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

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT
AXBOROT TEXNOLOGIYALARI UNIVERSITETI

B. B. MO‘MINOV

DASTURLASH I

O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta’lim vazirligi tomonidan oliy o‘quv yurtlarining talabalari uchun darslik sifatida tavsiya etilgan
(28.12.2020 yil 676 sonli buyrug‘i)

TOSHKENT – 2021

UO‘K: 000(000)
KBK 00.00
M00
Mo‘minov B.B., Dasturlash I. (Darslik). – T.: «Nihol print» OK, 2021. –280 b.

ISBN 978–9943–

Darslikda algoritm va dasturlashning asosiy tushunchalari, jumladan, til alifbosi, identifikator, kalit so‘zlar, satrli o‘zgaruvchilar, ma’lumotlar tipi, arifmetik ifoda va amallar, siljitish amallari, inkrement, dekrement, kutubxonalar va ularning funksiyalari hamda preprotsessor direktivalaridan foydalanish usullari, takrorlanuvchi jarayonlarni tashkil qiluvchi operatorlar va ulardan foydalanish usullari hamda shartli va shartsiz jarayonlarni samarali tashkil qilish usullari va misollari, funksiya va ularni tashkil qilish usullari, rekursiv funksiya va ularni qo‘llanilishi, bir va ko‘p o‘lchovli massiv hamda ularning elementlari ustida amallar bajarish algoritm va usullari, massivni funksiya paramatri sifatida qo‘llanilish yo‘llari, saralash algoritimlari, obyektga yo‘naltirilgan dasturlash tamoyllarini tashkil qilish usullari, sinflar va konstruktorlar hamda ulardan foydalanish yo‘llari, obyektlar va ularni parametr sifatida qo‘llanilish usullari, satrlar va belgili o‘zgaruvchilarga ishlov berish hamda ularni tashkil qiluvchi standart funksiyalar, ulardan foydalanish yo‘llari, matnli va binar fayllar bilan ishlash usullari, istisnoli holatlarni qayta ishlash yo‘llari, inkapsulyatsiyani tashkil qilish yo‘llari, merosxo‘rlik va ulardan foydalanish usullari, polimorfizm va uning turlari, operatorlarni qayta yuklash hamda shablonlar va ularni tashkil qilish shartlari va ketma-ketliklari bo‘yicha nazariy va amaliy materiallar bayon qilingan.

UO‘K: 000(000)
KBK 00.00

Taqrizchilar:

Madraximov Sh.F. – Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti “Algoritmlar va dasturlash texnologiyalari” kafedrasi mudiri, t.f.d., dosent;

Nishanov A.X. – Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti “Axborot texnologiyalarining dasturiy ta’minoti” kafedrasi professori, t.f.d.

ISBN 978–9943–

KIRISH

O‘zbekiston Respublikasi Prezidentining bir qator Qaror va Farmonlari Axborot-kommunikatsiya texnologiyalarini yanada jadal rivojlantirish maqsadida qabul qilinmoqda. O‘zbekiston Respublikasi Prezidentining 2020 yilning 28-aprelidagi PQ-4699-sonli “Raqamli iqtisodiyot va elektron hukumatni keng joriy etish chora-tadbirlari to‘g‘risi”dagi qarori va 2020 yil 5-oktabrda PF-6079-sonli “«Raqamli O‘zbekiston — 2030» strategiyasini tasdiqlash va uni samarali amalga oshirish chora-tadbirlari to‘g‘risi”dagi farmonlarida takidlanganidek «Bir million dasturchi» loyihasi doirasida 500 ming nafar yoshlarni qamrab olish orqali kompyuterli dasturlash asoslariga o‘qitishni tashkillashtirish maqsadida Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universitetining barcha yo‘nalish talabalariga mo‘ljallangan “Dasturlash I” fani uchun darslikni yaratish dolzarb masala hisoblanadi.

Darslikning birinchi bobida algoritm va dasturlashga kirish bayon etilgan. Algoritm va dasturlashning asosiy tushunchalari ochib berilgan. Jumladan, til alifbosi, identifikator, kalit so‘zlar, satrli o‘zgaruvchilar, ma’lumotlar tipi, arifmetik ifoda va amallar, siljitish amallari, inkrement, dekrement, kutubxonalar va ularning funksiyalari hamda preprotsessor direktivalaridan foydalanish usullari keltirilgan.

Darslikning ikkinchi bobida chiziqli, tarmoqlanuvchi va takrorlanuvchi jarayonlarni tashkil qiluvchi operatorlarga bag‘ishlangan. Tarmoqlanuvchi, takrorlanuvchi jarayonlarni tashkil qiluvchi operatorlar va ulardan foydalanish usullari hamda shartli va shartsiz jarayonlarni samarali tashkil qilish usullari va misollar bayon qilingan.

Darslikning uchinchi bobida funksiyalar va to‘plamlar bilan ishlash operatorlari, funksiya va ularni tashkil qilish usullari, rekursiv funksiya va ularni qo‘llanilishi, bir va ko‘p o‘lchovli massivlar hamda ularning elementlari ustida amallar bajarish algoritm va usullari, massivni funksiya paramatri sifatida qo‘llanilish yo‘llari, saralash algoritimlarini qamrab olgan.

Darslikning to‘rtinchi bobida obyektga yo‘naltirilgan dasturlash asoslari deb nomlangan bobda obyektga yo‘naltirilgan dasturlash tamoyllarini tashkil qilish usullari, sinflar va konstruktorlar hamda ulardan foydalanish yo‘llari, obyektlar va ularni parametr sifatida qo‘llanilish usullari o‘rgatishga qaratilgan.

Darslikning beshinchi bobida satrlar va fayllar bilan ishlash usullari bayon etilgan. Satrlar va belgili o‘zgaruvchilarga ishlov berish

hamda ularni tashkil qiluvchi standart funksiyalar, ulardan foydalanish yo‘llari, matnli va binar fayllar bilan ishlash usullari, istisnoli holatlarni qayta ishlash yo‘llari bayon qilingan.

Darslikning oltinchi bobida inkapsulyatsiya, merosxo‘rlik, polimorfizm va shablonlar bilan ishlashga bag‘ishlangan. Inkapsulyatsiyani tashkil qilish yo‘llari, merosxo‘rlik va ulardan foydalanish usullari, polimorfizm va uning turlari, operatorlarni qayta yuklash hamda shablonlar va ularni tashkil qilish shartlari va ketma-ketliklari bayon qilingan.

Darslik bo‘yicha takliflar, undagi aniqlangan kamchiliklar bo‘yicha fikr-mulohazalaringizni mbbahodir@gmail.com elektron pochtasiga yuborishingizni so‘raymiz. Sizning bu bildirgan fikr-mulohazalaringiz darslikning keyingi nashrlarida uni yanada mazmunliroq va kamchiliklardan holi tarzda chop etishimizga albatta asqotadi.

Muallif

1-BOB. ALGORITM VA DASTURLASHGA KIRISH

1.1. Dasturlashga kirish, dasturlashning asosiy tushunchalari

📖 Dasturlash tillarining tuzilmasi, tilning bazaviy tushunchalari, preprotsessor direktivalari va vositalari, identifikatorlar, ularning tiplari, ularga qiymat o‘zlashtirish usullari va operatorlari, amallar bajarilish ustivorligi va boshqa tilning asosiy tushunchalari keltirilgan. Bundan tashqari oddiy dasturning tuzilishi, o‘zgaruvchilar, operatsiyalar, iboralar, asosiy tiplarning o‘zgarishi, kiritish-chiqirish vositalari va dastur strukturasi, kutubxona funksiyalari, matematik funksiyalar haqida ma’lumotlar bayon etilgan.

Kalit so‘zlar. *kommunikatsiya, dasturiy ta’minot, kompilyator, loyihalash, foydalanuvchi interfeysi, foydalanuvchi, aniqlik, dasturchi, samaradorlik, iostream, kiritish, chiqarish amali.*

REJA:

1. Tilning bazaviy tushunchalari (til alifbosi, identifikator va leksemalar, kalit so‘zlar, konstanta satrlar, ma’lumotlar tipi)
2. Arifmetik ifoda va amallar, siljitish amallari, inkrement va dekrement
3. Bitlarga ishlov beruvchi operatorlar, ma’lumotlar tipini o‘zgartirish
4. Kutubxona funksiyalari
5. Preprotsessor direktivalari va vositalari.

C++ tili Byarn Straustrup tomonidan 1980 yilning boshlarida ishlab chiqilgan. C++ tilida yaxshi dastur tuzish uchun “aql, farosat va sabr” kerak bo‘ladi. Bu til asosan tizim sathida dasturlovchilar uchun yaratilgan.

C/C++ algoritmik tilining alifbosi:

1. 26 ta lotin harflari(katta va kichik);
2. 0 dan 9 gacha bo‘lgan arab raqamlari;
3. Maxsus belgilar: - + * / : ; . , % ? ! = “” <> {} [] () $ # & ^ va h.k.

Dastur bajarilishi jarayonida o‘z qiymatini o‘zgartira oladigan kattaliklar ***o‘zgaruvchilar*** deyiladi. O‘zgaruvchilarning nomlari harfdan boshlanuvchi harf va raqamlardan iborat bo‘lishi mumkin. O‘zguruvchilarni belgilashda katta va kichik harflar bir biridan farq qiladi(A va a harflari 2 ta o‘zgaruvchini bildiradi). Har bir o‘zgaruvchi o‘z nomiga, tipiga, xotiradan egallagan joyiga va son qiymatiga ega bo‘lishi kerak. O‘zgaruvchiga murojaat qilish uning nomi orqali bo‘ladi. O‘zgaruvchi uchun xotiradan ajratilgan joyning tartib raqami uning

manzili hisoblanadi. O‘zgaruvchi ishlatilishidan oldin u aniqlangan bo‘lishi lozim.

O‘zgaruvchilarning son qiymatlari quyidagi ko‘rinishda yoziladi:

- butun tipli o‘nlik sanoq tizimida: ular faqat butun sondan iborat bo‘ladilar. Masalan: 5; 76; -674;
- sakkizlik sanoq tizimidagi sonlar: 0 (nol) dan boshlanib, 0 dan 7 gacha bo‘lgan raqamlardan tashkil topadi. Masalan: *x*=0453217;
- char belgi tipi uning o‘zgarish intervali 0 dan 255 gacha yoki apostrof ichidagi ixtiyoriy 1 ta simvol. Xotiradan 1 bayt joy oladi. Simvollar ASCII kodlariga mos keladi. (ASCII – American Standart Code for Information Interchange);
- butun tipli o‘zgaruvchilar: int. Masalan: int a, i, j ; Bu yerda dasturda ishlatilayotgan a, i, j o‘zgaruvchilarining tipi butun ekanligi ko‘rsatildi. Bu tipdagi o‘zgaruvchilar 2 bayt joy egallaydi. Ularning o‘zgarish oralig‘i: -32768 dan +32767 gacha; (Hozirgi 32 razryadli kompyuterlarda 4 bayt joy oladi va oralig‘i 2 marta oshgan);
- butun tipli katta (uzun) o‘zgaruvchilar: long. Masalan: long s, s2, aa34; Bu tipdagi o‘zgaruvchilar xotiradan 4 bayt joy egallaydi. Ular –2147483648 dan +2147483647 gacha bo‘lgan oraliqdagi sonlarni qabul qilishi mumkin;
- ishorasiz butun o‘zgaruvchilar: unsigned short – 2 bayt joy oladi, o‘zgarish oralig‘i 0 dan 65535 gacha; unsigned long – 4 bayt joy oladi, o‘zgarish oralig‘i: 0 dan 4294967295 gacha;
  unsigned char 1 bayt joy oladi, o‘zgarish oralig‘i 0 dan 255 gacha;
- haqiqiy tipdagi o‘zgaruvchilar: float. Masalan: float a, b: Bu yerda dasturda ishlatilayotgan a, b o‘zgaruvchilarining tipi haqiqiy ekanligi ko‘rsatilgan. Bu tipdagi o‘zgaruvchilar 4 bayt joy egallaydi va qabul qilish chegarasi 10^{-38} dan 10^{+38} gacha;
- katta yoki kichik qiymatli o‘zgaruvchilarni ifoda etishda double tipi ishlatiladi. Ular uchun 8 bayt joy ajratiladi va qabul qilish chegarasi 10^{-304} dan 10^{+304} gacha;
- juda katta yoki juda kichik qiymatli o‘zgaruvchilar uchun long double tipi ishlatiladi, u 10 bayt joy oladi va qiymat qabul qilish chegarasi $3.4*10^{-4932}$ dan $1.1*10^{-4932}$ gacha;
- qator tipidagi o‘zgaruvchilar uchun ham char tipi belgilangan. Ular ham 1 bayt joy oladi va 0 dan 256 tagacha bo‘lgan simvollar ketma-ketligidan iborat bo‘lishi mumkin. Satr tipidagi o‘zgaruvchilar qo‘shtirnoq (“) ichida yoziladi.

C++ tilida o‘zgaruvchilarni initsializatsiya qilish degan tushuncha ham mavjud. Initsializatsiya qilish degani o‘zgaruvchini e’lon qilish barobarida unga boshlang‘ich qiymatini ham berish demakdir. Masalan: int a=5, b, c=-100;– a, b, c o‘zgaruvchilari butun tipli ekanligi ko‘rsatildi va a o‘zgaruvchisiga 5 (a=5), c o‘zgaruvchisiga esa –100 (c=-100) boshlang‘ich qiymatlar berildi.

Dastur bajarilishi jarayonida o‘z qiymatini o‘zgartira olmaydigan kattaliklar ***o‘zgarmaslar*** deyiladi. Masalan: x=1; bo‘lsa keyinchalik x=x+5 deb yozib bo‘lmaydi. O‘zgarmaslarni const so‘zi bilan ko‘rsatiladi. Maslan: const int x=95; float y=9.17; (o‘zgarmaslar simvol yoki NULL bo‘lishi ham mumkin).

1.1-jadval. C++ tilida standart funksiyalarning yozilishi

Funksiya	Ifodalanishi	Funksiya	Ifodalanishi
Sin x	sin(x)	$\sqrt{x}$	sqrt(x); pow(x,1/2.)
Cos x	cos(x)	$\lvert x \rvert$	abs(x) yoki fabs(x)
Tg x	tan(x)	Arctan x	atan(x)
e^x	exp(x)	Arcsin x	asin(x)
Ln x	log(x)	Arccos x	acos(x)
Lg x	log10(x)	$\sqrt[3]{x^2}$	pow(x,2/3.)
x^a	pow(x,a)	Log_2x	log(x)/log(2)

Masalan: $\frac{-b+\sqrt{b^2-4ac}}{2a}$ → (-b + sqrt(b*b-4*a*c)/(2*a); yoki (-b+pow(b*b-4*a*c,1/2.))/(2*a);

$e^{\sin x} + tg^2(x+3)$ → exp(sin(x)) + pow(tan(x+3),2);

k=(m*5)+((7 % n) / (9+x));

C++ tilidagi dastur quyidagilardan tashkil topadi:

1. Direktivalar – # include <file.h> direktiva – instruksiya degan ma’noni beradi. C++ tilida dasturning tuzilishiga, ya’ni ehtiyojiga qarab, kerakli direktivalar ishlatiladi. Ular < > belgisi orasida keltiriladi. C++ tilida jami 32 ta direktiva mavjud bo‘lib ulardan eng ko‘p qo‘llaniladiganlari quyidagilar:

- #include <stdio.h> - C da oddiy kiritish/chiqarish dasturi uchun. Bu yerda std - standart, i – input, o - output degani;
- #include <iostream.h> - C++ da kiritish/chiqarish uchun, oddiy amallar bajariladi;
- #include <math.h> - standart funksiyalarni ishlatish uchun;

- #include <conio.h> - dasturning tashqi koʻrinishini shakllantirish uchun;
- #include <string.h> - satr tipidagi oʻzgaruvchilar ustida amallar bajarish uchun;
- #include <stdlib.h> - standart kutubxona fayllarini chaqirish uchun;
- #include <time.h> - kompyuter ichidagi vaqt qiymatlaridan foydalanish uchun;
- #include <graphics.h> - C++ tilining grafik imkoniyatlaridan foydalanish uchun.

Bu direktivalar maxsus kutubxonalarni e'lon qilish fayllari hisoblanib, ular alohida INCLUDE deb nomlanadigan papkada saqlanadi. Hozirda vaqtda C++ kutubxonasi yangilanib undagi fayllarning nomlaridan .h (head – bosh ma'nosida) kengaytmasi olib tashlandi va oldiga **c** harfi qoʻshildi(C dan qolgan 18 tasiga). Bu fayllarda funksiya prototiplari, tiplari, oʻzgaruvchilar, oʻzgarmaslar tariflari yozilgan boʻladi.

Direktivalar dasturni uni kompilyatsiya qilinishidan oldin tekshirib chiqadi.

