

Madraximov Sh.F.

C++ OBYEKTGA YO'NALTIRILGAN DASTURLASH



681.3.01
14-14.

O'ZBEKISTON RESPUBLIKASI
OLIY VA MAXSUS TA'LIM VAZIRLIGI

MIRZO ULUG'BEK NOMIDAGI
O'ZBEKISTON MILLIY UNIVERSITETI

Madraximov Sh.F.

**C++. OBYEKTGA YO'NALTIRILGAN
DASTURLASH**

O'quv qo'llanma

Toshkent
"METODIST NASHRIYOTI"
2023

-14402/23-

6
7
7
9
1
2
3
3
1
5
1
1

UO'K: 681.3.01(075.8)
KBK: 32.973.26-018.2
M 14

Madraximov Sh.F.

C++. OBYEKTGA YO'NALTIRILGAN DASTURLASH

/ O'quv qo'llanma/ - Toshkent: "Metodist nashriyoti". 2023.
170 bet.

Qo'llanmada C++ tilida obyektga yo'naltirilgan dasturlash texnologiyasini amalga oshirilishi qaraladi. Unda obyektga yo'naltirilgan dasturlashni amalga oshirishda asosiy vositasi sinftushunchasi, sinfning a'zolari va ularga murojaat toifalari, sinfning do'stlari, sinflar vorisligi, polimorfizmni amalga oshirish, statik va dinamik polimorfizmlar, operatorlarni qayta yuklash, istisno holatni qayta ishlash, qoliplar va standart qoliplar kutubxonasi mavzulari qamrab olingan va namunaviy masalalar keltirilgan.

Qo'llanma dasturlashni o'zlashtirish bilan bog'liq ta'lim yo'nalishi va mutaxassisligi talabalariga, ta'lim beruvchilar, hamda mustaqil o'rganuvchilarga mo'ljallangan.

The manual focuses on implementation of technologies object-oriented programming in C++. There is considered theories and practical tasks of topics as the concept of class which basic tool for implementing technology object-oriented programming in C++, class members and types of call them, friends of classes, inheritance, implementation of polymorphism, static and dynamic polymorphism, overloading operators, exception handling, templates and standard templates library.

Guide is planned for students who educated on the direction and specialty related to the study programming and lecturer and self learners.

Taqrizchilar:

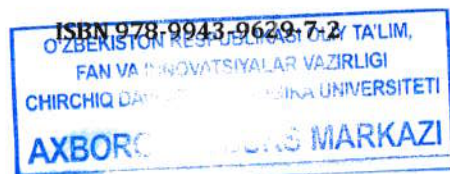
Ignatev N.A. O'zMU professori

Zaripov O.O. TDTU BAT kafedrası mudiri

Mas'ul muharrir:

Aripov M.M. O'zMU professori

Ushbu o'quv qo'llanma O'zbekiston Respublikasi Oliy va o'rta maxsus ta'lim vazirligining 2016-yil 25-avgustdagi 335-sonli buyrug'iga asosan nashr ruxsatnomasini olgan.



© Madraximov Sh. F., 2023.
© "METODIST NASHRIYOTI", 2023.

Mundarija

Kirish.....	6
1. OYD tamoyillari	7
1.1. OYD yuzaga kelishi va tamoyillari	7
1.2. OYD modelining konsepsiyalari	9
1.3. OYD tillari	11
1-bob bo'yicha nazorat uchun savollar va topshiriqlar	12
2. Sinflar	13
2.1. Sinf sintaksisi	13
2.2. Konstruktorlar va destruktorglar	21
2.3. Nusxalash konstruktori	26
2.4. this ko'rsatkichi	27
2.5. Joylashtiriladigan (inline) funksiya-a'zolar	29
2.6. Sinfning statik a'zolari	30
2.7. Sinfning konstanta obyektlari va konstanta funksiya - a'zolari	33
2.8. Sinf funksiya-a'zolariga ko'rsatkichlarni ishlatish	35
2.9. Sinf obyektlarining massivi	37
2-bob bo'yicha nazorat uchun savollar va topshiriqlar	45
3. Do'st funksiyalar va sinf do'stlari.....	47
3.1. Do'st funksiyalar.....	47
3.2. Do'st sinflar	50
3-bob bo'yicha nazorat uchun savollar va topshiriqlar	52
4. Vorislik	53
4.1. Sinflar o'rtasidagi munosabatlar	53
4.2. Oddiy vorislik	54
4.3. To'plamli vorislik	61
4.4. Virtual tayanch sinflar.....	65
4-bob bo'yicha nazorat uchun savollar va topshiriqlar	68
5. Funksiyalarni qayta yuklash	69
5.1. Funksiyalarni qayta yuklashning afzalligi	69
5.2. Sinfning funksiya-a'zolarini qayta yuklash	70
5.3. Konstruktorlarni qayta yuklash	73
5-bob bo'yicha nazorat uchun savollar va topshiriqlar	77
6. Operatorlarni qayta yuklash	78
6.1. Operatorlarni qayta yuklash tushunchasi	78

6.2. Binar operatorlarni qayta yuklash	81
6.3. Taqqoslash va mantiqiy operatorlarni qayta yuklash.....	87
6.4. Qiymat berish operatorini qayta yuklash.....	88
6.5. Unar operatorlarni qayta yuklash	91
6.6. Inkrement va dekrement operatorlarini qayta yuklash.....	92
6.7. Indeksash operatorini qayta yuklash	97
6.8. Funksiyalarni chaqirish operatorini qayta yuklash	99
6.9. Sinf a'zolariga murojaat operatorlarini qayta yuklash.....	100
6.10. new va delete operatorlarini qayta yuklash	102
6-bob bo'yicha nazorat uchun savollar va topshiriqlar.....	108
7. Polimorfizm va virtual funksiyalar	109
7.1. Vaqtli va kechiktirilgan bog'lanishlar. Dinamik polimorfizm	109
7.2. Virtual funksiyalar	110
7.2. Virtual va novirtual funksiyalar	114
7.3. Dinamik polimorfizmi qo'llash	119
7.4. Virtual destruktoral	123
7.5. Abstrakt sinflar va sof virtual funksiyalar	125
7-bob bo'yicha nazorat uchun savollar va topshiriqlar.....	128
8. Istisno holatlar.....	130
8.1. Istisno holatlarni qayta ishlash	130
8.2. Istisnolarni yuzaga keltirish	132
8.3. Istisnolarni ilib olish	135
8.4. Ichma-ich joylashgan try-catch bloklar	137
8.5. Kutilmagan istisnolar va tugatishni qayta ishlash.....	139
8-bob bo'yicha nazorat uchun savollar va topshiriqlar.....	143
9. Qoliplar	144
9.1. Umumlashgan dasturlash	144
9.2. Funksiyalar qoliplari	144
9.3. Sinflar qolipi	148
9-bob bo'yicha nazorat uchun savollar va topshiriqlar.....	150
10. Qoliplarning standart kutubxonasi (STL).....	151
10.1. STL asosiy konsepsiyasi	151
10.2. Konteynerlar	152
10.3. Iteratorlar	152
10.4. Konteynerlar umumiy xossalari	153

10.5. Algoritmlar	155
10.6. Konteynerlar adapterlari	156
10.7. Funksional obyektlar	159
10.8. Assotsiativ konteynerlardan foydalanish.....	160
10-bob bo'yicha nazorat uchun savollar va topshiriqlar.....	163
Glossariy.....	165
Foydalanilgan adabiyotlar.....	169

Kirish

Hisoblash texnikasining rivojlanishi va yechilayotgan masalalarni tobora murakkablashuvi dasturlashning turli modellari (paradigmalarini) yuzaga kelishiga sabab bo'lmoqda. Obyektga yo'naltirilgan dasturlash (OYD) - bu dasturlashga yangi bir yondashuvi bo'lib, protsedurali dasturlashda mavjud bo'lgan muammolarni bartaraf qilish maqsadida yuzaga kelgan. Birinchilar qatorida ishlab chiqilgan kompilyatorlarda (masalan, **FORTRAN** tili uchun) dasturlashning funksiyalardan foydalanishga asoslangan protsedura modelini qo'llab quvvatlagan. Bu model yordamida dastur tuzuvchi bir necha ming qatorli dasturlarni tuzishi mumkin edi. Rivojlanishning keyingi bosqichida dasturlarning tuzilmali modeli paydo bo'ldi va u **ALGOL**, **Pascal** va **C** tillari kompilyatorlarida o'z aksini topdi. Tuzilmali dasturlashning mohiyati - dasturni o'zaro bog'langan protseduralar (blokklar) va ular qayta ishlaydigan berilganlarning majmuasi deb qarashdan iborat. Ushbu model dastur blokklari keng qo'llashga, **goto** operatoridan imkon qadar kam foydalanishga tayangan va unda dastur tuzuvchi o'n mingdan ortiq qatorlarga ega monolit dasturlarni yarata olgan. Yaratilgan dasturlarni sozlash va nazorat qilish protsedurali modelga nisbatan oson kechgan.

Qo'llanmada **C++** tilida OYD amalga oshirishda asosiy vositasi sinf tushunchasi, sinfning a'zolari va ularga murojaat toifalari, sinfning do'stlari, sinflar vorisligi, polimorfizmni amalga oshirish, statik va dinamik polimorfizmlar, operatorlarni qayta yuklash, istino holatni qayta ishlash, qoliplar va standart qoliplar kutubxonasi mavzularining nazariyasi va amaliy masalalari qaratiladi.

1. OYD tamoyillari

1.1. OYD yuzaga kelishi va tamoyillari

Strukturali dasturlash asosida yaratilgan ilovalarning bajarilish oqimi sodda va chiziqli. Ilova xotiraga yuklanadi, **A** nuqtadan bajarilishni boshlaydi, **B** nuqtada ishni tugatadi va xotiradan chiqariladi. Ilova yo'l-yo'lakay boshqa narsalardan, masalan, berilganlar saqlagichidagi fayllardan, videokartadan foydalanishi mumkin, biroq qayta ishlashning asosiy qismi bitta joyda bajariladi. Odatda berilganlarni qayta ishlash murakkab bo'lmasdan, turli xil matematik va mantiqiy vositalardan foydalanilgan.

OYDda bunday chiziqli holat kam uchraydi. Xuddi shunday natijalarga erishilsa ham, jarayon ko'p hollarda umuman boshqacha ko'rinishda bo'ladi. OYDda asosiy e'tibor berilganlarning tuzilishi va mazmuniga, hamda bu berilganlarning boshqa berilganlar bilan o'zaro ta'siriga qaratiladi. O'z navbatida bu holat ilovani loyihalash bosqichida katta diqqat e'tiborni talab etadi va shuni hisobiga uni kengaytirish (takomillashtirish) imkonini beradi. Konkret berilganlar turlari haqida tasavvur bo'yicha aniq bir qarorga kelgandan keyin bu taqdimotni ilovaning keyingi naqllarida yoki umuman boshqa ilovalarda foydalanish mumkin bo'ladi. Bu turdagi kelishuvlar ilovalar yaratishni ancha soddalashtirish imkonini beradi. OYD, tasavvur bo'yicha kelishuv va nisbatan abstrakt berilganlarni qo'llash hisobiga dasturlashni soddalashtiradi. Texnologiyaning nomidan ko'rinib turibdiki, yuqorida qayd qilingan afzalliklarning barchasi obyektlar yordamida erishiladi.

Obyektlar - OYD ilovalarning "*qurilish blokidir*". Bunday qurilish bloki ilovaning qismini o'z ichiga oladi - hisoblash jarayonini, berilganlarning bir qismini yoki qandaydir yanada abstrakt bo'lgan obyektini.

Obyektlarni o'z ichida a'zo sifatida o'zgaruvchilar va funksiyalarni olgan tuzilma deb qarash mumkin. O'zgaruvchilar qiymatlari obyektida saqlanuvchi berilganlarni, funksiyalar esa obyektning amal imkoniyatlariga murojaat qilishni ta'minlaydi.

Obyekt turi uchun OYDda maxsus nom – class (*sinf*) mavjuddir. Sinfni aniqlash obyektlarni, ya'ni sinf nusxalarini yaratish imkoniyatini beradi.

Obyekt bilan xossa va maydon tushunchalari bog'liq bo'lib ular obyektidagi berilganlarga murojaatni ta'minlaydi. Ularning qiymatlari obyektlarni bir-biridan farqlash imkonini beradi, chunki sinfning turli obyektlarining maydonlari turli qiymatlarga ega bo'lishi mumkin. Xossa va maydonlar qiymatlari obyekt holatini aniqlaydi.

Xossa va maydonlar turlarga ega, shu sababli berilganlar string, int va boshqa turlarda saqlanishi mumkin. Xossaning maydondan farqi shundaki, u berilganlarga bevosita murojaat qilish imkonini bermaydi, unung asosiy vazifasi qiymat kiritilishida mazmuniy cheklovlarni qo'yishdir. Misol uchun shaxsning yoshi int turida bo'ladigan bo'lsa, uning qiymatlari int turi chegarasida bo'lishi mumkin. Yoshni manfiy qiymatini kiritish o'rinsizdir va qiymat kiritishda bu nazoratni xossa o'z zimmasiga oladi. Xossadan foydalanish orqali yosh qiymatlari diapazonini 1 dan 120 gacha qilib belgilash mumkin.

Metod termini bilan obyekt tomonidan taqdim etiluvchi funksiya tushuniladi. Metodlar obyektning funksional imkoniyatlariga murojaat qilishda qo'llaniladi. Xuddi xossa va maydonlar kabi metodlar ham zarur bo'lganda tashqi kodlardan murojaatlarga cheklovlar qo'yish uchun ochiq yoki yopiq bo'lishi mumkin.

OYD quyidagi asosiy tamoyillarga taynadi:

1. Ustida amal bajariladiganlarning barchasi obyektlar hisoblanadi;
2. Hisoblash obyektlar o'rtasida o'zaro ta'sir (berilganlarni almashish) yo'li bilan amalga oshiriladi va unda bir obyekt ikkinchisi tomonidan qandaydir ishni bajarishni talab qiladi. Obyektlar xabarlar vositasida bir-biri bilan o'zaro ta'sir qilishadi. *Xabarlar*- bu zarur bo'lishi mumkin bo'lgan argumentlar bilan to'ldirilgan biror ishni bajarilishga so'rovdir.

3. Har bir obyekt, uni tashkil etuvchilari uchun ajratilgan mustaqil xotiraga ega.

4. Har bir obyekt, uning umumiy xossalarini ifodalovchi sinfning vakilidir.

5. Sinfda obyektning hatti-harakati (funktional imkoniyati) beriladi. Shuning hisobiga sinfning nusxalari bo'lgan obyektlar ustida bir xil amallar to'plamini bajarish mumkin bo'ladi.

6. Sinflar vorislik shajarasi deb nomlanuvchi yagona daraxt ko'rinishidagi tuzilmani tashkil qiladi. Aniq bir sinf nusxalari bilan bog'liq xotira va amallar shajara daraxting quyi qismida joylashgan ixtiyoriy voris sinfga ham tegishli bo'ladi.

1.2. OYD modelining konsepsiyalari

OYD modeli bir nechta tayanch konsepsiyalarga asoslanadi.

Berilganlarni abstraktlash – berilganlarni yangi turini yaratish imkoniyati bo'lib, bu turlar bilan xuddi berilganlarning tayanch turlari bilan ishlagandek ishlash mumkin. Odatda yangi turlarni berilganlarning *abstrakt turi* deyiladi, garchi ularni soddaroq qilib *“foydalanuvchi tomonidan aniqlangan tur”* deb atash mumkin.

Inkapsulyatsiya – bu berilganlar va ularni qayta ishlovchi kodni birlashtirish mexanizmidir. Inkapsulyatsiya berilganlar va kodni tashqi ta'sirdan saqlash imkonini beradi.

Yuqoridagi ikkita konsepsiyani amalga oshirish uchun C++ tilida sinflar ishlatiladi. Sinf termini bilan obyektlar turi aniqlanadi. Sinfning har bir vakili (nusxasi) *obyekt* deb nomlanadi. Har bir obyekt o'zining alohida holatiga ega bo'ladi. Obyekt holati undagi *berilgan-a'zolarining* ayni paytdagi qiymati bilan aniqlanadi. Sinf vazifasi uning *funksiya-a'zolarining* sinf obyektlari ustidabajaradigan amallar imkoniyati bilan aniqlanadi.

Berilgan sinf obyektini yaratish *konstruktor* deb nomlanuvchi maxsus funksiya-a'zo tomonidan, o'chirish esa *destruktor* deb nomlanuvchi maxsus funksiya-a'zo orqali amalga oshiriladi.

Sinf ichki berilganlariga murojaatni cheklab qo'yishi mumkin. Cheklovlar berilganlarni *ochiq* (public), *yopiq* (private) va *himoyalangan* (protected) deb aniqlashlar bilan tayinlanadi.

Sinf, shu turdagi obyektning *“tashqi dunyo”* bilan o'zaro bog'lanishi uchun qat'iy muloqat shartlarini aniqlaydi. Yopiq berilganlarga yoki kodga faqat shu obyekt ichida murojaat qilish mumkin. Boshqa tomondan, ochiq berilganlarga va kodlarga, garchi ular obyekt ichida aniqlangan

bo'lsa ham, dasturning ixtiyoriy joyidan murojaat qilish mumkin va ular obyektni tashqi olam bilan muloqatni yaratishga xizmat qiladi. Yaratilgan obyektlarni, ularning funksiya-a'zolariga murojaat orqali amalga oshiriluvchi *xabarlar* (yoki *so'rovlar*) yordamida boshqarish mumkin. Keyinchalik Windows xabarlari bilan adashtirmaslik uchun "*so'rov*" termini ishlatiladi.

Vorislik – bu shunday jarayonki, unda bir obyekt boshqasining xossalarini o'zlashtirishi mumkin bo'ladi. Vorislik orqali mavjud sinflar asosida hosilaviy sinflarni qurish mumkin bo'ladi. Hosilaviy sinf (*avlod sinf*) o'zining "*ona*" sinfidan (*ajdod sinfdan*) berilganlar va funksiyalarni vorislik bo'yicha oladi, hamda ular qatoriga faqat o'ziga xos bo'lgan qirralarni amalga oshirishga imkon beruvchi berilgan va funksiyalarni qo'shadi. Ajdod sinfdagi himoyalangan berilgan-a'zolariga va funksiya-a'zolariga ajdod sinfda murojaat qilish mumkin bo'ladi. Bundan tashqari, hosilaviy sinfda ona sinf funksiyalari qayta aniqlanishi mumkin. Demak, vorislik asosida bir-biri bilan "*ona-bola*" munosabatidagi sinflar shajarasini yaratish mumkin. *Tayanch sinf* termini sinflar shajarasidagi ona sinf sinonimi sifatida ishlatiladi. Agar obyekt o'z atributlarini (berilgan-a'zolar va funksiya-a'zolar) faqat bitta ona sinfdan vorislik bilan olsa, *yakka* (yoki *oddiy*) *vorislik* deyiladi. Agar obyekt o'z atributlarini bir nechta ona sinflardan olsa, *to'plamli vorislik* deyiladi.

Polimorfizm – bu kodning, bajarilish paytidan yuzaga keladigan holatga bog'liq ravishda o'zini turlicha amal qilish xususiyatidir. Polimorfizm faqat obyektlar xususiyati bo'lmasdan, balki funksiya-a'zolar xususiyatidir va ular xususan, bitta nomdagi funksiya-a'zo vositasida har xil turdagi argumentlarga ega, bajaridagan amali unga uzatiladigan argumentlar turiga bog'liq bo'lgan funksiyalar turkumidan foydalanish imkoniyatida namoyon bo'ladi. Bu holatga *funksiyalarni qayta yuklash* deyiladi. Polimorfizm amallarga ham qo'llanishi mumkin, ya'ni amal mazmuni (natijasi) operand (berilgan) turiga bog'liq bo'ladi. Polimorfizمنى bunday turiga *amallarni qayta yuklash* deyiladi.

Polimorfizm yana bir ta'rifi quyidagicha: *polimorfizm* – bu tayanch sinfga ko'rsatkichlarning (murojaatlarning) ularni virtual funksiyalarni chaqirishdagi turli shakl (qiymatlam) qabul qilish imkoniyatidir. C++ tilining bunday imkoniyati *kechiktirilgan bog'lanish* natijasidir.

Kechiktirilgan bog'lanishda chaqiriladigan funksiya-a'zolarining xotiradagi adreslari dastur bajarilishi jarayonida dinamik ravishda aniqlanadi. An'anaviy dasturlash tillarida esa bu adreslar statik bo'lib, ular kompilyasiya paytida aniqlanadi (*oldindan bog'lanish*). Kechiktirilgan bog'lanish faqat virtual funksiyalar uchun o'rinni.

Yuqorida keltirilgan tushunchalarga tayangan holda OYDning boshqacha ta'rifini keltiramiz. OYD – bu dasturlash metodologiyasi bo'lib, unda dasturga obyektlar majmuasi sifatida qaraladi. Har bir obyekt aniq bir sinfning amalga oshirilishidir, sinflar esa vorislik tamoyiliga ko'ra shajara hosil qiladi.

Bu ta'rifdagi uchta asosiy qismni ko'rsatish mumkin: birinchidan, OYD element sifatida obyekt konstruksiyalari ishlatadi; ikkinchidan, har bir obyekt qandaydir aniq bir sinfni amalga oshirilishidir; uchinchidan, sinflar shajaraviy tarzda tashkil etilgan. Shu uchta talab amalga oshirilgandan keyingina dasturni OYD deyiladi.

1.3. OYD tillari

Ko'pgina zamonaviy dasturlash tillari OYDni osonlashtirish uchun maxsus yaratilgan. Biroq, OYD usulini obyektga yo'naltirilmagan tillarda ham amalga oshirish mumkin va aksincha, OYD tillarida yozilgan har qanday kod avtomatik ravisha obyektga yo'naltirilgan bo'lib qolmaydi.

Odatda OYD tillari quyidagi elementlardan tashkil topadi:

1. Tarkibida maydonlari (berilganlar-sinf a'zolari) va metodlari (funksiyalar - sinf a'zolari) sinflarni e'lon qilish;
2. Sinfni kengaytirish mexanizmi (vorislik) - mavjud ajdod sinfdan uning xususiyatlarini o'z tarkibiga avtomatik ravishda oluvchi avlod sinfini yaratish. Aksariyat OYD tillarida sodda (bittalik) vorislik qo'llab quvvatlanadi.
3. Bitta o'zgaruvchiga turli sinflarning nusxalarini qiymat sifatida berish imkoniyatini beruvchi polimorf o'zgaruvchilar va funksiyalar (metodlar) yaratish imkoniyati;
4. Virtual metodlardan foydalanish hisobiga sinf nusxalarining polimorfliğini ta'minlash. Ayrim OYD tillarida sinfning barcha metodlari virtual bo'ladi.

Hozirda OYD tillari sifatida ommalashgan Simula, C++, Visual Basic, Object pascal, Java, C#, PHP dasturlash tillarini keltirish mumkin.

1-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Obyektlar nima?
2. OYD asosiy tamoyillari sanab o'ting.
3. OYD modelining tayanch konsepsiyalari nimalardan iborat?
4. C++tilida OYD konsepsiyalarini amalga oshirish vositasi nima?
5. OYD tillarining umumiy va o'ziga xosliklarini ko'rsating.

2. Sinflar

2.1. Sinf sintaksisi

Sinf tushunchasi C++ tilidagi eng muhim tushunchalardan biridir. Sinf sintaksisi tuzilma (struct) sintaksisiga o'xshashdir va uning ko'rinishi quyidagicha:

```
class <sinf nomi>
{
    // sinfning yopiq berilgan-a'zolari va funksiya-a'zolari
public:
    // sinfning ochiq berilgan-a'zolari va funksiya-a'zolari
}
```

Odatda sinf tavsifida <obyektlar ro'yxati> qismi shart emas. Sinf obyektlari keyinchalik, zarurat bo'yicha e'lon qilinishi mumkin. Garchi, <sinf nomi> qismi ham majburiy bo'lmasada, uning bo'lgani ma'qul. Chunki <sinf nomi> berilganlarning yangi turining nomi bo'lib, uning yordamida shu sinf obyektlari aniqlanadi.

Sinf ichida e'lon qilingan funksiya va berilganlar shu sinf a'zolari hisobalanadi. Sinf ichida e'lon qilingan o'zgaruvchilar *berilgan-a'zolar*, sinf ichida e'lon qilingan funksiyalar *funksiya-a'zolar* deyiladi. Kelishuv bo'yicha sinf ichidagi barcha funksiya va o'zgaruvchilar yopiq hisoblanadi, ya'ni ularni faqat shu sinf a'zolari ishlatishi mumkin. Sinfning ochiq a'zolarini e'lon qilish uchun *public* kalit so'zi va ':' belgisidan foydalaniladi. Sinf e'lonidagi *public* so'zidan keyin e'lon qilingan funksiyalar va o'zgaruvchilarga sinfning boshqa a'zolari va dasturning shu sinf ishlatilgan ixtiyoriy joyidan murojaat qilish mumkin bo'ladi.

Sinf e'loniga misol:

```
class Sinf
{ int a; // Sinfning yopiq elementi
public:
    int qiymat_a();
    void a_qiymati(int son);
```

```
}
```

Garchi `int qiymat_a()` va `void a_qiymat(int son)` funksiyalari `Sinf` sinfi ichida e'lon qilingan bo'lsa ham, ular hali aniqlangani yo'q. Funksiyani aniqlashda "*ko'rish sohasiga ruxsat berish*"("::") amalidan foydalaniladi. Funksiya-a'zoni aniqlashning umumiy shakli quyidagicha:

```
<funksiya turi><sinf nomi>::<funksiya nomi>
(<parametrlar ro'yxati>)
```

```
{
// funksiya tanasi
}
```

Yuqorida e'lon qilingan `Sinf` sinfning `int qiymat_a()` va `void a_qiymat(int son)` funksiya-a'zolari aniqlashga misol keltirilgan:

```
int Sinf::qiymat_a(){return a;}
void Sinf::a_qiymat(int son){a=son;}
```

Faqat `Sinf` sinfini e'lon qilish bilan shu `sinf` turidagi obyektlari yuzaga kelmaydi. `Sinf` obyektlarini yuzaga keltirish uchun `sinf` nomini berilganlar turi spetsifikatori sifatida ishlatish zarur bo'ladi. Masalan,

```
Sinf obj1,obj2;
```

`Sinf` obyekti yaratilgandan keyin '.' yordamida `sinf`ning ochiq a'zolariga murojaat qilish mumkin bo'ladi. Misol uchun

```
obj1.a_qiymat(20);
obj2.a_qiymat(50);
```

murojaatlar orqali `obj1` va `obj2` obyektlarining `a` o'zgaruvchilariga qiymatlar beriladi. Har bir obyekt `sinf`da e'lon qilingan o'zgaruvchilarning o'z nusxalariga ega bo'ladi. Shu sababli, `obj1` obyektidagi `a` o'zgaruvchi `obj2` obyektidagi `a` o'zgaruvchidan farq qiladi.

`Sinf` elementlariga murojaatni ko'rsatkichlar orqali ham amalga oshirish mumkin. Quyidagi misol buni namoyon etadi.

```
class Nuqta
{
public:
int x,y;
void Koord_Qiymat_Berish(int _x,int _y);
```

```
};
```

```
void Nuqta::Koord_Qiymat_Berish(int _x,int _y){x=_x;y=_y;}
```

```
int main()
```

```
{
Nuqta x0y;
Nuqta * Koord_kursatgich=&x0y;
//...
x0y.x=0;
Koord_kursatgich->y=0;
Koord_kursatgich->Koord_Qiymat_Berish(10,15);
//...
return 0;
}
```

Shuni qayd qilish kerakki, `sinf` obyektlariga '.' va "-->" orqali murojaat qilishning kompilyatsiya nuqtai-nazaridan hech bir farqi yo'q. Kompilyator '.' bilan murojaatni "-->" bilan almashtiradi. Masalan,

```
obj1.a_qiymat(20);
ko'rsatmasi kompilyator tomonidan
(&obj1)->a_qiymat(20);
```

ko'rinishidagi ko'rsatma bilan almashtiriladi.

C++ tili `sinf`ning berilgan-a'zolariga ma'lum bir cheklanishlar qo'yadi:

- berilgan-a'zolar `auto`,`extern` yoki `register` modifikatorlari bilan aniqlanishi mumkin emas;
- `sinf`ning berilgan-a'zolari shu `sinf` turidagi obyekt bo'lishi mumkin emas, lekin ular shu `sinf`ga ko'rsatkich yoki adres olish amali (&) bo'lishi, boshqa `sinf` obyekti bo'lishi mumkin.

Ma'lum bir mazmunga ega `sinf`ga misol tariqasida stek tuzilmasi bilan amal qiladigan `sinf`ni yaratishni ko'raylik.

```
#include <iostream>
using namespace std;
#define ULCHAM 100
```

```

class STEK
{
    int stk[ULCHAM];
    int stk_uchi;
public:
    void Boshlash();
    void Joylash(int son);
    bool Olish(int & son);
};
void STEK::Boshlash()
{
    stk_uchi=0;
}
void STEK::Joylash(int son)
{
    if (stk_uchi==ULCHAM)
    {
        cout<<"Stek to'ldi!\n";
        return;
    }
    stk[stk_uchi]=son;
    stk_uchi++;
}
bool STEK::Olish(int & son)
{
    if (stk_uchi==0)
    {
        cout<<"Stek bo'sh!\n";
        return false;
    }
    stk_uchi--;
    son=stk[stk_uchi];

```

```

return true;
}
int main()
{
    STEK stek1,stek2;
    int son;
    stek1.Boshlash();
    stek2.Boshlash();
    stek1.Joylash(1);
    stek2.Joylash(2);
    stek1.Joylash(3);
    stek2.Joylash(4);
    if(stek1.Olish(son))cout<<son<<" ";
    if(stek1.Olish(son))cout<<son<<" ";
    if(stek2.Olish(son))cout<<son<<" ";
    if(stek2.Olish(son))cout<<son<<" ";
    if(stek2.Olish(son))cout<<son<<" ";
    return 0;
}

```

Dastur ishlashi natijasida ekranga quyidagilar chop etiladi:

3 1 4 2 Stek bo'sh!

Sinf, uning a'zolari ishlatilishidan oldin e'lon qilingan bo'lishi kerak. Biroq, ayrim hollarda sinfda hali e'lon qilinmagan sinfga ko'rsatkich yoki adres olish amalini (&) e'lon qilishga zarurat bo'lishi mumkin. Bu holda sinfning to'liq bo'lmagan e'lonidan foydalanishga to'g'ri keladi. Sinfning to'liq bo'lmagan e'loni quyidagi ko'rinishga ega:

```
class <sinf nomi>;
```

Misol ko'raylik.

```
class Sinf2; // Sinfning to'liqmas e'loni
```

```
class Sinf1
```

```
{
```

```
    int x;
```

```
Sinf2 * sinf2; // Sinf2 sinfiga ko'rsatkich
```

```

public:
    Sinf1(int _x) {x=_x;}
};
int main()
{
    //...
    return 0;
}
class Sinf2      // Sinf2 sinfining to'liq e'loni
{
    int a;
public:
    Sinf2();
};

```

Shuni qayd etish kerakki, sinf e'loni tuzilma (**struct**) e'loniga o'xshash, farqli ravishda:

- sinf e'lonida **public**, **protected** yoki **private** murojaat modifikatorlari ishlatiladi;
- **struct** kalit so'zi o'rnida **class** yoki **union** kalit so'zlari ishlatilishi mumkin;
- odatda sinf tarkibida berilganlardan tashqari funksiya-a'zolar kiradi;
- sinf konstruktor yoki destruktur deb nomlanuvchi maxsus funksiya-a'zolariga ega bo'ladi.

Quyida **struct** va **union** kalit so'zlari bilan aniqlangan sinflarga misol keltirilgan.

```

struct Nuqta
{
private:
    int x; int y;
public:
    int Olish_X();
    int Olish_Y();
};

```

```

void Qiymat_Berish_X(int _x);
void Qiymat_Berish_Y(int _y);
};
union Bit
{
    Bit(unsigned int n);
    void Bit_Chop_Qilish();
    unsigned int num;
    unsigned char c[sizeof(unsigned int)];
};

```

Murojaat spetsifikatori, undan keyin joylashgan barcha sinf elementlariga qo'llaniladi, toki boshqa spetsifikator uchramaguncha yoki sinf e'loni tugamaguncha.

Quyida sinfga murojaat spetsifikatorlari tavsiflari keltirilgan:

private – berilgan-a'zolariga va funksiya-a'zolariga faqat shu sinf funksiya-a'zolari murojaat qilishi (ishlatishi) mumkin;

protected – berilgan-a'zolariga va funksiya-a'zolariga faqat shu sinf va shu sinfdan hosil bo'lgan sinflar funksiya-a'zolari murojaat qilishi (ishlatishi) mumkin;

public – berilgan-a'zolariga va funksiya-a'zolariga faqat shu sinf funksiya-a'zolari va sinf obyekti mavjud bo'lgan dastur funksiyalari murojaat qilishi (ishlatishi) mumkin.

C++ tilida tuzilma va birlashmalar sinf turlari deb qaraladi. Tuzilma va sinflar bir-biriga o'xshash, faqat kelishuv bo'yicha murojaat spetsifikatori bilan farq qiladi: tuzilmada kelishuv bo'yicha barcha elementlar **public** murojaat spetsifikatoriga ega bo'lsa, sinfda ular **private** murojaat spetsifikatorida bo'ladi. Birlashmada ham, xuddi tuzilmadek kelishuv bo'yicha elementlar **public** murojaat spetsifikatorida bo'ladi.

Tuzilma va birlashmalar o'z konstruktor va destrukturlariga ega bo'lishi mumkin. Shu bilan birgalikda birlashmalarni sinf sifatida ishlatishga ma'lum bir cheklovlar mavjud. Birinchidan, ular birorta sinf vorisi bo'lishi mumkin emas, o'zlari ham boshqa sinflar uchun tayanch sinf bo'la olmaydi. Ular **static** atributli a'zolariga, hamda ular o'z tarkibida konstruktor va destrukturi bo'lgan obyektlarga ega bo'lishi mumkin emas.

