoqfunksiyalarning qaytarilayotgan qiymatlari bilan muomalasida muvofiqlik qoidalarini belgilaydi.

Pascal implementation tarkibli sinf Ob’ektli 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’ektlar uyumning dinamik xotirasida new operatori yordamida yaratiladi.

Xususiyatlarni e’lon qilish

C++Builder tarkibli sinflar xususityalarini identifikatsiya qilish uchun _property modifikatoridan foydalanadi. 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 metodi > ma’lumotlar a’zosining qiymatini olish usulini aniqlaydi;
- default=<bul konstatntasi > .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’ektlar Inspektori tomonidan aks ettiriladigan komponentalar xususiyatlarini spetsifikatsiyalash uchun _published modifikatoridan foydalanadi. Agar komponentaning ishlab chiquvchisi biron-bir xususiyat qiymatini modifikatsiyalashga ruxsat berishni 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’ektlar Inspektoriga RTTI axboroti uzatiladi.

Voqealar ishlatgichlarining e’lonlari

C++Builder voqealar ishlatgichlari funksiyalarining e’loni uchun _closure modifikatoridan foydalanadilar:

<tur>(_closure<name>)(<pametrlar 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 xam uzatadi (joriy sinf ekzemplyariga txis 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 xam beradi. Bu qobiliyat Ob’ektli Paskaldan o‘zlashtirilgan edi, _closure yesa Vizual Komponentalar Kutubxonasidagi voqealar mexanizmini ishga tushirishda xavodek zarur bo‘lib qoldi.

Funksiyalarning tez chaqirilishi

Parametrlari protsessorli registrlar orqali uzatiladigan funksiyalarni e’lon qilishda _fastcall modifikatori qo‘llanadi:

<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‘llanmaydi,

Xolisanillo aytganda, funksiyalarning tez chaqirilishi C++Builder kompilyatorininggina vazifasiga kirmaydi. Voqealarga ishlov berish funksiyalarini e’lon qilishda_fastcall ning qo‘llanishiga aloxida e’tibor berish kerak. Bu voqealarni C++Builder avtomatik tarzda generatsiya qiladi.

Nomlar fazosi

Oddiy ilovalarning ko‘pi dastlabki dastur matniga ega bo‘lgan bir nechta fayldan iborat. Bu fayllar dasturchilar guruxi 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 yokitranslyatsiya 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 xolda xar 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 xar bir tarmoq tizim nomlarni tanlashda erkin ish tutadi, xamda uning muallifi xuddi shunday ismlardan biron boshqa kimsa foydalanishi mumkinligiga qayg‘urmasa xam bo‘ladi. Xar bir tarmoq tizim global nomlar umumiy fazosida o‘zining paydo bo‘lganini namespace kalit-so‘zdan 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 qirish:
- using namespace::ALPHA;//ALPHA dagi barcha nomlarga kirish
- Nomlarning lokal fazosida yangi identifikatorning e’lon qilinishi:
- using :: new_name;//identifikatorning qo‘shilishi

Ochiq-oydin e’lonlar

Odatda bitta parametrli konstruktor e’lon qilingan sinf ob’ektlariga turlari avtomatik tarzda (yashirish xolda) o‘z sinfi turiga qayta o‘zgaradigan qiymatlarni berish mumkin. Konstruktorni e’lon qilishda explicit konstruktoridan foydalanish mumkin:

explicit<konstruktorni e’lon qilish >

Bu xolda berilgan sinf konstruktorlarini explicit kalit-so‘z bilan e’lon qilishda sinfning barcha ob’ektlariga faqat shunday qiymatlarni berish mumkinki, bu qiymatlar turlari o‘z sinfi turiga ochiq-oydin qayta o‘zagaradigan 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’ektlariga berilgan bo‘lsa, ular xuddi shu sinfiy turga qayta o‘zgartirilishini talab qiladi.

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‘llanadi:

```
volatile<tur><ob’ekt nomi>;
```

C++da volatile kalit-so‘zning qo‘llanishi sinflar va a’zo-funksiyalarga xam tegishlidir. Bu kalit-so‘z ko‘rsatilgan ob’ekt qiymatiga nisbatan taxminlar qilishni kompilyatorga ta’qiqlaydi, chunki bunday qilinsa, ushbu ob’ektni o‘z ichiga olgan ifodalarni xisoblashda, uning qiymati xar 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
tickC++;
void wait (int interval)
ticks =0;
while (ticks < interval); // Kutish tsikli
}
```

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 tsikl ichidagi xar 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‘zning 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 main(){
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 foydalanish.
5. Ochiq oydin va o‘zgaruvchan e’lonlar nima uchun ishlatiladi?