2. Makroslar - # define makro qiymati. Masalan:

```
    #define y sin(x+25) – y = sin(x+25) //qiymati berildi;
    #define pi 3.1415   – pi = 3.1415
    #define s(x) x*x    – s(x) = x*x    //(; belgisi qoʻyilmaydi)
```

Global oʻzgaruvchilarni e'lon qilish. Asosiy funksiya ichida e'lon qilingan oʻzgaruvchilar lokal, funksiyadan tashqarida e'lon qilinganlari esa global oʻzgaruvchilar deyiladi. Global oʻzgaruvchilar dastur davomida ishlaydi va xotiradan ma'lum joyni egallaydi. Oʻzgaruvchini bevosita ishlatishdan oldin e'lon qilsa ham boʻladi, u holda oʻzgaruvchi lokal boʻladi. Global oʻzgaruvchilar nomi lokal oʻzgaruvchilar nomi bilan bir xil boʻlishi ham mumkin. Bunday holatda lokal oʻzgaruvchining qiymati joriy funksiya ichidagini qiymatini oʻzgartiradi, funksiyadan chiqishi bilan global oʻzgaruvchilar ishlaydi.

Asosiy funksiya - main() hisoblanadi. Bu funksiya dasturda boʻlishi shart. Umuman olganda C++ dagi dastur funksiyalardan iborat deb qaraladi. main() funksiyasi { boshlanadi va dastur oxirida berkitilishi shart }. main – asosiy degan ma'noni beradi. Bu funksiya oldida uning tipi koʻrsatiladi. Agar main() funksiyasi beradigan (qaytaradigan) javob oddiy soʻz yoki gaplardan iborat boʻlsa, hech

qanday natija qaytarmasa, void so‘zi keltiriladi. main ( ) funksiyasi dastur tomonidan emas, balki OS tomonidan chaqiriladi. OSga qiymat qaytarish shart emas, chunki u bu qiymatdan foydalanmaydi. Shuning uchun main( ) funksiyasining tipini *void* deb ko‘rsatganimiz ma’qul. Har bir funksiyaning o‘z argumenti bo‘ladi, shuning uchun main funksiya ( ) belgilari ichiga uning parametri keltiriladi. Ba’zan u bo‘sh bo‘lishi ham mumkin. Bu funksiyadan chiqish uchun odatda *return* operatori ishlatiladi. 0 (nol) qiymatining qaytarilishi operasion tizimga ushbu dastur normal bajarilib turganini bildiradi. return orqali qaytadigan qiymat tipi funksiya e’lonidagi qaytish tipi bilan bir xil bo‘lishi kerak. Masalan int main ( ) va 0 (nol) qiymat butun tiplidir. Bu funksiyadan so‘ng lokal o‘zgaruvchilar, qism dasturlar, ularning haqiqiy parametrlari e’lon qilinadi. So‘ngra dasturning asosiy operatorlari (kiritish/chiqarish, hisoblash va h.k.) yoziladi. Agar bu operatorlar murakkab tipli bo‘lsa, ularni alohida {} qavslarga olinadi. C++ tilida dastur kichik harflarda yoziladi. Ba’zi operatorlar katta harflar bilan kelishi mumkin, bunday xollarda ular alohida aytib o‘tiladi. Operatorlar oxiriga ; belgisi qo‘yiladi. Operatorlar bir qatorga ketma-ket yozilishi mumkin. Dasturda izohlar ham kelishi mumkin, ular /* ....*/ belgisi orasiga olinadi. Agar izoh bir qatorda tugasa, uni // belgisidan keyin yoziladi. Masalan:

```
main ( ) // C++ tilining asosiy funksiyasi
```

Tilda quyidagi amallardan foydalanish mumkin:

Arifmetik amallar: +, -, /, *, %. Barcha amallar odatdagidek bajariladi, faqat bo‘lish amali butunga bo‘lish bajariladi, ya’ni agar butun sonlar ustida bajarilayotgan bo‘lsa, natija doim butun bo‘ladi, ya’ni kasr qism tashlab yuboriladi (9/5=1; vaxolanki 1,8 bo‘lishi kerak). Shuning uchun surat yoki maxrajiga nuqta (.) qo‘yilsa, natija ham haqiqiy bo‘ladi (9./5=1.8). % belgisi (modul operatori) esa butun sonni butun songa bo‘lgandan hosil bo‘ladigan qoldiqni bildiradi.

Masalan: 9 % 5=4

Taqqoslash amallari: = = (tengmi?); != (teng emas); < ; > ; >=; <=

Mantiqiy amallar: && (and) mantiqiy ko‘paytirish; || (or) mantiqiy qo‘shish; ! (not) mantiqiy inkor. Mantiqiy amallarni ixtiyoriy sonlar ustida bajarish mumkin. Agar javob rost bo‘lsa, natija 1 bo‘ladi, agar javob yolg‘on bo‘lsa, natija 0 bo‘ladi. Umuman olganda 0 (nol) dan farqli javob rost deb qabul qilinadi.

Masalan: i>50 && j==24 yoki s1 < s2 && (s3>50 || s4<=20);
Yoki $6 \le x \le 10$ yozuvi x>=6 && x<=10 deb yoziladi

Qiymat berish amallari:

a=5; b = 2*c; x = y = z =1; a = (b = c)*d // 3=5 deb yozib bo‘lmaydi

Qabul qildim va almashtirdim deb nomalanadigan amallar:

+ = : a+=b → a = a + b;

- = : a-=b → a = a - b;

* = : a*=b → a = a * b;

/ = : a/=b → a = a / b;

% = : a%=b → a = a % b;

- inkrement operatsiyasi (++) ikki ma’noda ishlatiladi: o‘zgaruvchiga murojaat qilinganidan keyin uning qiymati 1 ga oshadi (a++ postfiks ko‘rinishi) va o‘zgaruvchining qiymati unga murojaat qilishdan oldin 1 ga oshadi (++a prefiks ko‘rinishi);
- dekrement operatsiyasi (--), xuddi inkrement operatsiyasi kabi, faqat kamaytirish uchun ishlatiladi. Masalan: s = a + b++ (a ga b ni qo‘shib keyin b ning qiymatini 1 ga oshiradi); s = a+(--b) (b ning qiymatini 1 ga kamaytirib, keyin a ga qo‘shadi).

Yuqoridagi standart funksiyalardan tashqari quyidagi funksiyalar ham ishlatiladi:

- ceil (x) - x ni x dan katta yoki unga teng bo‘lgan eng kichik butun songacha yaxlitlash. Masalan: ceil (12.6) = 13.0; ceil (-2.4) = -2.0;
- floor (x) - x ni x dan kichik bo‘lgan eng katta butun songacha yaxlitlash. Masalan: floor (4.8) = 4.0; floor (-15.9) = -16.0; floor(12.1) = 12; floor(-12.1)=-13;
- fmod (x,y) – x / y ning qoldig‘ini kasr son ko‘rinishida berish. Masalan: fmod(7.3, 1.7) = 0.5;

C++ tilidagi dastur tuzilishi va uning kompilyatsiyasi. C++ tilida dastur yaratish bir nechta bosqichlardan iborat bo‘ladi. Dastlab, matn muharririda (odatda dasturlash muhitining muharririda) dastur matni teriladi, bu faylning kengaytmasi «.cpp» bo‘ladi. Keyingi bosqichda dastur matni yozilgan fayl kompilyatorga uzatiladi, agarda dasturda xatoliklar bo‘lmasa, kompilyator «.obj» kengaytmali obyekt modul faylini hosil qiladi. Oxirgi qadamda komponovka (yig‘uvchi) yordamida «.exe» kengaytmali bajariluvchi fayl - dastur hosil bo‘ladi. Bosqichlarda yuzaga keluvchi fayllarning nomlari boshlang‘ich matnli faylning nomi bilan bir xil bo‘ladi.

Kompilyatsiya jarayonining o‘zi ham ikkita bosqichdan tashkil topadi. Boshida preprotsessor ishlaydi, u matndagi kompilyatsiya direktivalarini bajaradi, xususan #include direktivasi bo‘yicha ko‘rsatilgan kutubxonalardan C++ tilida yozilgan modullarni dastur

tarkibiga kiritadi. Shundan so‘ng kengaytirilgan dastur matni kompilyatorga uzatiladi. Kompilyator o‘zi ham dastur bo‘lib, C++ tilida yozilgan dastur matni uning uchun kiruvchi ma’lumot hisoblanadi. Kompilyator dastur matnini leksema (atomar) elementlarga ajratadi va uni leksik, keyinchalik sintaktik tahlil qiladi. Leksik tahlil jarayonida u matnni leksemalarga ajratish uchun «probel ajratuvchisini» ishlatadi. Probel ajratuvchisi - probel belgisi (‘␣‘), ‘\t‘ - tabulyasiya belgisi, ‘\n‘- keyingi qatorga o‘tish belgisi, boshqa ajratuvchilar va izohlar hisoblanadi.

Dastur matni tushunarli bo‘lishi uchun izohlar ishlatiladi. Izohlar kompilyator tomonidan «o‘tkazib» yuboriladi va ular dasturning bajarilishiga hech qanday ta’sir qilmaydi.

C++ tilida izohlar ikki ko‘rinishda yozilishi mumkin.

Birinchisi “**/***” dan boshlanib, “***/**” belgilar oralig‘ida joylashgan barcha belgilar ketma-ketligi izoh hisoblanadi. Ikkinchisi «satriy izoh» deb nomlanadi va u “//” belgilardan boshlangan va satr oxirigacha yozilgan belgilar ketma-ketligi bo‘ladi. Izohning birinchi ko‘rinishida yozilgan izohlar bir necha satr bo‘lishi va ulardan keyin C++ operatorlari davom etishi mumkin.

Misol.

```
int main(){
 // bu qator izoh hisoblanadi
 int a=0; // int d;
 int c;
 /* int b=15 */
 /* - izoh boshlanishi
 a=c;
 izoh tugashi */
 return 0; }
```

Dasturda d, b o‘zgaruvchilar e’lonlari inobatga olinmaydi va a=c amali bajarilmaydi.

Quyida C++ tilidagi sodda dastur matni keltirilgan.

```
# include <iostream.h> // sarlavha faylni
qo‘shish
int main ()          // bosh funksiya tavsifi
{                 // blok boshlanishi
 cout << ”Salom Olam!\n”;// satrni chop etish
 return 0;           // funksiya qaytaradigan
qiymat
```

```
}                    // blok tugashi
```

Dastur bajarilishi natijasida ekranga “Salom Olam!” satri chop etiladi.

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

Dastur tanasida konsol rejimida belgilar ketma-ketligini oqimga chiqarish amali qo‘llanilgan. Ma’lumotlarni standart oqimga (ekranga) chiqarish uchun quyidagi format ishlatilgan:

```
cout <<  <ifoda>;
```

Bu yerda <ifoda> sifatida o‘zgaruvchi yoki sintaksisi to‘g‘ri yozilgan va qandaydir qiymat qabul qiluvchi til ifodasi kelishi mumkin (*keyinchalik, burchak qavs “<>” ichiga olingan o‘zbek tilidagi matn C++ tili tarkibiga kirmaydi deb qabul qilish kerak).*

Masalan:

```
int uzg=324;
cout<<uzg;  // butun son chop etiladi
```

Ma’lumotlarni standart oqimdan (odatda klaviaturadan) o‘qish quyidagi formatda amalga oshiriladi:

```
cin >> <o‘zgaruvchi>;
```

Bu yerda <o‘zgaruvchi> qiymat qabul qiluvchi o‘zgaruvchining nomi.

Misol:

```
int Yosh;
cout<<”Yoshingizni kiriting_”;
cin>>Yosh;
```

Butun tipidagi Yosh o‘zgaruvchisi kiritilgan qiymatni o‘zlashtiradi. Kiritilgan qiymatni o‘zgaruvchi tipiga mos kelishini tekshirish mas‘uliyati dastur tuzuvchisining zimmasiga yuklanadi.

Bir paytning o‘zida probel vositasida bir nechta va har xil tipdagi qiymatlarni oqimdan kiritish mumkin. Qiymat kiritish <enter> tugmasini bosish bilan tugaydi. Agar kiritilgan qiymatlar soni o‘zgaruvchilar sonidan ko‘p bo‘lsa, «ortiqcha» qiymatlar bufer xotirada saqlanib qoladi.

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

```
    int x,y;
    float z;
    cin>>x>>y>>z;
    cout<<O‘qilgan qiymatlar\n”;
    cout<<x<<‘\t‘<<y<<‘\t‘<<z;
    return 0;          }
```

O‘zgaruvchilarga qiymat kiritish uchun klaviatura orqali

```
    10 20 3.14 <enter>
```

harakati amalga oshiriladi. Shuni qayd etish kerakki, oqimga qiymat kiritishda probel ajratuvchi hisoblanadi. Haqiqiy sonning butun va kasr qismlari ‘.‘ belgisi bilan ajratiladi.

C++ tilida leksemalar.

Alfavit belgilaridan tilning leksemalari shakllantiriladi: identifikatorlar; kalit (xizmatchi yoki zahiralangan) so‘zlar; o‘zgarmaslar; amallar belgilanishlari; ajratuvchilar.

Identifikatorlar va kalit so‘zlar. Dasturlash tilining muhim asosiy tushunchalaridan biri - identifikator tushunchasidir. *Identifikator* deganda katta va kichik lotin harflari, raqamlar va tag chiziq (‘_‘) belgilaridan tashkil topgan va raqamdan boshlanmaydigan belgilar ketma-ketligi tushuniladi. Identifikatorlarda harflarning registrlari (katta yoki kichikligi) hisobga olinadi. Masalan, RUN, run, Run - bu har xil identifikatorlardir. Identifikator uzunligiga chegara qo‘yilmagan, lekin ular kompilyator tomonidan faqat boshidagi 32 belgisi bilan farqlanadi.

Identifikatorlar kalit so‘zlar, o‘zgaruvchilar, funksiyalar, nishonlar va boshqa obyektlarni nomlashda ishlatiladi.

C++ tilining kalit so‘zlariga quyidagilar kiradi:

asm, auto, break, case, catch, char, class, const, continue, default, delete, do, double, else, enum, explicit, extern, float, for, friend, goto, if, inline, int, long, mutable, new, operator, private, protected, public, register, return, short, signed, sizeof, static, struct, swith, template, this, throw, try, typedef, typename, union, unsigned, virtual, void, volatile, while.

Yuqorida keltirilgan identifikatorlarni boshqa maqsadda ishlatish mumkin emas.

Protsessor registrlarini belgilash uchun quyidagi so‘zlar ishlatiladi:

_AH, _AL, _AX, _EAX, _BH, _BL, _BX, _EBX, _CL, _CH, _CX, _ECX, _DH, _DL, _DX, _EDX, _CS, _ESP, _EBP, _FS, _GS, _DI, _EDI, _SI, _ESI, _BP, _SP, _DS, _ES, _SS, _FLAGS.

Bulardan tashqari «__» (ikkita tagchiziq) belgilaridan boshlangan identifikatorlar kutubxonalar uchun zahiralangan. Shu sababli '_' va «__» belgilarni identifikatorning birinchi belgisi sifatida ishlatmagan ma'qul. Identifikator belgilar orasida probel ishlatish mumkin emas. Zarur bo'lganda uning o'rniga '_' ishlatish mumkin: Silindr_radiusi, aylana_diametiri.

O'zgarmaslar. ***O'zgarmas* (*literal*) - bu fiksirlangan sonni, satrni va belgini ifodalovchi leksemadir.**

O'zgarmaslar beshta guruhga bo'linadi - *butun*, *haqiqiy* (*suzuvchi nuqtali*), *sanab o'tiluvchi*, *belgi* (*literli*) va *satr* («*string», literli satr*).

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

Butun o'zgarmaslar. Butun o'zgarmaslar quyidagi formatlarda bo'ladi:

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

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

Manfiy o'zgarmas - bu ishorasiz o'zgarmas bo'lib, unga faqat ishorani o'zgartirish amali qo'llanilgan deb hisoblanadi.

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

023; 0777; 0.

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

0x1A; 0X9F2D; 0x23.

Harf belgilar ixtiyoriy registrlarda berilishi mumkin.

Kompilyator sonning qiymatiga qarab unga mos tipni belgilaydi. Agar tilda belgilangan tiplar dastur tuzuvchini qanoatlantirmasa, u oshkor ravishda tipni ko'rsatishi mumkin. Buning uchun butun o'zgarmas raqamlari oxiriga, probelsiz l yoki L (long), u yoki U (unsigned) yoziladi. Zarur hollarda bitta o'zgarmas uchun bu belgilarning ikkitasini ham ishlatish mumkin:

45lu, 012Ul, 0xA2L.

Haqiqiy o'zgarmaslar. Haqiqiy o'zgarmaslar - suzuvchi nuqtali son bo'lib, u ikki xil formatda berilishi mumkin:

- o‘nlik fiksirlangan nuqtali formatda. Bu ko‘rinishda son nuqta orqali ajratilgan butun va kasr qismlar ko‘rinishida bo‘ladi. Sonning butun yoki kasr qismi bo‘lmasligi mumkin, lekin nuqta albatta bo‘lishi kerak. Fiksirlangan nuqtali o‘zgarmaslarga misollar: **24.56; 13.0; 66.; .87**;
- eksponensial shaklda haqiqiy o‘zgarmas 6 qismdan iborat bo‘ladi:
  1) butun qismi (o‘nli butun son);
  2) o‘nli kasr nuqta belgisi;
  3) kasr qismi (o‘nlik ishorasiz o‘zgarmas);
  4) eksponenta belgisi ‘e‘ yoki ‘E‘;
  5) o‘n darajasi ko‘rsatkichi (o‘nli butun son);
  6) qo‘shimcha belgisi (‘F‘ yoki ‘f‘ , ‘L‘ yoki ‘l‘).