Birlashmani sinf sifatida ishlatilishiga misol ko'raylik.

```
#include <iostream.h>
union Bit
{
    Bit(unsigned short int n);
    void Bit_Chop_Qilish();
    unsigned short int num;
    unsigned char c[sizeof(unsigned short int)];
};
Bit::Bit(unsigned short int n){num=n;};
void Bit::Bit_Chop_Qilish( )
{
    for (int j=0;j<sizeof(unsigned short int);j++)
    {
        cout<<j<<" - baytning ikkilik ko'rinishi"<<" ";
        for (int i=128;i>=1){if (i&c[j])cout<<"1"; else cout<<"0";}
        cout<<endl;
    }
}
int main()
{
    Bit bit(2000);
    bit.Bit_Chop_Qilish();
    return 0;
}
```

Bu misolda obyektga berilgan ishorasiz butun qiymatning ikkilik ko'rinishi chop etiladi. Berilgan ishorasiz butun 2000 soni uchun quyidagi satr paydo bo'ladi:

0- baytning ikkilik ko'rinishi: 11010000

1- baytning ikkilik ko'rinishi: 00000111

2.2. Konstruktorlar va destruktorelar

Odatda sinf obyektini yaratishda uni initsializatsiyalash kerak. Bu maqsadda C++ tilida *konstruktor* deb nomlanuvchi maxsus funksiya-a'zo aniqlangan. Sinf konstruktori har safar sinf objekti yaratilishi paytida avtomatik ravishda chaqiriladi. Konstruktor nomi o'zi a'zo bo'lgan sinf nomi bilan ustma-ust tushadi va qaytaruvchi qiymatga ega bo'lmaydi. Masalan,

```
#include <iostream.h>
class Sinf
{
    int var;
public:
    Sinf(); // Konstruktor
    void Chop_etish_var();
};
Sinf::Sinf()
{
    cout<<"Konstruktor ishladi!\n";
    var=0;
}
Sinf::Chop_etish_var(){cout<<var;}
int main()
{
    Sinf ob;
    ob.Chop_Etish_var();
    //...
    return 0;}

```

Bu misolda Sinf konstruktori ekranga xabar chiqaradi va yopiq var o'zgaruvchini initsializatsiya qiladi (boshlang'ich qiymat beradi).

Shuni qayd etish kerakki, dastur tuzuvchisi oshkor ravishda sinf konstruktorini chaqiradigan til ko'rsatmasini yozmasligi kerak. Barcha zarur ishni kompilyator amalga oshiradi. Yuqorida qayd qilingandek konstruktor u tegishli bo'lgan sinf objekti yaratilayotgan paytda

avtomatik ravishda chaqiriladi. O'z navbatida obyekt uni e'lon qiluvchi operator bajarilishida yaratiladi. Shuning uchun ham C++ tilida o'zgaruvchi e'loni bajariluvchi operator hisoblanadi.

Global obyektlar uchun konstruktor dastur bajarilishi boshlanganda chaqiriladi. Lokal obyektlar uchun konstruktor o'zgaruvchi e'lonining har bir bajarilishida chaqiriladi.

Konstruktorga nisbatan teskari amal bajaradigan funksiya-a'zolarga *destruktorlar* deyiladi. Bu funksiya-a'zo obyekt o'chirilishida (yo'qotilishida) avtomatik ravishda chaqiriladi. Odatda destruktor obyekt tomonidan egallangan dinamik xotirani bo'shatish uchun xizmat qiladi. Uning nomi sinf nomi bilan mos tushadi, faqat oldiga '~' belgisi qo'yiladi.

Quyida destruktor aniqlangan sinfga misol keltirilgan.

```
#include <iostream.h>
class Sinf;
{
    int var;
public:
    Sinf(); // Konstruktor
    ~Sinf(); // Destruktor
    void Chop_etish_var();
    Sinf::Sinf(){cout<<"Konstruktor ishladi!\n"; var=0;}
    Sinf::~Sinf(){cout<<"Destruktor ishladi!\n";}
};
void Sinf::Chop_etish_var(){cout<<var<<endl;}
int main()
{
    Sinf ob;
    ob.Chop_Etish_var();
    //...
    return 0;}

```

Destruktor obyekt o'chirilishida chaqiriladi. Global obyektlar dastur tugashida o'chiriladi. Lokal obyektlar – ularning ko'rish sohasidan chiqishda o'chiriladi.

Shuni alohida qayd etish kerakki, konstruktor va destruktorlarga ko'rsatkichlar hosil qilish mumkin emas.

Agar sinf o'zgaruvchilarini initsializatsiya qilish zarur bo'lsa, parametrlı konstruktor ishlatildi. Yuqorida keltirilgan misolga o'zgartirish kiritamiz.

```
#include <iostream.h>
class Sinf
{
    int a, b;
public:
    Sinf(int x,int y);
    ~Sinf();
    void Chop_etish_var();
};
Sinf::Sinf(int x,int y)
{
    cout<<"Konstruktor ishladi!\n";
    a=x; b=y;
}
Sinf::~Sinf(){cout<<"Destruktor ishladi!\n";}
void Sinf::Chop_etish_var(){cout<<a<<b<<endl;}
int main()
{
    Sinf ob(5,10);
    ob.Chop_Etish_var();
    //...
    return 0;
}

```

Bu erda **ob** obyekti e'lonida konstruktorga uzatilgan qiymatlar sinf tarkibidagi **a** va **b** yopiq o'zgaruvchilarini initsializatsiya qilishda ishlatiladi.

Parametrli konstruktorga qiymat uzatishda «**Sinf ob(5,10);**» va «**Sinf(5,10);**» ko'rsatmalar bir xil mazmunga ega. Aksariyat hollarda ikkinchi, qisqa shakl ishlatiladi.

Konstruktordan farqli ravishda destruktur parametrga ega bo'lishi mumkin emas, chunki o'chirilayotgan obyekt o'zgaruvchilariga qiymat berish ma'noga ega emas.

Konstruktur uchun aniqlangan bir nechta qoidalarini keltiramiz:

-konstruktur funksiyasi uchun qaytariluvchi qiymat turi ko'rsatilmaydi;

-konstruktur qiymat qaytarmaydi;

-konstruktur vorislik bilan o'tmaydi;

-konstruktur **const**, **volatile**, **static** yoki **virtual** modifikatorlari bilan e'lon qilinmaydi.

Agar sinf aniqlanishida konstruktur e'lon qilinmasa, kompilyator o'zi kelishuv bo'yicha parametrsiz konstruktorni yaratadi.

Destruktor uchun quyidagi qoidalar aniqlangan:

- destruktur funksiyasi parametrlarga bo'lishi mumkin emas;

- destrukturqiymat qaytarmaydi;

- destruktur vorislik bilan o'tmaydi;

- sinf bittadan ortiq destruktorga ega bo'lishi mumkin emas;

- destruktur **const**, **volatile**, **static** yoki **virtual** modifikatorlari bilan e'lon qilinmaydi.

Agar sinfda destruktur e'lon qilinmasa, kompilyator o'zi kelishuv bo'yicha destruktorni hosil qiladi.

Odatda sinf berilgan-a'zolari konstruktur tanasida initsializatsiyalanadi. Lekin initsializatsiyani boshqa usul bilan – *elementlarni initsializatsiyalash ro'yxati* orqali amalga oshirish mumkin. Elementlarni initsializatsiyalash ro'yxati funksiya sarlavhasi aniqlanishidan keyin ikki nuqtadan (':') keyin joylashadi va unda vergul bilan ajratilgan holda berilgan-a'zolar va tayanch sinflar yoziladi. Har bir element uchun qavs ichida initsializatsiyada ishlatiladigan bir yoki bir nechta parametrlar

ko'rsatiladi. Quyidagi misolda elementlarni initsializatsiyalash ro'yxati orqali sinf o'zgaruvchilarga boshlang'ich qiymat berish ko'rsatilgan.

```
class Sinf
{
    int a, b;
public:
    Sinf(int x, int y);
};
Sinf::Sinf(int x,int y):a(x),b(y)
{
    cout<<"Konstruktur ishladi!\n";
}
//...
```

Keltirilgan dastur bo'lagida konstruktur bajaradigan ish oldingi misoldagi konstruktur ishi bilan bir xil mazmunga ega.

Albatta, sinf elementlarini initsializatsiya qilishning qaysi shaklini qo'llash dastur tuzuvchiga bog'liq. Biroq, shunday holatlar bo'ladiki, unda elementlarni initsializatsiyalash ro'yxatidan foydalanmaslikning iloji yo'q:

– sinfning konstanta berilganlariga va adres olish amali qo'llanilgan

o'zgaruvchilarga boshlang'ich qiymat berishda;

– sinf a'zosi boshqa sinf obyekti bo'lganda va bu obyekt konstrukturi bir yoki bir nechta parametrlarga qiymat berishni talab qilgan hollarda.

Kelishuv bo'yicha konstrukturlarni ishlatishda kolliziyadan (bir narsani ikki xil tushunishdan) qochish kerak, ya'ni sinfda bir nechta konstruktur bo'lganda kompilyator ularning qaysi birini qachon chaqirilishini aniq bilishi kerak. Quyidagi misolda bu holat bilan bog'liq xato ko'rsatilgan.

```
class S
{
public:
    S(); // Kelishuv bo'yicha konstruktur
```

```

S(int i=0); // Kelishuv bo'yicha konstruktor o'rnida ishlatilgan
konstruktor
};
int main()
{
    S ob1(10); // S::S(int) konstruktori ishlatiladi.
    S ob2; // Noto'g'ri. S::S(int) yoki S::S() konstruktorlarining qaysi
biri?
    ...
    return 0;
}

```

Bu ziddiyatni hal qilish yo'li - bu sinf e'lonidan kelishuv bo'yicha konstruktorni o'chirishdir.

2.3. Nusxalash konstruktori

Nusxalash konstruktori sinf obyektini yaratishda shu sinfning mavjud obyektlaridan berilganlarni (ularning qiymatini) nusxasini oladi. Shu sababli u sinf obyektiga konstanta obyekt (`const S&`) yoki oddiy adresini olish amali (`S&`) bo'lgan yagona parametrga ega bo'ladi. Parametrlarning birinchisini ishlatish ma'qul hisoblanadi, u konstanta obyektlarni nusxalash imkonini beradi.

Quyidagi misolda nusxalash konstruktorini ishlatish ko'rsatilgan.

```

class Nuqta
{
    int x,y;
public:
    Nuqta(const Nuqta & koor);
    Nuqta(int x0, int y0);
    void Abstissa();
    void Ordinata();
    void Uzgartirish (int delta_x, int delta_y);
};
Nuqta::Nuqta(const Nuqta & koor){ x=koor.x; y=koor.y;};

```

```

Nuqta::Nuqta(int x0, int y0){x=x0; y=y0;};
void Nuqta::Abstissa(){cout<<" x="<<x;};
void Nuqta::Ordinata(){cout<<" y="<<y;};
void Nuqta::Uzgartirish(int delta_x, int delta_y){x+=delta_x;
y+=delta_y;};
int main( )
{
    Nuqta koord1(5,10);
    Nuqta koord2=koord1;
    Nuqta koord3(koord1);
    koord1.Uzgartirish(3, -2);
    koord2.Uzgartirish(1, 2);
    cout<<"\n koord1 obyekt berilgan-a'zolari qiymati:";
    koord1.Abstissa();
    koord1.Ordinata();
    cout<<"\nkoord2 obyekt berilgan-a'zolari qiymati:";
    koord2.Abstissa();
    koord2.Ordinata();
    cout<<"\nkoord3 obyekt berilgan-a'zolari qiymati:";
    koord3.Abstissa();
    koord3.Ordinata();
    return 0;
}

```

Dastur ishlashi natijasida ekranga quyidagilar chop etiladi:

```

koord1 obyekt berilgan-a'zolari qiymati:x=8 y=8
koord2 obyekt berilgan-a'zolari qiymati:x=6 y=12
koord3 obyekt berilgan-a'zolari qiymati: x=5 y=10

```

2.4. this ko'rsatkichi

C++ tilidagi har bir obyekt kompilyator tomonidan yaratiladigan va ayni shu obyektga ko'rsatuvchi `this` deb nomlanuvchi maxsus ko'rsatkichga ega. `this` ko'rsatkichining turi "`S*`" ko'rinishda bo'lib, `S` - ushbu obyekt sinfining turidir. `this` ko'rsatkichi sinfda aniqlanganligi

sababli uning amal qilish sohasi o'zi aniqlangan sinf bo'ladi. Boshqacha aytganda, `this` ko'rsatkichini kompilyator tomonidan qo'shiluvchi sinfning yashiringan parametri deb qarash mumkin. Sinf funksiya-a'zosi chaqirilganda unga `this` ko'rsatkichi, go'yoki birinchi argument sifatida uzatiladi, ya'ni funksiya-a'zoni chaqirishning quyidagi ko'rinishi

```
Obekt1.Funksiya(arg1,arg2);
```

kompilyator tomonidan

```
Obekt1.Funksiya(&Obekt1, arg1,arg2);
```

ko'rinishida talqin qilinadi.

Funksiya chaqirilganda uning qavs ichidagi argumentlari stekka o'ngdan chapga tomonga qarab joylashtirildi. Birinchi argument (`this`) stekka eng oxirda joylashtiriladi. Funksiya-a'zo ichida obyektlar adresi `this` orqali aniqlanadi. Quyida `this` ko'rsatkichini ishlatish bilan bog'liq dastur matni keltirilgan.

```
#include <iostream.h>
#include <string.h>
class S
{
    char lsm[20];
public:
    S(char*);
    void Salom();
};
S::S(char *_lsm)
{
    strcpy(lsm, _lsm);
    Salom(); // Uchta murojaat
    this->Salom(); // o'zaro
    (*this).Salom(); // ekvivalent
}
void S::Salom()
{
    cout<<"Salom "<<lsm<<"\n";
```

```
cout<<"Salom "<<this->lsm<<"\n";
```

```
}
```

```
int main()
```

```
{
```

```
S obekt("Muqumov Rustam"); return 0;
```

```
}
```

Dastur matnidan ko'rinish turibdiki, funksiya-a'zo ichida boshqa funksiya-a'zolarga va berilgan-a'zolarga bevosita ularning nomlari bilan yoki `this` ko'rsatkichi orqali murojaat qilish mumkin. Shu sababli amaliyotda `this` ko'rsatkichidan kam foydalaniladi, u asosan funksiya qaytaruvchi qiymati sifatida (`return this;` yoki `return*this;`) va operatorlarni qayta yuklash bilan bog'liq masalalarda keng qo'llaniladi.

2.5. Joylashtiriladigan (inline) funksiya-a'zolar

Funksiyalar bo'limida `inline` funksiya haqida ma'lumot berilgan edi. Ularning boshqa funksiyalardan farqi - kompilyator dasturda bunday funksiya chaqirilgan joyga funksiya chaqirish buyrug'ini emas, balki funksiya tanasini qo'yadi (mumkin bo'lsa). `inline` funksiya afzalligi shunda ediki, dasturda oddiy funksiya chaqirishdagi argumentlarni uzatish, stekka qiymatlarni joylashtirish va qayta olish bilan bog'liq qo'shimcha amallar bajarilmaydi. Noqulay tomoni, `inline` funksiya murojatlar ko'p bo'lganda dastur hajmi oshib ketadi. Bu o'rinda `inline` funksiya qo'llanishini makroslarni ishlatishga o'xshatish mumkin. Shu sababli, juda sodda funksiya uchun `inline` funksiya sifatida ishlatiladi. Funksiyalar aniqlanishda `inline` spetsifikatorini qo'yishni kompilyatorga funksiya chaqirilgan joylarga funksiya tanasini qo'yishga talab deb qaraladi. Agar kompilyator funksiya tanasini qo'yishni amalga oshira olmasa, `inline` funksiya oddiy funksiya sifatida ishlatiladi. Kompilyator quyidagi holatlarda `inline` funksiya chaqiruv joyiga qo'ya olmaydi, agarda funksiya:

- tanasida takrorlash operatorlari ishlatilgan bo'lsa (`for`, `while`, `do-while`);

- tanasida `switch`, `goto` operatorlari bo'lsa;

- **static** o'zgaruvchilarni ishlatasa;
- rekursiv bo'lsa;
- **void** turidan farqi qaytaruvchi turga ega va tanasida **return** operatori bo'lmasa;

– tanasida assembler kodli bo'laklar mavjud bo'lsa.

Joylashtiriladigan funksiyalar sifatida nafaqat oddiy funksiyalar, balki sinfning funksiya-a'zolari ham aniqlanishi mumkin. Buning uchun sinf e'lonida funksiya-a'zoning aniqlanishini qo'yish etarli yoki sinfdan tashqaridagi funksiya aniqlanishida, funksiya sarlavhasi oldiga **inline** spetsifikatorini qo'yish zarur bo'ladi. Quyida keltirilgan misolda joylashtiriluvchi funksiya-a'zolarining ikki xil variantdagi tavsiflanishi ko'rsatilgan.

```
class Nuqta
{
    int x, y;
public:
    Qiymat_x(){return x;}
    Qiymat_y(){return y;}
    void x_Qiymati(int _x);
    void y_Qiymati(int _y);
};
inline void Nuqta::x_Qiymati(int _x){x=_x;}
inline void Nuqta::y_Qiymati(int _y){y=_y;}
```

2.6. Sinfning statik a'zolari

Sinf a'zolari **static** modifikatori bilan e'lon qilinishi mumkin. Sinf statik a'zosini sinf sohasi chegarasida murojaat qilish mumkin bo'lgan global o'zgaruvchi yoki funksiya deb qarash mumkin. Sinfning statik deb e'lon qilingan berilgan-a'zolari sinfning barcha obyektlari tomonidan birgalikda ishlatiladi, chunki bunday o'zgaruvchi yagona nusxada bo'ladi. Amalda sinfning statik berilganlari uchun xotiradan joy, hattoki sinfning birorta obyekti e'lon qilinmagan bo'lsa ham ajratiladi. Shu sababli sinf statik berilganni e'lon qilib qolmasdan, uni aniqlashi shart. Masalan:

```
class Sinf
```

```
{
```

```
public:
```

```
    Sinf();
```

```
    static int Sanagich; // Statik berilgan-a'zo e'loni
```

```
};
```

```
int Sinf::Sanagich=0; // Statik berilgan-a'zoga boshlang'ich qiymat
```

berish

Bu misolda **Sanagich** statik berilgan-a'zo **public** bo'limida e'lon qilingan va unga sinf obyekti nomini ishlatish yordamida murojaat qilish mumkin.

```
Sinf sinf1;
```

```
Sinf1.Sanagich++;
```

```
Sinf sinf2;
```

```
Sinf2->Sanagich--;
```

Statik berilgan-a'zolarga sinf nomi orqali murojaat qilgan ma'qul bo'ladi.

```
Sinf::Sanagich++;
```

Bu holat **Sanagich** statik berilgan-a'zo barcha sinf obyektlari uchun yagona ekanligini ta'kidlaydi.

Agarda statik berilganlar yopiq deb e'lon qilingan bo'lsa, ularga funksiya-a'zolar orqali murojaat qilish mumkin.

Umuman olganda, statik berilgan-a'zolarini ishlatishda quyidagi tavsiyalarni berish mumkin:

- statik berilgan-a'zolarini bir nechta sinf obyektlari tomonidan birgalikda ishlatish uchun aniqlash zarur;

- statik berilgan-a'zolarini **private**, **protected** modifikatorlar bilan e'lon qilish orqali ularga murojaatni cheklash kerak.

Sinfning statik berilgan-a'zoni ishlatishga misol.

```
class S;
```

```
{
```

```
public:
```

```
    S(){ob_soni++;}
```

```

~S(){ob_soni--;}
static int ob_soni;
private:
int x;
};
int S::ob_soni=0;
int main()
{
S * p_ob=new S[5];
cout<<"Sinfning "<<S::ob_soni<<" obyekt mavjud.\n";
delete [ ] p_ob;
return 0;
}

```

Dastur ishlash natijasida ekranga quyidagi satr chop etiladi
Sinfning 5 obyekt mavjud.

Sinfning statik funksiyalarni ishlatishning o'ziga xosligi shundaki, ular ham yagona nusxada aniqlanadi va birorta sinf obyektining "shaxsiy" funksiyasi bo'lmaydi. Shu sababli, bu funksiyalarga **this** ko'rsatkichi uzatilmaydi. Statik funksiyalarning bunday xususiyatidan Windows OC uchun dasturlashda keng foydalaniladi.

Yuqorida aytilgan fikrlardan bir nechta muhim xulosalar kelib chiqadi:

- statik funksiya-a'zolari sinfning birorta ham vakili (obyekti) mavjud bo'lmasa ham chaqirish mumkin;
- sinfning statik funksiyasi faqat sinfning statik berilganlarini qayta ishlashi mumkin va u faqat sinfning statik funksiya-a'zolarini chaqirishi mumkin;
- statik funksiya-a'zo **virtual** modifikatori bilan e'lon qilinishi mumkin emas.

Quyida keltirilgan dastur statik funksiya-a'zoni ishlatishga misol bo'ladi:

```

#include <iostream.h>
class S

```

```

{
public:
S(){sanagich++;};
~S(){sanagich--};
// ...
static int Sinf_Sanagichi(){return sanagich;}
private:
int x;
static int sanagich;
};
int S::sanagich=0;
int main()
{
S * pOb=new S[10];
cout<<"S sinfning " <<S::Sinf_Sanagichi() <<" obyekt mavjud"<<endl;
delete [ ] pOb;
return 0;
}

```

2.7. Sinfning konstanta obyektlari va konstanta funksiya - a'zolari

Sinfning funksiya-a'zolari parametrlar ro'yxatidan keyin keluvchi **const** modifikatori bilan e'lon qilinishi mumkin. Bunday funksiya sinf berilgan-a'zolari qiymatlarini o'zgartira olmaydi va sinfning konstanta bo'lmagan funksiya-a'zolarini chaqirishi mumkin emas. Konstanta funksiya-a'zosi bo'lgan sinfga misol keltiramiz.

```

class Nuqta
{
int x,y;
public:
Nuqta(int _x, int _y);
void Qiymat_Berish(int x0,int y0);

```

```
void Qiymat_Olish(int xx,int yy) const; // Konstanta funksiya-a'zo
e'loni
```

```
};
Nuqta::Nuqta(int _x,int _y){x=_x; y=_y;}
void Nuqta::Qiymat_Berish(int x0,int y0){x=x0;y=y0;}
void Nuqta::Qiymat_Olish(int &xx,int &yy) const {xx=x; yy=y;}
```

Xuddi shunday, konstanta obyektlarni yaratish mumkin. Buning uchun obyekt e'loni oldiga **const** modifikatori qo'yish yetarli. Masalan, **const Nuqta koord(3,6);**

til ko'rsatmasidagi **const** kalit so'zi kompilyatorga ushbu obyektning holati o'zgarmasligi kerakligini bildiradi. Shu sababli obyekt berilgan-a'zolari qiymatini o'zgartiradigan funksiya-a'zoni chaqirishi uchrasa, kompilyator xatolik haqida xabar beradi. Bu qoidaga konstanta funksiya-a'zolari chaqirish rioya qilmaydi, chunki o'z mazmuniga ko'ra ular berilgan-a'zolari qiymatini o'zgartira olmaydi. Yuqorida e'lon qilingan Nuqta sinfini konstanta obyekt ishlatishiga misol.

// Bu erda Nuqta sinfi e'loni va aniqlanishi yoziladi.

```
int main()
{
    Nuqta nuqta1(3,7);
    const Nuqta nuqta2(8,10); // Konstanta obyekt
    int a,b;
    nuqta1.Qiymat_Olish(a,b);
    nuqta2.Qiymat_Berish(2,3); // Xato
    nuqta2.Qiymat_Olish(a,b); // To'g'ri
    return 0;
}
```

Konstanta funksiya-a'zolarining berilgan-a'zolar bilan ishlash bilan bog'liq cheklavlarni "*aylanib o'tish*" uchun **mutable** kalit so'zi aniqlangan. Bu kalit so'z sinfning qaysi berilgan-a'zo konstanta funksiya-a'zolar tomonidan o'zgartirilishi mumkin ekanligini ko'rsatadi. Statik va konstanta berilgan-a'zolariga **mutable** kalit so'zini ishlatish mumkin emas, u berilganlar turining modifikatori sifatida ishlatiladi.

Misol.

```
#include <iostream.h>
class Sinf
{
    mutable int count;
    mutable const int * intPtr; // Amal o'rinli, garchi ko'rsatkich
```

konstanta

// butun songa ko'rsatsa ham o'zi konstanta emas.

```
public:
    int Funksiya(int i=0) const
    { count=i++;
      intPtr=&i;
      cout<<*intPtr;
      return count;
    }
};
int main()
{
    Sinf S;
    S.Funksiya();
    return 0;
}
Dastur ishlashi natijasida ekranga
1
chop etiladi.
```

2.8. Sinf funksiya-a'zolariga ko'rsatkichlarni ishlatish

Sinfning funksiya-a'zosiga ko'rsatkichni aniqlash mumkin. Uning sintaksisi quyidagicha:

<qiymat turi> (<sinf nomi>::<ko'rsatkich nomi>)(<parametrlar>);*

Misol.

```
void (S::*F_kursatkich)(int x,int y);
```

Quyida keltirilgan dasturda funksiya-a'zosiga ko'rsatkich ishlatilgan.

```
class S
{
    int i;
public:
    S(int l):i(l){};
    void FunAzo(){cout<<"Funksiya-a'zo ishladi!";}
    void FunChaqirish(void (S::*F_Kursatgich)())    {(this->*F_Kursatgich)();}
}
int main()
{
    void (S::*FKursatgich)()=&S::FunAzo;
    S obekt(100);
    obekt.FunChaqirish(FKursatgich);
    return 0;
}
```

Dasturning 1-satrida FunKursatgich ko'rsatkichiga FunAzo funksiya-a'zo adresi berilgan, 3-satrida FunChaqirish funksiya-a'zo FunKursatgich ko'rsatkich bo'yicha FunAzo funksiyasini chaqiradi va ekranga

Funksiya-a'zo ishladi!
xabari chiqadi.

Sinfning funksiya-a'zoga ko'rsatkichni ishlatishda ma'lum bir cheklovlar mavjud:

- sinfning funksiya-a'zoga ko'rsatkichni sinfning statik funksiya-a'zolariga ko'rsata olmaydi, chunki ularga **this** ko'rsatkichi uzatilmaydi;
- sinfning funksiya-a'zoga ko'rsatkichni qandaydir sinfning a'zosi bo'lmagan oddiy funksiya ko'rsatkichiga aylantirish mumkin emas.

2.9. Sinf obyektlarining massivi

Sinf obyektlaridan massivlar qurish mumkin. Obyektlar massivini e'lon qilish sintaksisi oddiy o'zgaruvchilar massivini e'loniga o'xshash. Masalan, 10 ta S sinf obyektidan iborat massiv e'loni quyidagicha bo'ladi:

```
S obekt_massiv[10];
```

Kompilyator tomonidan bu massivni yaratishda sinf obyektlarining kelishuv bo'yicha konstruktoridan foydalanishi shart. Shu sababli, kompilyator o'zi sinf uchun konstruktor yaratishiga tayanmasdan, sinf obyektlarining massivini e'lon qilishda sinf tarkibiga kelishuv bo'yicha konstruktor kiritish zarur. Agar foydalanuvchi tomonidan konstruktor aniqlanadigan bo'lsa, sinf obyektlari massivining har bir elementining boshlang'ich qiymati oshkor ravishda ko'rsatilishi kerak.

Obyektlar massiviga murojaat xuddi ixtiyoriy turdagi elementlardan iborat massivga murojaatdek bo'ladi:

```
#include <iostream.h>
class Sinf
{
    int a;
public:
    Sinf(int n){a=n;}
    int a_Qiymati(){return a;}
};
```

```
main()
{
    Sinf ob_Sinf[5]={12,23,34,56,67};
    for(int i=0;i<4;i++)
    {
```

```
        cout<<ob_Sinf[i].a_Qiymati()<<' ';    // Massiv elementlariga
murojaat
    }
    cout<<"\n";
    return 0;
}
```

```
}
```

Dastur ekranga Sinf sinfi har obyekt berilgan-a'zosining initsializatsiyada olgan qiymatini chop etadi. Initsializatsiyaning bu usulidan konstruktor bitta parametr ega bo'lgan holda ishlatish mumkin. Agar konstruktor bittadan ortiq parametr ega bo'lsa, initsializatsiyaning to'liq shaklidan foydalanishga to'g'ri keladi.

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

```
class Koord
```

```
{
```

```
int x,y;
```

```
public:
```

```
Koord(int _x,int _y){x=_x; y=_y;}
```

```
int x_Qiymati(){return x;}
```

```
int y_Qiymati(){return y;}
```

```
};
```

```
main()
```

```
{
```

```
Koord koord_mas[4][2]={Koord(1,2);Koord(3,4); Koord(5,6);
```

```
Koord(7,8);
```

```
Koord(9,10);Koord(11,12); Koord(13,14); Koord(15,16); }
```

```
int i,j;
```

```
for(i=0;i<4;i++)
```

```
for(j=0;j<2;j++)
```

```
{ cout<<koord_mas[i][j].x_Qiymati()<<' ';
```

```
cout<<koord_mas[i][j].y_Qiymati()<<'\n'; }
```

```
return 0;
```

```
}
```

Ushbu dastur sinf obyektlar massivining har bir elementining x,y berilgan-a'zolarining qiymatini chop etadi.

Sinf obyektlari massivga ko'rsatkich orqali murojaat qilish mumkin. Bunda ko'rsatkichlar arifmetikasi ham o'rinli bo'ladi. Masalan, ko'rsatkich inkrementi massivning keyingi elementiga ko'rsatishiga olib kelsa, ko'rsatkich dekrementi - ko'rsatkich o'zi ko'rsatib turgan massiv

obyektidan oldingi obyektga ko'rsatadi. Yuqorida keltirilgan dasturning ko'rsatkich ishlatilgan variantini ko'raylik.

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

```
class Koord
```

```
{
```

```
int x,y;
```

```
public:
```

```
Koord(int _x,int _y){x=_x; y=_y;}
```

```
int x_Qiymati(){return x;}
```

```
int y_Qiymati(){return y;}
```

```
};
```

```
main()
```

```
{
```

```
Koord koord_mas[4]={Koord(1,2);Koord(3,4);Koord(5,6); Koord(7,8);}
```

```
int i;
```

```
Koord * kursat_ob;
```

```
kursat_ob=koord_mas;
```

```
for(i=0;i<4;i++)
```

```
{ cout<<kursat_ob->x_Qiymati()<<' ';
```

```
cout<<kursat_ob->y_Qiymati()<<'\n';
```

```
kursat_ob++;
```

```
}
```

```
return 0; }
```

Dastur ishlashi natijasida ekranga

```
1 2
```

```
3 4
```

```
5 6
```

```
7 8
```

qiymatlari chop etiladi.

Sinf obyektlar massivini dinamik xotirada hosil qilish mumkin:

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

```
class Koord
```

```
{
```

```

int x,y;
public:
Koord(int _x,int _y){x=_x;y=_y;}
Koord(){x=0;y=0;}
int x_Qiymati(){return x;}
int y_Qiymati(){return y;}
};
main()
{
int i;
Koord * kursat_ob;
kursat_ob=new Koord[4];
for(i=0;i<4;i++)
{
cout<<kursat_ob->x_Qiymati()<<' ';
cout<<kursat_ob->y_Qiymati()<<'\n';
kursat_ob++;
}
delete [] kursat_ob;
return 0;
}

```

Dastur Koord sinfining obyektlarining dinamik massivi hosil qilinishida foydalanuvchi tomonidan kelishuv bo'yicha aniqlangan parametrsiz konstruktor amal qiladi va dastur ishlashi natijasida ekranga

```

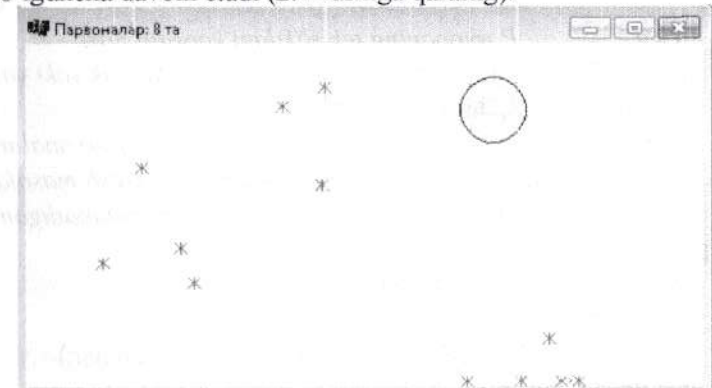
0 0
0 0
0 0
0 0

```

qiymatlari chop etiladi.

Misol. C++ Builder muhitida yorug'lik manbai (chiroq) atrofida xoatik ravishda uchadigan hashorotlar - parvonalar harakatini ko'rsatuvchi dastur tuzish masalasi qaraladi. Parvonalar harakati Form1 sirtida ro'y beradi. Sirtning qaysidir qismida (chap yuqori burchagida) yorug'lik

manbai shar ko'rinishida joylashadi. Boshida parvonalar sirt yuzida tasodifiy ravishda joylashadi va muntazam ravishda xoatik ravishda harakatlanishadi, yani ular tasodifiy ravishda bir joydan ikkinchi joyga ko'chadi. Agar parvona chiroqqa ma'lum bir masofaga yaqin kelib qolsa, ishqilik ta'sirida halok bo'ladi. Ilovaning amal qilishi barcha parvonalar halok bo'lguncha davom etadi (2.1-rasmga qarang).



2.1 -rasm. PARVONA dasturining ishlash holati.

Har bir parvonani ifodalash uchun PARVONA sinfi yaratiladi. Sinf berilgan-a'zolari sifatida parvonaning sirt tekisligidagi joylashuvi koordinatalari, unung ayni paytdagi holatlari (tirik yoki holak bo'lganligini, halok bo'lgandan keyin yerga tushgan yoki yo'qligi) parametrlarini o'z ichiga oladi. Barcha parvonalar uchun umumiy bo'lgan parametrlar - chiroqning koordinasi va radiusi, ayni paytdagi tirik parvonalar soni, parvonaning chiroqqa xavfli yaqinlashish va ko'chish masofasi, hamda parvonalar harakatlanadigan sirt komponentalari PARVONA sinfining statik a'zolari qilib aniqlagan. Sinfining funktsiya-a'zolari esa parvonaning boshlang'ich parametrlarini berish uchun konstruktor, uning harakati va holatini aniqlaydigan funksiyalar tashkil qiladi. Parvonalar majmuasi PARVONA sinfi obektlari massivi ko'rinishida amalga oshiriladi.

Quyida dastur matni keltiriladi.