Eksponensial shakldagi o‘zgarmas sonlarga misollar: **1e2; 5e+3; .25e4; 31.4e-1** .

Belgi o‘zgarmaslar. Belgi o‘zgarmaslar qo‘shtirnoq (‘,‘-apostroflar) ichiga olingan alohida belgilardan tashkil topadi va u char kalit so‘zi bilan aniqlanadi. Belgi o‘zgarmas uchun xotirada bir bayt joy ajratiladi va unda butun son ko‘rinishidagi belgining ASCII kodi joylashadi. Quyidagilar belgi o‘zgarmaslarga misol bo‘ladi: **‘e‘, ‘@‘, ‘7‘, ‘z‘,‘W‘,‘+‘,‘sh‘,‘*‘,‘a’, ‘s‘.**

1.2-jadval. C++ tilida escape -belgilar jadvali

Escape belgilari	Ichki kod (16 son)	Nomi	Amal
\\	0x5S	\	Teskari yon chiziqni chop etish
\‘	0x27	‘	Apostrofni chop etish
\”	0x22	“	Qo‘shtirnoqni chop etish
\?	0x3F	?	So‘roq belgisi
\a	0x07	bel	Tovush signalini berish
\b	0x08	bs	Kursorni 1 belgi o‘rniga orqaga qaytarish
\f	0x0C	ff	Sahifani o‘tkazish
\n	0x0A	lf	Qatorni o‘tkazish
\r	0x0D	cr	Kursorni ayni qatorning boshiga qaytarish
\t	0x09	ht	Navbatdagi tabulyasiya joyiga o‘tish
\v	0x0D	vt	Vertikal tabulyasiya (pastga)
\000	000		Sakkizlik kodi
\xNN	0xNN		Belgi o‘n oltilik kodi bilan berilgan

Ayrim belgi o‘zgarmaslar ‘\‘ belgisidan boshlanadi. Bu belgi birinchidan, grafik ko‘rinishga ega bo‘lmagan o‘zgarmaslarni belgilaydi,

ikkinchidan, maxsus vazifalar yuklangan belgilar - apostrof belgisi(‘), so‘roq belgisini (‘?‘), teskari yon chiziq belgisini (‘\‘) va ikkita qo‘shtirnoq belgisini (‘“‘) chop qilish uchun ishlatiladi. Undan tashqari, bu belgi orqali belgini ko‘rinishini emas, balki oshkor ravishda uning ASCII kodini sakkizlik yoki o‘n oltilik shaklda yozish mumkin. Bunday belgidan boshlangan belgilar escape ketma-ketliklar deyiladi (1.1-jadval).

C++ tilida qo‘shimcha ravishda wide harfli o‘zgarmaslar va ko‘p belgili o‘zgarmaslar aniqlangan.

wide harfli o‘zgarmaslar tipi milliy kodlarni belgilash uchun kiritilgan bo‘lib, u wchar_t kalit so‘zi bilan beriladi, hamda xotirada 2 bayt joy egallaydi. Bu o‘zgarmas L belgisidan boshlanadi:

L‘\013\022‘, L‘cc‘

Ko‘p belgili o‘zgarmas tipi int bo‘lib, u to‘rtta belgidan iborat bo‘lishi mumkin:

‘abc‘, ‘\001\002\003\004‘.

Satr o‘zgarmaslar. Ikkita qo‘shtirnoq (“,”) ichiga olingan belgilar ketma-ketligi *satr o‘zgarmas* deyiladi:

”Bu satr o‘zgarmas va uning nomi string\n”

Satr ichida escape ketma-ketligi ham ishlatilishi mumkin, faqat bu ketma-ketlik apostrofsiz yoziladi.

Probel bilan ajratib yozilgan satrlar kompilyator tomonidan yagona satrga ulanadi (konkantenatsiya):

”Satr - bu belgilar massivi” /* bu satr keyingi satrga qo‘shiladi */ ”, uning tipi char[]”;

Bu yozuv

”Satr - bu belgilar massivi, uning tipi char[]”;

yozuvi bilan ekvivalent hisoblanadi.

Uzun satrni bir nechta qatorga yozish mumkin va buning uchun qator oxirida ‘\‘ belgisi qo‘yiladi:

”Kompilyator har bir satr uchun kompyuter xotirasida
**satr uzunligiga teng sondagi baytlardagi alohida **
xotira ajratadi va bitta - 0 qiymatli bayt qo‘shadi”;

Yuqoridagi uchta qatorda yozilgan satr keltirilgan. Teskari yon chiziq (‘\‘) belgisi keyingi qatorda yozilgan belgilar ketma-ketligini yuqoridagi satrga qo‘shish kerakligini bildiradi. Agar qo‘shiladigan satr boshlanishida probellar bo‘lsa, ular ham satr tarkibiga kiradi.

Satr xotirada joylashganda uning oxiriga ‘\0‘ (0 kodli belgi) qo‘shiladi va bu belgi satr tugaganligini bildiradi. Shu sababli satr uzunligi, uning «haqiqiy» qiymatidan bittaga ko‘p bo‘ladi.

Ma’lumotlar tiplari va o‘zgaruvchilar. Dastur bajarilishi paytida ishlatiladigan ma’lumotlarni(qiymatlarni) saqlab turish uchun o‘zgaruvchilar va o‘zgarmaslardan foydalaniladi. O‘zgaruvchi - dastur obyekti bo‘lib, xotiradagi bir nechta yacheykalarni egallaydi va qiymatlarni saqlash uchun xizmat qiladi. O‘zgaruvchi nomga, o‘lchamga va boshqa atributlarga(ko‘rinish sohasi, amal qilish vaqti va boshqa xususiyatlarga) ega bo‘ladi. O‘zgaruvchilarni ishlatish uchun ular albatta e’lon qilinishi kerak. E’lon qilish natijasida o‘zgaruvchi uchun xotiradan joy zahiralanadi, zahiralangan joyning o‘lchami esa o‘zgaruvchining tipiga bog‘liq bo‘ladi. Shuni qayd etish zarurki, bitta tipga turli apparat platformalarda turlicha joy ajratilishi mumkin.

O‘zgaruvchini e’lon qilish uning tipini aniqlovchi kalit so‘z bilan boshlanadi. Teng(=) belgisi orqali boshlang‘ich qiymat berilishi ham mumkin. Bitta kalit so‘z bilan bir nechta o‘zgaruvchini e’lon qilish mumkin. Bunda o‘zgaruvchilar bir-biridan vergul(,) belgisi bilan ajratiladi. E’lonlar nuqta vergul(;) belgisi bilan tugaydi. O‘zgaruvchi nomi 255 ta belgidan oshmasligi kerak.

C++ tilining asosiy ma’lumotlar tiplari, ularning baytlardagi o‘lchamlari va qiymatlarining chegaralari 1.3-jadvalda keltirilgan.

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

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

1.3-jadval. C++ tilining asosiy tiplari

Tip nomi	Baytlardagi o‘lchami	Qiymat chegarasi
bool	1	true yoki false
unsigned short int	2	0..65535
short int	2	-32768..32767
unsigned long int	4	0..42949667295
long int	4	-2147483648..2147483647
int (16 razryadli)	2	-32768..32767

int (32 razryadli)	4	-2147483648..2147483647
unsigned int (16 razryadli)	2	0..65535
unsigned int (32 razryadli)	4	0..42949667295
unsigned char	1	0..255
char	1	-128..127
float	4	1.2E-38..3.4E38
double	8	2.2E-308..1.8E308
long double (32 razryadli)	10	3.4e-4932..-3.4e4932
void	2 yoki 4	-

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

Mantiqiy tip. Bu tipdagi o‘zgaruvchi bool kalit so‘zi bilan e’lon qilinadi. Bu tipdagi o‘zgaruvchi 1 bayt joy egallaydi va 0 (false, yolg‘on) yoki 0 qiymatidan farqli qiymat (true, rost) qabul qiladi. Mantiqiy tipdagi o‘zgaruvchilar qiymatlar o‘rtasidagi munosabatlarni ifodalaydigan mulohazalarni rost yoki yolg‘on ekanligini tavsiflashda qo‘llaniladi va ular qabul qiladigan qiymatlar matematik mantiq qonuniyatlariga asoslanadi.

Matematik mantiq - fikrlashning shakli va qonuniyatlari haqidagi fan. Uning asosini mulohazalar tashkil qiladi. **Mulohaza** - bu ixtiyoriy jumla bo‘lib, unga nisbatan rost yoki yolg‘on fikrni bildirishi mumkin. Masalan «3>2», «5 - juft son», «Moskva-Ukraina poytaxti» va hakozo. Lekin «0.000001 kichik son» jumlasi mulohaza hisoblanmaydi, chunki «kichik son» tushunchasi juda ham nisbiy, ya’ni kichik son deganda qanday sonni tushunish kerakligi aniq emas. Shuning uchun yuqoridagi jumlani rost yoki yolg‘onligi haqida fikr bildirish qiyin.

Mulohazalarning rostligi holatlarga bog‘liq ravishda o‘zgarishi mumkin. Masalan «bugun - chorshanba» jumlasini rost yoki yolg‘onligi ayni qaralayotgan kunga bog‘liq. Xuddi shunday «x<0» jumlasi x

o‘zgaruvchisining ayni paytdagi qiymatiga mos ravishda rost yoki yolg‘on bo‘ladi.

C++ tilida mantiqiy tip nomi angliyalik matematik Jorj Bul sharafiga bool so‘zi bilan ifodalangan. Mantiqiy amallar «Bul algebrasi» deyiladi.

Mantiqiy mulohazalar ustida uch xil amal bajarish mumkin:

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

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

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

Yurqorida keltirilgan fikrlar asosida mantiqiy amallar uchun rostlik jadvali aniqlangan (1.4-jadval).

1.4-jadval. Mantiqiy amallar uchun rostlik jadvali

Mulohazalar		**Mulohazalar ustida amallar**		
A	B	!A	A && B	A || B
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

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

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

void tipi. void tipidagi dastur obyekti hech qanday qiymatga ega bo‘lmaydi va bu tipdan operatorning til sintaksisiga mos kelishini ta’minlash uchun ishlatiladi. Masalan, C++ tili sintaksisi funksiya

qiymat qaytarishini talab qiladi. Agar funksiya qiymat qaytarmaydigan bo‘lsa, u void kalit so‘zi bilan e’lon qilinadi.

Misollar.

```
int a=0,A=1; float abc=17.5;
double Ildiz;
bool Ok=true;
char LETTER=‘z‘;
void Mening_Funktsiyam();/* funksiya qaytaradigan qiymat inobatga olinmaydi */
```

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

Tiplangan o‘zgarmas e’lonida const kalit so‘zi, undan keyin o‘zgarmas tipi va nomi hamda albatta initsializatsiya qismi bo‘ladi.

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

```
#include <iostream.h>
int main(){
 const double pi=3.1415;
 const int Radius=3;
 double Square=0;
 Square=pi*Radius*Radius;
 cout<<Square<<‘\n‘;
 return 0;}
```

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

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

```
    enum <sanab o‘tiladigan tip nomi>      { <nom1>=<qiymat1>,
            <nom2>=<qiymat2>,  … <nomn>=<qiymatn>  };
```

Bu yerda, ***enum*** - kalit so‘z (inglizcha enumerate - sanamoq);

<sanab o‘tiladigan tip nomi>- o‘zgarmaslar ro‘yxatining nomi;

<nom_i> - butun qiymatli konstantalarning nomlari;

<$qiymat_i$>- shart bo‘lmagan initsializatsiya qiymati (ifoda).

Misol uchun hafta kunlari bilan bog‘liq masala yechishda hafta kunlarini dush (dushanba), sesh (seshanba), chor (chorshanba), paysh (payshanba), juma (juma), shanba (shanba), yaksh (yakshanba) o‘zgarmaslarini ishlatish mumkin va ular sanab o‘tiluvchi tip yordamida bitta satrda yoziladi:

enum Hafta {dush,sesh,chor,paysh,juma,shanba,yaksh};

Sanab o‘tiluvchi o‘zgarmaslar quyidagi xossaga ega: agar o‘zgarmas qiymati ko‘rsatilmagan bo‘lsa, u oldingi o‘zgarmas qiymatidan bittaga ortiq bo‘ladi. Kelishuv bo‘yicha birinchi o‘zgarmas qiymati 0 bo‘ladi.

Initsializatsiya yordamida o‘zgarmas qiymatini o‘zgartirish mumkin:

```
    enum Hafta {dush=8,sesh,chor=12,paysh=13,juma=16,
                shanba, yaksh=20};
```

Bu e’londa sesh qiymati 9, shanba esa 17 ga teng bo‘ladi.

Sanab o‘tiluvchi o‘zgarmaslarning nomlari har xil bo‘lishi kerak, lekin ularning qiymatlari bir xil bo‘lishi mumkin:

```
  enum{nol=0,toza=0,bir,ikki,juft=2,uch};
```

O‘zgarmasning qiymati ifoda ko‘rinishda berilishi mumkin, faqat ifodadagi nomlarning qiymatlari shu qadamdagacha aniqlangan bo‘lishi kerak:

```
  enum {ikki=2,turt=ikki*2};
```

O‘zgarmasni qiymatlari manfiy son bo‘lishi ham mumkin:

```
    enum {minus2=-2,minus1,nul,bir};
```

Tipni boshqa tipga keltirish. **C++ tilida bir tipni boshqa tipga keltirishning oshkor va oshkormas usullari mavjud.**

Umuman olganda, tipni boshqa tipga oshkormas keltirish ifodada har xil tipdagi o‘zgaruvchilar qatnashgan hollarda bajariladi (aralash tiplar arifmetikasi). Ayrim hollarda, xususan asosiy tiplar bilan bog‘liq tipga keltirish amallarida xatoliklar yuzaga kelishi mumkin. Masalan, hisoblash natijasidagi sonning xotiradan vaqtincha egallagan joyi uzunligi, uni o‘zlashtiradigan o‘zgaruvchi uchun ajratilgan joy

uzunligidan katta bo‘lsa, qiymatga ega razryadlarni yo‘qotish holati yuz beradi.

Oshkor ravishda tipga keltirishda, o‘zgaruvchi oldiga qavs ichida boshqa tip nomi yoziladi:

```
#include <iostream.h>
int main(){
 int Integer_1=54;
 int Integer_2;
 float Floating=15.854;
 Integer_1=(int)Floating; // oshkor keltirish;
 Integer_2=Floating; // oshkormas keltirish;
 cout<<“Yangi Integer(Oshkor):
“<<Integer_1<<”\n”;
 cout<<“Yangi Integer(Oshkormas):
“<<Integer_2<<”\n”;
 retipn 0;}
```

Dastur natijasi quyidagi ko‘rinishida bo‘ladi:

```
Yangi Integer(Oshkor): 15
Yangi Integer(Oshkormas): 15
```

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

Dastur matni:

```
#include <iostream.h>
int main(){
 unsigned char A;
 cout<<"Belgini kiriting: ";
 cin>>A; cout<<‘\‘‘<<A<<"‘-belgining ASCCI
kodi="<<(int)A<<‘\n‘;
 return 0;}
```

Dasturning

Belgini kiriting:

so‘roviga

A <enter>

amali bajarilsa, ekranga

‘A’-belgining ASCCI kodi=65

satri chop etiladi.

Arifmetik amallar. Qiymat berish operatori. Ma’lumotlarni qayta ishlash uchun C++ tilida amallarning juda keng majmuasi

aniqlangan. *Amal* - bu qandaydir harakat bo‘lib, u bitta (unar) yoki ikkita (binar) operandlar ustida bajariladi, hisob natijasi uning qaytaruvchi qiymati hisoblanadi.

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

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

<o‘zgaruvchi><amal>=<ifoda>;

Quyidagi dastur matnida ayrim amallarga misollar keltirilgan.

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

Dastur bajarilishi natijasida ekranga quyidagi sonlar qatori paydo bo‘ladi:

4 188 2 12 9 1 482 2

Ifoda tushunchasi. C++ tilida *ifoda* - amallar, operandlar va tinish (punktatsiya) belgilarining ketma-ketligi bo‘lib, kompilyator tomonidan ma’lumotlar ustida ma’lum bir amallarni bajarishga ko‘rsatma deb qabul qilinadi. Har qanday ‘;‘ belgi bilan tugaydigan ifodaga *til ko‘rsatmasi* deyiladi.

C++ tilidagi til ko‘rsatmasiga misol:

```
x=3*(y-2.45);
y=Summa(a,9,c);
```

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

Operandga nisbatan bu amallarning prefiks va postfiks koʻrinishlari boʻladi. Prefiks koʻrinishda amal til koʻrsatmasi boʻyicha ish bajarilishidan oldin operandga qoʻllaniladi. Postfiks holatda esa amal til koʻrsatmasi boʻyicha ish bajarilgandan keyin operandga qoʻllaniladi.

Prefiks yoki postfiks amal tushunchasi faqat qiymat berish bilan bogʻliq ifodalarda oʻrinli:

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

Bu yerda *y* oʻzgaruvchining qiymatini *x* oʻzgaruvchisiga oʻzlashtiriladi va keyin bittaga oshiriladi, *i* oʻzgaruvchining qiymati bittaga kamaytirib, index oʻzgaruvchisiga oʻzlashtiriladi.

Razryadli mantiqiy amallar. Dastur tuzish tajribasi shuni koʻrsatadiki, odatda qoʻyilgan masalani yechishda biror holat roʻy bergan yoki yoʻqligini ifodalash uchun 0 yoki 1 qiymatini qabul qiluvchi *bayroqlardan* foydalaniladi. Bu maqsadda bir yoki undan ortiq baytli oʻzgaruvchilardan foydalanish mumkin. Masalan, bool tipidagi oʻzgaruvchini shu maqsadda ishlatsa boʻladi. Boshqa tomondan, bayroq sifatida baytning razryadlaridan foydalanish ham mumkin. Chunki razryadlar faqat ikki xil qiymatni - 0 yoki 1 sonlarini qabul qiladi. Bir baytda 8 razryad boʻlgani uchun unda 8 ta bayroqni kodlash imkoniyati mavjud.

Faraz qilaylik, qoʻriqlash tizimiga 5 ta xona ulangan va tizim taxtasida 5 ta chiroqcha (indikator) xonalar holatini bildiradi: xona qoʻriqlash tizimi nazoratida ekanligini mos indikatorning yonib turishi(razryadning 1 qiymatini) va xonani tizimga ulanmaganligini indikator oʻchganligi(razryadning 0 qiymatini) bildiradi. Tizim holatini ifodalash uchun bir bayt yetarli boʻladi va uning kichik razryadidan boshlab beshtasini shu maqsadda ishlatish mumkin:

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

Masalan, baytning quyidagi holati 1, 4 va 5 xonalar qoʻriqlash tizimiga ulanganligini bildiradi:

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

Quyidagi jadvalda C++ tilida bayt razryadlari ustida mantiqiy amallar ro‘yxati keltirilgan.

1.5-jadval. Bayt razryadlari ustida bajariladigan mantiqiy amallar

Amallar	Mazmuni
&	Mantiqiy VA (ko‘paytirish)
|	Mantiqiy YOKI (qo‘shish)
^	Istisno qiluvchi YOKI
~	Mantiqiy INKOR (inversiya)

Razryadli mantiqiy amallarning bajarish natijalarini jadval ko‘rinishida ko‘rsatish mumkin.

1.6-jadval. Razryadli mantiqiy amallarning bajarish natijalari

A	B	C=A&B	C=A|B	C=A^B	C=~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Yuqoridagi keltirilgan misol uchun qo‘riqlash tizimini ifodalovchi bir baytli char tipidagi o‘zgaruvchini e’lon qilish mumkin:

char q_taxtasi=0;

Bu yerda q_taxtasi o‘zgaruvchisiga 0 qiymat berish orqali barcha xonalar qo‘riqlash tizimiga ulanmaganligi ifodalanadi:

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

Agar 3-xonani tizimga ulash zarur bo‘lsa

q_taxtasi=q_taxtasi|0x04;

amalini bajarish kerak, chunki $0x04_{16}=00000100_2$ va mantiqiy YOKI amali natijasida q_taxtasi o‘zgaruvchisi bayti quyidagi ko‘rinishda bo‘ladi:

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

Xuddi shunday yo‘l bilan boshqa xonalarni tizimga ulash mumkin, zarur bo‘lsa birdaniga ikkitasini (zarur bo‘lsa barchasini):

q_taxtasi=q_taxtasi|0x1F;

Mantiqiy ko‘paytirish orqali xonalarni qo‘riqlash tizimidan chiqarish mumkin:

q_taxtasi=q_taxtasi&0xFD; // $0xFD_{16}=11111101_2$

Xuddi shu natijani ‘~‘ amalidan foydalangan holda ham olish mumkin. Ikkinchi xona tizimga ulanganligi bildiruvchi bayt qiymati - 00000010_2, demak shu holatni inkor qilgan holda mantiqiy ko‘paytirishni bajarish kerak.

q_taxtasi=q_taxtasi&(~0x02);

Va nihoyat, agar 3-xona indikatorini, uni qanday qiymatda bo‘lishidan qat’iy nazar qarama-qarshi holatga o‘tkazishni «inkor qiluvchi YOKI» amali yordamida bajarish mumkin:

q_taxtasi=q_taxtasi^0x04; // $0x04_{16}=00000100_2$

Razryadli mantiqiy amallarni qiymat berish operatori bilan birgalikda bajarilishining quyidagi ko‘rinishlari mavjud:

&= - razryadli VA qiymat berish bilan;

|= - razryadli YOKI qiymat berish bilan;

^= - razryadli istisno qiluvchi YOKI qiymat berish bilan.

Chapga va o‘ngga surish amallari. **Baytdagi bitlar qiymatini chapga yoki o‘ngga surish uchun, mos ravishda “<<” va “>>” amallari qo‘llaniladi. Amaldan keyingi son bitlar nechta o‘rin chapga yoki o‘nga surish kerakligini bildiradi.**

Masalan:

unsigned char A=12; // $A=00001100_2=0x0C_{16}$
A=A<<2; // $A=00110000_2=0x30_{16}=48_{10}$
A=A>>3; // $A=00000110_2=0x06_{16}=6_{10}$

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

char B=-120; // $B=10001000_2=0x88_{16}$
B=B<<2; // $B=00100000_2=0x20_{16}=32_{10}$
B=-120; // $B=10001000_2=0x88_{16}$
B=B>>3; // $B=11110001_2=0xF1_{16}=-15_{10}$

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

Taqqoslash amallari. C++ tilida qiymatlarni solishtirish uchun taqqoslash amallari aniqlangan (1.7-jadval). Taqqoslash amali binar amal bo‘lib, quyidagi ko‘rinishga ega:

<operand$_1$> <taqqoslash amali> < operand$_2$>

Taqqoslash amallarining natijasi - taqqoslash sharti o‘rinli bo‘lsa, true (rost), aks holda false (yolg‘on) bo‘ladi. Agar taqqoslashda arifmetik ifoda qatnashsa, uning qiymati 0 qiymatidan farqli holatlar uchun 1 deb hisoblanadi.

1.7-jadval. Taqqoslash amallari va ularning qo‘llanishi

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

«Vergul» amali

Til operatorlaridagi bir nechta ifodalarni kompilyator tomonidan yaxlit bir ifoda deb qabul qilishi uchun «vergul» amali qo‘llaniladi. Bu amalni qo‘llash orqali dastur yozishda ma’lum bir samaradorlikka erishish mumkin. Odatda «vergul» amali if va for operatorlarida keng qo‘llaniladi. Masalan, if operatori quyidagi ko‘rinishda bo‘lishi mumkin:

if(i=CallFunc(),i<7)...

Bu yerda, oldin CallFunc() funksiyasi chaqiriladi va uning natijasi *i* o‘zgaruvchisiga o‘zlashtiriladi, keyin *i* ning qiymati 7 bilan solishtiriladi.

Quyidagi jadvalda C++ tilida ishlatiladigan amallar (operatorlar), ularning ustunlik koeffitsientlari va bajarilish yo‘nalishlari ($\Leftarrow$ - o‘ngdan chapga, $\Rightarrow$ - chapdan o‘ngga) keltirilgan.

1.8-jadval. Amallarning ustunliklari va bajarilish yo‘nalishlari

Operator	Tavsifi	Ustunlik	Yo‘nalish
::	Ko‘rinish sohasiga ruxsat berish	16	$\Rightarrow$
[ ]	Massiv indeksi	16	$\Rightarrow$
( )	Funksiyani chaqirish	16	$\Rightarrow$
.	Struktura yoki sinf elementini tanlash	16	$\Rightarrow$
->			
++	Postfiks inkrement	15	$\Leftarrow$
- -	Postfiks dekrement	15	$\Leftarrow$
++	Prefiks inkrement	14	$\Leftarrow$
- -	Prefiks dekrement	14	$\Leftarrow$
sizeof	O‘lchamni olish	14	$\Leftarrow$
(<tip>)	Tipga akslantirish	14	

~	Razryadli mantiqiy INKOR	14	⇐
!	Mantiqiy inkor	14	⇐
-	Unar minus	14	⇐
+	Unar plyus	14	⇐
&	Adresni olish	14	⇐
*	Vositali murojaat	14	⇐
new	Dinamik obyektni yaratish	14	⇐
delete	Dinamik obyektni yo‘q qilish	14	⇐
casting	Tipga keltirish	14	
*	Ko‘paytirish	13	⇒
/	Bo‘lish	13	⇒
%	Bo‘lish qoldig‘i	13	⇒
+	Qo‘shish	12	⇒
-	Ayirish	12	⇒
>>	Razryad bo‘yicha o‘ngga surish	11	⇒
<<	Razryad bo‘yicha chapga surish	11	⇒
<	Kichik	10	⇒
<=	Kichik yoki teng	10	⇒
>	Katta	10	⇒
>=	Katta yoki teng	10	⇒
==	Teng	9	⇒
!=	Teng emas	9	⇒
&	Razryadli VA	8	⇒
^	Razryadli istisno qiluvchi YOKI	7	⇒
|	Razryadli YOKI	6	⇒
&&	Mantiqiy VA	5	⇒
||	Mantiqiy YOKI	4	⇒
?:	Shart amali	3	⇐
=	Qiymat berish	2	⇐
*=	Ko‘paytirish qiymat berish bilan	2	⇐
/=	Bo‘lish qiymat berish bilan	2	⇐
%=	Modulli bo‘lish qiymat berish bilan	2	⇐
+=	Qo‘shish qiymat berish bilan	2	⇐
- =	Ayirish qiymat berish bilan	2	⇐
<<=	Chapga surish qiymat berish	2	⇐

	bilan		
>>=	O‘ngga surish qiymat berish bilan	2	⇐
&=	Razryadli VA qiymat berish bilan	2	⇐
^=	Razryadli istisno kiluvchi YOKI qiymat berish bilan	2	⇐
| =	Razryadli YOKI qiymat berish bilan	2	⇐
throw	Istisno holatni yuzaga keltirish	2	⇐
,	Vergul	1	⇒

C++ tili dastur tuzuvchisiga amallarning bajarilish tartibini o‘zgartirish imkoniyatini beradi. Xuddi matematikadagidek, amallarni qavslar yordamida guruhlash mumkin. Qavs ishlatishga cheklov yo‘q.

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

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

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

x=184

y=348 satrlari chop etiladi.

Preprotsessor vositalari. Sodda dastur tuzilishi. Dastur preprotsessor buyruqlari va bir necha funksiyalardan iborat bo‘lishi mumkin. Bu funksiyalar orasida main nomli asosiy funksiya bo‘lishi shart. Agar asosiy funksiyadan boshqa funksiyalar ishlatilmasa dastur quyidagi ko‘rinishda tuziladi:

Preprotsessor_buyruqlari

int main(){//Dastur tanasi.}

Preprotsessor direktivalari kompilyatsiya jarayonidan oldin preprotsessor tomonidan bajariladi. Natijada dastur matni preprotsessor direktivalari asosida o‘zgartiriladi.

Preprotsessor buyruqlaridan ikkitasini koʻrib chiqamiz.

include <fayl_nomi> Bu direktiva standart kutubxonalardagi funksiyalarni dasturga joyjlash uchun foydalaniladi.

#define <almashtiruvchi ifoda> <almashinuvchi ifoda>

Bu direktiva bajarilganda dastur matnidagi almashtiruvchi ifodalar almashinuvchi ifodalarga almashtiriladi.

Misol tariqasida C ++ tilida tuzilgan birinchi dasturni keltiramiz:

```
#include <iostream.h>
int main(){
    cout << “\n Salom, Dunyo!  \n”;}
```

Bu dastur ekranga Salom, Dunyo! Jumlasini chiqaradi.

Define direktivasi yordamida bu dasturni quyidagicha yozish mumkin:

```
#include <iostream.h>
#define pr  cout << “\n Salom, Dunyo! \n”
#define begin {
#define end  }
int main()
  begin
    pr;
  end
```

Define direktivasidan nomlangan konstantalarni kiritish uchun foydalanish mumkin.

Misol uchun:

#define EULER 2.718282

Agar dasturda quyidagi matn mavjud boʻlsin:

Double mix=EULER

D=alfa*EULER

Preprotsessor bu matndagi har bir EULER konstantani uning qiymati bilan almashtiradi va natijada quyidagi matn hosil boʻladi.

Double mix=2.718282

D=alfa*2.718282

Dastur matni va preprotsessor. C++ tilida matnli fayl shaklida tayyorlangan dastur uchta qayta ishlash bosqichlaridan oʻtadi. Matnni preprotsessor direktivalari asosida oʻzgartiriladi. Bu jarayon natijasi yana matnli fayl boʻlib preprotsessor tomonidan bajariladi.

Kompilyatsiya. Bu jarayon natijasi mashina kodiga oʻtkazilgan obyektli fayl boʻlib, kompilyator tomonidan bajariladi.

Bog‘lash. Bu jarayon natijasi to‘la mashina kodiga o‘tkazilgan bajariluvchi fayl bo‘lib, bog‘lagich(компоновщик) tomonidan bajariladi.

Preprotsessorning vazifasi dastur matnini preprotsessor direktivalari asosida o‘zgartirishdir. Define direktivasi dasturda bir jumlani ikkinchi jumla bilan almashtirish uchun ishlatiladi. Bu direktivadan foydalanishning sodda misollarini biz yuqorida ko‘rib chiqdik. Include direktivasi ikki ko‘rinishda ishlatilishi mumkin.

#include fayl nomi direktivasi dasturning shu direktiva o‘rniga qaysi matnli fayllarni qo‘shish kerakligini ko‘rsatadi.

#include <fayl nomi> direktivasi dasturga kompilyator standart kutubxonalariga mos keluvchi sarlavhali fayllar matnlarini qo‘shish uchun mo‘ljallangan. Bu fayllarda funksiya prototipi, tiplar, o‘zgaruvchilar, konstantalar tariflari yozilgan bo‘ladi. Funksiya prototipi funksiya qaytaruvchi tip, funksiya nomi va funksiyaga uzatiluvchi tiplardan iborat bo‘ladi. Misol uchun cos funksiyasi prototipi quyidagicha yozilishi mumkin: double cos(double). Agar funksiya nomidan oldin void tipi ko‘rsatilgan bo‘lsa bu funksiya hech qanday qiymat qaytarmasligini ko‘rsatadi. Shuni ta’kidlash lozimki bu direktiva dasturga standart kutubxona qo‘shilishiga olib kelmaydi. Standart funksiyalarning kodlari bog‘lash ya’ni aloqalarni tahrirlash bosqichida, kompilyatsiya bosqichidan so‘ng amalga oshiriladi.

Kompilyatsiya bosqichida sintaksis xatolar tekshiriladi va dasturda bunday xatolar mavjud bo‘lmasa, standart funksiyalar kodlarisiz mashina kodiga utkaziladi. Sarlavhali fayllarni dasturning ixtiyoriy joyida ulash mumkin bo‘lsa ham, bu fayllar odatda dastur boshida qo‘shish lozimdir. Shuning uchun bu fayllarga sarlavhali fayl (header file) nomi berilgandir.

Dasturda kiritish va chiqarish funksiyalaridan masalan Cout<< funksiyasidan foydalanish uchun #include <iostream.h> direktivasidan foydalanish lozimdir Bu direktivada iostream.h sarlavhali fayl nomi quyidagilarni bildiradi: st- standart, i- input, o- output, h – head(sarlavha).

Nazariy bilimlarni tekshirish uchun savollar.

1. Algoritmlarni yozishda dasturiy algoritmik va og‘zaki so‘z o‘rtasidagi farq nima? Dasturlarga qanday talablar qo‘yiladi?
2. Obyektga yo‘naltirilgan dasturlash atamasi nimani anglatadi?

3. Kompilyatsiya interpretatsiya tushunchalari orasida qanday farq bor?
4. C ++ tilining tavsifi va maqsadini bilasiz? Tilning umumiy ma’lumotlari va xususiyatlari, tilning tarkibi nimalardan iborat?
5. Ma’lumotni kiritish printsiplari, oddiy ma’lumotlar tiplarining ierarxiyasi, ma’lumotlarning standart tiplari, xarakteristikalar jadvali. Nimalardan iborat?
6. Preprotsessor ko‘rsatmalari, sarlavha fayllari, kutubxona prototiplari vazifalari, ularni chaqirish nimalardan iborat?
7. Dastur matnini qayta ishlash bosqichlari. Matnlarini sarlavha fayllaridan olib qo‘shish.
8. Dasturning asosiy funksiyasi. Funktsiya tuzilishi, uning sarlavhasi nima?
9. C ++ tilining chiziqli operatorlari va oddiy ma’lumotlar tiplari nima?
10. C ++ tilidagi dasturning tuzilishi. Standart kutubxonalar va ularning aloqasi nimalardan iborat?

1.2. Dasturlash tillarining tuzilmasi

📖 *Hozirgi paytda algoritm sifatida biror masalani ishlash yoki biror ishni bajarish uchun qilinishi kerak bo‘lgan tartiblangan chekli sondagi aniq bir qiymatli ko‘rsatmalar ketma-ketligi tushuniladi. Algoritm tushunchasini keng ma’noda tahlil qilish mumkin. Biz asosan hisoblash algoritmlari haqida so‘z yuritamiz. Algoritmlarga xos bo‘lgan belgi va talablarni sanab o‘tamiz.*

✎ ***Kalit so‘zlar.*** *Algoritm, kompilyator, sizeof, ternar, preprotsessor direktivalar.*

REJA:

1. Dastur tarkibi. Konsoldan kiritish va chiqarish.
2. Simvollarni o‘qish va yozish.
3. Konsoldan formatli kiritish va chiqarish. Format modifikatorlari. printf(), scanf() funksiyalari. Format spesifikatorlari.
4. Sonlarni kiritish. Adreslarni kiritish.
5. Statik operator (sizeof), kompilyator va uning turlari;
6. Qiymat o‘zlashtirish operatorlari va ularning ishlash usullari;
7. Preprotsessor direktivalari.