...

```

TForm1 *Form1;
const int Parvonalar_Soni=15,
//Chiroqning koordinatalri va kattaligi (radiusi)
Chiroq_X=350,Chiroq_Y=50 ,Chiroq_R=25;
class PARVONA
{
    int x,y;          // Parvonaning tekislikdagi koordinatasi
    boolean Tirik, Yerda; // Parvonaning ayni paytda tirik yoki yo'qligi
    static int ChiroqX,ChiroqY,ChiroqR;
    static int Tirik_Parvonalar;    // Ayni paytda tirik parvonalar soni
    static int Masofa;              // Parvonaning ko'chish masofasi
    static TForm * Sirt;            // Parvonalar harakatlanadigan sirt.
public:
    PARVONA (int max_x, int max_y);
    void static SIRT(TForm * _Sirt){Sirt=_Sirt;}
    int static PARVONALAR_SONI(){return Tirik_Parvonalar;}
    void Paydo_bulish();
    void Yoq_bulish();
    void Yangi_Joy();
    void Kochish();
    bool Chiroqqa_Yaqin(int masofa);
};
int PARVONA:: Tirik_Parvonalar=Parvonalar_Soni;
int PARVONA:: ChiroqX=Chiroq_X;
int PARVONA:: ChiroqY=Chiroq_Y;
int PARVONA:: ChiroqR=Chiroq_R;
TForm * PARVONA:: Sirt=Form1;
int PARVONA:: Masofa=25;
PARVONA::PARVONA(int max_x, int max_y)
// max_x, max_y - parvona paydo bo'lish chegarali
{
    Tirik=true;

```

```

Yerda=false;
do
{
    x=random(max_x); y=random(max_y);
}
while(Chiroqqa_Yaqin(Masofa)); //Chiroqqa xavfli yaqinlikda emas
Paydo_bulish();
};
bool PARVONA:: Chiroqqa_Yaqin(int masofa)
{
    Return sqrt((ChiroqR+masofa)*(ChiroqR+masofa))>=
    sqrt(pow((ChiroqX-x),2)+pow((ChiroqY-y),2));
}
void PARVONA::Paydo_bulish()
{
    if(!Chiroqqa_Yaqin(Masofa))
    {
        if(Tirik) Sirt->Canvas->Font->Color=clGreen;
        // Tirik parvona yashil rangda,
        else Sirt->Canvas->Font->Color=clRed; // aks holda qizil.
        Sirt->Canvas->TextOutA(x,y,'Ж');
    }
}
void PARVONA:: Yoq_bulish()
{
    if(!Chiroqqa_Yaqin(Masofa))
    {
        Sirt->Canvas->Font->Color=Sirt->Color;
        Sirt->Canvas->TextOutA(x,y,'Ж');
    }
}
void PARVONA:: Yangi_Joy()

```

```

{
    if (Tirik)
    {
        do {
            int dx=random(4*Masofa)-2*Masofa;
            int dy=random(4*Masofa)-2*Masofa;
            if (x+dx<0) x+=fabs(dx);
            if (y+dy<0) y+=fabs(dy);
            if(x+dx<Sirt->ClientWidth-20)x+=dx; else x-=fabs(dx);
            if(y+dy<Sirt->ClientHeight-20)y+=dy; else y-=fabs(dy);
        } while(Chiroqqa_Yaqin(Masofa));
        Tirik=!Chiroqqa_Yaqin(Masofa+10);
        if(!Tirik)
        {
            Tirik_Parvonalar--;
            Sirt->Caption="Ayni          paytda          "          +
            IntToStr(Tirik_Parvonalar)+"parvonalar tirik!";
        }
    }
    else {
        if(y<Sirt->ClientHeight-10) y+=2*Masofa;
        else if(y!=(Sirt->ClientHeight-10))y=Sirt->ClientHeight-10;
        else Yerda=true;
        while (Chiroqqa_Yaqin(Masofa))y+=2*Masofa;
    }
}
void PARVONA:: Kochish()
{
    if(!Yerda)
    {
        Yoq_bulish();
        Yangi_Joy();
    }
}

```

```

Paydo_bulish();
}
}
PARVONA **parvona;
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    for(int i=0; i< Parvonalar_Soni; i++)
        parvona[i]-> Kochish();
}
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    PARVONA::SIRT(Form1);
    Form1->Caption="Parvonalar: " + IntToStr(Parvonalar_Soni)+" ta
qoldi";
    Form1->Canvas->Font->Height=12;
    Shape1->Left=Chiroq_X-Chiroq_R;
    Shape1->Top=Chiroq_Y-Chiroq_R;
    Shape1->Height=2*Chiroq_R;
    Shape1->Width=2*Chiroq_R;
    parvona = new PARVONA*[Parvonalar_Soni];
    for(int i=0; i<Parvonalar_Soni; i++)
        parvona[i]=new PARVONA (Form1->ClientWidth,Form1-
>ClientHeight);
}
void __fastcall TForm1::FormClick(TObject *Sender)
{
    Timer1->Enabled=!Timer1->Enabled;
}
}

```

2-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Sinf (class) sintaksisining struct va union sintaksislaridan farqi nimada?
2. Sinf a'zolari nima?

3. Sinf funksiya-a'zosi e'loni va aniqlanishi tavsiflansin.
4. Sinf a'zolariga murojaat spetsifikatorlarining vazifalari nimadan iborat?
5. Sinf konstruktori va destruktorni aniqlanishi va amal qilishini tushuntirilsin.
6. Nusxalash konstruktori nima?
7. **this** ko'rsatkichining mazmunini qanday?
8. Sinfning statik a'zolarini tavsiflansin.
9. Sinf obyektlari massivlarini aniqlashning o'ziga xosligi nimada?

3. Do'st funksiyalar va sinf do'stlari

3.1. Do'st funksiyalar

C++ tili OYD asosiy konsepsiyalaridan biri – berilganlarni inkapsulyatsiyalash konsepsiyasini “do'stlar” yordamida “buzish” mumkin. C++ tilida sinfning ikki turdagi sinf do'stlarini e'lon qilish imkonini beradi: *do'st funksiya* va *do'st sinf*.

Ma'lumki, sinfning yopiq elementlariga ochiq funksiya-a'zolar orqali murojaat qilish mumkin. Lekin C++ tili sinfning yopiq elementlariga boshqa yo'l orqali – do'st funksiyalar orqali murojaat qilishni qo'llab-quvvatlaydi. Do'st funksiyalar sinf a'zolari bo'lmagan holda shu sinfning yopiq a'zolariga murojaat qilish imkoniyatiga ega.

Birorta sinfga do'st funksiyani e'lon qilish uchun shu sinf tarkibida oldiga **friend** kalit so'zi qo'yilgan funksiya prototipi yoziladi.

Misol uchun:

```
#include <iostream.h>
class Dust_sinf
{
    int x,y;
public:
    Dust_sinf(int n, int m){x=n; y=m;}
    friend bool Karrali(Dust_sinf ob); // Do'st funksiya prototipi
};
bool Karrali(Dust_sinf ob)
{
    return (!(ob.x % ob.y));
}
main()
{
    Dust_sinf obekt(12,3);
    if (Karrali(obekt)) cout<<"Sonlar karrali.\n";
    else cout<<"Sonlar o'zaro karrali emas.\n";
    return 0;
}
```

```
}
```

Shuni qayd qilish kerakki, do'st funksiya o'zi e'lon qilingan sinfning a'zosi hisoblanmaydi. Shu sababli, do'st funksiya chaqirilishida obyekt nomini ko'rsatish yoki sinf a'zolariga murojaat amallarini (‘.’ yoki “->”) ishlatilmaydi. Do'st funksiyalar sinfning yopiq a'zolariga faqat sinf obyekti orqali murojaat qiladi. Bu obyekt o'z navbatida do'st funksiyada e'lon qilingan bo'lishi yoki unga argument sifatida berilishi mumkin. Do'st funksiya (sinf) vorislik bilan o'tmaydi, yani ular hosilaviy sinflarda amal qilmaydi. Boshqa tomondan, funksiya bir vaqtning o'zida bir nechta sinfga do'st bo'lishi mumkin.

Masalan:

```
include <iostream.h>
class Sinf1;    // Sinfning to'liqmas e'loni
class Sinf2;
{
    int d;
public:
    Sinf2(int _d){d= _d;}
    //Do'st funksiya prototipi
    friend bool Karrali_Sonlar(Sinf1 ob1, Sinf2 ob2);
};
class Sinf1
{
    int n;
public:
    Sinf1(int _n){n= _n;}
    // Do'st funksiya prototipi
    friend bool Karrali_Sonlar(Sinf1 ob1, Sinf2 ob2);
};
bool Karrali_Sonlar(Sinf1 ob1,Sinf2 ob2)
{
    return(!(ob1.n%ob2.d));
}
```

```
main()
```

```
{
    Sinf1 obekt1(15);
    Sinf2 obekt2(3);
    cout<<"Sonlar karrali";
    if(!Karrali_Sonlar(obekt1,obekt2)) cout<<" emas";
    cout<<'!';
    return 0;}
}
```

Bu misolda sinfning to'liqmas e'lonidan foydalanish ko'rsatilgan. Bu qurilmani qo'llamasdan hali e'loni berilmagan sinf nomini ishlatib bo'lmaydi.

Funksiya bir sinfning a'zosi bo'lgan holda boshqa sinfning do'sti bo'lishi mumkin.

Masalan:

```
include <iostream.h>
class Sinf1;    // Sinfning to'liqmas e'loni
class Sinf2;
{
    int d;
public:
    Sinf2(int _d){d= _d;}
    bool Karrali_Sonlar(Sinf1 ob1);
};
class Sinf1
{
    int n;
public:
    Sinf1(int _n){n= _n;}
    friend bool Sinf2::Karrali_Sonlar(Sinf1 ob1); // Do'st funksiya
    // prototipi
};
bool Sinf2::Karrali_Sonlar(Sinf1 ob1)
{
```

```

    return (!(ob1.n%d));
}
main()
{
    Sinf1 obekt1(15);
    Sinf2 obekt2(3);
    if(Sinf2.Karrali_Sonlar(obekt1)) cout<<"Sonlar karrali!";
    else cout<<"Sonlar karrali emas!";
    return 0;
}

```

3.2. Do'st sinflar

C++ tili nafaqat do'st funksiyalarni, balki do'st sinflarni e'lon qilish mumkin. Do'st sinfga uni do'st deb e'lon qilgan sinfning barcha a'zolariga murojaat qilishga to'la ruxsat beradi. Do'st sinfni e'lon qilish uchun sinf e'loniga do'st sinfni **friend** kalit so'zi bilan kiritish etarli.

Masalan:

```

class Sinf
{
    //...
    friend class Dust_Sinf;
};

```

Bu erda **Sinf** sinfga do'st **Dust_Sinf** sinfi e'lon qilingan.

Sinf o'zini boshqa bir ikkinchi sinfning do'sti deb e'lon qila olmaydi. Buning uchun u ikkinchi sinfda do'st deb e'lon qilingan bo'lishi kerak.

Masalan:

```

#include <iostream.h>
class Sinf1
{
    friend class Sinf2;    // Sinf2 sinfi Sinf1 sinfning do'sti
    int x;
    void IncX(){x++;}
}

```

```

public:
    Sinf1(){x=0;}
    Sinf1(int _x){x=_x;}
};
class Sinf2
{
    Sinf1 sinf1;
public:
    void x_qiymati();
};
void Sinf2::x_qiymati()
{
    cout<<sinf1.x<<endl;
    sinf1.IncX();
    cout<<sinf1.x<<endl;
}
int main()
{
    Sinf2 sinf2;
    sinf2.x_qiymati();
    return 0;
}

```

Sinflar bir-birini do'st deb e'lon qilishi mumkin. Lekin bu holat ijobiy baholanmaydi, u sinflar shajarasi chuqur o'ylamasdan tuzilganligining dalolati hisoblanadi. Shunga qaramasdan C++ tili bunga imkon beradi. O'zaro do'st sinflar quyidagicha e'lon qilingan bo'lishi kerak:

```

class Sinf2;    // Sinfning to'liqmas e'loni
class Sinf1
{
    friend class Sinf2;
    //...
};

```

```
class Sinf2
```

```
{
```

```
    friend class Sinf1;
```

```
    //...
```

```
};
```

Yuqorida keltirilgan dastur bo'lagida sinfning to'liqmas e'loni faqat **Sinf1** ichida **Sinf2** sinfiga ko'rsatkich bo'lgandagina zarur bo'ladi.

Do'st sinflar uchun quyidagi qoidalar amal qiladi:

- do'stlik o'zaro emas, ya'ni **Sinf1** sinfi **Sinf2** sinfining do'sti ekanligidan **Sinf2** sinfini **Sinf1** sinf do'sti ekanligi kelib chiqmaydi;
- do'stlik vorislik bo'yicha o'tmaydi, agar **Sinf1** sinfi **Sinf2** sinfining do'sti bo'lsa, **Sinf1** sinfini avlodlari **Sinf2** sinf do'stlari bo'lmaydi;
- do'stlik tayanch sinf avlodlariga o'tmaydi, agar **Sinf1** sinfi **Sinf2** sinfining do'sti bo'lsa, **Sinf1** sinfini **Sinf2** avlodlariga do'st bo'lmaydi.

3-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Sinf do'stlarini e'lon qilish nima uchun zarur?
2. Do'st sinflar uchun qoidalar nimadan iborat?
3. Sinflar o'zaro do'st bo'lishi mumkinmi?

4. Vorislik

4.1. Sinflar o'rtasidagi munosabatlar

Aksariyat OYD tillarida sinflar o'rtasida quyidagi munosabatlar qo'llab-quvvatlanadi:

- assotsiatsiya;
- vorislik;
- agregatsiya;
- bog'liqlik (foydalanish).

Assotsiatsiya. Agar ikkita sinf konseptuallik darajasida bir-biriga ta'sir o'tkazsa, u holda bunday o'zaro ta'sirga *assotsiatsiya* deyiladi. Masalan, savdo shoxobchasi ishini modellashtirishda ikkita abstraksiya (tushuncha) yuzaga keladi: mahsulotlar (**Mahsulot** sinfi) va sotish (**Sotish** sinfi). **Sotish** sinfining obyekti – bu qandaydir hodisa bo'lib, unda 1 dan n gacha **Mahsulot** sinfining obyektlari sotilganligini anglatadi.

Assotsiatsiya loyihalashning boshlang'ich qadamlarida aniqlanadigan ikkita sinf o'rtasidagi eng ko'p namoyon bo'lgan semantik bog'lanishlarni ifodalaydi. Assotsiativ bog'lanishlar vorislik, agregatsiya, bog'lanish munosabatlari orqali konkretlashtiriladi.

Agregatsiya. Sinflar o'rtasidagi *agregatsiya* munosabati bitta sinf o'zining tarkibiy qismi sifatida boshqa sinf obyektlarini o'z ichiga olgan holatlarda namoyon bo'ladi. Boshqacha aytganda bu munosabat "*butun/qism*" munosabatidir.

Agregatsiyani ikkita ko'rinishda amalga oshirish mumkin:

- qat'iymas agregatsiyalar (oddiy agregatsiyalar);
- qat'iy agregatsiyalar (kompozitsiyalar).

Qat'iymas agregatsiyada e'lon qilingan barcha *qismlar butunga* kirishi shart emas. Shu sababli, "*qismning butunga kirishi*" ko'rsatkich vositasida amalga oshiriladi va ularning barchasi initsializatsiya qilinishi shart emas (ayrim ko'rsatkichlar 0 qiymatiga ega bo'ladi). Yechilayotgan masalaga bog'liq ravishda bunday komponentalar dinamik ravishda paydo bo'lishi yoki yo'qotilishi mumkin.

Qat'iy agregatsiya munosabatida komponenta (*qism*) *butun* obyekt mavjudlik paytida yo'q bo'lishi mumkin bo'lmaydi. Masalan, uchburchak

(Uchburchak sinfi) uchta nuqtani (Nuqta sinfi obyektlarini) o'z ichiga oladi va nuqtalardan birortasi bo'lmasa, uchburchak yasash mumkin emas.

Kompozitsiyani amalga oshirishning sodda yo'li - kompozitsiya obyektlar-komponentalarni qiymatlari bilan kiritish yo'lidir (oddiy o'zgaruvchilar sifatida). Agar komponentalar ko'rsatkichlar vositasida amalga oshirilsa, ular quyidagi talablarga javob berishlari kerak: komponentalarning yashash davri *butun* obyekt yashash davri bilan ustma-ust tushishi kerak.

Bog'liqlik. Ikki sinf o'rtasida *bog'liqlik* (foydalanish) munosabatida bir sinf (Asosiy sinfi) boshqa sinf (Yordamchi sinfining) xizmatlaridan foydalanadi. Masalan:

- Asosiy sinf funksiya-a'zosi Yordamchi sinfning ayrim maydonlari qiymatlaridan foydalanadi;
- Asosiy sinf funksiya-a'zosi Yordamchi sinfning funksiya-a'zosini chaqiradi;
- Asosiy sinf funksiya-a'zosining prototipida Yordamchi sinf turidagi parametrlar qatnashadi.

Misol uchun, Uchburchak sinfida uchburchak tomonining uzunligini hisoblaydigan funksiya-a'zosi e'lonida Nuqta sinfi turidagi parametrlar qatnashadi:

double Uchburchak::Masofa(Nuqta a, Nuqta b);

Vorislik. Vorislik ("is a") sinf ostining (hosilaviy sinfning) bir yoki bir nechta *supersinflarda* (*tayanch sinflarda*) aniqlangan atribut va amallarni birgalikda ishlatadi. Masalan, Ko_pBurchak sinfining vorisi bo'lgan Uchburchak sinfi Ko_pBurchak tayanch sinfi a'zolarini ishlatishi mumkin (qayta aniqlamasdan).

4.2. Oddiy vorislik

C++ tili sinfga bir yoki bir nechta sinfning berilgan-a'zolari va funksiya-a'zolarini vorislik bilan olishga imkon beradi. Bu holda yangi sinfni *hosilaviy sinf* (*avlod sinf*) deyiladi. Elementlari vorislik bo'yicha olinadigan sinfga uning vorisiga nisbatan tayanch sinf (*ajdod sinf*) deyiladi.

Vorislik sinflarning umumiy xususiyatlarini yagona bitta tayanch sinfda abstraksiyalash imkonini beradi. Voris sinflar esa umumiy xususiyatlarni vorislik bilan olgan holda, ayrim funksiya-a'zolarini qayta aniqlash orqali yoki qo'shimcha berilgan-a'zolar va funksiya-a'zolarini kiritish orqali tayanch sinfga ma'lum bir o'zgartirish kiritishlari mumkin. Shu sababli, hosilaviy sinflarni aniqlash sezilarli ravishda kamayadi, chunki unga faqat tayanch sinfdan farqli elementlar qo'shiladi.

Hosilaviy sinfni e'lon qilish sintaksisi quyidagi ko'rinishda bo'ladi:

```
class <hosilaviy sinf nomi>:<murojaat xossasi><tayanch sinfi>,
    <murojaat xossasi><tayanch sinfi>,
    ...
    <murojaat xossasi><tayanch sinfi>
```

```
{
// hosilaviy sinf tarkibi
};
```

Bu erda <murojaat xossasi> - public, protected yoki private va ular bo'lishi shart emas. Agar <murojaat xossasi> bo'lmasa kelishuv bo'yicha sinflar uchun private, tuzilma va birlashma uchun public hisoblanadi. <tayanch sinfi> - oldindan aniqlangan tayanch sinflarning nomlari.

4.1-jadval. Sinflarning vorislik variantlari

Murojaat xossasi	Tayanch sinfdagi murojaat turi	Voris sinfdagi mos murojaat turi
public	public protected private	public protected murojaat yo'q
protected	public protected private	protected protected murojaat yo'q
private	public protected private	private private murojaat yo'q

Murojaat xossasi tayanch sinf elementlarining vorislik bo'yicha olish darajasini aniqlaydi. 4.1-jadvalda vorislik variantlari keltirilgan.

4.1-jadval bilan tanishish shuni ko'rsatadiki, vorislikning murojaat xossasi tayanch sinfda o'rnatilgan murojaatni qanday darajagacha tushganligini ko'rsatadi (ochiq a'zolarga nisbatan).

Agar tayanch sinfdagi a'zolarga hosilaviy sinflarda murojaat qilish zarur bo'lsa ularni public deb e'lon qilish kerak bo'ladi.

Misol.

```
class Tayanch
{
    int x,y;
public:
    Tayanch(int _x=5,int _y=10){x=_x; y=_y;}
    int X_Qiymati(){return x;}
    int Y_Qiymati(){return y;}
};
class Hosila : private Tayanch
{
public:
    int Tayanch::X_Qiymati();
};
main()
{
    int X,Y;
    Hosila ob;
    X=ob.X_Qiymati();
    Y=ob.Y_Qiymati();
    cout<<"X="<<X<<" , Y="<<Y<<endl;
    return 0;
}
```

Dasturda Hosilaviy hosilaviy sinfi tayanch sinfdagi X_Qiymati() va Y_Qiymati() funksiyalarini private darajasidagi murojaat bilan vorislik bo'yicha oladi va ekranga

X=5, Y=10
ifodalari chop qilinadi.

Shuni qayd etish kerakki, hosilaviy sinf tayanch sinfning faqat public yoki protected a'zolariga murojaat qilishi mumkin. Tayanch sinfning yopiq a'zolari har qanday vorislikda ham yopiqligicha qoladi.

Tayanch sinfning quyidagi elementlari vorislik bilan o'tmaydi:

- konstruktorlar;
- nusxalash konstruktorlari;
- destruktorlar;
- dastur tuzuvchisi tomonidan aniqlangan qiymat berish operatori;
- sinfning do'stlari.

Hosilaviy sinf bitta tayanch sinfga ega bo'lsa, bunday vorislikni *oddiy (yakka) vorislik* deyiladi.

Quyidagi misolda oddiy vorislik amalga oshirilgan.

```
#include <iostream.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(int _x, int _y){x=_x; y=_y;}
    Nuqta(){x=0; y=0;}
    int X_Qiymati(){return x;}
    int Y_Qiymati(){return y;}
    void Qiymat_X(int _x){x=_x;}
    void Qiymat_Y(int _y){y=_y;}
};
class Dekart : public Nuqta
{
public:
    Dekart (int _x, int _y): Nuqta(2*_x, 2*_y){};
    void Qiymat_XX(){cout<<X_Qiymat_X()<<' ';}
    void Qiymat_YY(){cout<<Y_Qiymat_X()<<' ';}
};
main()
{
```

```

Dekart * dkt;
dkt=new Dekart(10,15); // Obyektni uyumda yaratish
dkt->Qiymat_XX();
dkt->Qiymat_YY();
delete dkt; // Obyektni uyumdan o'chirish
return 0;
}

```

Ushbu dasturda Dekart hosilaviy sinfi yagona Nuqta tayanch sinfiga ega. Konstruktor vorislik bo'yicha o'tmaganligi sababli, voris sinfda konstruktor sintaksisi o'zgargan. Ma'lumki, voris sinf tayanch sinfning barcha berilganlarini vorislik bo'yicha oladi (garchi tayanch sinfning ayrim a'zolariga to'g'ridan-to'g'ri murojaat qila olmasa ham) va sinf obyektini aniqlanishida bu berilganlar initsializatsiya qilinishi kerak. Chunki, voris sinfda bu berilganlar ishlatilishi mumkin. Shu sababli hosilaviy sinf konstruktori quyidagi tuzilma ko'rinishida aniqlanadi:

```

<hosilaviy sinf konstruktori>(<parametrlar ro'yxati>);
<tayanch sinf konstruktori>(<argumentlar ro'yxati>){<konstruktor tanasi>}

```

Konstruktorida tayanch sinf konstruktoriga ko'rsatuvchi elementlarni initsializatsiyalash ro'yxatidan foydalanilgan. Voris sinf konstruktoriga uzatilgan argumentlarning ma'lum bir qismi tayanch sinf konstruktoriga argument sifatida uzatiladi. Keyin voris sinf konstruktori tanasida ayni shu sinf uchun zarur initsializatsiya amallarini bajarilishi mumkin. Yuqorida keltirilgan misolda Dekart voris sinfi konstruktori

```

Dekart(int x, int y):Nuqta(2*x, 2*y){};

```

ko'rinishida aniqlangan. Voris sinf obyektini initsializatsiya uchun uzatilgan argumentlar ikkiga ko'paytirilgan holda tayanch sinf berilgan a'zolarini initsializatsiyalash uchun ishlatimoqda. Konstruktor tanasi bo'sh, chunki bu sinf o'z berilganlariga ega emas.

Agar voris sinf konstruktori parametrsiz aniqlangan bo'lsa, ya'ni kelishuv bo'yicha konstruktor bo'lsa, hosilaviy sinf obyektini yaratilishida tayanch sinf konstruktori avtomatik ravishda chaqiriladi.

Hosilaviy sinf destruktoriga nisbatan quyidagi qoida amal qiladi: hosilaviy sinf destruktoriga tayanch sinf destruktoriga oldin ishlashi kerak.

Aks holda tayanch sinf destruktoriga ayni paytda ishlatilayotgan hosilaviy sinf berilgan a'zolarini o'chirishi mumkin. Destruktorlar to'g'ri ishlashini kompilyator o'z zimmasiga oladi.

Quyida keltirilgan dasturda tayanch va voris sinflar konstruktorlari va destruktorga ishlash ketma-ketligi ko'rsatilgan.

```

#include <iostream.h>
class Tayanch
{
public:
    Tayanch(){cout<<"Tayanch sinf konstruktori ishladi.\n";}
    ~Tayanch(){cout<<"Tayanch sinf destruktoriga ishladi.\n";}
};
class Hosila:public Tayanch
{
public:
    Hosila(){cout<<"Hosila sinf konstruktori ishladi.\n";}
    ~Hosila(){cout<<"Hosila sinf destruktoriga ishladi.\n";}
};
int main()
{
    Hosila ob;
    return 0;
}

```

Dastur ishlashi natijasida ekranga quyidagi xabarlar chiqadi:

Tayanch sinf konstruktori ishladi.

Hosila sinf konstruktori ishladi.

Hosila sinf destruktoriga ishladi.

Tayanch sinf destruktoriga ishladi.

Ko'p hollarda hosilaviy sinfda tayanch sinf a'zolari qatoriga yangi a'zolar qo'shiladi. Biroq tayanch sinf a'zolarini qayta aniqlash imkoniyati mavjud. Odatda tayanch sinf funktsiya a'zolari qayta aniqlanadi. Buning uchun tayanch sinf funktsiya a'zosini hosilaviy sinf e'loni prototipini berish va keyinchalik uni aniqlash etarli. Misol tariqasida Nuqta tayanch

sinfda aniqlangan `X_Qiymati()` va `Y_Qiymati()` funksiyalarini qayta aniqlashni qaraylik:

```
#include <iostream.h>
class Nuqta
{
protected:
    int x,y;
public:
    Nuqta(int _x, int _y){x=_x; y=_y;}
    Nuqta(){x=0; y=0;}
    int X_Qiymati(){return x;}
    int Y_Qiymati(){return y;}
    void Qiymat_X(int _x){x=_x;}
    void Qiymat_Y(int _y){y=_y;}
};
class Dekart : public Nuqta
{
public:
    Dekart (int _x, int _y):Nuqta(_x,_y){};
    int X_Qiymati(){return ++x;}
    int Y_Qiymati(){return ++y;}
    void Qiymat_XX(){cout<<X_Qiymat_X()<<' ';}
    void Qiymat_YY(){cout<<Y_Qiymat_X()<<' ';}
};
main()
{
    Dekart * dkt;
    dkt=new Dekart(10,15);    // Obyektni uyumda yaratish
    dkt-> Qiymat_XX();
    dkt-> Qiymat_YY();
    delete dkt;              // Obyektni uyumdan o'chirish
    return 0;
}
```

}

Dastur ishlashi natijasida ekranga berilgan koordinatalarning bittaga oshirilgan qiymatlari chop etiladi. Garchi dasturda tayanch sinfning `X_Qiymati()` va `Y_Qiymati()` funksiyalarini qayta aniqlangan bo'lsa ham ularning oldingi variantlariga murojaat qilish mumkin. Buning uchun "*ko'rinish sohasiga ruxsat berish*" amalidan foydalanish zarur bo'ladi. Yuqoridagi misolda keltirilgan dastur matniga

```
...
void Qiymat_XX(){cout<<Nuqta::X_Qiymat_X()<<' ';}
void Qiymat_YY(){cout<<Nuqta::Y_Qiymat_X()<<' ';}
...
```

o'zgartirish orqali qayta aniqlangan funksiyaning tayanch sinfdagi aniqlanishini chaqirish mumkin.

Hosilaviy sinf o'z navbatida boshqa uchinchi bir sinf uchun tayanch bo'lishi mumkin. Bunda tayanch sinf uchinchi sinf uchun *vositali tayanch sinf* bo'ladi.

4.3. To'plamli vorislik

Agar hosilaviy sinf bir nechta tayanch sinflarga ega bo'lsa, bunday vorislikka *to'plamli vorislik* deyiladi. To'plamli vorislik bitta sinfda bir nechta sinf xossalari va amallarini jamlash imkonini beradi.

Quyidagi misol to'plamli vorislikni namoyish qiladi:

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(int _x,int _y){x=_x;y=_y;}
    Nuqta(){x=0; y=0;}
    int X_Qiymati(){return x;}
    int Y_Qiymati(){return y;}
}
```

```

void Qiymat_X(int _x){x= _x;}
void Qiymat_Y(int _y){y= _y;}
};
class Xabar_Qabul_Qilish
{
    char Xabar[80];
public:
    Xabar_Qabul_Qilish(char * xbr){Xabarni_Saqlash(xbr);}
    void Xabarni_Saqlash(char *xabar){strcpy(Xabar,xabar);}
    void Xabar_Berish(){cout<<Xabar;}
};
class Xabarni_Chop_Qilish: public Nuqta,
    public Xabar_Qabul_Qilish
{
public:
    Xabarni_Chop_Qilish(int _x,int _y,char * xbr):
        Nuqta(_x,_y),Xabar_Qabul_Qilish(xbr){}
    void Chop_Qilish();
};
void Xabarni_Chop_Qilish::Chop_Qilish()
{
    gotoxy(X_Qiymati(),Y_Qiymati());    // Kursor xabar chiqarish
    nuqtasiga
    Xabar_Berish();
}
int main()
{
    Xabarni_Chop_Qilish * xbr_adress;
    xbr_adress=new Xabarni_Chop_Qilish(10,5,"To'plamli");
    xbr_adress->Chop_Qilish();
    xbr_adress->Qiymat_X(10);
    xbr_adress->Qiymat_Y(6);

```

```

xbr_adress->Xabarni_Saqlash("vorislik! ");
xbr_adress->Chop_Qilish();
delete xbr_adress;
return 0;
}

```

Ushbu misolda **Nuqta** sinfi ekrandagi belgi o'qi koordinatasini o'rnatish va saqlashga, **Xabar_Olish** sinfi xabarni qabul qilish va saqlashga xizmat qiladi. Ularning vorisi bo'lgan **Xabarni_Chop_Qilish** sinfi esa xabarni ekranning ko'rsatilgan koordinatada chop qiladi. Misol tayanch va voris sinflar konstruktorlarini qanday ishlashi kerakligini namoyon etadi. To'plamli vorislikda, xuddi oddiy vorislikdagidek, oldin tayanch sinflar konstruktorlari, keyinchalik voris sinf konstruktori ishlashi kerak. Tayanch sinf konstruktorlarining qanday ketma-ketlikda ishlashi, ularning voris sinfi e'lonining initsializatsiya ro'yxatidagi o'qi bilan aniqlanadi. Misolda **Xabarni_Chop_Qilish** sinfi vorislik haqida quyidagi e'loniga ega:

```

class Xabarni_Chop_Qilish: public Nuqta, public
Xabar_Qabul_Qilish
{
    Bu e'loniga quyidagi konstruktor mos keladi:
    Xabarni_Chop_Qilish(int _x,int _y,char * xbr): Nuqta(_x,_y),Xa-
        bar_Qabul_Qilish(xbr){}

```

Dasturning bosh funksiyasidan **Xabarni_Chop_Qilish** sinfini e'loni orqali konstruktoriga argumentlar uzatiladi:

```

xbr_adress=new Xabarni_Chop_Qilish(10,5,"To'plamli ");

```

Keyingi qatordagi **Chop_Qilish()** funksiyasi ekranning (10,5) koordinatasida "To'plamli" satrini chop qiladi. **Qiymat_X(10)**, **Qiymat_Y(6)** va **Xabarni_Saqlash("vorislik!")** funksiyalarini chaqirish orqali yangi berilganlar o'zlashtiriladi. Ikkinchi marta **Chop_Qilish()** funksiyasini chaqirish orqali ko'rsatilgan koordinatada boshqa xabar chiqadi.

Dastur ishlashi natijasida ekranga

To'plamli

vorislik!

xabari chop etiladi.

To'plamli vorislikda konstruktor va destruktorni amal qilish ketma-ketligini quyidagi dastur misolida ko'rish mumkin.