Algoritm — ijrochi uchun ma’lum bir masalani yechishga qaratilgan ko‘rsatmalarning aniq ketma-ketligi.

“Algoritm” so‘zi o‘rta osiyolik buyuk matematik al-Xorazmiyning nomi bilan bog‘liq. al-Xorazmiyning nomini lotincha ifodasi — Algorithm. Algoritm — informatika va matematikaning asosiy tushunchalaridan hisoblanadi.

Algoritm ijrochisi — algoritmda ko‘rsatilgan buyruqlarni bajara oladigan abstrakt yoki real (texnik, biologik yoki biotexnik) tizimdir.

Algoritmlarga xos xususiyatlar:

- oddiy harakatlar;
- buyruqlar tizimi.

Buyruqlar tizimi. Har bir ijrochi faqatgina ushbu ijrochi tushunadigan buyruqlarni (ya’ni, ijrochi bajaradigan buyruqlar ro‘yxatiga mansublarni) bajara oladi.

Ijrochi buyruqlarni bajarish jarayonida oddiy harakatlarni bajaradi.

Odatda ijrochiga algoritmning maqsadi ma’lum bo‘lmaydi. Shuning uchun ijrochi “nimaga” va “nima uchun” degan savollarni bermaydi. Informatikada algoritmlarning unversal ijrochisi kompyuterlar bo‘lib hisoblanadi.

Algoritmlarning asosiy xossalari quyidagilardan iborat:

- Tushunarlilik. Algoritm ijrochisi buyruqlar ketma-ketligini qanday bajarishni aniq bilishi kerak.
- Diskretlilik. Algoritm ijrochisi masalani yechish jarayonini alohida va sodda qadamlar ketma-ketligini bajarish deb tushunishi kerak.
- Aniqlik. Algoritmning har bir qoidasi, undagi amallar va buyruqlar bir ma’noli bo‘lishi kerak. Shu xossaga asosan algoritm ijrochisi buyruqlar ketma-ketligini mexanik bajarish imkoniyatiga ega bo‘ladi.
- Natijaviylik. Bu xossaning mazmuni shundan iboratki, har qanday algoritmning ijrosi oxir-oqibat ma’lum bir yechimga kelishi kerak.
- Ommaviylik. Masalani yechish algoritmi umumiy hollar uchun yaratiladi, ya’ni faqatgina boshlang‘ich qiymatlari bilan farqlanuvchi bir turdagi masalalar sinfi uchun tuziladi. Bunda boshlang‘ich qiymatlar algoritmning qiymatlar qabul qilishi mumkin bo‘lgan sohadan olinadi.

Algoritmlarni tasvirlash usullari. Amaliyotda algoritmlarni tasvirlashning keng tarqalgan usullari quyidagilar:

- so‘zlar yordamida (og‘zaki nutqda ishlatiladigan so‘zlar yordamida, tabiiy tilda);
- grafik usulda (grafik simvollar yordamida);
- dastur ko‘rinishida (dasturlash tillariga oid xizmatchi so‘zlar, operator va funksiyalar yordamida);
- formulalar yordamida (matematik formulalardan foydalangan holda, analitik ko‘rinishda);
- makrotildan foydalangan holda (dasturlovchi va EHMga tushunarli bo‘lgan makrobuyruqlar yordamida);
- jadval ko‘rinishida (mantiqiy algebra elementlaridan foydalangan holda).

Algoritmlarni so‘zlar yordamida tasvirlash. Algoritmlarni so‘zlar yordamida tasvirlashda bajariladigan buyruqlar va ko‘rsatmalar ketma-ket og‘zaki nutqda ishlatiladigan so‘zlar orqali yoziladi.

Masalan, Ikki sonning eng katta umumiy bo‘luvchisini (EKUB) topish algoritmi quyidagicha yozilishi mumkin:

1. Ikkita sonni kiriting;
2. Agarda bu sonlar teng bo‘lsa, u holda ulardan birini javob sifatida oling va ishni to‘xtating, aks holda esa davom ettiring;
3. Ikkita son ichida kattasini aniqlang;

4. Katta va kichik sonlarning ayirmasini katta son bilan almashtiring;

5. Algoritmni 2-qadamdan boshlab qaytaring.

Keltirilgan algoritmni har qanday natural sonlarning EKUBini topish uchun ishlatish mumkin.

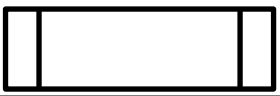
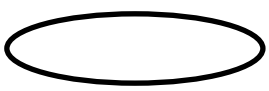
Algoritmlarni so‘zlar yordamida tasvirlashning bir qancha kamchiligi mavjud bo‘lib, aksariyat hollarda algoritmlarni tasvirlashda bu usuldan foydalanilmaydi.

Algoritmlarni grafik usulda tasvirlash. Algoritmlarni grafik usulda tasvirlashda har bir amal bir yoki bir nechta harakatni ifodalovchi o‘zaro bog‘liq funksional bloklar ketma-ketligi orqali tasvirlanadi.

Algoritmning bunday tasvirlash usuli algoritm sxemasi yoki blok-sxema deb ataladi. Blok-sxemada har bir harakat turi (boshlang‘ich qiymatlarni kiritish, ifodalar qiymatlarini hisoblash, shartlarni tekshirish, amallarni takrorlashni boshqarish, qayta ishlashni tugatish va h.k.) ma’lum bir geometrik figura orqali ifodalanadi.

Blokli belgilar (geometrik figuralar) chiziqlar orqali bog‘lanadi (bunda qaysi amal oldin, qaysinisi keyin bajarilishi ko‘rsatiladi).

1.9- jadval blok-sxemada ishlatiladigan bloklarni aks etadi.

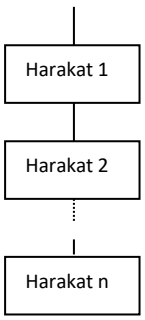
Amallarni belgilanishi	Izoh
	Oddiy harakat
	Shart tekshirish
	Sikl (takrorlanish) boshi
	Yordamchi algoritmga murojaat
	Ma’lumotlarni kiritish va chiqarishning umumiy ko‘rinishi
	Algoritmning boshi va oxiri
	Natijani bosmaga chiqarish

- “Oddiy harakat” belgisi orqali formulalar, hisob-kitob, o‘zlashtirish amallari ifodalaniladi. Bir nechta amallarni alohida yoki bitta belgi orqali ifodalash mumkin.
- “Shart tekshirish” bloki orqali amallar bajarilish yo‘nalishi shart bajarilishi asosida ko‘rsatiladi. Bunday blokning har birida savol, shart yoki munosabat ko‘rsatiladi.
- “Sikl” bloki amallarni takrorlash uchun ishlatiladi. Blok ichida siklning boshi va oxirini ko‘rsatuvchi parametr (i), parametrning o‘zgarish qadami ko‘rsatiladi.
- “Yordamchi algoritmga murojaat” bloki alohida va mustaqil ishlovchi qism dastur va yordamchi algoritmlarga murojaatni bildiradi.

Har qanday algoritmning mantiqiy tuzilishi uchta asosiy algoritm ko’rinishidan biri orqali ifoda qilinishi mumkin:

- *ketma-ketlik(chiziqli);*
- *tarmoqlanish;*
- *takrorlanish (sikl)*

1. Chiziqli algoritm tuzilmasi ketma-ket bajariladigan buyruqlar tizimidan iborat bo‘ladi:

Harakatlar	*Blok-sxema*
harakat 1 harakat 2 . . . . . . harakat n	

2. Tarmoqlanish. Bu tuzilma shart bajarilishi natijasiga qarab (ha yoki yo‘q) algoritmni bajarish yo‘nalishini belgilaydi.

Tarmoqlanish tuzilmasi to‘rtta ko‘rinishda bo‘lishi mumkin:

- agar-u holda;
- agar-u holda-aks holda;
- shartlar ketma-ketligi agar-u holda;
- shartlar ketma-ketligi agar-u holda-aks holda.

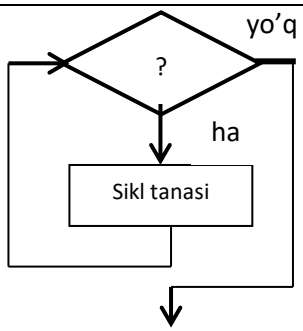
3. Sikl tuzilmasi buyruq, ko‘rsatma va amallarni ko‘p marotaba bajarilishini ta’minlaydi. Takrorlashni ta’minlashning asosiy turlari ushbu jadvalda ko‘rsatilgan:

Og‘zaki so‘zlar orqali	Blok-sxema tilida

Toki sikl turi

Toki so‘zidan keyin keluvchi shart bajarilgunga qadar sikl tanasida ko‘rsatilgan buyruqlar bajariladi.

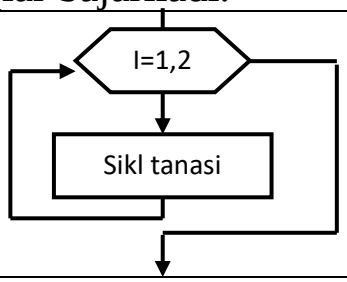
Sikl boshi toki shart
Sikl tanasi (buyruklar ketma-ketligi)
Sikl oxiri



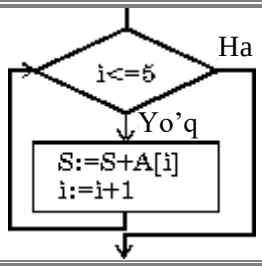
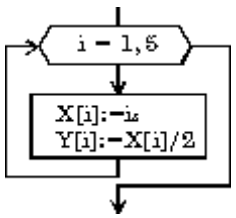
Uchun sikl turi

Sikl o‘zgaruvchisi (sikl parametri) barcha qiymatlarni qabul qilgunga qadar sikl tanasida ko‘rsatilgan buyruqlar bajariladi.

Sikl boshi i uchun 1 dan 2 gacha
Sikl tanasi (buyruqlar ketma-ketligi)
Sikl oxiri



Toki va *uchun* buyruqlariga misollar

So‘zlar orqali	Blok-sxema orqali
Sikl boshi toki i <= 5 S := S+A[i] i := i+1 sikl oxiri	
Sikl boshi i uchun 1 da 5 gacha X[i] := i*i*i Y[i] := X[i]/2 sikl oxiri	

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

Butun sonni butun songa bo‘lganda natija butun songacha yaxlitlanadi.

Misol uchun, 20/3 = 6; (-20)/3 = -6; 20/(-3) = -6.

Unar amallarga ishorani o‘zgartiruvchi unar minus “–” va unar plyus “+” amallari kiradi. Bundan tashqari inkrement “++” va dekrement “--” amallari ham unar amallarga kiradi.

Ternar operatori quyidagi shaklga ega:

amal₁ ? amal₂ : amal₃

exp 1 ifodasi har doim baholanadi. *amal₂* va *amal₃* bajarilishi *amal₁* natijasiga bog‘liq. Agar *amal₁* natijasi nolga teng bo‘lmasa, *amal₂* baholanadi, aks holda *amal₃* baholanadi.

Kamchiliklari:

amal₁ har qanday kamchiligi *amal₂* yoki *amal₃* dan oldin darhol baholanadi va yangilanadi. Boshqacha qilib aytganda, holatni uchlamchi nuqtai nazardan baholaganingizdan keyin ketma-ketlikda nuqta bor. Agar *amal₂* yoki *amal₃* kamchiligi bo‘lsa, ulardan faqat bittasi baholanadi.

Qaytish tipi:

Ternar operatori qaytish tipiga ega. Qaytish tipi odatiy/ortiqcha yuklangan konversiya qoidalariga muvofiq *amal₂* ga va *amal₃* ning *amal₂* ga konvertatsiyasiga bog‘liq. Agar ular o‘zgartirilmasa, kompilyator xato chiqaradi. Quyidagi misollarga qarang.

Quyidagi dastur xatosiz kompilyatsiya qiladi. Ternar operatorning qaytish tipi *float*(*amal₂* dagiday) bo‘lishi kutilmoqda va *amal₃* (ya’ni, haqiqiy nol - tipi *int*) noaniq ravishda *float* tipiga o‘tkaziladi.

```
#include <iostream>
using namespace std;
 int main() {
   int test = 0;
   float fvalue = 3.111f;
   cout << (test ? fvalue : 0) << endl;
   return 0; }
```

Quyidagi dastur kompilyatsiya qilmaydi, chunki kompilyator ternar operatorning qaytariladigan tipini topa olmaydi, yoki *exp 2* (belgilar qatori) va *exp 3* (int) o‘rtasida yashirin konversiya mavjud emas.

```
#include <iostream>
using namespace std;
  int main()
{    int test = 0;
   cout << test ? "A String" : 0 << endl;
     return 0; }
```

Quyidagi dastur kompilyatsiya qilishi mumkinmi? Yoki ish vaqtida ishlamay qoladimi?.

```
#include <iostream>
using namespace std;
  int main()
{    int test = 0;
   cout << (test ? "A String" : 0) << endl;
      return 0; }
```

test ? "A String" : 0 ifodasining qaytish tipi (*char* *) tipi bilan cheklangan, ammo *int* ni qaytaradi, shuning uchun dastur xato bilan tugaydi. Haqiqatdan ham, dastur ish vaqtida 0-chi manzilga bir qatorni bosib chiqarishga harakat qiladi.

Sizeof operatori. Har xil tipdagi o‘zgaruvchilar kompyuter xotirasida har xil sondagi baytlarni egallaydi. Bunda, hattoki bir tipdagi o‘zgaruvchilar ham qaysi kompyuterda va qaysi operatsion tizimda bajarilishiga qarab har xil o‘lchamdagi xotirani band qilishi mumkin.

C++ tilida ixtiyoriy tipdagi(asosiy va hosilaviy tipdagi) o‘zgaruvchilarning o‘lchamini sizeof operatori yordamida aniqlanadi. Bu operator konstantaga, tipga va o‘zgaruvchiga qo‘llanishi mumkin.

Quyidagi dastur kompyuterning aniq platformasi uchun asosiy tiplarning o‘lchamlarini chop qiladi.

```
          cout<<”int    tipining    o‘lchami:”    <<
sizeof(int)<<”\n”;
          cout<<”float   tipining    o‘lchami:”    <<
sizeof(float)<<”\n”;
          cout<<”double   tipining   o‘lchami:”    <<
sizeof(double)<<”\n”;
          cout<<”char    tipining    o‘lchami:”    <<
sizeof(char)<<”\n”;
```

Dastur bajarilishi natijasida *sizeof* operatori yordamida mos tiplarning o‘lchamlari hisoblanadi va ekranga chop etiladi.

***sizeof* amali operand** sifatida ko‘rsatilgan obyektning baytlarda xotiradagi hajmini hisoblash uchun ishlatiladi.

Bu amalning ikki ko‘rinishi mavjud:

- sizeof ifoda;
- sizeof (tip)

Shuni ta’kidlab o‘tish lozimki sizeof funksiyasi preprotsessor qayta ishlash jarayonida bajariladi, shuning uchun dastur bajarilish jarayonida vaqt talab etmaydi.

Misol uchun:

sizeof 3.14 = 8 *sizeof 3.14f = 4*

sizeof 3.14L = 10 *sizeof(char) = 1*

sizeof(double) = 8.

```
#include <stdio.h>
#include <iostream>
using namespace std;
int main() {
 printf("%lu\n", sizeof(char));
    printf("%lu\n", sizeof(int));
    printf("%lu\n", sizeof(float));
    printf("%lu", sizeof(double));
getchar();
    return 0; }
```

Natija quyidagicha bo‘ladi

```
1
4
4
8
```

Ifoda qavslarsiz yoki ko‘rsatilmasdan belgilanishi mumkin.

```
// First type
sizeof expression
  // Second type
sizeof(expression)
```

Ifoda faqat baholashni emas, balki operanda tipini olish uchun ishlatiladi. Masalan, quyidagi kod *i* ning qiymatini 5 sifatida va *i* ning hajmini ko‘rsatadi

```
#include <stdio.h>
  int main()
{
    int i = 5;
    int int_size = sizeof(i++);
      // Displaying the size of the operand
    printf("\n size of i = %d", int_size);
      // Displaying the value of the operand
    printf("\n Value of i = %d", i);
      getchar();
    return 0; }
```

Natija:

hajmi i = 4

qiymati i = 5

Kompilyator bu – dastur tuzish uchun, ya’ni kodlarning qonun qoida bo‘yicha terilganligini nazorat qiluvchi va dasturning natijasini chiqaruvchi amaliy dasturdir.