```
#include <iostream.h>
class Tayanch1
{
public:
    Tayanch1(){cout<<"Tayanch1 sinf konstruktori ishladi.\n"}
    ~Tayanch1(){cout<<"Tayanch1 sinf destruktorni ishladi.\n"}
};
class Tayanch2
{
public:
    Tayanch2(){cout<<"Tayanch2 sinf konstruktori ishladi.\n";}
    ~Tayanch2(){cout<<"Tayanch2 sinf destruktorni ishladi.\n";}
};
class Hosila: public Tayanch1, public Tayanch2
{
public:
    Hosila(){cout<<"Hosila sinf konstruktori ishladi.\n";}
    ~Hosila(){cout<<"Hosila sinf destruktorni ishladi.\n";}
};
int main()
{
    Hosila ob;
    return 0;
}
```

Dastur ishlashi natijasida ekranga quyidagi xabarlar chiqadi:
 Tayanch1 sinf konstruktori ishladi.
 Tayanch2 sinf konstruktori ishladi.
 Hosila sinf konstruktori ishladi.
 Hosila sinf destruktorni ishladi.
 Tayanch2 sinf destruktorni ishladi.
 Tayanch1 sinf destruktorni ishladi.

Misol shuni ko'rsatadiki, konstruktorlar ularning e'londagi tartibiga mos ravishda, destruktorni esa teskari ravishda ishg'a tushadi.

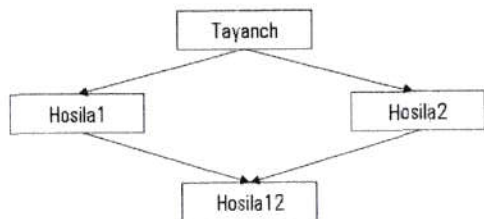
4.4. Virtual tayanch sinflar

To'plamli vorislikda sinflarning nisbatan murakkab shajarasida hosilaviy sinf vositali ravishda bitta sinfdan ikki yoki undan ortiq nusxasini vorislik orqali olishi mumkin. Quyida keltirilgan misol bu holatni namoyish etadi:

```
#include <iostream.h>
class Tayanch
{
    int x;
public:
    int X_Qiymati(){return x;}
    void Qiymat_X(int _x){x=_x;}
    double y;
};
class Hosila1 public Tayanch
{
    //...
};
class Hosila2 public Tayanch
{
    //...
};
class Hosila12 public Hosila1, public Hosila2;
{
    //...
};
main()
{
    Hosila12 ob;
    ob.y=5.0;
    ob.Qiymat_X(10);
    int z=ob.X_Qiymati();
}
```

```
return 0;
}
```

Bu misolda **Hosila12** sinfi o'zining tayanch sinflari – **Hosila1** va **Hosila2** sinflari orqali vositali ravishda **Tayanch** sinfini vorislik bo'yicha oladi. Bu holatning grafik shakli 4.1-rasmda keltirilgan.



4.1-rasm. Vositali vorislik.

Vorislikning bunday ko'rinishida **Hosila12** sinfida **Tayanch** sinfining ikkita nusxasi paydo bo'ladi va **Tayanch** sinfining **y** va **Qiymat_X()** a'zolariga

```
ob.y=5.0;
ob.Qiymat_X(10);
int z=ob.X_Qiymat();
```

ko'rinishidagi murojaatlarga kompilyatsiya xatosi yuzaga keladi. Chunki kompilyator bu ajdod sinflarining funksiya-a'zolarining qaysi nusxasiga murojaat bo'layotganligini aniqlay olmaydi. Bu o'rinda "*ko'rish sohasiga ruxsat berish*" amali yordamida qaysi tayanch sinf orqali ajdod sinfga murojaat bo'layotganligini ko'rsatish mumkin:

```
ob.Hosila1::y=5.0;
ob.Hosila1::Qiymat_X(10);
int z=ob.Hosila1::X_Qiymat();
```

Garchi "..." amalidan foydalanish noaniqlikni yo'qotgusa ham, **Hosila12** sinfida **Tayanch** sinfining ikkita nusxasi paydo bo'lishini yo'q qilmaydi. Bu muammoni hal qilish uchun kompilyatorga *virtual tayanch sinfni* ishlatish haqida ko'rsatma berish kerak bo'ladi. Virtual tayanch sinfni ko'rsatish uchun vorislik ko'rsatildigan joyda murojaat xossasidan oldin yoki keyin **virtual** kalit so'zini yozish kerak bo'ladi. Yuqoridagi

misolda **Hosila1** va **Hosila2** sinflar e'loniga quyidagi o'zgartirishlar qilish orqali **Hosila12** sinfida **Tayanch** sinfining ikkita nusxasi paydo bo'lishining oldi olinadi:

```
...
class Hosila1: virtual public Tayanch
```

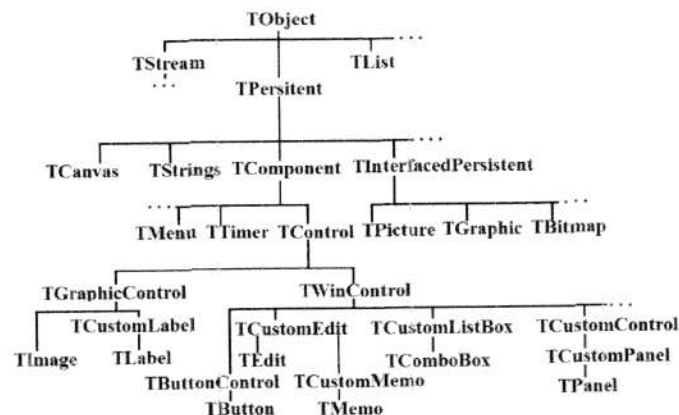
```
...
class Hosila2: virtual public Tayanch
```

```
...
Dasturling bu variantida
ob.y=5.0;
ob.Qiymat_X(10);
int z= ob.X_Qiymat();
```

murojaatlar kompilyatsiya xatosiga olib kelmaydi.

Shuni qayd qilish kerakki, sinflar shajarasida virtual sinflar bo'lganda konstruktorlar bajarilishi ketma-ketligi o'zgaradi: oldin virtual sinflar konstruktorlari ishga tushadi, keyinchalik novirtual ajdod sinflar konstruktorlari va oxirida hosila sinf konstruktori ishlaydi. Destruktorlar ishlashi shu ketma-ketlikka teskari ravishda amal qiladi.

Sinflar vorisligi samarali qo'llanishiga misol tariqasida **C++ Builder** muhitidagi vizual komponentalar kutubxonasi – **VCL** tuzilishini keltirish mumkin (4.2-rasm). Bu yerda **TObject** sinfi barcha **VCL** komponentalar sinflarining ajdodi hisoblanadi. Undan keyin **TPersistent** sinfi bo'lib, u komponentalarning fayllarda, xotirada saqlanishi va shunga o'xshash quyi darajadagi ishlarni bajarish bilan bog'liq xususiyatlarini aniqlab beradi. **TComponent** sinfi komponentalarning bevosita tayanch sinfi hisoblanadi. U barcha komponentalarning amal qilishini ta'minlaydi. **TComponent** sinfidan vizual va novizual komponentalar sinflari vorislik orqali yuzaga keladi. Vizual komponentalar uchun tayanch sinf – **TControl** sinfi bo'lib, unda vizual komponentalardan talab qilinadigan qo'shimcha funksiyalar aniqlangan. Foydalanuvchi ishlatadigan vizual komponentalar **TControl** sinfining vorislari **TGraphicControl** va **TWinControl** sinflarning avlodlari hisoblanadi.



4.2-rasm. VCL sinflari shajarasi.

4-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Sinflar o'rtasidagi munosabatlarni tavsiflab bering.
2. Vorislikdagi murojaat xossalari nimani anglatadi?
3. Oddiy va to'plamli vorislikni sxema ko'rinishida tavsiflab bering.
4. Tuzilma va birlashmalarda vorislikni amalga oshirsa bo'ladimi?
5. Vorislikda konstruktor va destruktorni amal qilish tartibi qanday?
6. Voris sinf konstruktorining o'ziga xos ko'rinishining sababi nimada?
7. To'plamli vorislikda virtual tayanch sinflar ishlatilishiga sabab nima?

5. Funksiyalarni qayta yuklash

5.1. Funksiyalarni qayta yuklashning afzalligi

Funksiyalarni qayta yuklash – bu C++ tili tomonidan qo'llab-quvvatlanadigan polimorfizm ko'rinishlaridan biridir. C++ tilida bir nechta funksiyalar bir xil nomga ega bo'lishi mumkin. Bu nomdagi funksiyaga *qayta yuklanuvchi funksiya* deyiladi. Argumentlarining turlari yoki soni bilan farqlanadigan funksiyalargina qayta yuklanishi mumkin. Qaytaruvchi qiymat turi bilan farq qiladigan funksiyalarni qayta yuklab bo'lmaydi. Funksiyalarni qayta yuklash orqali mazmunan har xil amal bajaruvchi funksiyalarni bir xil nomlash bilan dasturni soddalashtirish mumkin.

Funksiyalarni qayta yuklash uchun uni e'lonini berish va keyin zarur bo'lgan barcha chaqirish variantlarini aniqlash kerak bo'ladi. Kompilyatorning o'zi argumentlar soni va turiga mos ravishda funksiyaning kerakli variantini tanlaydi. Misol:

```

include <iostream.h>
// abs() funksiyasini qayta yuklash
int abs(int n);
long abs(long l);
double abs(double d);
main()
{
    int n=125; long l=-26L; double d=-15.0L;
    cout<<n<<" sonining absolyut qiymati="<<abs(n)<<endl;
    cout<<l<<" sonining absolyut qiymati="<<abs(l)<<endl;
    cout<<d<<" sonining absolyut qiymati="<<abs(d)<<endl;
    return 0;
}
int abs(int n){return n<0?-n:n;};
long abs(long l){return l<0?-l:l;};
double abs(double d){return d<0?-d:d;};
  
```

Ma'lumki, C++ tili kutubxonasida sonning absolyut qiymatini hisoblash uchun argument turiga mos ravishda `abs()`, `labs()` va `fabs()` funksiyalari aniqlangan. Yuqorida keltirilgan misolda sonning qanday turda bo'lishidan qat'iy nazar bitta `abs()` funksiyasi chaqirilgan va shuning hisobiga dastur matni soddalashtirishgan.

5.2. Sinfning funksiya-a'zolarini qayta yuklash

Funksiyalarni qayta yuklashda quyidagi holatlarga e'tibor berish kerak:

1) argumentlari turli xil initsializatorlarga ega funksiyalar har xil funksiya hisoblanadi va ularni qayta yuklash mumkin;

2) ikkita funksiya parametrlari ro'yxatida biror turdagi parametr bo'lib, ikkinchisida mos o'rinda shu turdagi adres oluvchi parametr bo'lsa, bunday funksiyalar qayta yuklash nuqtai-nazaridan aynan bir xil ko'rinish deb hisoblanadi va ular qayta yuklanmaydi. Masalan, e'lonlari

```
int func(int,int);
```

va

```
int func(int&,int&);
```

kompilyatsiya xatosiga olib keladi, chunki ular qayta yuklash shartlari bo'yicha bir xil hisoblanadi.

3) funksiyalarning qandaydir turga tegishli argumentlari bir-biridan `const` va `volatile` modifikatorlari bilan farqlansa, qayta yuklash nuqtai-nazaridan ular bir xil e'lon hisoblanadi va bu farq bilan funksiyalarni qayta yuklash mumkin emas;

4) funksiyaning parametrlari ro'yxatidagi adres oluvchi parametr bo'lgan holda ikkinchi funksiya-dagi mos o'rindagi adres oluvchi parametr `const` va `volatile` modifikatori bilan berilgan bo'lsa, qayta yuklash nuqtai-nazaridan bu funksiyalar farqlanadi va ularni qayta yuklash mumkin. Masalan, quyidagi qayta yuklashlar o'rinli.

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

```
class Sinf
```

```
{
```

```
Sinf(){cout<<"Sinf uchun kelishuv bo'yicha konstruktor\n";}
```

```
Sinf(Sinf& ob){cout<<"Sinf uchun nusxalash konstruktor\n";}
```

```
Sinf(const Sinf& ob){cout<<"Sinf uchun nusxalash konstruktor\n";}
```

```
};
```

```
int main()
```

```
{ Sinf ob1; // Kelishuv bo'yicha konstruktor amal qiladi
```

```
Sinf ob2(ob1); // Nusxalash konstruktor amal qiladi
```

```
const Sinf ob3;
```

```
Sinf ob4(ob3); // Konstanta obyektini nusxalash konstruktor
```

```
ishladi
```

```
return 0;
```

```
}
```

5) har qanday ikkita qayta yuklanuvchi funksiyalar turli xil parametrlar ro'yxatiga ega bo'lishi kerak;

6) funksiya-a'zolar faqat ularning biri `static`, ikkinchisida yo'qligi asosida qayta yuklanishi mumkin emas;

7) `typedef`-aniqlashlar qayta yuklash mexanizmiga ta'sir qilmaydi, chunki ular yangi turni yaratmaydi, ular mavjud turlarning sinonimi hisoblanadi. Masalan, quyidagi

```
typedef char *PSTR;
```

aniqlash, kompilyatorga «`void Satr(char * s);`» va «`void Satr(PSTR s);`» ko'rinishdagi funksiya e'lonlarini har xil deb hisoblashga asos bermaydi va ular qayta yuklanmaydi;

8) barcha `enum` turlari har xil hisoblanadi va ulardan funksiyalarni qayta yuklashda foydalanish mumkin;

9) massiv va ko'rsatkichlar qayta yuklash nuqtai-nazaridan bir xil e'lon deb qaraladi. Lekin bu fikr faqat bir o'lchamli massivlar uchun o'rinli. Shu sababli, «`void Satr(char*s);`» va «`void Satr(char s[ ]);`» ko'rinishidagi funksiyalarni qayta yuklashga urinish kompilyatsiya xatosiga olib keladi;

10) ko'p o'lchamli massivlarda ikkinchi va undan keyingi o'lchamlar berilganlar turining alohida bir qismi deb qaraladi va bu holat qayta yuklash uchun asos bo'ladi. Quyidagi funksiyalar qayta yuklanishi mumkin:

```
void Satr(char s[ ];
```

```
void Satr(char s[ ][5]);
```

Qayta yuklanuvchi funksiyalar e'lon qilinganda kompilyator bu funksiyalarning ko'rinish sohalarini tahlil qiladi. Masalan, hosilaviy sinfdagi tayanch sinfdagi funksiya-a'zo bilan bir xil nomdagi funksiya e'lon qilingan bo'lsa, u tayanch sinf funksiyasini "yashiradi" va kompilyator bu funksiyalarni qayta yuklash o'rimga *qayta aniqlaydi*, chunki ular turli ko'rinish sohalariga ega. Xuddi shunday kompilyator boshqa ko'rinish sohalarini ham kuzatib boradi. Agar funksiya fayl (boshlang'ich matn) ko'rinishi sohasida e'lon qilingan va u blokda e'lon qilingan funksiya bilan bitta ko'rinish sohasida bo'lsa ham ular kompilyator tomonidan qayta yuklanmaydi. Funksiyaning lokal e'loni uning global e'lonini qoplaydi (yashiradi). Masalan:

```
#include <iostream.h>
void func(int i) {cout<<"\nGlobal e'lon qilingan funksiya: "<<i; }
void func(char*s) {cout<<"\nLokal e'lon qilingan funksiya: "<<s; }
int main()
{ void func(char*); // Funksiyaning lokal e'loni.
  func(100); // Kompilyasiya xatosi,
              // chunki blokda funksiya qayta aniqlangan.
  func("\nQayta aniqlash ro'y berdi!"); // O'rinli.
  return 0; }
```

Qayta yuklanuvchi funksiya-a'zolar turli xil murojaat spetsifikatori bilan e'lon qilingan bo'lishi mumkin. Ular bitta ko'rinish sohasiga ega (sinf ichida) va shu sababli ular kompilyator tomonidan bitta funksiya deb qaraladi va qayta yuklanadi.

```
#include <iostream.h>
class Sinf
{
public:
  void Sinf_Azosi(double,char*);
  void Qayta_Yuklash1();
  void Qayta_Yuklash2();
private:
  void Sinf_Azosi(int);
```

```
};
void Sinf::Sinf_Azosi(double d,char*s){ cout<<"\ndouble="
"<<d<<"\t"
"<<"satr=" "<<s;}
void Sinf::Sinf_Azosi(int i) { cout<<"\nint=" "<<i<<endl; }
void Sinf::Qayta_Yuklash1() { Sinf_Azosi(10.5,"Qayta yuklash!"); }
void Sinf::Qayta_Yuklash2() { Sinf_Azosi(10); }
```

```
int main()
{
  Sinf sinf;
  sinf.Sinf_Azosi(100); // Xato, private – funksiya chaqirilmoqda
  sinf.Sinf_Azosi(12.5,"Satr qiymat"); // O'rinli qayta yuklash
  sinf.Qayta_Yuklash1();
  sinf.Qayta_Yuklash2();
  return 0;
}
```

Amalda sinfning statik funksiya-a'zolarini ham qayta yuklash mumkin, lekin bu ishdan mazmuniy samaraga erishib bo'lmaydi.

5.3. Konstruktorlarni qayta yuklash

Aksariyat hollarda qayta yuklash mexanizmidan konstruktorlarni qayta yuklashda foydalaniladi (destruktori qayta yuklash mumkin emas). Qayta yuklash qo'llanishidan maqsad foydalanuvchiga sinf vakilining (obyektning) turli variantlari bilan ishlashga imkon berishdir. E'tibor berilgan bo'lsa, oldin ham konstruktorlarning qayta yuklash uchragan edi, lekin unda qayta yuklash haqida gapirilmagan edi.

Konstruktori qayta yuklashga zarurat foydalanuvchiga konstruktor yordamida turlarni almashtirish imkonini berishdir. Masalan, argumentida Sinf sinf obyektiga murojaat bo'lgan funksiyani e'loni berilgan bo'lsin:

```
void func(Sinf*ob_k);
```

Ushbu funktsiyani boshqa turdagi argument bilan chaqirish kompilyasiya xatosiga olib keladi, chunki kompilyator turlarni qanday ravishda moslashtirishni aniqlay olmaydi. Kompilyatorga qandaydir T turini Sinf turiga olib kelish zarurligi haqida ko'rsatmani berish uchun Sinf sinfida

```
Sinf(T t);
konstruktorini e'lon qilish kerak. Bu holda kompilyator
func(t);
chaqirishni
func(Sinf(t));
mazmunida qabul qiladi va xatolik ro'y bermaydi.
class Sinf
{
    int x;
public:
    Sinf();
    Sinf(int);           // int turini Sinfturiga o'zgartirish
    Sinf(long);         // long turini Sinfturiga o'zgartirish
    Sinf(double);       // double turini Sinfturiga o'zgartirish
    int x_qiymati();
};
Sinf::Sinf(){x=0;}
Sinf::Sinf(int _x){x=_x;}
Sinf::Sinf(long _x){x=int(_x);}
Sinf::Sinf(double _x){x=_x;}
int Sinf::x_qiymati(){return x;}
void func(Sinf & ob_k) {cout<<ob_k.x_qiymati()<<"\n";}
int main()
{
    func(10); func(11L); func(12.15);
    Sinf sinf;
    func(sinf);           // Argument turini o'zgarishga hojat yo'q
    return 0;
}
```

)
 Dastur ishlashi natijasida ekranga
 10 11 12 0
 qiymatlari chop etiladi.
 Bu usuldan birorta berilganni bir sinf turidan ikkinchi sinf turiga o'tkazishda foydalanish mumkin. Quyida keltirilgan misol buni namoyon qiladi.

```
#include <iostream.h>
class Sinf1
{
    int x;
public:
    Sinf1(){x=5;};
    int qiymat(){return x;};
};
class Sinf2
{
    int z;
public:
    Sinf2(){z=10;};
    Sinf2(Sinf1 & ob_k); // O'zgartirish konstruktori
    int qiymat(){return z;};
};
Sinf2::Sinf2(Sinf1 & ob1) { z=ob1.qiymat();}
void func(Sinf2 & ob2) {cout<<ob2.qiymat()<<endl;}
int main()
{
    Sinf1 sinf1; func(sinf1);           // Sinf1 turi Sinf2 turiga o'zgartiriladi.
    Sinf2 sinf2; func(sinf2);           // Tur o'zgartirishga hojat yo'q.
    return 0;
}
```

Shuni qayd etish kerakki, konstruktorlarni qayta yuklashning turli variantlarini ishlatish mumkin, lekin ularning samarasi dastur tuzuvchi zimmasida bo'ladi.

Umuman olganda, shuni inobatga olish kerakki, foydalanuvchi tomonidan ko'rsatiladigan turlarni o'zgartirish kompilyator tomonidan quyidagi holatlardagina amalga oshiradi:

- obyektlarni initsializatsiya qilishda;
- funksiyalarni chaqirishda;
- funksiya qiymat qaytarishida.

Turni o'zgartirish konstruktoridan turlarni oshkor ravishda keltirishda foydalanish mumkin:

```
...
void main()
{
    Sinf1 sinf1;
    Sinf2 sinf12=(Sinf2) sinf1; // Oshkor ravishda turga keltirish
    ...
}
```

Quyida nusxalash konstruktorini qayta yuklashga misol keltirilgan. Dasturda to'rtburchak bilan ishlash uchun sinflar ishlatilgan.

```
#include <iostream.h>
class TurtBurchak
{
public:
    TurtBurchak();
    TurtBurchak(int,int);
    TurtBurchak(int,int,int,int);
    TurtBurchak(const TurtBurchak &);
    TurtBurchak(const TurtBurchak &,int,int);
private:
    int x,y,w,h;
};
TurtBurchak::TurtBurchak(){x=y=w=h=0;}
```

```
TurtBurchak::TurtBurchak(int _x,int _y) {x=_x; y=_y; w=h=100;}
TurtBurchak::TurtBurchak(int _x,int _y,int _w,int _h)
    {x=_x; y=_y; w=_w; h=_h;}
TurtBurchak::TurtBurchak(const TurtBurchak & tb)
    {x=tb.x; y=tb.y; w=tb.w; h=tb.h;}
TurtBurchak::TurtBurchak(const TurtBurchak & tb, int _x,int _y)
    { x=_x; y=_y; w=tb.w; h=tb.h; }

int main()
{
    TurtBurchak tburchak1(5,10,10,100);
    TurtBurchak tburchak2(tburchak1,15,200);
    TurtBurchak tburchak3(50,50);
    TurtBurchak tburchak4(tburchak3);
    return 0;
}
```

Ushbu dasturda TurtBurchak sinfi bir nechta konstruktorga ega bo'lib, to'rtburchak sohani yaratishda u foydalanuvchi talabiga moslashuvchanlik xususiyatiga ega. Qayta yuklanuvchi konstruktorlar variantlari ichidagi TurtBurchak(const TurtBurchak&) va TurtBurchak(const TurtBurchak&, int, int) konstruktorlari nusxalash konstruktorlari hisoblanadi. Bu imkoniyat, qo'yilgan masala mazmunidan kelib chiqqan holda turli xil nusxalash konstruktorlarini yaratishda qo'l keladi.

5-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Funksiyalarni qayta yuklash afzalligi nimada?
2. Sinfning funksiya-a'zolarini qayta yuklash shartlarini sanab o'ting.
3. Funksiyalarni qayta yuklash va qayta aniqlash o'rtasidagi farqini izohlang.
4. Sinf konstruktorini qayta yuklash zarurati nimada?

6. Operatorlarni qayta yuklash

6.1. Operatorlarni qayta yuklash tushunchasi

C++ tilida o'rnatilgan *operatorlarni qayta yuklash* imkoniyati mavjud. Operatorlar global ravishda yoki sinf chegarasida qayta yuklanishi mumkin. Qayta yuklash **operator** kalit so'zi bilan funksiya ko'rinishida amalga oshiriladi. Qayta yuklanuvchi funksiya *operator-funksiya* deb nomlanadi va nomi **operatorX** ko'rinishida bo'lishi kerak, bu erda **X** – qayta yuklanuvchi operator. C++ tilida qayta yuklanishi mumkin bo'lgan operatorlar ro'yxati 6.1-jadvalida keltirilgan. Masalan, qo'shish operatorini qayta yuklash uchun **operator+** nomli funksiyani aniqlash kerak bo'ladi. Agar qo'shish qiymat berish amali bilan kelgan holini qayta yuklash uchun **operator+=** ko'rinishida funksiya aniqlash zarur bo'ladi. Odatda kompilyator dastur kodida qayta yuklangan operatorlar uchraganda ularni oshkormas ravishda qo'llaydi. Zarur bo'lganda ularni oshkor chaqirish mumkin:

```
Nuqta nuqta1, nuqta2, nuqta3;
```

```
// Qayta yuklangan qo'shish operatorini oshkor ravishda chaqirish
nuqta3=nuqta1.operator+(nuqta2);
```

6.1-jadval. Qayta yuklanuvchi operatorlar

Operator	Tavsifi	Toifasi
,	vergul	binar
!	mantiqiy inkor	unar
!=	teng emas	binar
%	bo'lish qoldig'i	binar
%=	modulli bo'lish qiymat berish bilan	binar
&	razryadli VA	binar
&	adresni olish	unar
&&	mantiqiy VA	binar
&=	razryadli VA qiymat berish bilan	binar
()	funksiyani chaqirish	-
*	ko'paytirish	binar
*	vositali murojaat	binar

*=	ko'paytirish qiymat berish bilan	binar
+	qo'shish	binar
+	unar plyus	unar
++	inkrement	unar
+=	qo'shish qiymat berish bilan	binar
-	ayirish	binar
-	unar minus	unar
--	dekrement	unar
- =	ayirish qiymat berish bilan	binar
->	elementini tanlash	binar
->*	elementini ko'rsatkich orqali tanlash	binar
/	bo'lish	binar
/=	bo'lishqiymat berish bilan	binar
<	kichik	binar
<=	kichik yoki teng	binar
<<	razryad bo'yicha chapga surish	binar
<<=	chapga surish qiymat berish bilan	binar
=	qiymat berish	binar
==	teng	binar
>	katta	binar
>=	katta yoki teng	binar
>>	razryad bo'yicha o'ngga surish	binar
>>=	o'ngga surish qiymat berish bilan	binar
[]	massiv indeksi	-
^	razryadli istisno qiluvchi YOKI	binar
^=	razryadli istisno qiluvchi YOKI qiymat berish bilan	binar
	razryadli YOKI	binar
=	razryadli YOKI qiymat berish bilan	binar
	mantiqiy YOKI	binar
~	bitli mantiqiy inkor	binar
delete	dinamik obyekttni yo'qotish	-
new	dinamik obyekttni yaratish	-

6.2-jadvalda qayta yuklanmaydigan operatorlar ro'yxati keltirilgan.

6.2-jadval. Qayta yuklanmaydigan operatorlar

Operator(lar)	Tavsifi
.	a'zoni tanlash
::	ko'rinish sohasiga ruxsat berish operatori
*	ko'rsatkich bo'yicha a'zoni tanlash
?:	shart amali
#,##	preprotessor belgilari

Qayta yuklanadigan operatorlarning operator funksiyalari, **new** va **delete** operatorlaridan tashqari, quyidagi qoidalarga bo'ysunishi kerak:

1) operator-funksiya sinfning **static** bo'lmagan funksiya-a'zosi bo'lishi kerak yoki operator-funksiya sinf yoki sanab o'tiladigan toifaga kiruvchi (**int**) turdagi argument qabul qilishi mumkin yoki

operator-funksiya sinf yoxud sanab o'tiladigan toifaga kiruvchi turga ko'rsatkich yoki adres olish amali (&) bo'lgan argumentlarni qabul qilishi mumkin.

Masalan,
class Nuqta

{

public:

Nuqta operator<(Nuqta&); // '<' operatori uchun operator-funksiya

...

// Qo'shish operatorlarini e'lon qilish

friend Nuqta operator+(Nuqta&,int);

friend Nuqta operator+(int,Nuqta&);

};

Bu misolda '<' operatori sinfning funksiya-a'zosi sifatida e'lon qilingan, qo'shish operatori esa sinfning do'sti sifatida e'lon qilingan va u bitta operatorni qayta yuklashning bir nechta varianti bo'lishi mumkinligini ko'rsatadi;

2) operator-funksiya operatorning argumentlar (operandlar) sonini, ularning ustunligi va bajarilish tartibini o'zgartira olmaydi;

3) sinf funksiya a'zosi sifatida e'lon qilingan unar operatorning operator-funksiyasi parametrarga ega bo'lmasligi kerak. Agar operator-funksiya global funksiya bo'lsa, u faqat bitta parametrarga ega bo'ladi;

4) sinf funksiya a'zosi sifatida e'lon qilingan binar operatorning operator-funksiyasi bitta parametrarga ega bo'lishi kerak. Agar operator-funksiya global funksiya bo'lsa, u faqat ikkita parametrarga ega bo'ladi;

5) operator-funksiya kelishuv bo'yicha qiymat qabul qiluvchi parametrlarga ega bo'lmasligi kerak;

6) sinf funksiya a'zosi sifatida e'lon qilingan operator funksiyaning birinchi parametri (agar u mavjud bo'lsa) sinf turida bo'lishi kerak. Chunki, aynan shu sinf obyekti uchun mazkur operator chaqiriladi. Birinchi argument ustida hech qanday turga keltirish amali bajarilmasligi kerak;

7) qiymat berish operatorining operator-funksiyasidan tashqari barcha operator funksiyalar vorislik bilan o'tadi;

8) '=', '()', '[]' va '>' operatorlarning operator-funksiyalari sinfning **static** bo'lmagan funksiya a'zolari bo'lishi kerak (va ular global funksiya bo'la olmaydi).

Operatorlarni qayta yuklash orqali, sinf chegarasida operatorning mohiyatini tubdan o'zgartirib yuborish mumkin. Lekin bu ishni zarurat bo'lgandagina amalga oshirgan ma'qul. Aks holda bajariladigan amallarda mazmuniy xatolar yuzaga kelishi mumkin.

6.2. Binar operatorlarni qayta yuklash

Binar operatorning operator-funksiyasi sinfning statik bo'lmagan funksiya-a'zosi sifatida e'lon qilinganda u quyidagi sintaksisga ega bo'lishi kerak:

<qaytariladigan qiymat turi> operatorX(<parametr turi><parametr>);

Bu erda *<qaytariladigan qiymat turi>* – funksiya qaytaradigan qiymat turi, *X*– qayta yuklanadigan operator, *<parametr turi>* – parametr turi va *<parametr>* – funksiya parametri.

Funksiya parametriga operatorning o'ng tomonidagi obyekt uzatiladi, operatorning chap tomonidagi obyekt esa oshkormas ravishda **this** ko'rsatkichi bilan uzatiladi.

Agar operator-funksiya global deb e'lon qilinsa, u quyidagi ko'rinishga ega bo'ladi:

```
<qaytariladigan qiymat turi> operatorX (<parametr  
turi1><parametr1>, <parametr turi2><parametr2>);
```

Bu erda funksiya parametrlarining kamida bittasi operator qayta yuklanayotgan sinf turida bo'lishi kerak.

Garchi, operator-funksiya qaytaradigan qiymat turiga hech qanday cheklov bo'lmasa ham, odatda u sinf turida yoki sinfga ko'rsatkich bo'ladi.

Operator funksiyalami yozishning bir nechta misollarini keltiramiz. Bu misollar operatorlarni qayta yuklashning to'liq imkoniyatlarini ochib bermasa ham, uning muhim qirralarini ko'rsatadi.

Birinchi navbatda operator funksiyaning sinfning funksiya-a'zosi ko'rinishida aniqlashni ko'ramiz.

Quyidagi dasturda Nuqta sinfi uchun qo'shish (+) va ayirish (-) operatorlarini qayta yuklash amalga oshirilgan.

```
#include <iostream.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(){x=0; y=0;}
    Nuqta(int _x,int _y){x=_x; y=_y;}
    void Nuqta_Qiymati(int & _x,int & _y){_x=x; _y=y;}
    Nuqta operator+(Nuqta& ob);
    Nuqta operator-(Nuqta& ob);
};
Nuqta Nuqta::operator+(Nuqta& ob)
{
    Nuqta OraliqOb;
    OraliqOb.x=x+ob.x;
    OraliqOb.y=y+ob.y;
    return OraliqOb;
```

```
);
Nuqta Nuqta::operator-(Nuqta& ob)
{
    Nuqta OraliqOb;
    OraliqOb.x=x-ob.x;
    OraliqOb.y=y-ob.y;
    return OraliqOb;
}
int main()
{
    int x,y;
    Nuqta A(100,200), B(50,100),C;
    C=A+B; // Qayta yuklangan qo'shish operatori amal qiladi.
    C.Nuqta_Qiymati(x,y);
    cout<<" C=A+B amal natijasi: "<<"C.x="<<x<<" C.y="<<y<<endl;
    A=A-B; // Qayta yuklangan ayirish operatori amal qiladi.
    A.Nuqta_Qiymati(x,y);
    cout<<" A=A-B amal natijasi: "<<"A.x="<<x<<" A.y="<<y<<endl;
    return 0;
}
```

Dastur ishlashi natijasida ekranga quyidagi ko'rinishidagi natijalar chop etiladi:

```
C=A+B amali natijasi: C.x=150 C.y=300
A=A-B amali natijasi: A.x=50 A.y=100
```

Dasturdagi shu narsaga e'tibor berish kerakki, operator-funksiya parametri sinf obyektini adresini olish ko'rinishida aniqlangan. Umuman olganda, argument sifatida obyektning o'zini ham chaqirish mumkin, lekin funksiyadan chiqishda bu obyekt destruktorga yordamida yo'qotiladi. Funksiya parametri sinf obyektiga adresini olish ko'rinishida bo'lishining afzalligi shundaki, funksiya chaqirilganda unga obyekt emas, balki obyektga ko'rsatkich uzatiladi va sinf nusxasi uchun chaqiriladigan destruktorga ta'sir qilmaydi. Operator-funksiyalarning qaytaruvchi qiymati ayni shu sinf turida bo'lib, bu hol obyektlarni nisbatan murakkab

ifodalarda qo'llash imkonini beradi. Masalan, quyidagi amallar dastur uchun ruxsat etilgan til ko'rsatmasi hisoblanadi:

```
C=A+B-C;
```

Ikkinchi tomondan, quyidagi ifoda ham o'rinni:

```
(A+B).Nuqta_Qiymati(x,y);
```

Bu ifodada qo'shish operatoring operator-funksiyasidagi vaqtincha (oraliq) obyektning Nuqta_Qiymati() funksiyasi ishlatiladi.

Keyingi misol operator-funksiya parametri sifatida butun turdagi berilgan holatini namoyon qiladi. Bu berilgan operatorning o'ng tomonida kelishiga e'tibor berish kerak.

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

```
class Nuqta
```

```
{
```

```
int x,y;
```

```
public:
```

```
Nuqta(){x=0; y=0;}
```

```
Nuqta(int _x,int _y){x=_x; y=_y;}
```

```
void Nuqta_Qiymati(int &_x,int &_y){_x=x; _y=y;}
```

```
Nuqta operator+(Nuqta& ob);
```

```
Nuqta operator+(int n);
```

```
};
```

```
Nuqta Nuqta::operator+(Nuqta& ob)
```

```
{
```

```
Nuqta OraliqOb;
```

```
OraliqOb.x=x+ob.x;
```

```
OraliqOb.y=y+ob.y;
```

```
return OraliqOb;
```

```
}
```

```
Nuqta Nuqta::operator+(int n)
```

```
{
```

```
Nuqta OraliqOb;
```

```
OraliqOb.x=x+n;
```

```
OraliqOb.y=y+n;
```

```
return OraliqOb;
```

```
}
```

```
int main()
```

```
{
```

```
int x,y;
```

```
Nuqta A(100,200), B(50,100),C;
```

```
C=A+B; // Parametri sinfturidagi obyekt bo'lgan
```

```
// qayta yuklangan qo'shish operatori amal qiladi.
```

```
C.Nuqta_Qiymati(x,y);
```

```
cout<<" C=A+B amal natijasi: "<<"C.x="<<x<<" C.y="<<y<<endl;
```

```
C=A+30; // Parametri sanab o'tiladigan turidagi obyekt bo'lgan
```

```
// qayta yuklangan qo'shish operatori amal qiladi.
```

```
C.Nuqta_Qiymati(x,y);
```

```
cout<<" C=A+30 amal natijasi: "<<"C.x="<<x<<" C.y="<<y<<endl;
```

```
return 0;
```

```
}
```

Dastur ishlashi natijasida ekranga quyidagi ko'rinishidagi natijalar chop etiladi:

```
C=A+B amali natijasi: C.x=150 C.y=300
```

```
C=A+30 amali natijasi: C.x=130 C.y=230
```

Operator-funksiya parametri operatorning o'ng tomonidagi operand ekanligi sababli kompilyator quyidagi ko'rsatmalarni to'g'ri "tushunadi":

```
C=A+30;
```

```
Lekin kompilyator
```

```
C=30+A;
```

ko'rsatmasini qabul qilmaydi.

Bu muammoni operator-funksiyaning "ichki" imkoniyatlari bilan hal qilib bo'lmaydi. Muammoni do'st operator-funksiyalardan foydalanish orqali yechish mumkin. Ma'lumki, do'st funksiyalarga yashiringan this ko'rsatkichi uzatilmaydi. Shuning uchun binar operator-funksiyasi ikkita argumentga ega bo'lishi kerak – birinchisi chap operand uchun, ikkinchisi o'ng operand uchun.

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

```

class Nuqta
{
    int x,y;
public:
    Nuqta(){x=0; y=0;}
    Nuqta(int _x,int _y){x=_x; y=_y;}
    void Nuqta_Qiymati(int & _x,int & _y){_x=x; _y=y;}
    friend class Nuqta operator+(Nuqta& ob1, Nuqta& ob2);
    friend class Nuqta operator+(Nuqta& ob,int n);
    friend class Nuqta operator+(int n, Nuqta& ob);
};
Nuqta operator+(Nuqta& ob1,Nuqta& ob2)
{
    Nuqta OraliqOb;
    OraliqOb.x=ob1.x+ob2.x;
    OraliqOb.y=ob1.y+ob2.y;
    return OraliqOb;
}
Nuqta operator+(Nuqta& ob,int n)
{
    Nuqta OraliqOb;
    OraliqOb.x=ob.x+n;
    OraliqOb.y=ob.y+n;
    return OraliqOb;
}
Nuqta operator+(int n, Nuqta& ob)
{
    Nuqta OraliqOb;
    OraliqOb.x=ob.x+n;
    OraliqOb.y=ob.y+n;
    return OraliqOb;
}

```

```

int main()
{ int x,y;
  Nuqta A(100,200),B(50,100),C;
  C=A+B;
  C.Nuqta_Qiymati(x,y);
  cout<<" C=A+B amal natijasi: "<<"C.x="<<x<<" C.y="<<y<<endl;
  C=A+30;
  C.Nuqta_Qiymati(x,y);
  cout<<" C=A+30: "<<"C.x="<<x<<" C.y="<<y<<endl;
  C=30+A;
  C.Nuqta_Qiymati(x,y);
  cout<<" C=30+A: "<<"C.x="<<x<<" C.y="<<y<<endl;
  return 0;
}
Do'st funksiyalarni qayta yuklash hisobiga
C=A+30;
C=30+A;

```

til ko'rsatmalarini bajarish imkoniyati yuzaga keldi.

6.3. Taqqoslash va mantiqiy operatorlarni qayta yuklash

Taqqoslash va mantiqiy operatorlari, garchi binar operatorlar bo'lsa ham ular alohida qaraladi. Chunki ularga mos keluvchi operator-funksiyalar o'zlari aniqlangan sinf turidagi qiymatni emas, balki mantiqiy qiymatlarni qaytarishi kerak (yoki, true va false sifatida qabul qilinuvchi butun son qiymatini). Misol tariqasida, "==" va "&&" operatorlarini qayta yuklashni ko'raylik.