Kompilyator turlari:

1. Dev;
2. CodeBlocks;
3. Visual Studio;
4. Borland C++Builder;
5. EmbarCadero.

C++ tilida katta va kichik harflarning farqi bor. Bundan tashqari kalit so‘zlar ham bor. Kompilyatorlarni turlari va versiyalariga qarab har xil xatoliklar kelib chiqishi mumkin:

1- kalit so‘zlarni noto‘g‘ri ishlatish;

2- o‘zgaruvchilarni yaratish va foydalanishda;

3- ingliz tilini bilish darajasiga ham bog‘liq;

4- operatorlarni no‘to‘gri ishlatish;

5- kutubxonalardan foydalanishda.

Preprotsessor direktivalari. Preprotsessor direktivalari kompilyatsiya jarayonidan oldin preprotsessor tomonidan bajariladi. Natijada dastur matni preprotsessor direktivalari asosida o‘zgartiriladi.

#include<fayl_nomi> bu direktiva standart kutubxonalardagi funksiyalarni dasturga joylash uchun foydalaniladi.

#define <almashtiruvchi ifoda> <almashinuvchi ifoda>

Bu direktiva bajarilganda dastur matnidagi almashtiruvchi ifodalar almashinuvchi ifodalarga almashtiriladi.

Misol:

```
#include <stdio.h>
#define begin {
#define end }
#define pr printf("\n Dasturlash \n");
int main(){
begin
pr;
end;}
```

Almashtiruvchi **define** direktivasidan nomlangan konstantalar kiritish uchun foydalanish mumkindir.

Misol uchun:

#define ZERO 0

Agar dasturda quyidagi matn mavjud bo‘lsin:

int d = ZERO;

Preprotsessor bu matnda har bir **ZERO** konstantani uning qiymati bilan almashtiradi va natijada quyidagi ifoda hosil bo‘ladi.

int d = 0;

Preprotsessorlarni boshqarish

•oldindan tayyorlangan simvollar ketma-ketligi bilan identifikatorlarni almashtirish;

•ko‘rsatilgan fayldagi matnni dasturga ulash(bog‘lash);

•dasturdan ba’zi qismlarni olib tashlash (shartli kompilyasiya).

Preprotsessor direktivalaridan tashqari preprotsessor amallari ham mavjud, ular buyruqlar bilan birgalikda batafsil o‘rganiladi.

defined - operandning rostligini tekshirish;

- preprotsessor leksemalarini birlashtiradi;

- operandni satr simvoliga akslantiradi (aylantiradi).

#include - dastur matniga ko‘rsatilgan fayldagi matnni ulash (bog‘lash, qo‘shish) imkonini beradi.

#undef - **#define** direktivasi aniqlagan amallarni bekor qiladi.

1. **#define** - makrosning aniqlanishi yoki preprotsessorning identifikatori ;
2. **#include** - fayldan matnni o‘qish;
3. **#undef** - identifikatorni va makrosni aniqlanishini bekor qilish;
4. **#if** -shart ifodani tekshirish;
5. **#ifdef** - identifikator aniqlanishini tekshirish;
6. **#ifndef** - identifikator noaniqligini aniqlash;
7. **#else** - #if uchun alternativ tarmoqning boshlanishi;
8. **#endif** - shart direktivasi #if ning oxiri;
9. **#elif** - tarkibiy direktiva #else/#if;
10. **#line** - keyingi satr raqamini almashtirish;
11. **#error** - translatsiya xatosi haqida xabarni shakllantirish;
12. **#pragma** – oldindan aniqlangan amallar;
13. # -bo‘sh direktivalar.

Nazariy bilimlarni tekshirish uchun savollar.

1. Algoritm nima? C++ dasturlash tilida tiplar turi va ularning xotiradagi hajmi (sizeof).
2. Format modifikatorlari deganda nimani tushunasiz? C++ dasturida nechta funksiya bo‘ladi?
3. C/C++ tilida o‘zgaruvchilarning tiplari va kompanovka bosqichlarini ayting.
4. Haqiqiy tipdagi o‘zgaruvchi float xotiradan qancha bayt egallaydi? Standart funksiyalarning qo‘llanishi.
5. O‘zgaruvchilar deb nimga aytiladi? O‘zgaruvchilarning oddiy tiplari.
6. Dasturda qaysi funksiya bo‘lishi shart? Standart matematik funksiyalar nimalar?
7. Butun tipdagi o‘zgaruvchi kompyuter xotirasidan qancha baytni egallaydi? Formatli kiritish operatorini ayting.
8. Uzunligi 64 bitdan kichik bo‘lmagan ma’lumotning haqiqiy tipi qaysi so‘z orqali ifodalanadi?
9. Ham asosiy dasturda ham boshqa funksiyalarda ishlashi mumkin bo‘lgan o‘zgaruvchilar qaysilar?
10. Xotira hajmini o‘lchaydigan operatorning nomini ayting.
11. C++ dasturlash tilida tiplar turlari va ularning xotiradagi hajmi.
12. Formatli chiqarish operatorini ayting. Dasturlashda xatoliklar bilan qanday ishlanadi?

2-BOB. CHIZIQLI, TARMOQLANUVCHI VA TAKRORLANUVCHI JARAYONLARNI TASHKIL QILUVCHI OPERATORLAR.

2.1. Tarmoqlanish va uzilishlarni tashkil etish operatorlari

📖 *Tarmoqlanuvchi hisoblash jarayonlarini algoritmlash va dasturlash. Ko‘pgina masalalarni yechishda ba’zi bir jarayonlar ma’lum shart yoki shartlarning qo‘yilishiga nisbatan bajariladi. Bunday jarayonlar tarmoqlanuvchi jarayonlar deb yuritiladi va bu jarayonlarning algoritmik tavsiflari bilan avvalgi boblarda tanishgan edik. Tarmoqlanuvchi hisoblash jarayonlari oddiy va murakkab bo‘lishi mumkin. Bu esa jarayondagi tarmoqlar soniga bog‘liq. Ma’lum bir tarmoqlanuvchi jarayon tarkibida yana tarmoqlanishlar bo‘lishi mumkin. Bunday tarmoqlanishlari bor bo‘lgan hisoblash jarayonlari murakkab tarmoqlanuvchi hisoblash jarayonlari deb ataladi. C++ tilida tarmoqlanuvchi jarayonlarni dasturlash uchun shartsiz, shartli o‘tish va tanlash operatorlaridan foydalaniladi: if, case.*

🖋 ***Kalit so‘zlar.*** *Tarmoqlanuvchi algoritm, if else, switch case, goto, nishon, ternar operatori, break, continue.*

REJA:

1. Shartli operator
2. To‘liq va qisqa tarmoqlanish (if)
3. tanlash operatori (swich-case va default) konstruksiyalari.
4. Ternar operatori. Shartsiz o‘tish operatori.
5. Uzilishni tashkil etish – break, continue.

Agar algoritm bajarilish ketma-ketligi bir nechta shartlarga bog‘liq bo‘lsa u ***tarmoqlanuvchi*** deb ataladi. Tarmoqlanuvchi operatorlar. Oldingi mavzularda misol tariqasida keltirilgan dasturlarda operatorlar yozilish tartibida ketma-ket va faqat bir marta bajarilgan holatlar, ya’ni chiziqli algoritmlar keltirilgan. Amalda esa kamdan–kam masalalar shu tariqa yechilishi mumkin. Aksariyat masalalar esa yuzaga keladigan turli holatlarga bog‘liq ravishda mos qaror qabul qilishni (yechimni) talab etadi. C++ tili dasturning alohida bo‘laklarini bajarilish tartibini boshqarishga imkon beruvchi operatorlarning yetarlicha katta majmuasiga ega. Masalan, dastur bajarilishining birorta qadamida qandaydir shartni tekshirish natijasiga ko‘ra dasturning u yoki bu bo‘lagiga boshqaruvni uzatishi mumkin (tarmoqlanuvchi algoritm). Tarmoqlanishni amalga oshirish uchun tarmoqlanuvchi operatorlardan foydalaniladi.

Shartli operatorlar. To‘liqsiz tarmoqlanish if operatori.

if operatori qandaydir shartni rostlikka tekshirish natijasiga ko‘ra dasturda tarmoqlanishni amalga oshiradi:

if (<tekshiriladigan shart>)<operator>1;

Bu yerda <tekshiriladigan shart> har qanday ifoda bo‘lishi mumkin, odatda u taqqoslash operatori bo‘ladi.

Agar tekshiriladigan_shart rost (true) bo‘lsa, <operator>1 bajariladi, aks holda(false) dastur keyingi operatorlarni bajarishga o‘tadi.

C++ tilida bir nechta amallarni blok(guruh)larga birlashtirish mumkin. Blok ‘{‘ va ‘}‘ belgi oralig‘iga olingan amallar ketma-ketligi bo‘lib, u kompilyator tomonidan yaxlit bir operator deb qabul qilinadi.

Quyida keltirilgan dasturda if operatoridan foydalanish ko‘rsatilgan.

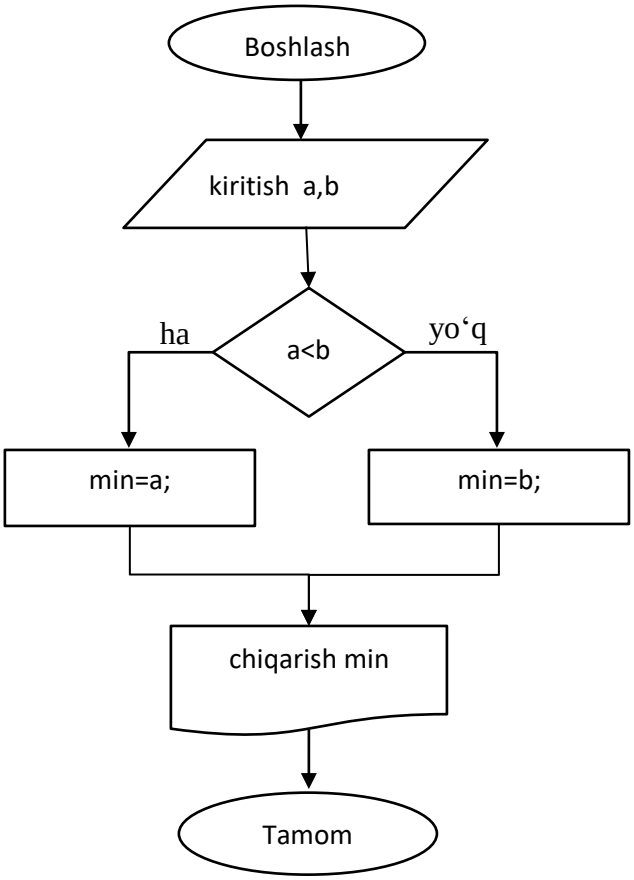
```
#include <iostream.h>
int main()  {
  int b;
  cin>>b;
  if (b>0)
    {           // b>0 shart bajarilgan holat
     cout << “b – musbat son”<<endl;
     cout<<“Uning ildizi ”<< sqrt(b)<< “ ga teng
”<<endl;
      }
          if (b<0)
                cout <<“b – manfiy son”; // b < 0
shart bajarilgan holat
  return 0;}
```

Dastur bajarilishi jarayonida butun tipdagi b o‘zgaruvchi e’lon qilinadi va klaviaturadan qiymati kiritiladi. Keyin *b* ning qiymatini 0 sonidan kattaligi tekshiriladi, agar shart bajarilsa (true) ‘{‘ va ‘}‘ belgilar ichidagi operatorlar bajariladi va ekranga “b – musbat son” va uning ildizi chiqariladi. Agar shart bajarilmasa, bu operatorlar cheklab o‘tiladi. Navbatdagi shart operatori *b* o‘zgaruvchi qiymatini manfiylikka tekshiradi, agar shart bajarilsa yagona cout ko‘rsatmasi bajariladi va ekranga “b – manfiy son” xabari chiqadi.

To‘liq tarmoqlanish. if – else operatori:

Misol. Ikkita butun sonni kiriting va ulardan kichigini ekranga chiqaring.

Blok-sxemasi



Shart operatorining if – else ko‘rinishi quyidagicha:

if (<shart-ifoda>) <operator>1; else <operator>2;

Bu yerda <shart-ifoda> rost (true) bo‘lsa, <operator>1 bajariladi, aks holda <operator>2 bajariladi. if – else shart operatori mazmuniga ko‘ra algoritmning tarmoqlanuvchi blokini ifodalaydi: <shart-ifoda> – shart bloki (romb) va <operator>1 blokning “ha” shoxiga, <operator>2 esa blokning “yo‘q” shoxiga mos keluvchi amallar bloklari deb qarash mumkin.

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

```
#include <iostream.h>
#include <math.h>
  int main()
  {      int a,b,c;
     float D,x1,x2;
     cout <<”ax^2+bx+c=0 tenglama ildizini topish
dastursi! ”;
     cout<<”\n a - koeffistientni kiriting: ”;
     cin>>a;
```

```
      cout<<”\n b - koeffistientni kiriting: ”;
      cin>>b;
      cout<<”\n c - koeffistientni kiriting: ”;
      cin>>a;
      D = b*b – 4 * a * c;
      if (D<0){
                  cout << “Tenglama haqiqiy
ildizlarga ega emas”;
         return 0;}
       if (D==0){
                   cout << “Tenglama yagona ildizga
ega:  ”;
          x1=x2= -b / (2 * a);
          cout<<”\n    x= “<<x1;
                   return 0;
        } else {
                   cout << “Tenglama ikkita ildizga
ega:  ”;
          x1 = (- b + sqrt(D)) / (2 * a);
          x2 = (- b - sqrt(D)) / (2 * a);
                   cout<<”\n x1=  “<<x1;
                   cout<<”\n x2=  “<<x2; }
            return 0; }
```

Dastur bajarilishi jarayonida birinchi navbatda tenglama koeffisentlari – *a, b, c* o‘zgaruvchilar qiymatlari kiritiladi, keyin diskriminant – *D* o‘zgaruvchi qiymati topiladi. Keyin *D* ning qiymati manfiy ekanligi tekshiriladi. Agar shart o‘rinli bo‘lsa, yaxlit operator sifatida keluvchi ‘{‘ va ‘}‘ belgilari orasida operatorlar bajariladi va ekranga “Tenglama haqiqiy ildizlarga ega emas” xabari chiqadi va dastur o‘z ishini tugatadi (return 0 operatorini bajarish orqali). Diskriminant 0 dan kichik bo‘lmasa, navbatdagi shart operatori uni 0 ga tengligini tekshiradi. Agar *D* ning qiymati nolga teng bo‘lsa keyingi qatorlardagi operatorlar bloki bajariladi va ekranga “Tenglama yagona ildizga ega:” xabari hamda *x1* o‘zgaruvchi qiymati chop etiladi va dastur shu yerda o‘z ishini tugatadi, aks holda, ya’ni *D* ning qiymati noldan katta bo‘lsa, else kalit so‘zidan keyingi operatorlar bloki bajariladi va ekranga “Tenglama ikkita ildizga ega: ” xabari, hamda *x1* va *x2* o‘zgaruvchilar qiymatlari chop etiladi. Shu bilan shart operatoridan

chiqiladi va asosiy funkstiyaning return koʻrsatmasini bajarish orqali dastur oʻz ishini tugatadi.

Ternar operatori. ?: shart operatori. Agar tekshirilayotgan shart nisbatan sodda boʻlsa, shart operatorining ?: koʻrinishini ishlatish mumkin. Bu operator quyidagi koʻrinishga ega:

```
<shart ifoda> ? <ifoda>1 : <ifoda>2;
```

if shart operatoriga oʻxshash holda bu shart operatori quyidagicha bajariladi: agar <shart ifoda> rost (true) boʻlsa <ifoda>1 bajariladi, aks holda <ifoda>2. Odatda ifodalar qiymatlari birorta oʻzgaruvchiga oʻzlashtiriladi.

Misol tariqasida ikkita butun son maksimumini topish masalasini koʻraylik.

```
#include <iostream.h>
        int main()
  {      int a,b,c;
     cout <<”a va b sonlar maksimumini topish
dasturi! ”;
     cout<<”\n a – qiymatini kiriting: ”;
             cin>>a;
     cout<<”\n b – qiymatini kiriting: ”;
     cin>>b;
     c = (a>b)? a : b;
     cout <<”\n sonlar maksimumi: “<<c;
     return 0;  }
```

Dasturdagi shart operatori qiymat berish operatorining tarkibiga kirgan boʻlib, *a* oʻzgaruvchi qiymati *b* oʻzgaruvchi qiymatidan kattaligi tekshiriladi, agar shart rost boʻlsa *c* oʻzgaruvchisi *a* oʻzgaruvchi qiymatini, aks holda *b* oʻzgaruvchi qiymatini oʻzlashtiradi va *c* oʻzgaruvchining qiymati chop etiladi.

?: operatorining qiymat qaytarish xossasidan foydalangan holda, uni bevosita cout koʻrsatmasiga qoʻyish orqali ham qoʻyilgan masalani yechish mumkin:

```
#include <iostream.h>
        int main()
  {      int a,b;
     cout <<”a va b sonlar maksimumini topish
dasturi! ”;
     cout<<”\n a – qiymatini kiriting: ”;
             cin>>a;
```

```
        cout<<"\n b – qiymatini kiriting: ";
        cin>>b;
        cout <<"\n sonlar maksimumi: "<<(a>b)? a : b;
        return 0;  }
```

Shartsiz o‘tish operatori. goto operatori.

Shartsiz o‘tish operatorining umumiy ko‘rinishi quyidagicha:

goto <nishon>;

goto operatoridan keyin boshqarilish <nishon> ga uzatiladi va dasturning bajarilishi shu yerdan davom etadi.

Nishon - bu oxiriga ikki nuqta ‘:‘ qo‘yilgan identifikator.

Misol uchun: nishon:

Nishon har qanday operator oldidan ishlatilishi mumkin, shuningdek shart operatori oldidan ham.

Misol: **N natural sonini kiritishni taklif qiluvchi dastur tuzilsin. Agar natural bo‘lmagan son kiritilsa, qayta kiritish taklif qilinsin.**

```
#include <iostream>
#include <math.h>
using namespace std;
int main(){
float n;
nishon:
cout << “natural son kiriting” << endl;
cin >> n;
if(( ceil(n) !=n) or  (n <= 0))
goto nishon;
cout << “Natural son kiritildi” << endl;
return 0;}
```

Dastur bajarilishi jarayonida birinchi navbatda ***n*** soni kiritiladi, keyin kiritilgan sonni natural son emasligi tekshiriladi. Agar shart rost(true) qiymat qaytarsa nishonga qaytadi va ***n*** soni qayta kiritilishi so‘raladi. Aks holda ya’ni ***n*** soni natural son bo‘lsa, “Natural son kiritildi” xabari chiqariladi.

Tanlash operatori. switch operatori. Shart operatorining yana bir ko‘rinishi switch tarmoqlanish operatori bo‘lib, uning sintaksisi quyidagicha:

```
switch (<ifoda>)
 {   case <konstanta ifoda> :
     <operatorlar guruhi>;
     break;
```

```
    case <konstanta ifoda> :
      <operatorlar guruhi>;
      break;
…
    default :
      <operatorlar guruhi>;    }
```

Bu operator quyidagicha ishlaydi: birinchi navbatda <ifoda> qiymati hisoblanadi, keyin bu qiymat case kalit so‘zi bilan ajratilgan <konstanta ifoda> bilan solishtiriladi. Agar ular ustma-ust tushsa, ‘:‘ belgisidan keyingi break kalit so‘zigacha bo‘lgan <operatorlar guruhi> bajariladi va boshqaruv tarmoqlanuvchi operatordan keyingi operatorga o‘tadi. Agar <ifoda> birorta ham <konstanta ifoda> ifoda bilan mos kelmasa, operatorning default kalit so‘zidan keyingi operatorlar guruhi bajariladi.

Misol uchun, kirish oqimidan “Jarayon davom etilsinmi?” so‘roviga foydalanuvchi tomonidan javob olinadi. Agar ijobiy javob olinsa, ekranga “Jarayon davom etadi!” xabari chop etiladi va dastur o‘z ishini tarmoqlanuvchi operatordan keyin davom ettiradi, aks holda “Jarayon tugadi!” javobi beriladi va dastur o‘z ishini tugatadi. Bunda, foydalanuvchining ‘y‘,‘Y‘,‘h‘,‘H‘ javoblari jarayonni davom ettirishni bildiradi, boshqa belgilar esa jarayonni tugatishni anglatadi.

```
    include <iostream.h>
    int main()   {
     char Javob =‘ ‘;
     cout<<”Jarayon davom etsinmi?
(‘y‘,‘Y‘,‘h‘,‘H‘): ”
     cin>> Javob;
     switch (Javob)      {
         case ‘Y‘ :
         case ‘y‘  :
         case ‘h‘  :
         case ‘H‘ :
           cout<<”Jarayon davom etadi!\n”;
           break;
         default :
           cout <<”Jarayon tygadi!\n”;
           return 0;    }
```

```
      …. // Jarayon
      return 0;}
```

Umuman olganda, tarmoqlanuvchi operatorda break va default kalit so‘zlarini ishlatish shart emas. Lekin bu holda operatorning mazmuni buzilishi mumkin. Masalan, default nomi bo‘lmaganda, agar <ifoda> birorta <konstanta ifoda> bilan ustma-ust tushmasa, operator hech qanday amal bajarmasdan boshqaruv navbatdagi operatorga o‘tib ketadi. Agar break bo‘lmasa dastur “to‘xtamasdan” keyingi qatordagi operatorlarni bajarishga o‘tib ketadi. Masalan, yuqoridagi misolda break operatori bo‘lmasa va jarayonni davom ettirish haqida ijobiy javob bo‘lgan taqdirda ekranga quyidagi natija chiqadi va dastur o‘z ishini tugatadi (return 0 operatorini bajarish natijasida).

```
Jarayon davom etadi!
Jarayon tugadi!
```

Tarmoqlanuvchi operator sanab o‘tiluvchi turdagi konstantalar bilan birgalikda ishlatilganda samarali bo‘ladi. Quyidagi dasturda ranglar gammasini tiplash masalasi yechilgan.

```
#include <iostream.h>
int main() {
   enum Ranglar {Qizil, Tuq_sariq, Sariq, Yashil,
Kuk, Zangori, Binafsha};
   Ranglar Rang;
   switch (Rang) {
       case Qizil:
       case  Tuq_sariq :
       case Sariq :
         cout << “Issiq gamma tanlandi.\n”;
         break;
       case Yashil :
       case Kuk :
       case Zangori:
       case Binafsha :
         cout << “Sovuq gamma tanlandi.\n”;
         break;
    default :
      cout <<”Kamalak bunday rangga ega emas.\n”;
  }   return 0;}
```

Dastur bajarilishida boshqaruv tarmoqlanuvchi operatorga kelganda, Rang qiymati Qizil yoki Tuq_sariq yoki Sariq bo‘lsa, ‘Issiq gamma tanlandi‘ xabari, agar Rang qiymati Yashil yoki Kuk yoki Zangori yoki Binafsha bo‘lsa, ekranga ‘Sovuq gamma tanlandi‘ xabari, agar Rang qiymati sanab o‘tilgan qiymatlardan farqli bo‘lsa, ekranga ‘Kamalak bunday rangga ega emas” xabari chop etiladi va dastur o‘z ishini tugatadi.

Nazariy bilimlarni tekshirish uchun savollar.

1. If operatorining nechta turi bor?
2. Tarmoqlanuvchi operatorlar qanday ishlaydi?
3. Tarmoqlanuvchi operatorlarga qandaydir kutubxona kerakmi?
4. Tanlash operatorlari qanday ishlaydi?
5. Tanlash operatorlariga qandaydir kutubxona kerakmi?
6. switch operatori qavsi ichiga nimalar yozish mumkin?
7. Ternar operatori qanday ishlaydi?
8. case deganda nimani tushunasiz?
9. default qaysi vaqtda ishga tushadi?
10. {} figurali bloklar nima uchun kerak?

2.2. Takrorlanish operatorlari

📖 *C++ dasturlash tilida muhim ahamiyatga ega bo‘lgan takrorlash operatorlari, takrorlanish jarayonini qanday tashkil etish usullariga bag‘ishlangan. Takrorlanish jarayonini tashkil etuvchi takrorlash operatorlarining turlari hamda ularning qo‘llanilish usullari. Takrorlash operatolari ishida ba’zan ma’lum bir qiymatlar uchun biror qism bajarilmasligi zarur bo‘lib qolganda yordamga kelgan o‘tish operatorlari haqida ham ma’lumotlar berilgan. Amaliy masalalar yordamida barcha takrorlash operatorlari aniq va to‘liq tushuntirilgan.*

✎ ***Kalit so‘zlar:*** *takrorlash, takrorlash tanasi, iteratsiya, takrorlanish, parametrli, oldshartli takrorlash, so‘ngshartli takrorlash.*

REJA:

1. Parametrli takrorlash operatori (for).
2. Old shartli va so‘ng shartli takrorlanuvchi sikl operatorlari (while, do while).
3. Takomillashtirilgan takrorlash operatorlari.
4. Takrorlanishni tarmoqlanish va shartsiz o‘tish orqali tashkil etish.

Har qanday dasturning strukturasi tarmoqlanish va takrorlanishlar to‘plamining kombinatsiyasidan iborat bo‘ladi. Qator masalalarni yechish uchun ko‘pincha bitta amalni bir necha marotaba bajarish talab qilinadi. Amaliyotda bu rekursiyalar va iterativ algoritmlar yordamida amalga oshiriladi. Iterativ jarayonlar – bu amallar ketma-ketligini zaruriy sonda takrorlanishidir. Takrorlanuvchi algoritmli dasturlarda aniq bir yoki bir necha amallar takror va takror bajarilish imkoniyati ko‘zda tutilgan bo‘ladi. Takrorlanishni amalga oshirilishi uchun dasturlash titlining takrorlash operatorlaridan foydalanish mumkin bo‘ladi. C++ dasturlash tilida takrorlash operatorlarining bir nechta turi mavjud. Takrorlash operatorlari “takrorlash sharti” deb nomlanuvchi ifodaning rost qiymatida dasturning ma’lum bir qismidagi operatorlarni (takrorlash tanasini) ko‘p marta takror ravishda bajaradi.

Takrorlash o‘zining kirish va chiqish nuqtalariga ega, lekin chiqish nuqtasi bo‘lmasligi mumkin, Bunday takrorlashlarga cheksiz takrorlash deyiladi. Cheksiz takrorlash uchun takrorlashni davom ettirish sharti doimo rost bo‘ladi.

Takrorlash shartini tekshirish takrorlash tanasidagi operatorlarni bajarishda oldin tekshirilishi mumkin (for, while operatolari) yoki tanasidagi operatorlar bir marta bajarilgandan keyin tekshirilishi mumkin (do-while operatori).

Takrorlanish jarayonlari. Takrorlanish – bu bir xil ketma-ketlikda bajariladigan ko‘pqirrali harakat.

Ma’lum qadamlar sonidagi takrorlanish va Noma’lum qadamlar sonidagi takrorlanish (shartli takrorlanish)

Masala. 1 dan 8 gacha (a dan b gacha) bo’lgan butun sonlarning kvadratlarini va kublarini ekranga chiqaring.

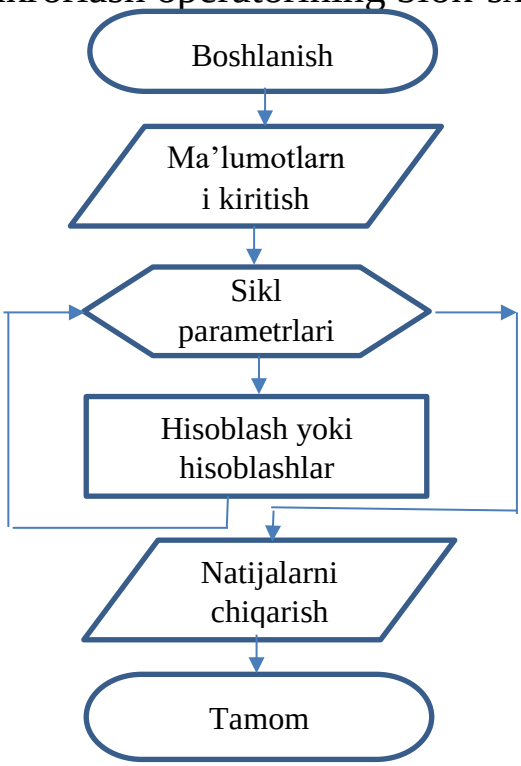
Xossa: bir xil harakatlar 8 marta bajariladi.

Takrorlanuvchi jarayonga misol: Avval berilgan ma’lumotlar kiritiladi. So‘ngra takrorlanuvch jarayonning, ya’ni siklning parametrlari o‘rnatiladi. Buni matematikada takrorlanish oralig‘i deb ham yuritiladi.

Masalan: $X \in [0;10]$ bo‘lsa, sikl parametrlari 0 dan 10 gacha hisoblanadi. Keyin hisoblash yoki bir nech hisoblashlar amalga oshiriladi. Natija 1 ta yoki bir nechta chiqishi mumkin, bu masalaning qo‘yilishiga bog‘liq bo‘ladi. Agar masalaning javobi bir nechta chiqadigan bo‘lsa, u holda chiqarish blogi ham sikl parametri ichida bo‘ladi.

for-parametrli takrorlash operatori.

for takrorlash operatorining blok-sxemasi



for takrorlash operatorining sintaksisi quyidagi ko‘rinishga ega:

for (<ifoda1]>; <ifoda2>;<ifoda3>) <operator yoki blok>;

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

Misol uchun 10 dan 20 gacha bo‘lgan butun sonlar yig‘indisini hisoblash masalasini ko‘raylik.

```
# include < iostream>
using namespace std;
int main () {
          int Summa=0;
          for ( int i= 10 ; i< = 20 ; i++ )
           Summ a+=i;
           cout<<" Yig‘indi= " << Summa;
return 0; }
```

Dasturdagi takrorlash operatori o‘z ishini, *i* takrorlash parametriga (takrorlash hisoblagichiga) boshlang‘ich qiymat - 10 sonini berishdan boshlaydi va har bir takrorlash qadamidan (iteratsiyadan) keyin qavs ichidagi uchinchi operator bajarilishi hisobiga uning qiymati bittaga oshadi. Har bir takrorlash qadamida takrorlash tanasidagi operator bajariladi, ya’ni *Summa* o‘zgaruvchisiga *i* ning qiymati qo‘shiladi. Takrorlash sanagichi *i* ning qiymati 21 bo‘lganda “*i<=20*” takrorlash sharti ***false(0)*** qiymatini qaytaradi va takrorlash tugaydi. Natijada boshqaruv takrorlash operatoridan keyingi ***cout*** operatoriga o‘tadi va ekranga yig‘indi chop etiladi. Yuqorida keltirilgan misolga qarab takrorlash operatorlarining qavs ichidagi ifodalariga izoh berish mumkin: <ifoda1> - takrorlash sanagichi vazifasini bajaruvchi o‘zgaruvchiga boshlang‘ich qiymat berishga xizmat qiladi va u takrorlash jarayoni boshida faqat bir marta hisoblanadi. Ifodada o‘zgaruvchi e’loni uchrashi mumkin va bu o‘zgaruvchi takrorlash operatori tanasida amal qiladi va takrorlash operatoridan tashqarida «ko‘rinmaydi», <ifoda2> - takrorlashni bajarishni yoki bajarilmasligini aniqlab beruvchi mantiqiy ifoda, agar shart rost bo‘lsa, takrorlash davom etadi, aks holda yo‘q. Agar bu ifoda bo‘sh bo‘lsa, shart doimo rost deb hisoblanadi; <ifoda3> - odatda takrorlash sanagichining qiymatini oshirish (kamaytirish) uchun xizmat qiladi yoki unda takrorlash shartiga ta’sir qiluvchi boshqa amallar bo‘lishi mumkin.

for operatorida takrorlash tanasi bo‘lmasligi ham mumkin. Yuqorida keltirilgan 10 dan 20 gacha bo‘lgan sonlar yig‘indisini bo‘sh tanali takrorlash operatori orqali hisoblash mumkin:

for (int i= 10; i< = 20 ; Summa+=i++) ;

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

Misol. Faktorialni hisoblash dasturi

```
#include<iostream>
using namespace std;
int main(){
      int n,i;
      long long fact=1;
      cout<<"n ni kiriting:";
      cin>>n;
      for(i=1; i<=n; i++)
      fact*=i;
     cout<<"natija="<<fact;
     return  0 ;}
```

Ichma-ich joylashgan for takrorlash operatori.

Misol: Takrorlash operatorining ichma-ich joylashuviga misol sifatida 20 gacha bo‘lgan sonlarning tub son yoki murakkab son ekanligi haqidagi ma’lumotni chop qilish masalasini ko‘rishimiz mumkin:

```
#include <iostream>
#include <math.h>
using namespace std;
int main(){
        const int m=20;
        int n[m];
        int i,j,f;
       for(i=0; i<=m; i++)
       n[i]=1;
       for(i=2; i<=m/2; i++)  {
      if (n[i]==1){
          for(j=i+1; j<=m; j++)
          if (n[j]==1)
          if (j%i==0)
              n[j]=0;     }   }
  for(i=2; i<=m; i++) {
      if (n[i]==1)
      cout<<i<<"=Tub son"<<endl;
      else
          cout<<i<<"=Murakkab son"<<endl;
  }return 0; }
```

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

for (;;)

cout <<”Cheksiz takrorlash…” ;

Agar takrorlash jarayonida bir nechta o‘zgaruvchilarning qiymati sinxron ravishda o‘zgarishi kerak bo‘lsa, takrorlash ifodalarida zarur operatorlarni ‘,‘ bilan yozish orqali bunga erishish mumkin:

for(int i=10; j=2; i<=20; i++; j=i+10) {...};

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

Xossa:

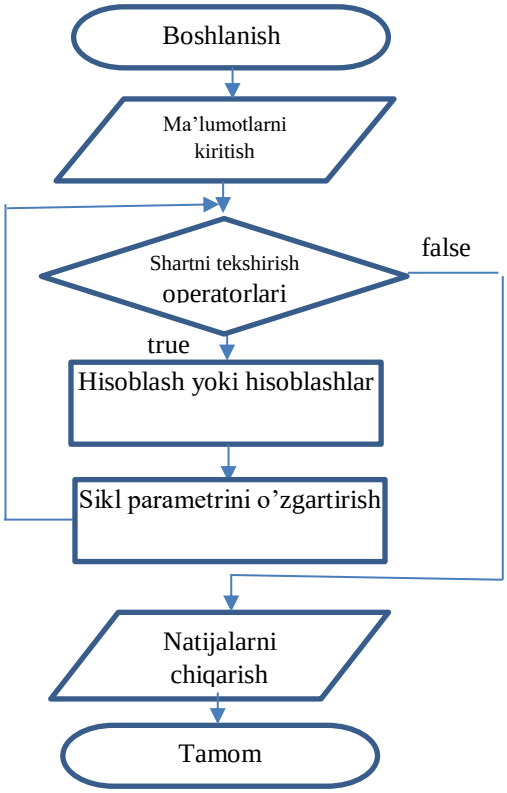
- Shart takrorlashning keyingi qadami boshlanishidan oldin tekshiriladi, agar u yolg‘on bo‘lsa takrorlash bajarilmaydi;
- o‘zgartirish (sarlavhaning uchinchi qismi) takrorlashning navbatdagi qadamining oxirida bajariladi;
- Agar shart yolg‘on bo‘lmasa takrorlash to‘xtovsiz ishlashi mumkin (sikl ichiga tushib qoladi)

While takrorlash operatori. **while** takrorlash operatori, operator yoki blokni takrorlash sharti yolg‘on (false yoki 0) bo‘lguncha takror bajaradi. U quyidagi sintaksisga ega:

while (<ifoda>) <operator yoki blok>;

Agar <**ifoda**> rost qiymatli o‘zgarmas ifoda bo‘lsa, takrorlash cheksiz bo‘ladi. Xuddi shunday, <**ifoda**> takrorlash boshlanishida rost bo‘lib, uning qiymatiga takrorlash tanasidagi hisoblash ta’sir etmasa, ya’ni uning qiymati o‘zgarmasa, takrorlash cheksiz bo‘ladi.