```

#include <iostream.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(int _x,int _y){x=_x; y=_y;}
    Nuqta(){x=0; y=0;}

```

```

    Qiymat_xy(int & _x, int & _y){_x=x; _y=y;}
    bool operator==(Nuqta ob);
    bool operator&&(Nuqta ob);
};
bool Nuqta::operator==(Nuqta ob) {return (x==ob.x && y==ob.y);}
bool Nuqta::operator&&(Nuqta ob) {return (x && ob.x) && (y &&
ob.y);}
int main()
{
    Nuqta nuqta1(10,20),nuqta2(10,25), nuqta3(10,20),nuqta4;
    if(nuqta1==nuqta2) cout<<"nuqta1 va nuqta2 o'zaro teng.\n";
    else cout<<"nuqta1 va nuqta2 o'zaro teng emas.\n";
    if(nuqta1==nuqta3) cout<<"nuqta1 va nuqta3 o'zaro teng.\n";
    else cout<<"nuqta1 va nuqta3 o'zaro teng emas.\n";
    if(nuqta1 && nuqta2) cout<<"nuqta1 && nuqta2 rost.\n";
    else cout<<"nuqta1 && nuqta2 yolg'on.\n";
    if(nuqta1 && nuqta4) cout<<"nuqta1 && nuqta4 rost.\n";
    else cout<<"nuqta1 && nuqta4 yolg'on.\n";
    return 0;
}

```

Dastur ishlashi natijasida ekranga quyidagilar chop etiladi:

```

nuqta1 va nuqta2 o'zaro teng emas.
nuqta1 va nuqta3 o'zaro teng.
nuqta1 && nuqta2 rost.
nuqta1 && nuqta4 yolg'on.

```

Operatorlarni qayta yuklash orqali koordinata nuqtalari orasidagi yangi mazmundagi munosabatlar aniqlandi.

6.4. Qiymat berish operatorini qayta yuklash

Qiymat berish operatori ham binar operator hisoblanadi, lekin uni qayta yuklash bir qator o'ziga xosliklarga ega:

- qiymat berish operatorining operator-funksiyasi global ravishda e'lon qilinishi mumkin emas, ya'ni u faqat sinfning funksiya-a'zosi bo'lishi kerak;

- qiymat berish operatorining operator-funksiyasi hosilaviy sinfga vorislik bilan o'tmaydi;

- kompilyator qiymat berish operatorining operator-funksiyasini hosil qilishi mumkin, agar u sinfda aniqlanmagan bo'lsa.

Kompilyator tomonidan kelishuv bo'yicha hosil qilingan qiymat berish operatori sinfning har bir statik bo'lmagan a'zolariga qiymat berish amalini bajaradi. Lekin, agar sinfda ko'rsatkichlar bo'ladigan bo'lsa, ularga bunday qiymat berish operatori ishlamaydi.

Qayd etish kerakki, qiymat berish operatori bajarilgandan keyin chap tomondagi operand o'zgaradi, chunki unga yangi qiymat beriladi. Shu sababli qiymat berish operatorining operator-funksiyasi uni chaqirilgan obyektga ko'rsatkichni qaytarishi shart. Buning uchun funksiya oshkormas ravishda sinf funksiyalariga birinchi parametr sifatida uzatiladigan *this* ko'rsatkichini qaytarishi etarli. O'z navbatida funksiyaning *this* ko'rsatkichini qaytarishi quyidagi ko'rinishdagi qiymat berish amallarini "*tushunish*" imkonini beradi:

```

nuqta1=nuqta2=nuqta3;

```

Quyidagi misol qiymat berish operatorini qayta yuklashni namoyon qiladi:

```

#include <iostream.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(int _x,int _y){x=_x; y=_y;}
    Nuqta(){x=0;y=0;}
    Qiymat_xy(int & _x,int & _y){_x=x; _y=y;}
    bool operator==(Nuqta ob);
    Nuqta & operator=(Nuqta & ob);
};

```

```

bool Nuqta::operator==(Nuqta ob){return (x==ob.x && y==ob.y);}
Nuqta & Nuqta::operator=(Nuqta & ob)
{
    if (this==&ob) return *this;
    x=ob.x; y=ob.y;
    return *this;
}
int main()
{
    int a,b;
    Nuqta nuqta1(10,20),nuqta2(20,25),nuqta3;
    nuqta3=nuqta2;
    if(nuqta2==nuqta3) cout<<"nuqta2 va nuqta3 o'zaro teng.\n";
    else cout<<"nuqta2 va nuqta3 o'zaro teng emas.\n";
    nuqta3=nuqta2=nuqta1;
    if(nuqta1==nuqta3) cout<<"nuqta1 va nuqta3 o'zaro teng.\n";
    else cout<<"nuqta1 va nuqta3 o'zaro teng emas.\n";
    nuqta3.Qiymat_xy(a,b);
    cout<<"nuqta3.x="<<a<<" nuqta3.y="<<b<<endl;
    return 0;
}

```

Dastur ishlashi natijasida ekranga
 nuqta2 va nuqta3 o'zaro teng.
 nuqta1 va nuqta3 o'zaro teng.
 nuqta3.x=10 nuqta3.y=20

xabarlari chop etiladi.

Qiymat berish amalini qayta yuklashda mazmunan xatoga olib
 keladigan

nuqta1=nuqta1;

ko'rinishdagi o'zini o'ziga yuklash holati alohida nazorat qilinishi kerak
 bo'ladi. Chunki qiymat berish operatori bajarilishida oldin chap tarafdagi
 operand xotirasi tozalanadi va keyinchalik shu joyga o'ng tomondagi

operandning haqiqatga to'g'ri kelmaydigan qiymatini joylashtirish ro'y
 beradi. Shu sababli, yuqoridagi misolda operator=() funksiyasi

if (this==&ob) return *this;

nazorat ko'rsatmasiga ega va u xatolik ro'y berishiga yo'l qo'ymaydi.

6.5. Unar operatorlarni qayta yuklash

Unar operatorlar uchun faqat bitta operand kerak bo'ladi. Unar ope-
 ratorni sinfning funksiya-a'zosi ko'rinishida qayta yuklashda bu yagona
 operand - bu amalni chaqirgan obyektning o'zi hisoblanadi. Shu sababli,
 unar operatorning operator-funksiyasi statik bo'lmagan funksiya-a'zo
 sifatida e'lon qilinadi va u quyidagi ko'rinishga ega bo'ladi:

<qaytaruvchi qiymat turi>operatorX();

bu erda <qaytaruvchi qiymat turi> funksiya qaytaradigan qiymat turi, X -
 qayta yuklanayotgan unar operator.

Agar operator-funksiya global ravishda e'lon qilinganda, u quyidagi
 sintaksisga javob berishi kerak:

<qaytaruvchi qiymat turi>operatorX(<parametr turi><parametr>);

bu erda <parametr turi> - parametr turi va <parametr> - funksiya
 parametri.

Quyida qo'shish va ayrish unar operatorlarni qayta yuklashga misol
 keltirilgan:

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

```
class Nuqta
```

```
{
```

```
    int x,y;
```

```
public:
```

```
    Nuqta(int _x,int _y){x=_x; y=_y;}
```

```
    Nuqta(){x=0; y=0;}
```

```
    Qiymat_xy(int & _x,int & _y){_x=x; _y=y;}
```

```
    Nuqta operator+();
```

```
    Nuqta operator-();
```

```
};
```

```
Nuqta & Nuqta::operator+(){ x+=x; y+=y; return *this;}
```

```

Nuqta & Nuqta::operator-(){ x=-x; y=-y; return *this;}
int main()
{
    int a,b;
    Nuqta n1(-10,20);
    n1+=n1; // Qayta yuklangan qo'shish operatorini chaqirish
    n1.Qiymat_xy(a,b);
    cout<<"n1.x="<<a<<" n1.y="<<b<<endl;
    n1=-n1; // Qayta yuklangan ayirish operatorini chaqirish
    n1.Qiymat_xy(a,b);
    cout<<"n1.x="<<a<<" n1.y="<<b<<endl;
    return 0;
}

```

Dastur ishlashi natijasida ekranda

n1.x=-10 n1.y=20

n1.x=10 n1.y=-20

satrlari paydo bo'ladi.

Xuddi shu natijalarga global operator funksiyalarni sinfning do'st funksiyalari ko'rinishida e'lon qilish orqali erishish mumkin:

```
friend Nuqta operator+(Nuqta & ob);
```

```
friend Nuqta operator-(Nuqta & ob);
```

Bu funksiyalar aniqlanishi

```
friend Nuqta operator+(Nuqta & ob) { ob.x+=ob.x; ob.y+=ob.y; return
```

```
ob;}
```

```
friend Nuqta operator-(Nuqta & ob) { ob.x=-ob.x; ob.y=-ob.y; return
```

```
ob;}
```

Ushbu funksiyalarni chaqirish natijalari yuqoridagi funksiyalar bilan bir xil bo'ladi.

6.6. Inkrement va dekrement operatorlarini qayta yuklash

Inkrement va dekrement operatorlari, ularning prefiks va postfiks ko'rinishlari bo'lishi hisobiga qayta yuklash nuqtai-nazaridan alohida

kategoriyaga tushadi. Prefiks va postfiks ko'rinishlarni farqlash uchun quyidagi qoidalarga amal qilinadi:

- prefiks ko'rinishni qayta yuklash, odatdagi unar operatorni qayta yuklash bilan bir xil;

- postfiks ko'rinish uchun operator-funksiya qo'shimcha int turidagi parametrga ega bo'ladi.

Amalda bu parametr ishlatilmaydi va funksiyani chaqirishda uning qiymati 0 bo'ladi (zarurat bo'lganda ishlatilishi mumkin). Bu parametrning vazifasi – kompilyatorga operatorni postfiks ko'rinish uchun ishlatilayotganligini bildirishdir.

Quyidagi misolda Nuqta sinfi uchun inkrement va dekrement operatorlarini qayta yuklash ko'rsatilgan:

```

#include <iostream.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(int _x,int _y){x=_x; y=_y;}
    Nuqta(){x=0; y=0;}
    Qiymat_xy(int & _x,int & _y){_x=x; _y=y;}
    Nuqta & operator++(); // Prefiks inkrement uchun
    Nuqta operator++(int); // Postfiks inkrement uchun
    Nuqta & operator--(); // Prefiks dekrement uchun
    Nuqta operator--(int); // Postfiks dekrement uchun
};
Nuqta & Nuqta::operator++(){ x++; y++; return *this;}
Nuqta Nuqta::operator++(int) { Nuqta Oraliq=*this; ++*this; return
Oraliq;}
Nuqta & Nuqta::operator--(){ x--; y--; return *this;}
Nuqta Nuqta::operator--(int) { Nuqta Oraliq=*this; --*this; return
Oraliq;}
int main()
{

```

```

int a,b;
Nuqta n1(-10,20, n2(15,25), n3;
++n1; // Prefiks inkrement operatorini chaqirish
n1.Qiymat_xy(a,b);
cout<<"(++n1).x="<<a<<"\t(++n1).y="<<b<<endl;
n1++; // Postfiks inkrement operatorini chaqirish
n1.Qiymat_xy(a,b);
cout<<"(n1++).x="<<a<<"\t(n1++).y="<<b<<endl;
n3--n2; // Prefiks dekrement operatorini chaqirish
n3.Qiymat_xy(a,b);
cout<<"n3--n2;=>n3.x="<<a<<"\tn3.y="<<b<<endl;
(n3=n1--).Qiymat_xy(a,b); // Postfiks dekrement operatorini
// chaqirish
cout<<"(n3=n1--).x="<<a<<"\t(n3=n1--).y="<<b<<endl;
n1.Qiymat_xy(a,b);
cout<<"n1.x="<<a<<"\tn1.y="<<b<<endl;
n2.Qiymat_xy(a,b);
cout<<"n2.x="<<a<<"\tn2.y="<<b<<endl;
return 0;
}

```

Dastur ishlashi natijasida ekranga

```

(++n1).x=11    (++n1).y=21
(N1++).x=12    (n1++).y=22
n3--n2;=>n3.x=14  n3.y=24
(n3=n1--).x=12 (n3=n1--).y=22
n1.x=11    n1.y=22
n2.x=14    n2.y=24

```

xabarlari chiqadi.

Dasturda inkrement va dekrement operatorlarining prefiks va postfiks ko'rinishlarini qayta yuklashni amalga oshirishda o'ziga xos yo'l tanlangan. Masalan, inkrement operatorining prefiks ko'rinishi uchun aniqlangan operator-funksiyaning qaytaradigan qiymati sinf obyektiga murojaat, chunki inkrement operatorining postfiks ko'rinish uchun

aniqlangan operator-funksiya shu funksiyani chaqiradi va o'zgargan obyektini qaytarib olishi kerak. Umuman olganda, bu funksiyalarni bir-biriga bog'liqmas ravishda aniqlash mumkin:

```
Nuqta Nuqta::operator++(){ x++; y++; return *this; }
```

```
Nuqta Nuqta::operator++(int) { x++; y++; return *this; }
```

Lekin bu variantda bir xil amallar ketma-ketligini takror yoziladi va inkrement operatorini turlicha talqin qilish bilan bog'liq xatolarni yuzaga kelishiga zamin bo'ladi. Ma'qul variant - bu operatorning prefiks ko'rinishining operator-funksiyasida operator mazmuni aniqlanadi va postfiks ko'rinishni qayta yuklash unga tayanadi.

Endi inkrement va dekrement operatorlarini do'st funksiyalar orqali qayta yuklashni ko'ramiz. Shunga e'tibor berish kerakki, do'st funksiyaga adres olish (&) ko'rinishidagi argument uzatilishi va u o'zgartirilib funksiya tomonidan qaytarilishi kerak bo'ladi.

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

```
class Nuqta
```

```
{
```

```
int x,y;
```

```
public:
```

```
Nuqta(int _x,int _y){x=_x; y=_y;}
```

```
Nuqta(){x=0; y=0;}
```

```
Qiymat_xy(int & _x,int & _y){_x=x; _y=y;}
```

```
friend Nuqta & operator++(Nuqta &); // Prefiks inkrement
```

```
friend Nuqta operator++(Nuqta&, int); // Postfiks inkrement
```

```
friend Nuqta & operator--(Nuqta&); // Prefiks dekrement
```

```
friend Nuqta operator--(Nuqta&, int); // Postfiks dekrement
```

```
};
```

```
Nuqta & operator++(Nuqta & ob) { ob.x++; ob.y++; return ob; }
```

```
Nuqta operator++(Nuqta & ob,int) { Nuqta Oraliq=ob; ++ob; return
```

```
Oraliq; }
```

```
Nuqta & operator--(Nuqta & ob) { ob.x--; ob.y--; return ob; }
```

```
Nuqta operator--(Nuqta & ob,int) { Nuqta Oraliq=ob; --ob; return
```

```
Oraliq; }
```

```

int main()
{
    int a,b;
    Nuqta n1(-10,20);n2(15,25),n3;
    ++n1;    //Prefiks inkrement operatorini chaqirish
    n1.Qiymat_xy(a,b);
    cout<<"(++n1).x="<<a<<"\t(++n1).y="<<b<<endl;
    n1++;    // Postfiks inkrement operatorini chaqirish
    n1.Qiymat_xy(a,b);
    cout<<"(n1++).x="<<a<<"\t(n1++).y="<<b<<endl;
    n3--n2; // Prefiks dekrement operatorini chaqirish
    n3.Qiymat_xy(a,b);
    cout<<"n3--n2; => n3.x="<<a<<"\t n3.y="<<b<<endl;
    (n3=n1--).Qiymat_xy(a,b);    // Postfiks dekrement operatorini

```

chaqirish

```

    cout<<"(n3=n1--).x="<<a<<"\t (n3=n1--).y="<<b<<endl;
    n1.Qiymat_xy(a,b);
    cout<<"n1.x="<<a<<"\t n1.y="<<b<<endl;
    n2.Qiymat_xy(a,b);
    cout<<"n2.x="<<a<<"\t n2.y="<<b<<endl;
    return 0;
}

```

Dastur ishlashi natijasida ekranga xuddi oldingi misoldagidek xabarlar chop etiladi.

Yuqorida qayd qilingandek, int turidagi argument odatda belgilash uchun ishlatiladi, lekin zarur bo'lganda ishlatilishi mumkin. Bunga misol:

```

#include <iostream.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(int _x,int _y){x=_x; y=_y;}
    Nuqta(){x=0; y=0;}

```

```

    Qiymat_xy(int &_x,int &_y){_x=x; _y=y;}
    Nuqta &operator++(int);
};
Nuqta &Nuqta::operator++(int n)
{
    if (n!=0) { x+=n; y+=n; }
    else {x++; y++;}
    return *this;}
int main()
{
    Nuqta n1(10,20);
    n1.operator++(100); //100 soniga inkrement
    return 0;
}

```

Bu holatning o'ziga xosligi shundaki, postfiks inkrement operatorining operator-funksiyasini oshkor ravishda chaqirishga to'g'ri keladi. Chunki kompilyator «n1++(100);» ko'rsatmasining operatorgacha bo'lgan qismini alohida ajratib «n1++;» ko'rsatmasi deb tushunadi.

6.7. Indekslish operatorini qayta yuklash

Kvadrat qavslar ('[', ']') bilan yoziladigan indekslash operatori binar operator hisoblanadi va u qayta yuklanishida operator-funksiya sinfnining bitta argumentli statik bo'lmagan funksiya-a'zosi sifatida aniqlanishi kerak. Funksiya argumenti ixtiyoriy turda bo'lishi va u sinf obyektlari massivining indeksi deb qabul qilinadi. Quyidagi misol buni namoyon qiladi:

```

#include <iostream.h>
class BS_Massiv
{
public:
    int max_index;
    int * k_butun;
    BS_Massiv(int elem_soni);

```

```

~BS_Massiv(){delete k_butun;}
int & operator[](int index);
};
BS_Massiv::BS_Massiv(int elem_soni)
{
    k_butun=new int[elem_soni];
    max_index=elem_soni;
}
int & BS_Massiv::operator[](int index)
{
    static int index_xato=-1;
    if(index>=0 && index<max_index) return k_butun[index];
    else
    {
        cout<<"Xato: Massiv chegarasidan chiqildi!" <<endl;
        return index_xato;
    }
}
main()
{
    BS_Massiv vector(5);
    for(int i=0;i<5;i++) vector[i]=i;
    for(int i=0;i<=5;i++) cout<<" vector["<<i<<"]= "<<vector[i]<<endl;
    return 0;
}

```

Dasturda 5 ta butun son turidagi elementlardan tashkil topgan vector massivi BS_Massiv sinfining obykti sifatida e'lon qilingan va unga qiymatlar berilib, keyin chop qilingan. Indeks argumenti i=5 bo'lganda xatolik haqida xabar beriladi. Dastur ishlashi natijasida ekranga quyidagilar chiqadi:

```

vector[0]=0
vector[1]=1

```

```

vector[2]=2
vector[3]=3
vector[4]=4
Xato: Massiv chegarasidan chiqildi!
vector[5]=-1

```

Shunga e'tibor berish kerakki, operator[] funksiyasi adres orqali qiymat qaytaradi (sonni o'zini emas) va shu sababli bu funksiyani qiymat berish operatorining ikki tomonida ham qo'llash imkoni yuzaga keladi.

6.8. Funksiyalarni chaqirish operatorini qayta yuklash

Qavslar vositasida amalga oshiriladigan funksiyani chaqirish operatori quyidagi sintaksisga ega bo'lgan binar operator hisoblanadi:

<ifoda>(<ifodalar ro'yxati>).

Bu erda *<ifoda>* – birinchi operand, hamda *<ifodalar ro'yxati>* – majburiy bo'lmagan ikkinchi operanddir. Funksiyani chaqirish operatorining operator-funksiyasi sinfnining statik bo'lmagan funksiya-a'zo ko'rinishida e'lon qilinishi kerak. Funksiyani chaqirish operatorini qayta yuklashga zarurat, odatda ko'p parametrlarni talab qiladigan amallarni bajarishda yuzaga keladi.

Funksiyani qayta yuklash operatori qayta yuklanganda, u faqat qavs ichidagi o'zi e'lon qilingan sinf obyektlariga bo'lgan murojaatni o'zgartiradi, lekin funksiya chaqirilishi jarayoniga ta'sir qilmaydi.

Funksiyani chaqirish operatorini qayta yuklashga misol ko'raylik:

```

#include <iostream.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(int _x,int _y){x=_x; y=_y;}
    Nuqta(){x=0; y=0;}
    Qiymat_xy(int & _x,int & _y){_x=x; _y=y;}
    Nuqta &operator()(int dx, int dy)
    {
        x+=dx; y+=dy;
    }
}

```

```

    return *this;
}
};
int main()
{
    int x,y;
    Nuqta n1,n2;
    n2=n1(5,10); // Operator-funksiyani chaqirish
    n2.Qiymat_xy(x,y);
    cout<<"1-chaqirishda: N2.x="<<x<<" N2.y="<<y<<endl;
    n2=n2(1,2); // Qayta yuklangan funksiyani chaqirish operatori
    n2.Qiymat_xy(x,y);
    cout<<"2-chaqirishda: N2.x="<<x<<" N2.y="<<y<<endl;
    return 0;
}

```

Dastur ishlashi natijasida ekranga quyidagi satrlar chiqadi:

1-chaqirishda: n2.x=5 n2.y=10

2-chaqirishda: n2.x=6 n2.y=12

Keltirilgan misol uchun «Nuqta n1(5,10);» va «n1(5,10);» ifodalari o'zaro chalkashtirmaslik kerak. Birinchi ifoda sinfning parametrli konstruktoriga murojaatni anglatadi, ikkinchisi – qayta yuklangan operator funksiyani chaqirish kerakligini bildiradi. Misol shuni ko'rsatadiki, funksiyani chaqirish operatorini qayta yuklash amalga oshirilgan sinf obyektiga funksiya vositasida murojaat qilish mumkin.

6.9. Sinf a'zolariga murojaat operatorlarini qayta yuklash

Sinf a'zolariga murojaat operatorlarini qayta yuklash a'zolariga murojaat operatorini ("→") qayta yuklash orqali amalga oshiriladi ("→" operatori qayta yuklanmaydi). Bu operator unar hisoblanadi va uning operator-funksiyasi sinfning statik bo'lmagan a'zosi qilib e'lon qilinishi kerak. Mos operator funksiyani ko'rinishi quyidagicha:

```
<sinf nomi> * operator->() {<til ko'rsatmalari>;};
```

Bu erda <sinf nomi>- operator tegishli bo'lgan sinf nomi. Sinf a'zolarini tanlash operatorini qayta yuklash, odatda qo'llanishi ayni

paytda o'rinli yoki yo'qligini nazorat qiluvchi "ongli" ko'rsatkichlarni amalga oshirishda ishlatiladi.

Sinf a'zolarini tanlash operatorini qayta yuklashga misol keltiramiz:

```

#include <iostream.h>
class Nuqta
{
    int x,y;
public:
    Nuqta(int _x,int _y){x=_x; y=_y;}
    Nuqta(){x=0; y=0;}
    Qiymat_X(){return x;}
    Qiymat_Y(){return y;}
    Nuqta &operator->();
};

```

```
Nuqta * Nuqta::operator->()
```

```
{
    cout<<"Obyekt elementiga murojaat: ";
    return this;
}
```

```
int main()
```

```
{
    Nuqta n(5,10);
    // murojaat operatorining operator-funksiyasini chaqirish
    cout<<"n->x = "<<n->Qiymat_X()<<endl;
    cout<<"n->y = "<<n->Qiymat_Y()<<endl;
    return 0;
}
```

Dastur bajarilganda, ekranga quyidagi satrlar chop etiladi:

Obyekt elementiga murojaat: n->x = 5

Obyekt elementiga murojaat: n->y = 10

Kompilyator tomonidan «n->Qiymat_X()» ko'rsatmasini

(n.operator->())->Qiymar_X()

ko'inishida talqin qilinishi operator funksiyani qanday bajarilishining ichki mohiyatini ochib beradi.

6.10. new va delete operatorlarini qayta yuklash

Xotirani dinamik ravishda ajratish va tozalash (qaytarish) vazifasini amalga oshiruvchi **new** va **delete** operatorlari bajarilganda mos ravishda standart aniqlangan operator **new()** (yoki **operator new []()**- massiv uchun qo'llanilganda) va operator **delete()** (yoki **operator delete []()**- massiv uchun qo'llanilganda) maxsus funksiyalari chaqiriladi.

Bundan keyin alohida zarurat bo'lmasa, bu funksiyalarning massiv varianti qaralmaydi va bittasi uchun aytilgan fikrlar ikkinchisi uchun ham o'rinli deb hisoblanadi.

Umuman olganda, **new** va **delete** operatorlari ikki xil variantda qayta yuklanishi mumkin:

```
::operator new()           // Global (standart);
::operator delete()        // Global (standart);
<sinfnomi>::operator new() // Sinf funksiyasi;
<sinfnomi>::operator delete() // Sinf funksiyasi.
```

Sinfda aniqlangan operator **new()** operator-funksiya sinfnining statik a'zosi bo'lib, u sinf obyektlari uchun global **::operator new()** funksiyasini yashiradi (berkitadi).

Global **::operator new()** – operator funksiyaning o'zi ham qayta yuklanishi (sinfdan tashqarida) va qayta yuklanuvchi funksiyalarning turli prototipga ega bir nechta variantlari bo'lishi mumkin.

Foydalanuvchi tomonidan aniqlanadigan **new** operatorning operator-funksiyasi **void*** qiymatini qaytarishi kerak va birinchi parametr sifatida **size_t** turidagi parametrga ega bo'lishi kerak. Oxirgi parametr **<stddef.h>** sarlavha faylida **unsigned int** ko'inishida aniqlangan. **new** operatorini qayta yuklash uchun quyidagi prototipdan foydalaniladi:

```
void * operator new(size_t size); // Bitta obyekt
void * operator new[](size_t size); // Obyektlar massivi
```

Bu prototiplar **<new.h>** sarlavha faylida joylashgan. Shu sababli, agar **new** va **delete** operatorlarini qayta yuklash zarur bo'lganda bu faylni dasturga qo'shish kerak bo'ladi.

Shunga e'tibor berish kerakki, **new** operator-funksiyasidagi **size** parametrining baytlardagi o'lchamini (**sizeof(size)**), ya'ni xotiradan ajratilishi kerak sohaning baytlardagi o'lchamini operator-funksiya uchun kompilyatorning o'zi hisoblab beradi. Operator funksiyaga bu parametrning qo'yilishiga sabab shundaki, hosilaviy sinflar **operator new()** va **operator new[]()** funksiyalarini vorislik bilan oladi va hosilaviy sinf obyektlarining o'lchami tayanch sinf obyektlari o'lchamidan farq qilishi mumkin.

Agar xotiradagi oldin ajratilgan joyni qaytadan "taqsimlash" zarur bo'lsa, **new** operatorining qo'shimcha parametrga ega bo'lgan *joylashuvchi shaklidan* foydalanish mumkin. Mos operator-funksiya sintaksisi quyidagi ko'inishiga ega:

```
void* operator new(size_t size, void* p); // Obyektlar uchun
void* operator new(size_t Type_size, void* p); // Obyektlar massivi uchun
```

Odatda **new** operatorining joylashuvchi shaklidan global obyektlar uchun, turga keltirish amali bajarilgan holda qo'llaniladi. Bu variantda absolyut adres bo'yicha oldindan ajratilgan joyga obyektни joylashtirish amalga oshiriladi. Ko'rsatilgan adres uyumdan bo'lishi shart emas. Joylashadigan obyekt o'lchami ko'rsatilgan sohada o'lchamidan kichik bo'lgan hollarda ajratilgan sohani korrekt ravishda tozalash foydalanuvchi zimmasiga yuklatiladi, chunki **delete** operatori to'g'ri ishlashiga kafolat yo'q.

Qayta yuklangan **new** operatorini chaqirish sintaksisi quyidagicha:

```
<::>new<tur uzunligi><tur nomi><(<initsializator>)>
```

yoki

```
<::>new<tur uzunligi> (<tur nomi>) (<(<initsializator>)>).
```

Bu erda: **<::>** – shart bo'lmagan, ko'inish sohasiga ruxsat berish operatori; **<tur uzunligi>** – operator **new()** funksiyasining **size** parametri, ko'rsatilmaligi mumkin; **<tur nomi>** – xotira ajratiladigan berilganning turi; **<(<initsializator>)>** – **<tur nomi>** turining konstrukturi uchun

uzatiladigan boshlang'ich qiymatlar ro'yxati (ro'yxat ko'rsatilmashligi mumkin).

Sintaksis shuni ko'rsatadiki, **new** operatorini obyektga boshlang'ich qiymat berish bilan chaqirish mumkin. Lekin, bu operatorni obyektlar massivi uchun chaqirilganda initsializatsiyani ishlatib bo'lmaydi va obyektlarning boshlang'ich qiymatlari noaniq bo'ladi.

Global **new** va **delete** operatorlarini qayta aniqlash va qayta yuklashga misol keltiramiz:

// Global new operatorini qayta aniqlash

```
void* operator new(size_t size)
```

```
{
    cout<<"Xotiradan "<<size<<" bayt ajratishga so'rov bo'ldi\n";
    return malloc(size);
}
```

```
void operator delete(void *p)
```

```
{
    cout<<"Xotirani bo'shatish!\n";
    free(p);
}
```

// Global new operatorini qayta yuklash

```
void* operator new(size_t size, char * fname, int line)
```

```
{
    cout<<fname<<" faylining "<<line<<"-qatorida \n";
    cout<<size<<" bayt ajratishga so'rov bo'ldi!\n";
    return malloc(size);
}
```

```
int main()
```

```
{
    char fayl_nomi[]="new_quyk.cpp";
    int qator=5;
    long * plnt = new long;      // Global new operatorini chaqirish
    delete plnt;                // Global delete operatorini chaqirish
}
```

```
plnt = new(fayl_nomi,qator) long; // Qayta yuklangan new
operatorini
```

// chaqirish.

```
delete plnt;
```

// Global delete operatorini

chaqirish

```
return 0;
```

```
}
```

Dastur ishlashi natijasida ekranga quyidagi satrlar chop etiladi:

Xotiradan 4 bayt ajratishga so'rov bo'ldi

Xotira bo'shatildi!

new_quyk.cpp faylining 5-qatorida

4 bayt ajratishga so'rov bo'ldi!

Xotira bo'shatildi!

Navbatdagi misolda global **new** operatorining joylashuvchi shaklidan foydalanish ko'rsatilgan.

// Global new operatorini qayta aniqlash

```
void* operator new(size_t size,void * krst)
```

```
{
    cout<<"Obyekt ko'rstatilgan adresga joylandi\n";
    return krst;
}
```

```
void Tizimni_tekshirish(){cout<<"Tizim normal ishlaypti!\n";}
```

```
class Nuqta
```

```
{
    int x,y;
public:
    Nuqta(int _x,int _y)
```

```
{
    x=_x; y=_y;
    cout<<" Obyektlar berilgan qiymatlar bilan initsializatsiyalandi.\n";
    cout<<" x="<<x<<" y="<<y<<"\n";
}
Nuqta()
```

```

{
    x=0; y=0;
    cout<<" Obyektlar kelishuv bo'yicha initsializatsiyalandi.\n";
    cout<<" x="<<x<<" y="<<y<<"\n";
}
~Nuqta() { cout<<"Nuqta::~Nuqta() ishladi\n"; }
};
Nuqta nuqta;           // Global obyekt
int main()
{
    Tizimni_tekshirish();    // Qandaydir ishni bajarish
    Nuqta * krst=new(nuqta) Nuqta(10,20);
    // Endi obyektini initsializatsiyalash mumkin!
    krst->Nuqta::~Nuqta();
    return 0;
}
Dastur natijasi quyidagi xabarlar bo'ladi:
Obyektlarni kelishuv bo'yicha initsializatsiyalandi.
x=0 y=0
Tizim normal ishlaypti!
Obyekt ko'rstatilgan adresga joylandi
Obyektlar berilgan qiymatlar bilan initsializatsiyalandi.
x=10 y=20
Nuqta::~Nuqta() ishladi
Nuqta::~Nuqta() ishladi
Keyingi dastur lokal new operatorini qayta yuklashni sinf ichida
joylashuvchi shaklidan foydalangan holda amalga oshirishga misol:
class Satr
{
    union { char ch; char buf[81]; };
public:
    Satr(char c='\0'): ch(c) { cout<<"Satr sinfining belgisi
konstruktori\n"; }

```

```

    Satr(char * s) {cout<<"Satr sinfining satrli konstruktori\n";
    strcpy(buf,s); }
    ~Satr() { cout<<"Satr::~Satr()"<<endl; }
    // new operatorining joylashuvchi sintaksisi
    void* operator new(size_t, void * buffer) { return buffer; }
};
char satr_buffer[sizeof(Satr)]; // Xotira buferi
int main()
{
    Satr * krst=new(satr_buffer) Satr("C++"); // Satrni buferga
    joylashtirish
    cout<<"satr_buffer"<<satr_buffer<<endl;
    krst->Satr::~Satr(); // Destruktorni oshkor ravishda chaqirish
    krst=new(satr_buffer)Satr("c"); // 'c' belgisini satr boshiga
    joylashtirish
    cout<<"satr_buffer[0]="<<satr_buffer[0]<<endl;
    cout<<"Buferning yangi qiymati"<<endl;
    cout<<"satr_buffer="<<satr_buffer<<endl;
    krst->Satr::~Satr(); // Destruktorni oshkor ravishda chaqirish
    return 0;}
Dasturda global obyekt satr_buffer satr yaratiladi va shu xotira
sohasiga Satr sinfi obyektini "C++" qiymati bilan joylashtiriladi. Oshkor
ravishda sinf destruktoriga chaqirish orqali sinf obyektini yo'qotiladi. Keyin,
xuddi shu adresda Satr sinfining ikkinchi obyektini 'c' belgisi bilan
yaratiladi, xotiraning satr_buffer adresli sohasida satr qiymat chop etiladi
va sinf obyektini yo'qotiladi.
Dastur ishlashi natijasida ekranda
Satr sinfining satrli konstruktori
satr_buffer=C++
Satr::~Satr()
Satr sinfining belgisi konstruktori
satr_buffer[0]=c
Buferning yangi qiymati

```

satr_buffer=c++
Satr::~Satr()
xabarlar chop etiladi.

6-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Operatorlarni qayta yuklash mohiyati nimada?
2. Nima uchun ayrim operatorlar qayta yuklanmaydi?
3. Qaysi holatlarda operator –funksiya do'st funksiya ko'rinishida aniqlanadi?
4. Binar operatorning operator-funksiyasining birinchi parametriga qanday talablar qo'yiladi?
5. Inkrement va dekrement amallarini qayta yuklashda ularning postfix yoki infix ko'rinishlari qanday inobatga olinadi?
6. Qiymat berish operatorini qayta yuklashning o'ziga xosligi nimada?

7. Polimorfizm va virtual funksiyalar

7.1. Vaqtli va kechiktirilgan bog'lanishlar. Dinamik polimorfizm

C++ tilida polimorfizm ikki usulda qo'llab-quvvatlanadi. Birinchisi, funksiya va operatorlarni qayta yuklash vositasi bilan kompilyasiya paytida. Polimorfizmning bu ko'rinishiga *statik polimorfizm* deyiladi, chunki u dastur bajarilishidan oldin, ya'ni kompilyatsiya va jamlash (komponovka) paytida funksiya identifikatorlarini fizik adreslar bilan *vaqtli bog'lash* orqali amalga oshiriladi. Ikkinchisida, dastur bajarilishida virtual funksiyalar vositasida. Dastur kodida virtual funksiyaga murojaatni uchratgan kompilyator, bu chaqirishni faqat belgilab qo'yadi, funksiya identifikatorini adres bilan bog'lashni dasturni bajarish bosqichiga qoldiradi. Bu jarayonga *kechiktirilgan bog'lanish* deyiladi.

Virtual funksiya – bu shunday funksiyaki, uni chaqirish va mos amallarni bajarish, uni chaqirgan obyekt turiga bog'liq bo'ladi. Obyekt dastur bajarilish jarayonida qaysi funksiyani chaqirish kerakligini aniqlaydi. Polimorfizmning bu ko'rinishiga *dinamik polimorfizm* deyiladi. Dinamik polimorfizمنى amalga oshirishning asosi sifatida C++ tilidagi tayanch sinfga ko'rsatkichni aniqlanishidir. O'z navbatida, bu ko'rsatkich nafaqat tayanch sinfga, balki shu sinfning vorisi bo'lgan ixtiyoriy sinf obyektiga ko'rsatishi mumkin. Sinflarning bu xossasi vorislikdan kelib chiqadi, chunki har qanday voris sinf obyektini tayanch sinf turida bo'ladi. Dasturni yig'ish paytida (komponovka paytida) tayanch sinfga ko'rsatkich egasi bo'lgan foydalanuvchi tomonidan qaysi sinf obyektini yaratilishi noma'lum bo'ladi. Shu sababli, ko'rsatkich o'z obyektini bilan faqat dastur ishlashi paytidagini, ya'ni dinamik ravishda bog'lanishi mumkin. Tarkibida hech bo'lmaganda bitta virtual funksiyasi bo'lgan sinf *polimorf sinf* deyiladi. Har bir polimorf turdagi berilganlar uchun kompilyator *virtual funksiyalar jadvalini* yaratadi va shu jadvalga sinfning har bir obyektiga yashiringan ko'rsatkichni joylashtiradi. Kompilyator virtual funksiyalar jadvaliga ko'rsatkichni initsializatsiya qiluvchi kod bo'lagini polimorf sinf konstruktori boshlanishiga joylashtiradi. Virtual funksiya chaqirilganda ushbu kod virtual funksiyalar jadvaliga ko'rsatkichni topadi va jadvaldagi mos funksiya adresiga o'tish bilan funksiya chaqirilishi ro'y beradi.

Jadval 7.1. Sinf virtual funksiyalar jadvali

Sinf shajarasi va ko'rsatkichlar	Ko'rsatkich qiymatining turi	Sinflardagi virtual funksiyalarning adreslari		
		f1()	f2()	...
Tayanch * tayanch;	Tayanch	Adres ₁₁	Adres ₂₁	...
Voris1 * voris1; // <i>Tayanch vorisi</i>	Voris1	Adres ₁₂	Adres ₂₂	...
Voris2 * voris2; // <i>Tayanch vorisi</i>	Voris2	Adres ₁₃	Adres ₂₃	...
...	...	...	...	...

Ma'lumki, hosilaviy sinf obyektini yaratilishida tayanch sinf konstruktori chaqiriladi. Ayni shu bosqichda virtual funksiyalar jadvali va unga ko'rsatkich hosil bo'ladi. Hosilaviy sinf konstruktori chaqirilgandan keyin virtual funksiyalar jadvaliga ko'rsatkich, shu sinf obyektini uchun qayta aniqlangan virtual funksiya variantini ko'rsatish uchun moslanadi (agar u mavjud bo'lsa).

Ko'rinib turibdiki, kechiktirilgan bog'lanishni amalga oshirish muayyan bir resurslar sarflashni taqozo etadi va undan oqilona foydalanish zarur bo'ladi.

7.2. Virtual funksiyalar

Mazmunidan kelib chiqqan holda virtual funksiyaga boshqa tavsiflarni berish mumkin:

1) chaqirish interfeysi (prototipi) ma'lum, amalga oshirilishi umumiy ko'rinishda berilishi mumkin bo'lmasdan, faqat konkret holatlardagini aniqlanadigan funksiyalarga *virtual funksiyalar* deyiladi;

2) *virtual funksiya* – bu chaqirilishi uchun qanday ifoda ishlatilishidan qat'iy nazar obyekt uchun to'g'ri (mos) funksiya chaqirilishini kafolatlaydigan funksiyadir.

Faraz qilaylik, tayanch sinfda funksiyaning virtual e'loni, hosilaviy sinfda ham xuddi shu funksiya e'loni bo'lsin. U holda hosilaviy sinf obyektlari uchun hosilaviy sinf funksiyasi chaqiriladi, agar ular chaqirilishida tayanch sinfga ko'rsatkich yoki murojaat ishlatilgan bo'lsa ham.

```
class Tayanch
{
public:
    Tayanch(int _x) {x=_x;}
    virtual int Qiymat_X(){return x;}
    virtual void Chop_X();
private:
    int x;
};

void Tayanch::Chop_X() {cout<<"Tayanch::x="<<Qiymat_X()<<"\n";}

class Hosila1: public Tayanch
{
public:
    Hosila1(int _x): Tayanch(_x){}
    void Chop_X();
};

void Hosila1::Chop_X() {cout<<"Hosila1::x="<<Qiymat_X()<<"\n";}

class Hosila2: public Tayanch
{
public:
    Hosila2(int _x): Tayanch(_x){}
    void Chop_X();
};

void Hosila2::Chop_X() {cout<<"Hosila2::x="<<Qiymat_X()<<"\n";}

int main(int argc, char* argv[])
{
    Tayanch * tayanch=new Tayanch(10);
    Hosila1 * hos1=new Hosila1(20);
    Hosila2 * hos2=new Hosila2(30);
    tayanch->Chop_X();
    tayanch=hos1;
    tayanch->Chop_X();
}
```

```

tayanch=hos2;
tayanch->Chop_X();
return 0;}

```

Hosilaviy sinflardagi `Chop_X()` funksiyalari virtual hisoblanadi, chunki u `Tayanch` tayanch sinfida virtual deb e'lon qilingan. Virtual funksiyalarni chaqirish uchun quyidagi kodlar ishlatilgan:

```

tayanch=hos1;
tayanch->Chop_X();
tayanch=hos2;
tayanch->Chop_X();

```

Sinflardagi `Chop_X()` funksiyalari virtual bo'lganligi uchun har bir obyektning o'z funksiyasi chaqiriladi. `Hosila1` va `Hosila2` sinflarida `Chop_X()` funksiyalar `Tayanch` tayanch sinfidagi `Chop_X()` funksiyasini qayta aniqlaydi. Agar hosilaviy sinfda `Chop_X()` funksiyasi qayta aniqlanmasa, kelishuv bo'yicha tayanch sinfdagi `Chop_X()` funksiyasi amal qiladi.

Yuqoridagi dastur ishlashi natijasida ekranga

```

Tayanch::x=10
Hosila1::x=20
Hosila2::x=30

```

natijalar chop etiladi:

Funksiyalarni ko'rsatkich va adresni olish amallari yordamida chaqirishda quyidagi qoidalarga amal qilinadi:

- virtual funksiyani chaqirish uni chaqirayotgan obyekt turiga mos ravishda hal qilinadi;

- virtual bo'lmagan funksiyalarni chaqirish ko'rsatkich turiga mos ravishda amalga oshiriladi.

Virtual funksiyalar faqat qandaydir sinfga tegishli obyektlar uchun chaqirilishini inobatga oladigan bo'lsak, global yoki statik funksiyalarni virtual deb e'lon qilish mumkin emas. `virtual` kalit so'zi hosilaviy sinfda funksiyani qayta aniqlashda ishlatilishi mumkin, lekin bu majburiy emas.

`Tayanch` sinfdagi virtual funksiyalar shu sinfda aniqlanishi kerak, agar ular sof virtual deb e'lon qilingan bo'lmasa. Hosilaviy sinfda e'lon

qilingan funksiya tayanch sinfdagi virtual funksiyani qayta aniqlaydi, agar uning nomi virtual funksiya nomi bilan mos tushsa, ular bir xil miqdordagi va turlari mos kelgan parametrlarga ega bo'lsa. Agar funksiya virtual funksiyadan hattoki bitta parametri bilan farq qilsa, u holda hosilaviy sinfdagi bu funksiya yangi hisoblanadi va qayta aniqlash ro'y bermaydi. Hosilaviy sinfdagi funksiya virtual funksiyadan faqat qaytaruvchi qiymati bilan farqlanmaydi, ularning parametrlar ro'yxati turlicha bo'lishi kerak.

Quyidagi misolda virtual funksiya tayanch sinfdagi o'zi bilan bir xil prototipga ega virtual funksiyani qayta aniqlaydi.

```

#include <iostream.h>
class Tayanch
{
    int x;
public:
    virtual void Qiymat(int _x) { x=_x; cout<<"Tayanch::x = "<<x<<"\n";
}

    virtual void Chop_Qilish(Tayanch * pOb) { Qiymat(10); }
};

class Hosila: public Tayanch
{
    int x,y;
public:
    virtual void Qiymat(int _x,int _y)
    {
        x=_x; y=_y;
        cout<<"Hosila::x = "<<x<<" Hosila::y = "<<y<<"\n";
    }

    virtual void Chop_Qilish(Tayanch * pOb) { Qiymat(15,20); }
};

int main()
{
    Tayanch * pOb1=new Tayanch;
    Tayanch * pOb2=new Hosila;
}

```

```
//Tayanch sinfidan virtual Chop_Qilish() funksiyasini chaqirish
pOb1->Chop_Qilish(pOb1);
// Hosila sinfidan virtual Chop_Qilish() funksiyasini chaqirish
pOb2->Chop_Qilish(pOb1);
// Hosila sinfidan virtual Chop_Qilish() funksiyasini chaqirish
pOb2->Chop_Qilish(pOb2);
return 0;
}
```

Dastur ishlashi natijasida ekranga quyidagilarni chop etadi:

Tayanch::x = 10

Hosila::x =15 Hosila::y = 20

Hosila::x =15 Hosila::y = 20

Keltirilgan misolda tayanch va hosilaviy sinflar ikkita bir xil nomdagi virtual funksiyalarga ega. Lekin kompilyator ularni turlicha talqin qiladi. Qiymat() funksiyasining prototipi hosilaviy sinfda o'zgarganligi sababli, u mutlaqo boshqa virtual funksiya deb qaraladi. Ikkinchi tomondan, hosilaviy sinfdagi Chop_Qilish() funksiyasi tayanch sinfdagi mos virtual funksiyaning qayta aniqlanishi deb qaraladi.

7.2. Virtual va novirtual funksiyalar

Quyidagi misolda ko'rsatkich orqali chaqirilganda virtual va novirtual funksiyalar o'zini qanday tutishi ko'rsatilgan:

```
#include <iostream.h>
class Tayanch
{
public:
    virtual void Virtual_Fun(){cout<<"Tayanch::Virtual_Fun()\n"; }
    void NoVirtual_Fun(){cout<<"Tayanch::NoVirtual_Fun()\n";}
};
class Hosila: public Tayanch
{
public:
    virtual void Virtual_Fun(){ cout<<"Hosila::Virtual_Fun()\n";}
```

```
void NoVirtual_Fun(){cout<<"Hosila::NoVirtual_Fun()\n";}
};
int main()
{
    Hosila hosila;
    Hosila * pHosila = &hosila;
    Tayanch * pTayanch = &hosila;
    // Hosila sinfidan Virtual_Fun() funksiyasini chaqirish
    pTayanch->Virtual_Fun();
    //Tayanch sinfidan NoVirtual_Fun() funksiyasini chaqirish
    pTayanch->NoVirtual_Fun();
    // Hosila sinfidan Virtual_Fun() funksiyasini chaqirish
    pHosila->Virtual_Fun();
    // Hosila sinfidan NoVirtual_Fun() funksiyasini chaqirish
    pHosila->NoVirtual_Fun();
    return 0;
}
```

Dastur bajarilishi natijasida ekranga quyidagi satrlar chop etiladi:

Hosila::Virtual_Fun()

Tayanch::NoVirtual_Fun()

Hosila::Virtual_Fun()

Hosila::NoVirtual_Fun()

Shunga e'tibor berish kerakki, Virtual_Fun() funksiyasi qaysi sinfga –Tayanch yoki Hosila ko'rsatkich orqali chaqirilishidan qat'iy nazar Hosila sinfidan Virtual_Fun() funksiyasi chaqiriladi. Bunga sabab – Virtual_Fun() funksiyasi virtual va pTayanch hamda pHosila ko'rsatkichlari Hosila turidagi obyektga ko'rsatadi. Ikkinchi tomondan, tayanch sinfga ko'rsatkichi pTayanch garchi novirtual funksiyalarga ega hosilaviy sinf obyektiga ko'rsatsa ham, tayanch sinfdagi mos funksiyaning chaqirilishi.

Ko'rish sohasiga ruxsat berish operatori vositasida kechiktirilgan bog'lanishni man qilish mumkin:

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

```

class Tayanch
{
public:
    virtual void Virtual_Fun(){cout<<"Tayanch::Virtual_Fun()\n";}
};
class Hosila: public Tayanch
{
public:
    virtual void Virtual_Fun(){cout<<"Hosila::Virtual_Fun()\n";}
};
int main()
{
    Tayanch * pTayanch = new Hosila;
    // Hosila sinfidan Virtual_Fun() funksiyasini chaqirish
    pTayanch->Virtual_Fun();
    //Tayanch sinfidan Virtual_Fun() funksiyasini chaqirish
    pTayanch->Tayanch::Virtual_Fun();
    return 0;
}
Dastur ekranga
Hosila::Virtual_Fun()
Tayanch::Virtual_Fun()
xabarlarini chop etadi.
Ko'rinib turibdiki,
pTayanch->Tayanch::Virtual_Fun();
ko'rsatmasi kechiktirilgan bog'lanishni yo'qqa chiqaradi.
Sinflar shajarasida virtual funksiyalar nomi bilan bir xil qayta
yuklanuvchi funksiyalarni e'lon qilish mumkin. Lekin, bunday novirtual
funksiyalar yashiringan bo'ladi.
Yuqoridagi fikrlarni tasdiqlaydigan misol keltiramiz:
#include <iostream.h>
#include <string.h>
class Tayanch

```

```

{
public:
    Tayanch(char * nom){strcpy(Nom,nom);}
    virtual void Fun(char c)
    {
        cout<<"Virtual "<<Nom<<":Fun "<<c<<" parametrini qabul qildi.\n";
    }
protected:
    char Nom[20];
};
class Hosila1: public Tayanch
{
public:
    Hosila1(char * nom):Tayanch(nom){}

    void Fun(const char * s)
    { cout<<Nom<<":Fun "<<s<<" parametrini qabul qildi.\n";}
    void Fun(int n){cout<<Nom<<":Fun "<<n<<" parametrini qabul
qildi.\n";}
    virtual void Fun(char c)
    {
        cout<<"Virtual "<<Nom<<":Fun "<<c<<" parametrini qabul qildi.\n";
    }
};
class Hosila11: public Hosila1
{
public:
    Hosila11(char * nom):Hosila1(nom){}
    void Fun(const char * s)
    {cout<<Nom<<":Fun "<<s<<" parametrini qabul qildi.\n";}
    void Fun(double d)
    {cout<<Nom<<":Fun "<<d<<" parametrini qabul qildi.\n";}
}

```

```

virtual void Fun(char c)
{
    cout<<"Virtual "<<Nom<<":Fun "<<c<<" parametrini qabul qildi.\n";
}
};
int main()
{
    Tayanch tayanch("Tayanch");
    Hosila1 hosila1("Hosila1");
    Hosila11 hosila11("Hosila11");
    tayanch.Fun('X');
    hosila1.Fun('Y');
    hosila1.Fun(10);
    hosila1.Fun("Hos1");
    hosila11.Fun('Z');
    hosila11.Fun(10.1234);
    hosila11.Fun("Hos11");
    return 0;
}

```

Dastur bajarilgandan keyin ekranda quyidagi satrlar paydo bo'ladi:

Virtual Tayanch::Fun X parametrini qabul qildi.

Virtual Hosila1::Fun Y parametrini qabul qildi.

Hosila1::Fun 10 parametrini qabul qildi.

Hosila1::Fun Hos1 parametrini qabul qildi.

Virtual Hosila11::Fun Z parametrini qabul qildi.

Hosila11::Fun 10.1234 parametrini qabul qildi.

Hosila11::Fun Hos11 parametrini qabul qildi.

Keltirilgan misolda chiziqli vorislikni hosil qiluvchi uchta sinf aniqlangan. Tayanch sinfida Fun(char) virtual funksiya e'lon qilingan. Hosila1 sinfi Fun(char) virtual funksiyasining o'z variantini va ikkita qayta yuklanuvchi novirtual Fun(const char*) va Fun(int) funksiyalarni e'lon qilgan. O'z navbatida, Hosila11 sinfi Fun(char) virtual funksiyasining o'z

variantini va ikkita qayta yuklanuvchi novirtual Fun(const char*) va Fun(double) funksiyalarni e'lon qilgan. Garchi, Fun(const char*) funksiyasi Hosila1 sinfidagi analogi bilan to'la ustma-ust tushsa ham, u Hosila11 sinfida qayta e'lon qilingan. Chunki, Hosila1 sinfida xuddi shu nomdagi virtual va novirtual funksiyalar mavjudligi sababli, Fun(const char*) funksiyasi yashiringan bo'ladi. Xuddi shunday, Fun(char) virtual funksiyasi Hosila1 va Hosila11 sinflarida qaytadan e'lon qilishga to'g'ri keladi, chunki ular ham sinflarda xuddi shu nomdagi qayta yuklanuvchi funksiyalarni mavjudligi sababli yashiringan bo'ladi.

Agar voris sinflardagi virtual funksiyalar e'lonlari o'chirilsa, funksiyaning belgi argumentli chaqirishda, amalda Hosila1 sinfidagi Fun(int) funksiyasini chaqirish ro'y beradi. Tayanch sinfidagi virtual funksiyani chaqirish zarur bo'lsa, ko'rish sohasiga ruxsat berish amaldan foydalanish mumkin:

```
Hosila1.Tayanch::Fun('Y');
```

7.3. Dinamik polimorfizmni qo'llash

Dinamik polimorfizm vositasida dastur bajarilishini boshqarishning moslanuvchan boshqarishni amalga oshirish mumkin. Quyida, butun sonlarning bog'langan ro'yxati ko'rinishida amalga oshirilgan *stek* va *navbat* tuzilmalari ustida ishlash qaralgan. Ma'lumki, navbat – "*birinchi kelgan – birinchi ketadi*", stek – "*oxirda kelgan – birinchi ketadi*" tamoyili bo'yicha berilganlarni saqlash va qayta ishlashni amalga oshiruvchi tuzilmalar hisoblanadi. Dasturda bog'langan ro'yxatni yaratish, unga qiymat joylashtirish va o'chirishni amalga oshiruvchi *Ruyxat* tayanch sinfi va uning vorislari sifatida navbat hosil qiluvchi mos ravishda *Navbat* va *Stek* sinflari yaratiladi. Garchi bu tuzilmalar bilan ishlash turlicha amalga oshirilsa ham, ularni ishlatishda yagona interfeysdan foydalaniladi.

```

#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>
class Ruyxat
{

```

```

public:
    Ruyxat(){Boshi=Oxiri=Keyingi=0;}
    virtual void Joylash(int n)=0;
    virtual bool Olish(int& n)=0;
    void Qiymat_n(int n){Son=n;}
    int n_Qiymat(){return Son;}
    Ruyxat * Boshi;
    Ruyxat * Oxiri;
    Ruyxat * Keyingi;
private:
    int Son;
};
class Navbat: public Ruyxat
{
public:
    void Joylash(int n);
    bool Olish(int& n);
};
void Navbat::Joylash(int n)
{
    Ruyxat * Yangi;
    Yangi=new Navbat; // Navbatning yangi elementini yaratish
    Yangi->Qiymat_n(n);
    Yangi->Keyingi=NULL;
    if(Oxiri) Oxiri->Keyingi=Yangi; // Elementni navbatning oxiriga
    joylash
    Oxiri=Yangi;
    if(!Boshi) Boshi=Yangi;
}
bool Navbat::Olish(int& n)
{
    Ruyxat * Element;

```

```

if(!Boshi){n=0; return false;}
n=Boshi->n_Qiymat();
Element=Boshi;
Boshi=Boshi->Keyingi;
delete Element;
return true;
}
class Stek: public Ruyxat
{
public:
    void Joylash(int n);
    bool Olish(int& n);
};
void Stek::Joylash(int n)
{
    Ruyxat * Yangi;
    Yangi=new Stek; // Stekning yangi elementini yaratish
    Yangi->Qiymat_n(n);
    Yangi->Keyingi=NULL;
    if(Boshi) Yangi->Keyingi=Boshi; // Elementni stek boshiga joylash
    Boshi=Yangi;
    if(!Oxiri) Oxiri=Yangi;
}
bool Stek::Olish(int& n)
{
    Ruyxat * Element;
    if(!Boshi){n=0; return false;}
    n=Boshi->n_Qiymat();
    Element=Boshi;
    Boshi=Boshi->Keyingi;
    delete Element;
    return true;
}

```

```

}
int main()
{
    Ruyxat * ruyxat;
    Navbat navbat;
    Stek stek;
    int son;
    char stek_navbat;
    cout<<"Sonlarni navbat va stekka joylash:\n";
    do
    {
        cout<<"Sonni kiriting(0-tamom): ";
        cin>>son;
        if(son)
        {
            do
            {
                cout<<"Joylashtirish? Stekka(S) yoki Navbatga(N):";
                cin>>stek_navbat;
            }
            while (stek_navbat!='S' && stek_navbat!='s'
                && stek_navbat!='N' && stek_navbat!='n');
            switch(stek_navbat)
            {
                case 'S': case 's': ruyxat=&stek; break;
                case 'N': case 'n': ruyxat=&navbat;
            }
            ruyxat->Joylash(son);
        }
    }
    while (son);
    for(;;)

```

```

{
    do
    {
        cout<<"O'qish? Stekdan(S) yoki Navbatdan(N):\n";
        cout<<"Dasturdan chiqish (Q):\n";
        cin>>stek_navbat;
    }
    while(stek_navbat!='S' && stek_navbat!='s' &&
        stek_navbat!='N' && stek_navbat!='n' &&
        stek_navbat!='Q' && stek_navbat!='q');
    switch(stek_navbat)
    {
        case 'S':
        case 's': ruyxat=&stek; break;
        case 'N': case 'n': ruyxat=&navbat; break;
        case 'Q': case 'q': return 0;
    }
    if(ruyxat->Olish(son)) cout<<son<<endl;
    else cout<<"Ro'yxat bo'sh!"<<endl;
}
}

```

Dastur kirish oqimidan butun sonlarni o'qiydi va foydalanuvchi tanlagan ro'yxatga- navbat yoki stekka joylashtiradi. Sonlarni kiritish jarayoni navbatdagi son tariqasida 0 soni kiritilganda to'xtaydi. Keyinchalik, foydalanuvchi ko'rsatgan ro'yxatdan son qiymatlari o'qiladi va ekranga chop qilinadi. Dinamik polimorfizm ruyxat ko'rsatkichi navbat yoki stek obyektlariga ko'rsatishiga mos ravishda virtual Olish() va Joylash() funksiyalarini chaqirishida namoyon bo'ladi.

7.4. Virtual destruktorelar

Konstruktorlar virtual bo'lmaydi, lekin destruktorelar virtual bo'lishi mumkin va aksariyat holatlarda shunday bo'ladi. Tayanch sinfiga ko'rsatkich hosilaviy sinf obyektiga ko'rsatib turganda, agar destruktore

virtual qilib e'lon qilingan bo'lsa, hosilaviy sinf destruktori chaqiriladi. Hosilaviy sinf destruktori o'z navbatida tayanch sinf destruktoriga chaqiradi va obyekt to'g'ri (to'laligicha) o'chiriladi. Aks holda ko'rsatkich turiga mos ravishda tayanch sinf destruktoriga chaqiriladi, hosilaviy sinf uchun ajratilgan xotira bo'shatilmay qoladi – xotirada band qilingan, lekin qayta ishlatish imkoni bo'lmagan xotira bo'lagi – “xotira axlati” paydo bo'ladi.

Misol ko'raylik:

```
#include <iostream.h>
class Tayanch
{
    int * px;
public:
    Tayanch(int _x) {px=new int; *px=_x; }
    /*virtual*/
    ~Tayanch() { cout<<"Tayanch sinf destruktoriga ishladi!\n"; delete px; }
};
class Hosila: public Tayanch
{
    int * pxx;
public:
    Hosila (int n):Tayanch(n) { pxx=new int; *pxx=n*n; }
    ~Hosila() { cout<<"Hosila sinf destruktoriga ishladi!\n"; delete pxx; }
};
int main()
{
    Tayanch * pTayanch=new Hosila(5);
    delete pTayanch;
    return 0;
}
```

Hosilaviy sinf - Hosila sinfida destruktoriga virtual deb e'lon qilinmagan va delete pTayanch;

til ko'rsatmasi bajarilishi natijasida ekranga

Tayanch sinf destruktoriga ishladi!

xabari chiqadi. Bu holat xotiradagi Hosila sinf obyektiga uchun ajratilgan xotira bo'shatilmay qolganligini bildiradi. Agar tayanch sinf destruktoriga virtual deb e'lon qilinsa, obyektga o'chirish to'g'ri ro'y beradi – oldin hosilaviy sinf destruktoriga, keyin tayanch sinf destruktoriga bajariladi. Dasturning ekranga chiqaradigan

Hosila sinf destruktoriga ishladi!

Tayanch sinf destruktoriga ishladi!

xabarlarini buni isbotlaydi.

7.5. Abstrakt sinflar va sof virtual funksiyalar

Sinflar, shu turga tegishli bo'lgan obyektlarning o'zaro bajaradigan amallari qoidalarini oldindan aniqlab berish uchun yaratilishi mumkin. Bunday sinflarga *abstrakt sinflar* deyiladi. Abstrakt sinflarning obyektlarini yaratib bo'lmaydi. Ular faqat hosilaviy sinflarni yaratish uchun xizmat qiladi.

Abstrakt sinf kamida bitta virtual funksiyaga ega bo'lishi kerak. Tayanch sinfning sof virtual funksiyalari hosilaviy sinflarda albatta aniqlanishi kerak, aks holda hosilaviy sinf ham virtual hisoblanadi.

Sof virtual funksiya quyidagi sintaksis bilan e'lon qilinadi:

virtual <funksiya nomi>(<parametrlar ro'yxati>)=0;

Misol ko'raylik. Faraz qilaylik, sinflar shajarasini yaratish zarur bo'lsin va tayanch sinf umumiy funksional imkoniyatlarni ta'minlashi kerak bo'lsin. Lekin, tayanch sinfi shu darajada umumlashgan bo'lib, natijada undagi ayrim funksiyalarni konkretlashtirish imkoni bo'lmashligi mumkin. Bunday tayanch sinfi abstrakt sinf uchun eng yaxshi nomzod hisoblanadi:

Misol uchun jonivorlar shajarasini tavsiflovchi *Jonivor* abstrakt tayanch sinf va uning vorislari *Kuchuk* va *Mushuk* sinflarini e'lon qilishni ko'raylik.

```
class Jonivor
{
public:
```

```

Jonivor(char * nomi)
{
    Nomi=new char[15];
    strcpy(Nomi,nomi);
};
virtual void Ovozi()=0;
virtual void Ozuqasi()=0;
protected:
    char * Nomi;
};
class Kuchuk : public Jonivor
{
public:
    Kuchuk(char * nomi):Jonivor(nomi){};
    void Ovozi(){cout<<Nomi<<" ovozi: Vov"<<endl;}
    void Ozuqasi(){cout<<Nomi<<" ozuqasi: Go'sht"<<endl;}
};
class Mushuk: public Jonivor
{
public:
    Mushuk(char * nomi):Jonivor(nomi){};
    void Ovozi(){cout<<Nomi<<" ovozi: Miyov"<<endl;}
    void Ozuqasi(){cout<< Nomi<<" ozuqasi: Sut"<<endl;}
};
int main()
{
    Mushuk mushuk("Baroq");
    Kuchuk kuchuk("Tuzik");
    mushuk.Ovozi();
    mushuk.Ozuqasi();
    kuchuk.Ovozi();
    kuchuk.Ozuqasi();
}

```

)

Bu misolning e'tiborli tomoni shundaki, **Jonivor** sinfida e'lon qilingan **ovozi()** va **ozuqasi()** funksiya-a'zolar abstrakt funksiyalardir. Bu funksiyalarni konkretlashtirishning imkoni yo'q, chunki **Jonivor** sinfi hayvonlar shajarasini aniqlab beruvchi, umumlashtiruvchi sinf va konkret hayvon aniqlanmaguncha uning ovozinig qanday bo'lishi va nima bilan oziqlanishini bilib bo'lmaydi. Lekin, aksariyat hayvonlar ovoz chiqaradi va albatta oziqlanadi. Shu sababli, **ovozi()** va **ozuqasi()** funksiyalar umumiy bo'lib, u **Jonivor** sinfida mavhum holda e'lon qilingan. Misolda bu funksiyalar **Kuchuk** va **Mushuk** sinflarida konkretlashtirilgan (majburiy ravishda).

Dastur ishlashi natijasida ekranga quyidagi xabarlar chiqadi:

Baroq ovozi: Miyov

Baroq ozuqasi: Sut

Tuzik ovozi: Vov

Tuzik ozuqasi: Go'sht

Abstrakt sinf bilan bog'liq yana bir o'ziga xos holat shundan iboratki, agar abstrakt sinf konstruktori bevosita yoki bilvosita sof abstrakt funksiyani chaqirsa, nima ro'y berishini oldindan aytishning iloji yo'q.

Sof abstrakt funksiyalar tavsifiga "zid" ravishda bunday funksiyalar abstrakt sinfda nafaqat e'lon qilinishi, balki aniqlanishi ham mumkin. Ular quyidagi sintaksis asosida bevosita chaqirilishi mumkin:

```

<abstrakt sinf nomi>::<abstrakt funksiya nomi>
(<parametrlar ro'yxati>)

```

Odatda bu sintaksisdan sof virtual destruktorga ega sinflar shajarasini yaratishda foydalaniladi:

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

```
class Tayanch
```

```
{
```

```
public:
```

```
Tayanch(){};
```

```
virtual ~Tayanch()=0; // Sof virtual destruktur
```

```
};
```

```
Tayanch::~Tayanch(){} // Destruktorni aniqlash
class Hosila: public Tayanch
```

```
{
public:
    Hosila(){};
    ~Hosila(){};
};
```

```
void main()
{
    Hosila * pHosila = new Hosila;
    delete pHosila;
}
```

Ma'lumki, destruktorki virtual bo'lganda, oldin hosilaviy sinf destruktorki, keyin tayanch sinf destruktorki bajariladi. Sof virtual destruktorkining aynan tayanch sinfda aniqlanishi, uning qandaydir amalga oshirilgan variantini yaratadiki, u destruktorklar ketma-ketligini to'g'ri bajarilishini ta'minlaydi.

Xulosa sifatida abstrakt sinflarga qo'llaniladigan qoidalarni keltiramiz:

- abstrakt sinfni funksiyaga uzatiladigan argumentning turi sifatida ishlatib bo'lmaydi;
- abstrakt sinfni funksiya qaytaradigan qiymatning turi sifatida ishlatib bo'lmaydi;
- obyekt turini oshkor ravishda abstrakt sinf turiga keltirish mumkin emas;
- abstrakt sinf obyektini yaratib bo'lmaydi;
- abstrakt sinfga ko'rsatkich yoki adres olish amalini e'lon qilish mumkin.

7-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Kompilyatsiya jarayonidagi vaqtli va kechiktirilgan bog'lanishlar tushunchalarini izohlab bering.

2. C++ tilida dinamik polimorfizmni amalga oshirish mexanizmi qanday?

3. Sinflar shajarasida virtual va novirtual funksiyalar amal qilishini tushuntiring.

4. Sinf destruktorkini virtual qilib aniqlanishiga sabab nima?

5. Sof virtual funksiya vazifasi nimadan iborat?

8. Istisno holatlar

8.1. Istisno holatlarni qayta ishlash

Istisno – bu dasturning normal bajarilishini uzadigan holatdir. C++ tilida istisno holatni qayta ishlashning *yakuniy model* deb nomlanuvchi varianti amalga oshirilgan: istisno holat ro'y berib, u qayta ishlangandan keyin boshqaruv istisno ro'y bergan kodga qaytib kelmaydi, ya'ni dasturning aynan uzilish ro'y bergan joydan davom etishi ta'minlanmaydi. Ikkinchi tomondan, C++ tilida apparat qurilmalardan keladigan istisnolar (uzilishlar) qayta ishlanmaydi, faqat qandaydir funksiya tomonidan yuzaga keltirilgan istisnolar qayta ishlanadi. Istisnolarni boshqarish uchun C++ tilida uchta kalit so'z ishlatiladi: **try**, **catch** va **throw**.

Istisno holatni yuzaga keltirishi mumkin bo'lgan kod bloki **try** kalit so'zi bilan belgilanadi. Bu blok figurali qavsga olinadi va *himoyalangan* yoki *try-blok* deyiladi:

```
try
{
    // Himoyalangan soha
}
```

try-blok ichida chaqiriladigan funksiyalar ham himoyalangan sohaga tegishli bo'ladi. Himoyalangan sohadagi funksiya yoki til ko'rsatmasi istisno yuzaga keltirishi mumkin. Agar istisno ro'y bersa, mos funksiya yoki til ko'rsatmasining bajarilishi to'xtatiladi, **try**-blokdagi qolgan ko'rsatmalar chetlab o'tiladi (bajarilmaydi) va boshqaruv blokdan tashqariga uzatiladi.

catch kalit so'zi bevosita **try**-blokdan keyin keladi va istisno ro'y berganda boshqaruv o'tishi kerak bo'lgan kod bo'lagini belgilaydi. Kod bo'lagi figurali qavsga olinadi va *catch-blok* (istisnoni qayta ishlovchi) deyiladi. **catch** kalit so'zidan keyin qavs ichiga olingan istisno turi va o'zgaruvchidan tashkil topgan *istisno tavsifi* keladi.

```
catch (<istisnoturi><istisno o'zgaruvchisi>)
{
    // Istisnoni qayta ishlash
}
```

Istisno turining nomida qayta ishlanadigan istisno turi ko'rsatiladi va **catch**-blok aynan shu turdagi istisnoni "*ilib*" oladi. Agar istisno ilib olingan bo'lsa, istisno o'zgaruvchisi istisno turi haqidagi qiymatni qabul qiladi. Agar istisno haqidagi ma'lumotga zarurat bo'lmasa, istisno o'zgaruvchisi ishlatilmasligi mumkin. Istisno o'zgaruvchisi ixtiyoriy turda bo'lishi mumkin. Bunga foydalanuvchi yaratgan sinf turi ham kiradi.

Bitta **try**-blokdan keyin bir nechta **catch**-blok kelishi mumkin. Istisno turining nomida nuqtalar bo'lgan **catch**-blok ixtiyoriy turdagi istisnolarni ilib oladi va u **try**-blokdan keyin keluvchi **catch** operatorlar ro'yxatining oxirida kelishi kerak.

Istisno holatni qayta ishlovchi misolni ko'raylik:

```
#include <iostream.h>
#include <exception.h>
using std::cout;
int AdivB(int A, int B) // Xato ro'y berishi mumkin bo'lgan funksiya
{
    // Istisno yuzaga keladigan joy
}
int main()
{ int a,b,c;
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  try
  { c=AdivB(a,b);
    cout<<"Istisno ro'y bermadu!\n";
    cout<<"c="<<c;
    return 0;
  }
  catch (int)
  { cout<<" int turidagi istisno ro'y berdi!\n";
    return 1;
  }
  catch(...)
```

```
{
    cout<<"Qandidir istisno ro'y berdi!\n";
    return 2;
}
}
```

Dasturdagi AdivB() funksiyasi istisno holatni yuzaga keltirishi mumkin, masalan nolga bo'lish. Funksiyada yuzaga keladigan istisnoni qayta ishlash uchun ikkita catch-blok aniqlangan. Birinchisida int turidagi istisno ilib olinadi. Agar boshqa turdagi istisno ro'y bersa uni catch(...) operatori qayta ishlaydi.

8.2. Istisnolarni yuzaga keltirish

Dasturdagi yuzaga keladigan istisno va uning turini ko'rsatish uchun quyida keltirilgan sintaksisga ega throw kalit so'zi ishlatiladi:

throw<ifoda>;

bu erda <ifoda> - hisoblangan qiymati, shu ifoda turi bilan aniqlangan vaqtinchalik obyektini initsializatsiya qiluvchi ifodadir.

throw operandi bo'lmasligi mumkin. Bu ko'rinish ayni paytda qayta ishlanayotgan istisno turidagi istisnoni qayta yuzaga keltirish uchun ishlatiladi va u faqat catch-blok ichida ishlatilishi mumkin.

Dasturning throw operandi joylashgan joyiga "istisnoni yuzaga keltirish nuqtasi" deyiladi.

Istisnoni yuzaga keltirishga misol keltiramiz.

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

```
#include <exception.h>
```

```
using std::cout;
```

```
class Xato1{};
```

```
class Xato2{};
```

```
int AdivB(int a, int b)
```

funksiya

```
{
```

```
    if(!b)throw Xato1();
```

```
    if(!a)throw Xato2();
```

// Xato ro'y berishi mumkin bo'lgan

```
    return a/b;
```

```
}
```

```
int main()
```

```
{
```

```
    int a,b,c;
```

```
    cout<<"a="; cin>>a;
```

```
    cout<<"b="; cin>>b;
```

```
    try
```

```
{
```

```
        c=AdivB(a,b);
```

```
        cout<<"Istisno ro'y bermadu!\n";
```

```
        cout<<"c="<<c;
```

```
        return 0;
```

```
}
```

```
catch (Xato1 & xato1)
```

```
{
```

```
    cout<<"Xato1 turidagi istisno ro'y berdi!\n";
```

```
    return 1;
```

```
}
```

```
catch (...)
```

```
{
```

```
    cout<<"Qandidir istisno ro'y berdi!\n";
```

```
    return 2;
```

```
}
```

```
}
```

Dastur ishlashida, bosh funksiyadan AdivB() funksiya chaqirilganda, agar:

– b argument qiymati 0 bo'lsa, Xato1 turidagi istisno yuzaga keltiriladi va catch(Xato1 & xato1) blok tomonidan ilib olinadi va qayta ishlanadi;

– a argument qiymati 0 bo'lsa, Xato1 turidagi istisno yuzaga keltiriladi va catch (...) blok tomonidan ilib olinadi va qayta ishlanadi;

– a va b argumentlar noldan farqli bo'lsa, istisno holati ro'y bermaydi, hamda dastur o'z ishini normal tugatadi.

throw kalit so'zini nafaqat istisno yuzaga keltirish uchun, balki birorta funksiya tomonidan yuzaga keltirishi mumkin bo'lgan *istisnolarni tasniflash* uchun ishlatish mumkin. Istisnolarni tasniflash quyidagi formatda beriladi:

throw (<tur₁><tur₂>,...)

Istisnolarni tasniflash funksiya parametrlari ro'yxatidan keyin keladi va u funksiya qanday turdagi istisno yuzaga keltirishi mumkinligi ko'rsatadi. Lekin bu holat funksiya boshqa turdagi istisnolarni yuzaga keltirmaydi degani emas. Ko'rsatilgan turdan farqli yuzaga kelgan istisnolarga *kutilmagan istisnolar* deyiladi va ular ham alohida yo'l bilan qayta ishlanadi. Agar funksiya birorta istisnoni yuzaga keltirishni man qilish zarur bo'lsa, turi ko'rsatilmagan istisno tasnifidan foydalaniladi. Ikkinchi tomondan, istisno tasnifiga ega bo'lmagan funksiya har qanday istisnoni yuzaga keltirishi mumkin.

Misol keltiraylik.

```
#include <stdio.h>
bool test;
class Xato1{};
void Func1(bool bul) throw(Xato1) { if (bul) throw Xato1(); }
void Func2() throw()
{
    try { Func1(true); }
    catch (Xato1 & xato)
    {
        puts("Xato1 turidagi istisno qayta ishlandi!");
        test=true;
    }
}
int main()
{
    try
```

```
(
    test=false;
    Func2();
    test? puts("Func2() funksiyasi istisnoni qayta ishladi"):
        puts("Hech qanday istisno holat ro'y bermadi");
)
catch(...) { puts("Func2()funksiyasi istisnoni yuzaga keltirdi!"); }
return 0;
}
```

Ushbu misol istisno tasnifi istisno holatni funksiyadan tashqariga chiqishga yo'l qo'ymasligini ko'rsatadi. Shu sababli, bosh funksiyadagi catch(...) bloki hech qachon ishlamaydi. Funksiya ichida yuzaga keladigan istisnoni funksiya ichida qayta ishlanishi mumkin. Agar Func1() funksiyasi Xato1 turidan farqli istisnoni yuzaga keltirsa, u kutilmagan istisno bo'ladi.

8.3. Istisnolarni ilib olish

Dasturning birorta nuqtasida istisno yuzaga kelsa, dastur bajarilishi uziladi va quyidagilar ro'y beradi:

- agar asos turdagi o'zgaruvchi yoki qiymat bo'yicha sinf obyekti paydo qilingan bo'lsa, o'zgaruvchining nusxasi yaratiladi (obyekt uchun nusxalash konstruktori ishlatiladi);
- agar murojaat bo'yicha o'zgaruvchi paydo qilingan bo'lsa, nusxalash ro'y bermaydi;
- paydo qilingan o'zgaruvchi bilan mos keluvchi parametrga qabul qila oladigan eng yaqin qayta ishlash bloki izlanadi;
- agar qayta ishlash bloki topilgan bo'lsa, stek shu nuqtagacha "bo'shatiladi", bunda ko'rinish sohasidan chiqqan lokal obyektlar destruktorga chaqiriladi;
- boshqaruv topilgan qayta ishlash blokiga uzatiladi;
- istisnoni qayta ishlaydigan blok topilmagan holat uchun dastur tuzuvchisi dasturni tugatish qayta ishlovchisi – set_terminate()

funksiyasini chaqirish orqali dasturni tugatishi mumkin, aks holda tizimning o'zi tugatish funksiyasini chaqiradi.

Istisnoga mos qayta ishlash blokini izlashda kompilyator quyidagi qoidalarga rioya qiladi. Istisnoga qayta ishlovchi mos hisoblanadi:

- paydo qilingan o'zgaruvchi turi qayta ishlovchi kutayotgan tur bo'lsa. Boshqacha aytganda, agar paydo bo'lgan o'zgaruvchi turi T bo'lsa, parametrlari T, const T, T& va const T& turida bo'lsa;

- paydo bo'lgan o'zgaruvchi ko'rsatkich bo'lib, uning turini qayta ishlovchining turiga keltirish mumkin bo'lsa;

- paydo bo'lgan o'zgaruvchi qandaydir voris sinf obyeksi bo'lib, qayta ishlovchi turi unga nisbatan tayanch sinf turida va vorislik public murojaat xossasi bo'yicha hosil qilingan bo'lsa.

Shuni qayd qilish kerakki, kompilyator paydo bo'lgan o'zgaruvchi turi bilan mos keluvchi parametrlarni qabul qiladigan eng yaqin qayta ishlovchini qidiradi. Shu sababli, mazkur try-blok uchun aniqlangan catch-bloklar ketma-ketligiga e'tibor berish zarur bo'ladi. Tayanch sinf istisnosini kutayotgan qayta ishlovchi voris sinf qayta ishlovchilarini berkitadi. Xuddi shunday void * turidagi ko'rsatkich uchun aniqlangan qayta ishlovchi ixtiyoriy turdagi ko'rsatkich turidagi qayta ishlovchilarini yashiradi.

Misol ko'raylik.

```
#include <iostream.h>
class Tayanch {};
class Hosila: public Tayanch {};
void Fun_Hosila_Istisnosi() { Hosila hosila; throw hosila; }
void Fun_Istisno() { throw "Fun_Istisno() funksiyasida xato!\n"; }
int main()
{
    try
    {
        Fun_Hosila_Istisnosi();
    }
    catch (Hosila &) { cout<<"Hosila& istisnosi ilib olindi!\n"; }
```

```
catch (Tayanch &) { cout<<"Tayanch& istisnosi ilib olindi!\n"; }
try
{
    Fun_Istisno();
}
catch (const char * s)
{
    cout<<"const char * istisnosi ilib olindi!\n";
    cout<<"Istisno manzili: "<<s<<"\n";
}
catch (void *) { cout<<"void* istisnosi ilib olindi!\n"; }
return 0;
}
```

Dastur ishlashi natijasida ekranga quyidagi satrlar chiqadi:

Hosila& istisnosi ilib olindi!

const char * istisnosi ilib olindi!

Istisno manzili: Fun_Istisno() funksiyasida xato!

Agar dasturda catch(Tayanch&) bloki catch(Hosila&) blokidan oldin yozilgan bo'lsa, u Hosila sinfi turidagi qayta ishlovchisini yashiradi va shu turdagi istisnoni o'zi ilib oladi. Xuddi shunday, catch(void*) bloki catch(const char*s) blokdan oldin kelsa, ko'rsatkich turidagi barcha istisnolarni ilib oladi, xususan, const char* turidagi istisnoni ham. Yuqoridagi dasturda bunday o'rin almashtirishlar qilinsa, ekranga

Tayanch& istisnosi ilib olindi!

void* istisnosi ilib olindi!

xabarlar chop etiladi.

8.4. Ichma-ich joylashgan try-catch bloklar

C++ tili ichma-ich joylashgan try-catch bloklarni ishlatishga imkon beradi. Bunday ko'rinishlar uchun yagona talab – har bir try-bloktan keyin albatta catch-blok kelishi kerak.

Misol:

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

```

class Tayanch {};
class Hosila: public Tayanch {};
void Fun_Hosila_Istisnosi() { Hosila hosila; throw hosila; }
int main()
{
    try
    {
        //...
        try
        {
            Fun_Hosila_Istisnosi();
        }
        catch (Hosila &) { cout<<"Hosila& istisnosi ilib olindi!\n"; }
        //...
        Tayanch tayanch;
        //...
        throw tayanch;
    }
    catch (Tayanch &) { cout<<"Tayanch& istisnosi ilib olindi!\n"; }
    return 0;
}

```

Dastur bajarilganda ekranga quyidagi natijalar chiqadi:

Hosila& istisnosi ilib olindi!

Tayanch& istisnosi ilib olindi!

Aksariyat hollatlarda ichma-ich joylashgan try-catch bloklar nooshkor ravishda yuzaga keladi: himoyaga olingan funksiyaning tanasida himoyalangan bloklar bo'lganda. Quyidagi misolda aynan shunday holat ko'rsatilgan:

```

#include <iostream.h>
void Fun_Istisno()
{
    //...
    throw "Fun_Istisno() funksiyasida xato!\n";
}

```

```

}
void Fun_Fun_Istisno()
{
    //...
    char* XatoSatr="Fun_Fun_Istisno() funksiyasida xato!\n";
    try
    {
        throw XatoSatr;
    }
    catch(char *s) { cout<<s; }
    Fun_Istisno();
}
void main()
{
    try
    {
        Fun_Fun_Istisno();
    }
    catch(char *s) { cout<<s; }
}

```

Dastur ekranga quyidagi satrlarni chop etadi.

Fun_Fun_Istisno()funksiysida xato!

Fun_Istisno() funksiyasida xato!

Ichma-ich try-catch bloklar ishlatishga qandaydir funksiyaning kutilmaganda istisnani yuzaga keltirish mumkinligi sabab bo'ladi. Bunday holatlarda istisnani hal qilishning yo'llaridan biri – bosh funksiya tanasini to'laigicha himoyaga olish va uning qayta ishlovchisi sifatida catch(...) ko'rinishi ishlatishdir.

8.5. Kutilmagan istisnalar va tugatishni qayta ishlash

Dastur bajarilishida funksiya, uning istisno tasnifida ko'rsatilmagan istisnani yuzaga keltirsa, *kutilmagan istisno* yuzaga keldi deyiladi. Bu holda unexpected() funksiyasi chaqiriladi. U o'z navbatida kutilmagan

istisnoning qayta ishlovchisini chaqiradi. Kelishuv bo'yicha bu `terminate()` funksiyasi hisoblanadi. `terminate()` funksiyasini chaqirilishi dastur bajarilish jarayonini tugatishga olib keladi.

`unexpected()` funksiyasining quyidagi sintaksisga ega:

```
void unexpected();
```

`set_unexpected()` funksiyasi yordamida dastur tuzuvchining o'zi kutilmagan istisnolarni qayta ishlovchi funksiyani aniqlashi mumkin. Bu funksiyaning prototipi quyidagi ko'rinishga ega:

```
unexpected_handler set_unexpected(unexpected_handler ph)
throw();
```

bu erda `unexpected_handler` kutilmagan istisnoni qayta ishlovchi funksiyaga ko'rsatkich:

```
typedef void(*unexpected_handler)();
```

`set_unexpected()` funksiyasi `ph` parametri ko'rsatib turgan funksiyani kutilmagan istisnoni yangi qayta ishlovchisi qilib o'rnatadi va oldingi qayta ishlovchining adresini qaytaradi. Yangi qayta ishlovchi parametrga ega bo'lmasligi kerak va u hech qanday qiymat qaytarmaydi. Bu funksiya uni chaqirgan kodga boshqaruvni qaytarmaydi va u quyidagi yo'llarning biri bilan dasturni bajarish jarayonini tugatadi:

- kutilmagan istisnoni yuzaga keltirgan funksiyaning istisnolar tasnifida keltirilgan turdagi istisnoni yuzaga keltirish orqali;
- `bad_exception` turidagi istisnoni yuzaga keltirish orqali;
- `terminate()`, `abort()` yoki `exit(1)` funksiyalardan birini chaqirish orqali.

Yuqorida qayd qilingandek, istisno qayta ishlovchisi `terminate()` funksiyasini chaqiradi.

`bad_exception` istisnosi

```
class bad_exception: public exception{};
```

ko'rinishida aniqlangan bo'lib, u `unexpected_handler` tomonidan yuzaga keltirilishi mumkin bo'lgan istisnolarni tavsiflaydi.

Kutilmagan istisnoni qayta ishlashga misol keltiramiz:

```
#include <iostream>
```

```
#include <exception>
```

```
using namespace std;
```

```
void f() throw()//Istisno yuzaga keltirmaydigan funksiya
```

```
{
```

```
    throw "Kutilmagan istisno!"; // Kutilmagan istisnoni yuzaga keltirish
```

```
}
```

```
void Shaxsiy_Qayta_Ishlovchi()
```

```
{
```

```
    cout<<"Kutilmagan istisnoni qayta ishlandi!";
```

```
    exit(-1);
```

```
}
```

```
int main()
```

```
{
```

```
    set_unexpected(Shaxsiy_Qayta_Ishlovchi());
```

```
    f();
```

```
    cout<<"Bu satr hech qachon chop qilinmaydi!";
```

```
    return 0;
```

```
}
```

Dastur ekranga

Kutilmagan istisnoni qayta ishlandi!

xabarini chop etadi.

Endi yuzaga kelgan istisnoga mos qayta ishlovchi topilmaganda nima holat ro'y berishini ko'ramiz. Bunday holatlarda kelishuv bo'yicha `terminate()` funksiyasi chaqiriladi. O'z navbatida `terminate()` funksiyasi, dasturni favqulotda to'xtashiga olib keluvchi `abort()` funksiyasini chaqiradi. Dastur tuzuvchisining dasturni tugashini qayta ishlovchisini `set_terminate()` funksiyasi vositasida o'rnatish orqali dasturni to'xtatish jarayoniga o'zgartirish kiritish mumkin.

`set_terminate()` funksiyasi quyidagi prototipga ega:

```
terminate_function set_terminate(terminate_function term_func);
```

bu erda `terminate_function` dasturni tugatuvchi funksiyaga ko'rsatkich:

```
typedef void(*terminate_function)();
```

Tugatish funksiya adresi - `term_func` parametri `set_terminate()` funksiyasining yagona kiruvchi parametri hisoblanadi va oldingi tugatish funksiya adresi uning qaytaruvchi qiymati bo'ladi. Agar funksiyani chaqirish `set_terminate(NULL)` ko'rinishida bo'lsa, kelishuv bo'yicha tugatish funksiyasi chaqirish holati tiklanadi.

`terminate()` funksiyasi prototipi

`void terminate();`

ko'rinishida va taqdim qilamayotgan tugatish funksiyalari ham xuddi shunday prototipga ega bo'lishi shart.

Xotirani tozalash bilan bog'liq zarur amallar bajarilgandan keyin `terminate()` funksiyasi dasturdan chiqish uchun `exit()` funksiyasini chaqirishi kerak bo'ladi (uni chaqirgan funksiyaga yoki istisno yuzaga kelgan kodga qaytmaslik uchun). Aks holda `abort()` funksiyasi chaqiriladi.

Misol ko'raylik:

```
#include <iostream>
#include <exception.h>
using std::cout;
void Tugatish_Funksiyasi()
{
    cout<<"Tugatish_Funksiyasi()funksiyasiterminate()tomonidan
chaqirildi.\n";
    exit(-1);
}
int main()
{ int i=10, j=0, rez;
  set_terminate(Tugatish_Funksiyasi);
  try
  {
      if(j==0) throw "Nolga bo'lish";
      else rez=i/j;
  }
  catch (int) { cout<<"int turidagi istisno ilib olindi!\n"; }
  cout<<"Bu satr hech qachon chop qilinmaydi!\n";
```

```
return 0;
```

```
}
```

Dastur ishlagandan keyin ekranga quyidagi satr chop qilinadi:

Tugatish_Funksiyasi() funksiyasi terminate() funksiyasi tomonidan chaqirildi.

8-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Dastur bajarilishida istisno qachon ro'y beradi?
2. Istisnolarni boshqarish uchun C++ tilida uchta kalit so'zlarni sanab ko'rsating.
3. Dasturdagi yuzaga keltirish va qayta ishlash mexanizmi qanday?
4. Istisnolarni ilib olishda istisnoni ilib olish blokini aniqlashda parametr roli nimada?
5. Kutilmagan istisnolar qachon ro'y beradi?

9. Qoliplar

9.1. Umumlashgan dasturlash

Ma'lumki, obyektga yo'naltirilgan dasturlashda e'tibor berilganlarga qaratiladi, ya'ni obyektning tavsiflovchi berilganlar va ular ustida bajariladigan metodlarni (funksiyalarni) aniqlaydi. Bunga qarama-qarshi *umumlashgan dasturlashda* esa e'tibor algoritmlarga qaratiladi. Umumlashgan dasturlash maqsadi berilganlar turlariga bog'liq bo'lmagan dastur kodini yozishdir. C++ tilidagi qoliplar umumlashgan dasturlashning vositasi hisoblanadi.

Qoliplar - berilganlar turlaridan, funksiyalar va sinflar aniqlanishlaridan foydalanish ma'nosida umumlashtirishga imkon beruvchi tushunchalardir. Shu sababli ularni *parametrlashtirilgan funksiyalar* yoki *parametrlashtirilgan sinflar* deyiladi. Aksariyat hollarda "qolipli funksiyalar" va "qolipli sinflar" terminlari ishlatiladi.

9.2. Funksiyalar qoliplari

Funksiya qolipi funksiyaning umumlashgan aniqlanishi bo'lib, uning asosida kompilyator foydalanuvchi tomonidan berilgan turdagi funksiya vakilini yaratadi.

Funksiya qolipini e'lonining sintaksisi quyidagi ko'rinishga ega:

```
template <class T1|T1 <identifiktor1>,  
class T2|T2 <identifiktor2>,  
...  
class Tn| Tn <identifiktorn>>  
<Qaytaruvchi qiymat turi> <funksiya nomi>  
(<parametrlar ro'yxati>)  
{  
    // funksiya tanasi  
}
```

Bu yerda **template** kalit so'zidan keyin burchakli qavs ichida bir-biridan vergul bilan ajratilgan parametrlar ro'yxati keladi. Har bir parametr - class kalit so'zi yoki tur nomi va undan keyin keluvchi identifikatordan iborat

bo'ladi. Ayrim hollarda, berilganlarning parametrlashgan turini berish uchun **class** so'zi o'rniga **typename** kalit so'zi ishlatilishi mumkin.

E'londagi **class** yoki **typename** kalit so'zidan keyingi qolip parametrlariga *parametrlashgan turlar* deyiladi. Ular kompilyatorga qolipda parametr sifatida qanday berilganlar turi ishlatilayotganini bildiradi. Tur nomi va identifikatordan iborat qolip parametri kompilyatorga qolip parametri ko'rsatilgan turdagi konstanta ekanligini bildiradi.

Funksiya qolipini aniqlash va chaqirishga misol ko'raylik.

```
#include <iostream.h>  
template <class T>  
T Kvadrat(T x){return x*x;}  
template <class T>  
T* Almashtirish(T * t, int ind1, int ind2)  
{  
    T Vaqtincha=t[ind1]; t[ind1]=t[ind2]; t[ind2]=Vaqtincha;  
    return t;  
}  
template <typename T1,typename T2>  
void Ekranga(T1 x,T2 y)  
{ cout<<x<<"\t"<<y<<endl; }  
template <class T,int siljish>  
void Obect_Adresi(T * obj, unsigned int * pAdres)  
{ *pAdres=(unsigned int) & obj[0]+siljish*sizeof(T); }  
int main()  
{  
    int n=10,kv_n,i=1,j=3;  
    double d=10.21, kv_d;  
    char * satr="Qolip";  
    kv_n=Kvadrat(n); kv_d=Kvadrat(d);  
    int Massiv[10];  
    unsigned int adres=0;  
    cout<<"n="<<n<<" Qvadrat n="<<kv_n<<endl;
```

```

cout<<"d="<<n<<" Qvadrat d="<<kv_d<<endl;
cout<<"satr="<<satr<<"O'zgargan
satr="<<Almashtirish(satr,i,j)<<endl;
cout<<"Son va uning qvadrati:\n";
Ekranga(n,d);
Ekranga(kv_n,kv_d);
Obect_Adresi<int,5>(Massiv, &adres);
cout<<"Arr[5]element adresi = "<<hex<<showbase<<adres;
return 0;
}
Dastur ishlashi natijasida ekranga quyidagilar chop etiladi:
n=10 Qvadrat n=100
d=10 Qvadrat d=104.244
satr=Qolip O'zgargan satr=Qilop
Son va uning qvadrati:
10 10.21
100 104.244
Arr[5] element adresi = 0x12ff50
Xuddi oddiy funksiyalardek funksiyalar qolipining prototipini e'lon
qilish mumkin. Masalan:
template <class T>
T Kvadrat(T x);
Funksiya qolipi va uning prototipidagi parametrlar nomi ustma-ust
tushmasligi mumkin:
template <class T,class S> // Funksiya qolipi prototipi
void Fun(T,S);
...
template <class U,class V> // Funksiya qolipi aniqlanishi
void Fun(U,V)
{
// Funksiya tanasi
}

```

Funksiya qolipining prototipidagi har bir parametr qolip aniqlanishida parametr sifatida ishlatilishi kerak. Masalan, quyidagi holat xatolikka olib keladi:

```

template <class T,class S>
T Fun(S);

```

Funksiya qolipining parametrlarining nomlari bir xil bo'lmazligi kerak. Quyidagi e'lon xato hisoblanadi:

```

template <class T,class T>
T Fun(T,T);

```

Qolipli funksiya tashqi, statik va joylashuvchi deb e'lon qilinishi mumkin. Buning uchun mos kalit so'zlar **template** qatorining oxirida yozilishi kerak bo'ladi:

```

template <class T, class S>extern // Tashqi qolipli funksiya
prototipi
void Fun2(T,S);
template <class T>static // Statik qolipli funksiya
prototipi
T Fun2(T);
template <class T,class S>static // Joylashuvchi qolipli funksiya
prototipi
T* Fun3(T,S);

```

Qolipli funksiyaning har bir tur bilan birinchi chaqirilishida uning vakili yaratiladi va bu jarayonga *qolipli funksiyani konkretlash* deyiladi. Yuqorida keltirilgan dasturdagi

```
kv_n=Kvadrat(n); kv_d=Kvadrat(d);
```

ko'rsatmalarining birinchisi butun tur uchun Kvadrat() funksiyasini, ikkinchi ko'rsatmani bajarish uchun esa bu funksiyasining haqiqiy tur uchun vakilini yaratadi.

Funksiya qolipini konkretlashtirishda uning parametrlari uchun boshqa turdan oshkor ravishda kutilgan turga keltirish amalini bajarish mumkin.

Masalan:

```

template <class T>
void Fun1(T){...};

```

...

```
void Fun2(char ch){ Fun1<int>(ch);}
```

Ushbu misolda kompilyator char turidagi ch o'zgaruvchisini int turiga o'tkazadi.

9.3. Sinf lar qolipi

C++ tilida sinflar qolipini yaratish ham qo'llab quvvatlanadi. Abstrakt turdagi sinf qolipi aniqlangandan keyin u konkret sinflarni yaratishda ishlatilishi mumkin. Kompilyator tomonidan sinf qolipi va qolip argumentlari bo'yicha konkret sinfni aniqlash jarayoniga *qolipni instansiyalash* deyiladi (*template instantiation*).

Misol uchun tekislikdagi nuqtani qaraylik. Nuqta koordinatasi double turida bo'lgan Nuqta sinfini yaratishimiz mumkin. Boshqa bir ilovada nuqtaning butun sonlar koordinatalarida int turida aniqlash zarur bo'lishi mumkin. Xuddi shunday koordinatalar turlari short, unsigned char turida talab qilinishi mumkin. Har bir holat uchun zarur turdagi koordinatalari bo'lgan Nuqta sinfini aniqlash orqali masalalani yechish mumkin. Bu albatta muammoning ma'qul echimi emas. To'g'ri yechim – Nuqta sinfining qolipini yaratishdir.

Sinf qolipini aniqlash quyidagi ko'rinishga ega:

```
template <qolip parametrlari> class <sinf nomi> { <qolip tanasi> }
```

Misol uchun Nuqta sinfining qolipi aniqlanishi

```
template <class T> class Nuqta
{
public:
    Nuqta(T _x=0, T _y=0): x(_x), y(_y){}
    void Chop_Qilish() const { cout<<" ("<<x<<" , "<<y<<" )"<<endl; }
private:
    T x,y;
};
```

Bu erda T – qandaydir abstrakt tur bo'lib, uning aniqlanishidagi class so'zi sinfni emas, balki T – qolip parametri ekanligini bildiradi.

Qolip ichidagi funksiya-a'zolarining aniqlanishi xuddi oddiy sinflar aniqlanishidek bo'ladi. Agar funksiya-a'zolar aniqlanishi sinf aniqlanishidan tashqarida bo'ladigan bo'lsa, ularni sintaksisi biroz murakkablashadi. Masalan, Nuqta sinfining Chop_Qilish() funksiya-a'zosining sinfdan tashqaridagi aniqlanishi quyidagicha bo'ladi:

```
template <class T> void class Nuqta<T>::Chop_Qilish() const
{ cout<<" ("<<x<<" , "<<y<<" )"<<endl; }
```

Dastur tarkibiga sinf qolipi kiritilganda abstrakt tur (T) o'miga konkret bir tur ko'rsatilib sinf nusxasi yaratilmaguncha birorta ham sinf yaratilmaydi. Abstrakt tur o'miga konkret tur qo'yishga *qolipni aktuallash* deyiladi. Bu jarayon xuddi oddiy sinflar obyektini e'lon qilishdek amalga oshiriladi, masalan:

```
Nuqta <int> nuqta_int(10,5);
```

yoki aktuallashtirilgan qolip turiga ko'rsatkich e'loni qilish va unga new orqali adres berish paytida:

```
Nuqta <double*> nuqta_k=new Nuqta<double>(3.14,2.7);
```

Quyida sinf qolipi ishlatilgan dastur matni keltirilgan.

```
#include <iostream.h>
template <class T> class Nuqta
{
public:
    Nuqta(T _x=0, T _y=0): x(_x), y(_y){}
    void Chop_Qilish() const { cout<<" ("<<x<<" , "<<y<<" )"<<endl; }
private:
    T x,y;
};
int main()
{
    Nuqta <int> nuqta_i;
    Nuqta <double> nuqta_d(3.14,2.7);
    Nuqta <short> nuqta_s(3,2);
    nuqta_i.Chop_Qilish();
    nuqta_d.Chop_Qilish();
}
```

```
nuqta_s.Chop_Qilish();  
return 0; }
```

Shuni qayd qilish kerakki, sinf qoliplari sinflar konteynerlarining (boshqa obyektlarni saqlashga mo'ljallangan obyektlar) standart kutubxonalarini (STL) yaratishda keng qo'llanilgan. Konteynerlar vektorlar, ro'yxatlar, navbatlar, to'plamlar va shunga o'xshash standart tuzilmalar bilan ishlashga mo'ljallangan.

9-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. Obyektga yo'naltirilgan dasturlashda va umumlashgan dasturlash o'rtasida farq nimada?
2. Funksiya qolipini e'lonining sintaksisini tahlil qiling.
3. Sinflar qolipini aniqlashga zarurat nimada?
4. Sinflar qolipini aniqlash qoidalarini izohlansin.

10. Qoliplarning standart kutubxonasi (STL)

10.1. STL asosiy konsepsiyasi

C++ tili tarkibiga kiruvchi STL kutubxonasi algoritmlarning umumlashgan holda tavsiflashga imkon beradi. Shuni qayd etish kerakki, STL kutubxonasi obyektga yo'naltirilgan dasturlash namunasi emas, garchi u qolip sinflar vositasida aniqlangan bo'lsa ham. Boshqacha aytganda, STL kutubxonasi umumlashgan dasturlash timsoli bo'lib, foydalanuvchiga "u (STL kutubxonasi) qanday amallarni (funksiyalarni) bajara oladi va ular qanday amalga oshirilgan" asnosida qiziqish uyg'otadi.

STL ikkita asosiy qismlardan iborat: konteyner sinflar majmuasi va umumlashgan algoritmlar majmuasi.

Konteynerlar – bu boshqa bir guruhdagi obyektlarni o'z ichiga olgan obyektlardir. Konteyner sinflar qoliplar hisoblanadi, shu sababli ularda saqlanadigan obyektlar o'rnatilgan yoki foydalanuvchi tomonidan aniqlangan turlarda bo'lishi mumkin. Bu obyektlarga nisbatan nusxalash va qiymat berish amallarini qo'llash mumkin. Konteyner sinflar berilganlarning namunaviy turlari – stek, ro'yxat, navbat va boshqa turlari ustidagi amallarni o'z ichiga oladi.

Umumlashgan algoritmlar – konteynerga qo'llanadigan ko'p sonli amallarning, masalan, izlash, tartiblash, qo'shish va shu kabi amallarning bajarilishidir. Biroq ular konteyner sinflarning metodlari hisoblanmaydi. Ular STL kutubxonasidagi global qolip funksiyalar ko'rinishida bo'ladi va shuning hisobiga universallikka erishiladi: bu funksiyalar nafaqat turli konteyner sinflar obyektlariga qo'llanib qolmasdan, ularni turli toifadagi elementlarga ega massivlarga ham qo'llash mumkin. Algoritmlarning konteyner turlariga nisbatan bog'liq bo'lmasligi ularning konteynerlar bilan vositali bog'lanishi hisobiga bo'ladi: funksiyaga konteyner uzatilmaydi, balki qayta ishlanadigan elementning (diapazonni aniqlovchi *first* va *last*) adreslar juftligi uzatiladi. Bu o'zaro ta'sir mexanizmi iteratorlardan foydalanish hisobiga amalga oshiriladi.

Iteratorlar – bu ko'rsatkichlarning umumlashtirilgan konsepsiyasidir: ular konteyner elementlariga ko'rsatadi (murojaat qiladi). Ularga nisbatan oddiy ko'rsatkichlardagi inkrement amalini qo'llash mumkin (konteyner bo'ylab ketma-ket harakatlanish uchun), '*' amali bilan qiymatga murojaat qilish mumkin.

10.2. Konteynerlar

Konteynerlarni ikki toifaga bo'lish mumkin: ketma-ket va assotsiativ.

Ketma-ket konteynerlar bir turdagi chekli obyektlarni uzluksiz ketma-ketlik ko'rinishida saqlashni ta'minlaydi. Tayanch ketma-ket konteynerlarga vektorlar (**vector**), ro'yxatlar (**list**), va ikki tomonli navbatlar (**deque**), bundan tashqari maxsus konteynerlar (konteynerlar adapterlari) steklar (**stack**), navbatlar (**queue**) va ustunlikli navbatlar (**priorety_queue**) kiradi.

Dasturda konteynerlardan foydalanish uchun mos sarlavha faylni qo'shish kerak. Konteynerlarda saqlanadigan obyektlar turi qolip argumenti yordamida beriladi.

Masalan:

```
vector<int> Avec; // int turidagi Avec vektori yaratiladi
```

```
list<Talaba> Guruh; // Talaba turidagi ro'yxatni yaratish
```

Assotsiativ konteyner kalit bo'yicha berilganlarga murojaat qilishni ta'minlaydi. Besh turda assotsiativ konteynerlar mavjud: lug'atlar (**map**), dublikatli lug'at (**multimap**), to'plamlar (**set**), dublikatli to'plamlar (**multiset**) va bitli to'plamlar (**bitset**).

10.3. Iteratorlar

Konteyner elementlari turli xil ravishda xotirada joylashishi mumkin, masalan, massivlar xotirada ketma-ket uzluksiz joylashadi, unda bir elementdan ikkinchisiga o'tish ko'rsatkich arifmetikasi orqali amalga oshirilishi mumkin. Chiziqli ro'yxat ko'rinishida obyektlar uchun ko'rsatkichlar arifmetikasini qo'llab bo'lmaydi, chunki ro'yxatning navbatdagi elementini xotiraning ixtiyoriy adresida joylashishi mumkin, ya'ni ro'yxat elementlari xotirada uzluksiz joylashishi shart emas. Shu

sababli har bir sinf uchun o'zining iteratori bo'lishi kerak. Barcha STL konteyner sinflari uchun **iterator** turi aniqlangan bo'lib, lekin ularni amalga oshirish turlicha. Masalan, **vector** sinfi uchun undagi elementlar ketma-ket joylashganligi uchun iterator **typedef T * iterator** ko'rinishida aniqlanadi, **list** sinfi uchun iterator alohida sinf ko'rinishida amalga oshiriladi.

Har bir iterator tomonidan bajariladigan asosiy amallar:

- iterator qiymatini olish: **p** – iterator, ***p** – u murojaat qiladigan obyekt qiymati;
- bir iteratorni ikkinchisiga qiymat sifatida berish;
- iteratorlarni tenglik (**==**) va teng emaslikka tekshirish (**!=**);
- konteyner bo'yicha prefiks ravishda (**++r**) yoki postfiks (**r++**) harakatlanish.

Har bir sinf uchun iterator, ko'rish sohasiga ruxsat berish (**"::"**) orqali ko'rsatiladi.

Masalan,

```
vector<int>::iterator iter1;
```

```
list<Talaba>::iterator iter2;
```

Konteyner elementlarini *"aylanib chiqish"* uchun takrorlash operatori o'ziga hos ko'rinishda aniqlanadi. Agar **i** – qandaydir iterator bo'lsa

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

o'miga

```
for (i=first; i!=last; ++i)
```

yoziladi. Bu erda **first** – konteyner birinchi elementiga ko'rsatkich qiymati, **last** – konteyner oxirgi elementidan keyin *"mavhum"* elementga ko'rsatkich qiymati, '<' amali '! ' amali bilan almashtirilgan.

10.4. Konteynerlar umumiy xossalari

Quyida aksariyat konteyner sinflarda **typedef** yordamida aniqlanidan turlar ro'yxati keltirilgan.

10.1-jadval. STLda aniqlangan unifikatsiya qilingan turlar

Maydon	Izoh
--------	------

value_type	Konteyner elementi turi
size_type	Indekslar turi, elementlar hisoblagichi va hokazo (unsigned int)
Iterator	Iterator
const_iterator	O'zgarmas iterator
reference	Elementga ko'rsatkich
const_reference	Elementga o'zgarmas ko'rsatkich
key_type	Kalit turi (assotsiativ konteynerlar uchun)
key_compare	Taqqoslash kriteriyasi turi (assotsiativ konteyner uchun)

10.2-jadval. Barcha konteynerlar uchun umumiy amallar va metodlar

Amal yoki metod	Izoh
==, !=	true yoki false qaytaradi
=(qiymat berish)	Bir konteynerni ikkinchisiga nusxalaydi
clear	Barcha elementlarni o'chiriladi
insert	Bitta elementni yoki elementlar diapazonini qo'shish
erase	Bitta elementni yoki elementlar diapazonini o'chirish
size_type size ()	Elementlar sonini qaytaradi
const	
size_type max_size() const	Konteynerning mumkin bo'lgan maksimal o'lchamini qaytaradi
bool empty ()const	Konteyner bo'sh bo'lsa, true qaytaradi
iterator begin ()	Iteratorni konteyner boshiga qaytaradi
iterator end ()	Iteratorni konteyner oxiriga o'tkazadi
reverse iterator begin ()	Reversiv iteratorni konteyner oxiriga o'tkazadi (to'g'ri yo'nalishdagi iteratsiya tugatiladi)
reverse iterator end ()	Reversiv iteratorni konteyner boshiga qaytaradi (teskari yo'nalishdagi iteratsiya tugatiladi)

10.5. Algoritmalar

Algoritm – konteyner (konteynerlar) elementlari ustida qandaydir amallarni bajaradigan funksiya. Umumlashgan algoritmnlarni dasturda qo'llash uchun <algorithm> sarlavha faylini kiritish kerak.

10.3-jadval. STL algoritmnlari (qisman)

Algoritm	Vazifasi
accumulate	Berilgan diapazondagi elementlar yig'indisini hisoblash
copy	Birinchi elementdan boshlab ketma-ketlikki nusxalash
count	Ketma-ketlikka kiruvchi qiymatlar miqdorini hisoblash
count_if	Ketma-ketlikdagi shart bajariluvchi elementlar soni
equal	Ikkita ketma-ketlikning mos elementlari juftligining tengligini tekshirish
fill	Ketma-ketlikning barcha elementlarini berilgan qiymat bilan almashtirish
find	Qiymatning ketma-ketlikka birinchi kirishini topish
find_first_of	Bir ketma-ketlikning ikkinchi ketma-ketlikdagi birinchi kirishini topish
find_if	Ketma-ketlikdagi berilgan shartga mos keluvchi birinchi elementini topish
for_each	Ketma-ketlikning har bir elementi uchun funktsiyani chaqirish
merge	Tartiblangan ketma-ketliklarni qo'shish
remove	Berilgan qiymatli elementlarni ko'chirish
replace	Berilgan qiymatli elementlarni almashtirish
search	Bir ketma-ketlikni ikkinchi ketma-ketlikka birinchi kirishini topish
sort	Tartiblash
swap	Ikkita elementlarni o'zaro almashtirish
transform	Ketma-ketlikning har bir elementi ustida berilgan amalni bajarish

Ro'yxatda keltirilgan algoritmnlarning boshdagi ikkita parametrlari qayta ishlanadigan elementlarning [first, last) – yarim interval

ko'rinishidagi diapazonini beradi, bu erda **first** – diapazon boshini ko'rsatuvchi iterator, **last** – diapazondan tashqariga chiqqanlikni ko'rsatuvchi iterator.

Masalan, agar sonlar massivi

```
int massiv[7]={5,9,-3,6,1,3,2}
```

ko'rinishida aniqlangan bo'lsa, uni **sort** algoritmi bilan tartiblash quyidagi ko'rinishda amalga oshiriladi:

```
sort(massiv,massiv+7);
```

Bu erda massiv iterator sifatida ishlatiluvchi **int*** turidagi ko'rsatkich.

Misol.

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

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

```
#include <vector.h>
```

```
#include <algorithm.h>
```

```
int main()
```

```
{ ifstream in("sonlar.txt");
```

```
if (!in) {cerr<<"Fayl topilmadi!"; exit(1);}
```

```
vector<int> v;
```

```
int x;
```

```
while (in>>x) v.push_back(x);
```

```
sort(v.begin(), v.end());
```

```
vector<int>::const_iterator i;
```

```
for(i=v.begin(); i!=v.end(); ++i)
```

```
cout<<*i<<" ";
```

```
return 0; }
```

10.6. Konteynerlar adapterlari

Maxsus ketma-ket konteynerlar – stek, navbat va ustunli navbat mustaqil konteyner sinflari bo'lmasdan, yuqorida qaralgan sinflar asosida amalga oshirilgan va shu sababli *konteynerlar adapterlari* deyiladi.

Stek – **stack** qolip sinfi (**<stack>** sarlavha fayli) quyidagicha aniqlangan:

```
template <class T, class Container = deque <T>> class stack
{ /* ... */ },
```

bu yerda **Container** parametri prototip sinfni aniqlaydi. Kelishuv bo'yicha stek uchun **deque** prototip sinf hisoblanadi.

10.4-jadval. **stack** sinfi interfeysi

stack sinf metodlari	Prototip sinf metodlar
push()	push_back()
pop()	pop_back()
top()	back()
empty()	empty()
size()	size()

Shuni qayd etish kerakki, **pop()** metodi o'chirilgan qiymatni qaytarmaydi. Shu sababli, stek uchidagi qiymatni o'qish (olish) uchun **top()** metodi qo'llaniladi.

Stek bilan ishlashga misol.

```
int main ()
```

```
{
```

```
ifstream in ("sonlar.txt");
```

```
stack<int> s;
```

```
int x;
```

```
while (in >> x) s.push (x);
```

```
while ( !s.empty() )
```

```
{
```

```
cout <<s.top() <<" ";
```

```
s.pop();
```

```
}
```

```
return 0;
```

```
}
```

Agar stek elementi nisbatan murakkab turdagi qiymat bo'lishi kerak bo'lsa, masalan, ro'yxat asosida u quyidagicha e'lon qilinadi.

```
stack<int, list<int>> s;
```

Navbat - qolip sinfi (sarlavha fayli) adapter bo'lib, unda ikki tomonlama navbat yoki ro'yxat amalga oshirilgan. Navbat berilganlarni bir tomondan joylash, ikkinchi tomondan olish uchun ishlatiladi.

10.5-jadval. *queue* sinfi interfeysi

queue sinf metodlari	Prototip sinf metodlar
push ()	push_back()
pop ()	pop_front()
front ()	front()
back ()	back()
empty ()	empty()
size ()	size()

Ustunli navbat – *priority queue* qolip sinfi (<queue> sarlavha fayli) qo'llab-quvvatlaydigan amallar oddiy sinfidan farqli ravishda qiymat olishda konteynerdan maksimal qiymatli element olinadi. Shu sababli har bir o'qishdan keyin maksimal qiymatli element konteyner boshiga suriladi. Agar ustunli navbat elementi murakkab bo'lsa, xususan, sinf obyektlari uchun tashkil qilinsa, sinfda '<' amali aniqlangan bo'lishi kerak. Ustunli navbat bilan ishlash bo'yicha misol:

```
int main()
{
    priority_queue<int> P;
    P.push (17); P.push (5); P.push (400); P.push (17); P.push (1);
    while (! P.empty ())
    {
        cout << P.top () << " "<< P.pop ();
    }
    return 0;
}
```

Dastur ishlash natijasida quyidagilar chop etiladi:
2500 400 17 5 1

10.7. Funktsional obyektlar

Qandaydir sinf obyekti *funktsional obyekt* deyiladi, agar uning uchun yagona *operator ()* funksiyani chaqirilishi aniqlangan bo'lsa.

Standart kutubxonada C++ tilidagi taqqoslash amali uchun funktsional obyektlarning qoliplari aniqlangan bo'lib, u *bool* turidagi qiymatini qaytaradi.

10.6-jadval. Funktsional obyektlar

Amal	Ekvivalent predikat (funktsional obyekt)
=	equal_to
!=	not_equal_to
>	greater
<	less
>=	greater_equal
<=	less_equal

Algoritm argumenti sifatida qo'yilganda bu qoliplarni installyasiyalash kerak bo'ladi. Masalan, oldingi keltirilgan misoldagi *V* vektorni tartiblashda *sort()* algoritmini chaqirishni quyidagicha almashtirish mumkin:

```
sort (V. begin, V.end (), greater<double>());
```

Teskari iterator - bu iterator konteynerda oxiridan boshiga harakatlanish uchun mo'ljallangandir. Masalan, *vector<double>V* konteyner aniqlangan bo'lsa, unda teskari harakatlanish uchun quyidagi dastur fragmentini yozish mumkin:

```
vector<double>::reverse_iterator ri;
ri=V. R begin ();
while (ri=V. Rend ()) cout << *ri++<< " ";
```

Shunga e'tibor berish kerakki, iterator uchun qo'llanilgan inkrement amali konteynerdagi ko'rsatkichini ayni paytda u ko'rsatib turgan elementidan oldingisiga o'tishini anglatadi.

10.8. Assotsiativ konteynerlardan foydalanish

Assotsiativ konteynerlarda elementlar chiziqli ketma-ketlikda joylashmagan bo'ladi. Ular nisbatan murakkab tuzilmalar ko'rinishida tashkil qilingan bo'lib, izlashda katta yutuqlarni beradi. Izlash bitta son yoki satr ko'rinishidagi kalitlar yordamida amalga oshiriladi.

Quyida STL kutubxonasidagi assotsiativ konteynerlarning ikkita asosiy kategoriyasi qaraladi: to'plamlar va lug'atlar. *To'plamda* (set) obyektlar, ularning atributi hisoblangan qandaydir kalit bo'yicha tartiblanadi. Masalan, to'plam "Talaba" sinfi obyektlarini talabaning familiya, ismi va sharifi maydoni bo'yicha tartiblangan ketma-ketligini saqlash mumkin.

Lug'atga (map) ikkita ustundan iborat jadval deb qarash mumkin. Birinchi ustunda o'z ichida kalit bo'lgan obyekt, ikkinchida qiymat – obyektlar joylashadi. To'plam va lug'atlarda barcha kalitlar yagona hisoblanadi (har bir kalitga faqat bitta qiymat mos keladi). Multito'plam (multiset) va multilug'atlarda (multimap) esa bitta kalitga bir nechta qiymat mos kelishi mumkin.

To'plam qolipi ikkita parametrga ega: kalit turi va "kichik" munosabatini aniqlovchi funksional obyekt turi:

```
template < class key, class compare = less <key>> class set { /* ... */ }
```

To'plam qolipi (set) turidagi konteynerlarni ishlatish uchun <set> sarlavha faylini qo'shish kerak bo'ladi.

Uch xil usulda set turidagi obyektни aniqlash mumkin:

```
set<int>set1; // Bo'sh to'plamni yaratish
```

```
int[5] = {1,2,3,4,5};
```

set<int>set2 (a,a+5); // Massivni nusxalash orqali to'plam hosil qilish

```
set<int>set3 (set2 ); // Boshqa to'plamdan nusxalash
```

To'plamga elementni qo'shish uchun insert() metodini, o'chirish uchun erase() metodini qo'llash mumkin.

Barcha assotsiativ konteynerlarda count() metodi bo'lib, ular berilgan kalitli elementlar sonini qaytaradi. To'plamlar va lug'atlarda

kalitlar yagona bo'lgani uchun count() metodi element topilsa – 1, topilmasa 0 qaytaradi.

To'plam uchun to'plamlar nazariyasidagi quyidagi amallar aniqlangan:

– includes algoritmi bitta ketma-ketlikni ikkinchisining ichida kirishini aniqlaydi. Natija true bo'ladi, agar [first2,last2) ketma-ketligining har bir elementi [first1, last1) ketma-ketlikda mavjud bo'lsa;

– set_intersection algoritmi bir paytda birinchi va ikkinchi to'plamga kiruvchi tartiblangan to'plamlar kesishmasini hosil qiladi;

– set_union algoritmi birinchi va ikkinchi to'plamlarning elementlari takrorlanmaydigan birlashmasini qaytaradi.

Yuqorida keltirilgan algoritmlardan foydalanishga misol:

```
#include <vcl.h>
#include <set.h>
#include <algorithm.h>
#include "Unit1.h"

...
typedef set<String> setS;
void print(String s, setS S, TMemor* memor)
{
    setS::iterator i;
    for (i=S.begin(); i!= S.end(); ++i) s+=" "+*i;
    memor->Lines->Add(s);
}

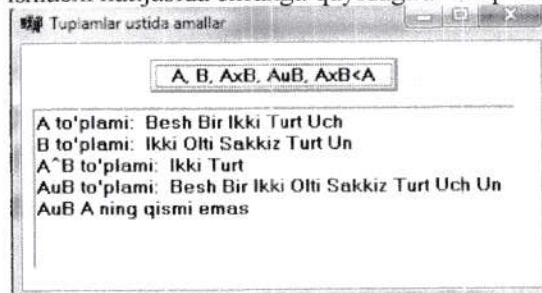
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    const int N=5;
    String S1[N]={"Bir","Ikki","Uch","Turt","Besh"};
    String S2[N]={"Ikki","Turt","olti","Sakkiz","Un"};
    setS A(S1,S1+N);
    setS B(S2,S2+N);
```

```

print("A to'plam: ", A, Memo1);
print("B to'plam: ", B, Memo1);
setS AxB, AuB;
set_intersection(A.begin(), A.end(), B.begin(), B.end(),
                inserter(AxB, AxB.begin()));
print("A^B to'plami: ", AxB, Memo1);
set_union(A.begin(), A.end(), B.begin(), B.end(),
          inserter(AuB, AuB.begin()));
print("AuB to'plami: ", AuB, Memo1);
if(includes(A.begin(), A.end(), AuB.begin(), AuB.end()))
Memo1->Lines->Add("AuB A ning qismi");
else Memo1->Lines->Add("AuB A ning qismi emas");
}

```

Dastur ishlashi natijasida ekranga quyidagilar chop etiladi:



Kalit bo'yicha elementga murojaat [] amali orqali amalga oshiriladi:
T& operator [] (const Key &x);

Uning yordamida nafaqat element qiymatini olish mumkin, balki lug'atga yangi element qo'shish ham mumkin.

map turidagi konteynerni ishlatish uchun <map> sarlavha fayli qo'shilishi kerak. Quyida misol tariqasida lug'at asosida shaxsning familiyasi bo'yicha uning yoshini aniqlab beruvchi dastur matni keltirilgan.

```

#include <iostream.h>
#include <map.h>

```

```

#include <vcl.h>
int main(int argc, char* argv[])
{
    map <String, int, less<String>> fish_yosh;
    fish_yosh["Boboyev B."]=18;
    fish_yosh["Abdullayev A."]=34;
    fish_yosh["Davlatov D."]=20;
    cout<<"Abdullaev A. yoshi - "<<fish_yosh["Abdullayev A."]<<endl;
    map <String, int, less<String>>:: iterator it;
    for(it=fish_yosh.begin(); it!=fish_yosh.end();++it)
    cout<<it->first<<" yoshi: " <<it->second<<endl;
    return 0;
}

```

Dasturda fish_yosh nomi bilan assosiativ massiv e'lon qilingan bo'lib, unda String turidagi kalit aniqlangan. Lug'atning ma'lum bir shart asosida tartiblangan holda bo'lishini inobatga oladigan bo'lsak, Massivni tartiblash uchun Compare obyekti sifatida less<String> olingan (massiv o'sish bo'yicha tartiblanadi).

Lug'atga shaxsning familiyasi va yoshi yaqqol ravishda beriladi. Keyin aniq bir familiya bo'yicha yosh chop etiladi. Undan keyin iterator yordamida lug'atdagi barcha juftliklar chop etiladi.

Dastur ishlashi natijasida ekranga quyidagilar chop etiladi.

Abdullaev A. yoshi - 34

Abdullayev A. yoshi: 34

Boboyev B. yoshi: 18

Davlatov D. yoshi: 20

Ushbu dasturni kalit va uning qiymatini har xil berilganlar turlari (sinf turidagi obyektlar) bilan e'lon qilish asosida rang-barang masalalarni yechish mumkin.

10-bob bo'yicha nazorat uchun savollar va topshiriqlar

1. STLning ikkita asosiy qismlarini sanab o'ting va izohlang.
2. Ketma-ket konteynerlar tashkil qiluvchilarini sanab o'ting.

3. Konteyner elementlari ustida amal qilishda iterator nima uchun kerak?
4. STL algoritmlari deganda nima tushuniladi?
5. Konteynerlar adapterlariga qanday konteynerlar kiradi?

Glossariy

Abstrakt sinf – obyektlar nusxasi aniqlash mumkin bo‘lmagan sinf. U keyinchalik tuzilmaviy va amaliy qismi bo‘yicha to‘ldirish uchun yaratiladigan sinfdir. Odatda abstrakt sinfning metod qismi to‘liq bo‘lmaydi va u shu sinfning hosilaviy sinflarida aniqlashni talab qiladi.

Agregatsiya – bu bitta sinf o‘zining tarkibiy qismi sifatida boshqa sinf obyektlarini o‘z ichiga olgan holat bo‘lib, “*butun/qism*” munosabatini aniqlaydi.

Ajdod sinf – elementlari vorislik bo‘yicha olinadigan sinfga nisbatan tayanch sinf deyiladi.

Algoritm – aniq shakllantirilgan qoidalar tizimi bo‘lib, u mumkin bo‘lgan boshlang‘ich berilganlarning (kiruvchi ma’lumotlar) cheklangan qadamlarda qayta ishlash orqali kutilgan natijaga olib keladi.

Axborot texnologiyalari – boshlang‘ich ma’lumot deb hisoblanuvchi berilganlarni saqlash, qayta ishlash va uzatish vosita va usullarining majmuasidir.

Berilganlar-a’zolar – sinf ichida e’lon qilingan o‘zgaruvchilar.

Berilganlarni abstraksiyalash – berilganlarni yangi turini yaratish imkoniyati bo‘lib, bu turlar bilan xuddi berilganlarning tayanch turlari bilan ishlagandek ishlash mumkin. Odatda yangi turlarni berilganlarning abstrakt turi deyiladi, garchi ularni soddaroq qilib «*foydalanuvchi tomonidan aniqlangan tur*» deb atash mumkin.

Destruktorlar – sinf ob’ekti o‘chirilishida chaqiriladigan maxsus funksiya-a’zo.

Dinamik polimorfizm – bunda sinf obyekti tomonidan qaysi polimorf funksiyaning chaqirish kerakligi dastur bajarilish jarayonida aniqlanadi.

Do’st funksiya – sinf a’zolari bo‘lmagan holda shu sinfning yopiq a’zolariga murojaat qilish imkoniyatiga ega funksiya.

Do’sst funksiya – sinf tarkibiga kirmaydigan, lekin uning a’zolariga murojaat qila oladigan funksiya.

Funksiya qolipi – funksiyaning umumlashgan aniqlanishi bo‘lib, uning asosida kompilyator foydalanuvchi tomonidan berilgan turdagi funksiya vakili yaratiladi.

Funksiya-a’zolar – sinf ichida e’lon qilingan funksiyalar.

Hosilaviy sinf (avlod sinf) – bir yoki bir nechta sinfning berilganlar-a'zolari va funksiyalar-a'zolarini vorislik bilan oladigan yangi sinf.

Inkapsulyasiya – bu berilganlar va ularni qayta ishlovchi kodni birlashtirish mexanizmidir. Inkapsulyasiya berilganlar va kodni tashqi ta'sirdan saqlash imkonini beradi.

Istisno – bu dasturning normal bajarilishini uzadigan holat.

Kompyuter(EHM) – hisoblash va axborot masalalarini avtomatik qayta ishlashga mo'ljallangan texnik vositalar majmuasi.

Konstruktor – sinf ob'ekti yaratilishda uni initsializatsiy qiluvchi maxsus funksiya-a'zo. Sinf konstruktori har safar sinf ob'ekti yaratilishi paytida avtomatik ravishda chaqiriladi. Konstruktor nomi o'zi a'zo bo'lgan sinf nomi bilan ustma-ust tushadi va qaytaruvchi qiymatga ega bo'lmaydi.

Konteyner sinf – obyektlari boshqa sinf obyektlardan tashkil topgan sinfdir. Konteyner sinfi bir jinsli (bitta sinf obyektlaridan) yoki bir jinsli bo'lmagan (turli sinf obyektlaridan) obyektlar qatoridan tashkil topadi. Konteyner sinflar umumlashgan bo'lishi mumkin yoki tarkibidagi obyektlar sinflarini belgilovchi parametrlarga ega bo'lishi mumkin.

Metod termini bilan obyekt tomonidan taqdim etiluvchi funksiya tushuniladi. Metodlar obyektning funksional imkoniyatlariga murojaat qilishda qo'llaniladi. Xuddi xossa va maydonlar kabi metodlar ham zarur bo'lganda tashqi kodlardan murojaatlarga cheklovlar qo'yish uchun ochiq yoki yopiq bo'lishi mumkin.

Obyektlar – OYD ilovalarning “qurilish blokidir”. Bunday qurilish bloki ilovaning qismini o'z ichiga oladi – hisoblash jarayonini, berilganlarning bir qismini yoki qandaydir yanada abstrakt bo'lgan obyektini.

Oddiy (yakka) vorislik – hosilaviy sinf bitta tayanch sinfga ega bo'lgan vorislik.

Operativ xotira – dastur bajarilishi uchun vaqtincha foydalaniladigan xotira bo'lib programma bajarilish tezligini aniqlaydi.

Operatorlarni qayta yuklash – bu global ravishda yoki sinf chegarasida operator mazmunini (amalini) o'zgartirish imkoniyatidir. Operatorni qayta yuklash maxsus ko'rinishdagi funksiyalar vositasida amalga oshiriladi.

Polimorfizm – bu kodning, bajarilish paytidan yuzaga keladigan holatga bog'liq ravishda o'zini turlicha amal qilish xususiyatidir.

private – sinf berilganlar-a'zolariga va funksiya-a'zolariga faqat shu sinf funksiya-a'zolari murojaat qilishi (ishlatishi) mumkin.

public – sinf berilganlar-a'zolariga va funksiya-a'zolariga faqat shu sinf funksiya-a'zolari va sinf ob'ekti mavjud bo'lgan programma funksiyalari murojaat qilishi (ishlatishi) mumkin.

Qoliplar – berilganlar turlaridan, funksiyalar va sinflar aniqlanishlaridan foydalanish ma'nosida umumlashtirishga imkon beruvchi tushunchalardir.

Sinf – tarkibiga berilganlar va ular ustida amallarni (funksiyalarni) oluvchi tuzilmadir.

Sinf do'sti – sinf a'zolariga murojaat qila oladigan ikkinchi bir sinf.

Sinflar tuzilmasi – tizim iyerarxik tuzilishini akslantirish vositasidir. U graf bo'lib, uning uchlari sinflarga mos keladi, uning yoylari – sinflar o'rtasidagi munosabatni aniqlaydi.

Sinfning konstanta obyektlari – obyekt e'loni oldiga **const** modifikatori qo'yish orqali beriladi va **const** kalit so'zi kompilyatorga ushbu obyektning holati o'zgarishligi kerakligini bildiradi. Shu sababli obyekt berilgan-a'zolari qiymatini o'zgartiradigan funksiya-a'zoni chaqirishi uchrasa, kompilyator xatolik haqida xabar beradi.

Sinfning statik a'zosi – bu **static** modifikatori bilan e'lon qilingan sinfning a'zosi bo'lib, uni sinf sohasi chegarasida murojaat qilish mumkin bo'lgan global o'zgaruvchi yoki funksiya. Sinfning statik deb e'lon qilingan berilgan-a'zolari sinfning barcha obyektlari tomonidan birgalikda ishlatiladi. Sinfning statik berilganlari uchun xotiradan joy, hattoki sinfning birorta ob'ekti e'lon qilinmagan bo'lsa ham ajratiladi. Shu sababli sinf statik berilganni e'lon qilib qolmasdan, uni aniqlashi shart.

Statik polimorfizm – polimorfizmni funksiya va operatorlarni qayta yuklash vositasi bilan kompilyatsiya paytida amalga oshirish holati.

Tayanch sinf – sinflash tuzilmadagi eng umumiy obyektlar sinfidir. U sinflar iyerarxiyasidagi ildiz sinf bo'lib, boshqa ixtiyoriy hosilaviy sinflar uchun birlamchi (boshlang'ich) sinf bo'ladi.

To'plamli vorislik – hosilaviy sinf bir nechta tayanch sinflarga ega bo'lgan vorislik.

Virtual funksiya – chaqirish interfeysi (prototipi) ma'lum, amalga oshirilishi umumiy ko'rinishda berilishi mumkin bo'lmagan, faqat konkret holatlardagini aniqlanadigan funksiya.

Vorislik – bu shunday jarayonki, unda bir ob'ekt boshqasining xossalarini o'zlashtirishi mumkin bo'ladi. Vorislik orqali mavjud sinflar asosida hosilaviy sinflarni qurish mumkin bo'ladi. Hosilaviy sinf (*sinf-avlod*) o'zining ona sinfidan (*sinf-ajdod*) berilganlar va funksiyalarni vorislik bo'yicha oladi, hamda ular qatoriga faqat o'ziga xos bo'lgan qirralarni amalga oshirishga imkon beruvchi berilgan va funksiyalarni qo'shadi.

- 14402/23 -

O'ZBEKISTON RESPUBLIKASI OLIY TA'LIM,
FAN VA INNOVATSIYALAR VAZIRLIGI
CHIRCHIQ DAVLAT PEDAGOGIKA UNIVERSITETI
AXBOROT RESURS MARKAZI

Foydalanilgan adabiyotlar

1. Страуструп Б. Язык программирования C++. Специальное издание.-М.:ООО «Бином-Пресс», 2006.-1104 с.
2. Глушаков С.В., Коваль А.В., Смирнов С.В. Язык программирования C++; Учебный курс.- Харьков: Фолио; М.: ООО «Издательство АСТ», 2001.-500с.
3. Павловская Т.А. C++. Программирование на языке высокого уровня - СПб.: Питер. 2005.- 461 с.
4. Шильт Г. Полный справочник по C++. 4е-изд.- М.:Издательский дом «Вильямс», 2003. 800 с.
5. Подбельский В.В. Язык СИ++.- М.; Финансы и статистика-2003 562с.
6. Павловская Т.С. Щупак Ю.С. C/C++. Структурное программирование. Практикум.-СПб.: Питер,2002-240с
7. Павловская Т.С. Щупак Ю.С. C++. Объектно-ориентированное программирование. Практикум.-СПб.: Питер,2005-265с
8. Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование: Учеб. для вузов/Под редакцией Г.С. Ивановой.-М.: Из-во МГТУ им.Н.Э.Баумана, 2001.-302с.
9. Калверт Ч., Рейсдорф К. Borland C++ Builder 5. Энциклопедия программиста: К.: - Издательство «Диасофт», 2001.-944с.
10. Мадрахимов Ш.Ф., Гайназаров С.М. C++ тилида программалаш асослари. Услубий қўлланма. ЎзМУ, 2009 й, 196 бет.
11. Xaldjigitov A.A., Madraximov Sh.F., Adamboev U.E. Informatika va dasturlash. O'quv qo'llanma, O'zMU, 2005 yil, 145 bet.
12. Madraximov Sh.F., Ikromov A.M., Babajanov M. C++ tilida dasturlash bo'yicha masalalar to'plami. O'quv qo'llanma, O'zMU, 2014 yil, 164 bet.

Madraximov Sh. F.

**C++. OBYEKTGA YO'NALTIRILGAN
DASTURLASH**

O'quv qo'llanma

Toshkent - "Metodist nashriyoti" - 2023

Muharrir: Xolsaidov F. B.

Bichimi 60x90. "Cambria" garniturası. Ofset bosma usulida bosildi.
Shartli bosma tabog'i 11. Nashr bosma tabog'i 10,75.
Adadi 200 nusxa.

"Metodist nashriyoti" MCHJ matbaa bo'limida chop etildi.
Manzil: Toshkent shaxar, Yakkasaroy ko'chasi 5 uy



+99893 552-11-21

Muallif va nashriyot rozilgisiz chop etish ta'qiqlanadi.

ISBN 978-9943-9629-7-2



9 789943 962972