While takrorlash operatorining blok-sxemasi



while takrorlash shartini oldindan tekshiruvchi takrorlash operatori hisoblanadi. Agar takrorlash boshida <**ifoda**> yolg‘on bo‘lsa, **while** operatori tarkibidagi <**operator yoki blok**> qismi bajarilmasdan cheklab o‘tiladi.

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

Agar rost ifodaning qiymati noldan farqli o‘zgarmas bo‘lsa, cheksiz takrorlash ro‘y beradi.

Masalan:

while(1); // **cheksiz takrorlash**

Xuddi ***for*** operatoridek, ‘,‘ yordamida <**ifoda**> da bir nechta amallar sinxron ravishda bajarish mumkin.

Misol. Son va uning kvadratlarini chop qilinadigan dasturda ushbu holat ko‘rsatilgan:

```
#include <iostream>
using namespace std;
int main(){
     int n,n2;
     cout<<"Sonni kiriting (1..10):=";
    cin>>n;
    n++;
    while(n--,n2=n*n,n>0)
    cout<<" n soni = "<<n<<"        sonning
kvadrati="<<n2<<endl;
    return 0; }
```

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

Misol: Ixtiyoriy natural sonlar kiritiladi, qachonki char tipidagi biron belgi kiritilguncha va kiritilgan sonlar yig‘indisi hisoblanadi.

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{    int a, ans;
    char s;
    cin >> a;
    ans = a;
    while(cin >> s >> a)
           {
```

```
                ans += a;          }
      cout << ans;
      return 0;}
```

while takrorlash operatori yordamida samarali dastur kodi yozishga yana bir misol bu - ikkita natural sonning eng katta umumiy bo‘luvchisini (EKUB) Yevklid algoritmi bilan topish masalasini keltirishimiz mumkin:

```
#include <iostream>
#include <stdio.h>
using namespace std;
 int main ()
 {          int a, b ;
         cout<< " A va B natural sonlar EKUBini
topish \ n " ;
         cout<< " A va B natural sonlarni kiriting
: " ;
          cin >> a >> b ;
            while ( a != b ) a > b ? a -= b : b -=
a ;
          cout<< " Bu soniar EKUBi= "<< a ;
return 0 ; }
```

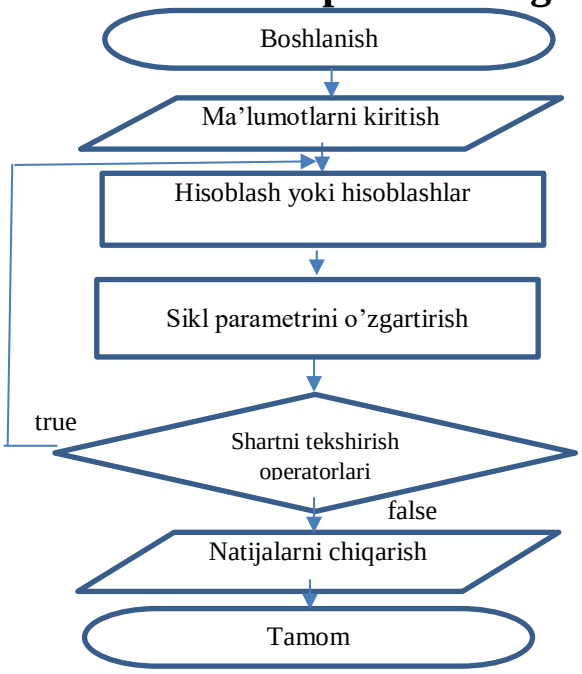
Bu misolda butun tipdagi ***a*** va ***b*** qiymatlari oqimdan o‘qilgandan keyin toki ularning qiymatlari o‘zaro teng bo‘lmaguncha takrorlash jarayoni ro‘y beradi. Takrorlashning har bir qadamida ***a*** va ***b*** sonlarning kattasidan kichigi ayriladi. Takrorlashdan keyingi ko‘rsatma vositasida ***a*** o‘zgaruvchisining qiymati natija sifatida chop etiladi.

do-while takrorlash operatori. do-while takrorlash operatori while operatoridan farqli ravishda oldin operator yoki blokni bajaradi, keyin takrorlash shartini tekshiradi. Bu operator takrorlash tanasini kamida bir marta bajarilishini ta’minlaydi. do-while takrorlash operatori quyidagi sintaksisga ega:

do <operator yoki blok>; while (<ifoda>);

Bunday takrorlash operatorining keng qo‘llaniladigan holatlari - bu takrorlashni boshlamasdan turib, takrorlash shartini tekshirishning iloji bo‘lmagan holatlar hisoblanadi.

do-while takrorlash operatorining blok-sxemasi.



Masalan, birorta jarayonni davom ettirish yoki to‘xtatish haqidagi so‘rovga javob olish va uni tekshirish zarur bo‘lsin. Ko‘rinib turibdiki, jarayonni boshlamasdan oldin bu so‘rovni berishning ma’nosi yo‘q. Hech bo‘lmaganda takrorlash jarayonining bitta qadami amalga oshirilgan bo‘lishi kerak:

```
    #include <iostream.h>
  int main(){
             char javob;
            do  {                    ...// dastur tanasi
                cout<<”Jarayonni to‘xtatish (N):_ ”;
                cin>>javob;
            }
     while(javob !=N)
     return 0;  }
```

Dastur toki ”Jarayonni to‘xtatish (N):_ ” so‘roviga ‘N‘ javobi kiritilmaguncha davom etadi.

Bu operator ham cheksiz takrorlanishi mumkin:

do; while(1);

Misol. Har qanday 7 dan katta butun sondagi pul miqdorini 3 va 5 so‘mliklarda berish mumkinligi isbotlansin. Qo‘yilgan masala **p=3n+5m** tenglamasi qanoatlantiruvchi **m, n** sonlar juftliklarini topish masalasidir (**p-pul miqdori**). Bu shartning bajarilishini m va n o‘zgaruvchilarining mumkin bo‘lgan qiymatlarining barcha kombinatsiyalarida tekshirish zarur bo‘ladi.

```
#include <iostream>
#include <math.h>
using namespace std;
int main() {
  unsigned int Pul;          //Pul- kiritiladigan
pul miqdori
  unsigned n3,m5;          //n-3 so‘mliklar ,    m-5
so‘mliklar soni
  bool xato=false;          //Pul qiymatini
kiritishdagi hatolik
  do  {    if(xato)cout<<"Pul qiymati 7 dan
kichik!";
  xato=true;                    // keyingi
takrorlalanish xato hisoblanadi
  cout<<"\nPul qiymatini kiriting (>7): ";
  cin>>Pul;
          }
  while(Pul<=7);          // Toki  7 dan katta son
kiritilguncha
  n3=0;          //Birorta ham 3 so‘mlik yo‘q
  do {m5=0;                // Birorta ham 5 so‘mlik
yo‘q
  do  {          if (3*n3+5*m5==Pul)
        cout<<n3<<" ta 3 so‘mlik+"<<m5<<" ta 5
so‘mlik\n";
        m5++;                          // 5
so‘mliklar 1 taga oshiriladi
}
while(3*n3+5*m5<=Pul);
  n3++;                //3  so‘mliklar bittaga
oshiriladi
 } while(3*n3<=Pul);
return 0;}
```

O‘tish operatorlari, break operatori. Ba’zi hollarda takrorlash bajarilishini ixtiyoriy joyda to‘xtatishga to‘g‘ri keladi. Bu vazifani ***break*** operatori bajarishga imkon beradi. Bu operator darhol sikl bajarilishini to‘xtatadi va boshqaruvni sikldan keyingi operatorlarga uzatadi. ***Break*** operatorini takrorlash operatori tanasining ixtiyoriy (zarur) joylariga

qo‘yish orqali shu joylarda takrorlashdan chiqishni amalga oshirish mumkin.

Misol: Bu misolda ***n*** o‘zgaruvchiga xoxlagancha qiymat kiritishmiz mumkin, qachonki n ga 1 yoki 0 kiritilganda ***break*** operatori ishga tushadi.

Misolning dasturi:

```
#include <iostream>
using namespace std;
 int main() {       int n;
      while (1){           cin>>n;
           if(n==1 || n == 0)
          break;
         }
 cout<<"Takrorlanish tugadi";
return 0;}
```

Bu misolda while(l) operatori yordamida cheksiz takrorlanish hosil qilinadi. Agar 1 yoki 0 soni kiritilsa, takrorlash to‘xtatiladi.

continue operatori. Takrorlanish bajarilishiga ta’sir o‘tkazishga imkon beradigan yana bir operator ***continue*** operatoridir. Bu operator takrorlanish qadamining bajarilishini to‘xtatib, for va while da ko‘rsatilgan shartli tekshirishga o‘tkazadi.

Misol:

```
#include <iostream>
using namespace std;
 int main(){
       int n;
       for(;;)
          {
             cin>>n;
             if(n==4 || n == 2)
             continue;
             break;
         }
cout<<"Takrorlanish tugadi";
return 0;}
```

Bu misolda for(;;) operatori yordamida cheksiz takrorlanish hosil qilinadi. Agar 4 yoki 2 sonlaridan farqli son kiritilsa, takrorlanish to‘xtatiladi.

goto o‘tish operatori. O‘tish operatorining ko‘rinishi:

goto <identifikator>.

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

Misol uchun goto Al;

Al: y = 5;

Strukturali dasturlashda ***goto*** operatoridan foydalanmaslik maslahat beriladi. Lekin ba’zi hollarda o‘tish operatoridan foydalanish dasturlashni osonlashtiradi.

Misol. Bir necha takrorlashdan birdan chiqish kerak bo‘lib qolganda, to‘g‘ridan to‘g‘ri break operatorini qo‘llab bo‘lmaydi, chunki u faqat eng ichki takrorlashdan chiqishga imkon beradi.

```
#include <iostream>
 using namespace std;
int main(){
        int n = 5, s=0;
        int i, j;
        for(i =1; i<5; i++)
             {
                   cout<<endl;
                  cout<<"i = "<<i<<endl;
                  for(j =1; j<5; j++)
                     {
                           cout<<"j = "<<j<<" ";
                           if(i*j > n) goto A;
                           s ++ ;
                          cout<<"s = "<<s<<" ";
                     }
             }
    A:
   cout<<endl;
    cout<< "Oxirgi natija ="<<s;
    return 0;}
```

Quyidagi misolda ma’lum bir rost (true) holatdan yolg‘on (false) holatiga o‘zgarguncha dastur blokini takrorlashini kuzatamiz.

```
#include <iostream>
 using namespace std;
int main(){
        int qadam = 0;
```

```
          bool takrorlash = true;
         while ( takrorlash != false )
             {
                qadam++;
               cout << "Takrorlanish bajarildi = "
<< qadam << " - marta\n";
               if ( qadam == 5 )
              takrorlash = false;
           }
  return 0; }
```

Biz C++ dasturlash tilida 3 xil ko‘rinishdagi takrorlash operatorlari borligini ko‘rib chiqdik. Bular for, while, do-while. Ular funksional nuqtai nazardan bir xil ish bajaradi, ya’ni ularni funksiyasi ma’lum bir amalni (yoki amallarni) ketma-ket bir necha marta takrorlashdan iborat. Ilmiy tilda siklik takrorlanishni *«iteratsiya»* deb ataydilar**.**

Ammo bu operatorlar mazkur funksiyani (ya’ni takrorlash jarayonini tavsiflashni) turlicha amalga oshiradi. Bu operatorlarning asosiy farqi quydagilardan iborat:

- for operatori faqat iteratsiyalar (ya’ni takrorlashlar) soni ma’lum bo‘lgan holda, while operator esa iteratsiyalar soni noma’lum bo‘lgan holda ham ishlaydi.
- do-while operatori ham xuddi while operatori kabi iteratsiyalar soni noma’lum bo‘lgan holda ham ishlaydi. Ammo u bir iteratsiyadan so‘ng “shart”ni tekshiradi, while operatori esa avval “shart”ni tekshiradi va so‘ngra birinchi iteratsiyani amalga oshiradi.
- Parametrli takrorlash for operatorini parametrning boshlang‘ich va oxirgi qiymati hamda o‘zgarish qadami aniq bo‘lganda qo‘llash juda qulay!

Nazariy bilimlarni tekshirish uchun savollar

1. C++ daturlash tilida takrorlash operatorlari nima uchun kerak?
2. *while* operatorining umumiy ko‘rinishi. *do while* operatori *while* operatoridan qanday farq qiladi?
3. *for* operatorining umumiy ko‘rinishi. Ichki sikllar qanday tashqil etiladi? *for* operatori qanday qo‘llaniladi?
4. For operatorining takrorlanishini tugashi uning qaysi qismiga bo‘liq?
5. While operatori qanday ishlaydi? Qahday holatda boshqaruv while opertoridan keyingi operatorga uzatiladi?

6. Qahday holatda while opertori tanasidagi amallar ketma-ketligi bajariladi va yana shart tekshirishga qaytiladi?
7. *for* bilan *while* va *do-while* operatorlarining qanday farqli tomonlari bor?
8. Takrorlanishlar algoritmlarda qanday ko‘rinishda bo‘ladi?
9. *while* operatorida qachon cheksiz takrorlash ro‘y beradi?
10. Takrorlanishlar ichma-ich bo‘lishi mumkinmi?
11. *do-while* operatori qanday ishlatiladi?
12. Takrorlashning takrorlanishlar soni qanday aniqlanadi?
13. Nima uchun goto operatori ko‘p ishlatilmaydi?
14. *for* operatori ichida while operatori yordamida takrorlashni tashkil etish mumkinmi?
15. Hech qachon tugallanmaydigan takrorlanishni tashkil etish mumkinmi?
16. *while* va *do-while* operatorlarining qanday farqi bor?
17. Takrorlash tanasiga nechta operator yozish (joylash) mumkin?

3-BOB. FUNKSIYALAR VA TO‘PLAMLAR BILAN ISHLASH OPERATORLARI.

3.1. Funksiyalar

📖 *C++ da funksiyalarni yaratish va ulardan foydalanish, shuningdek foydalanuvchi kutubxonasini yaratish hamda undan foydalanish haqida ma’lumotlar olish uchun tayyorlangan. Yuqoridagilardan tashqari funksiyalarning turlari, rekursiv funksiya, lokal va global o‘zgaruvchilar batafsil yoritilgan. Funksiyaning turlari bo‘yicha to‘liq ma’lumotga ega bo‘lish uchun amaliy masalalar yordamida aniq va to‘liq tushuntirilgan.*

✎ ***Kalit so‘zlar.*** *funksiya, prototip, foydalanuvchi kutubxonasi, local, global, protsedura, qayta yuklash.*

REJA:

1. Funksiya tavsifi va qiymatlarni qaytarish;
2. Parametrlarni qiymat va adressga ko‘ra jo‘natish.
3. Havolalar va qiymat berish parametrlari.
4. Funksiyalar parametr sifatida.
5. Rekursiv funksiyalar va funksiyalarni qayta yuklash.
6. Foydalanuvchi kutubxonasini tashkil etish.

Umuman olganda C/C++ tilida barcha yozuvlar funksiyadan iborat deb qaraladi. Funksiya bu ma’nosiga ko‘ra bajariluvchi modul bo‘lib hisoblanadi. Funksiya boshqa dasturlash tillarida qism dastur, protsedura, protsedura funksiya deb yuritiladi. C/C++ tilida funksiya standart formaga asosan quyidagicha ifodalanadi :

<funksiya tipi> funksiya_nomi(rasmiy parametrlar ro‘yxati)

{ funksiya tanasi }

Funksiya tipi istalgan tip yoki void (bo‘sh) tip bo‘lishi mumkin.

Funksiya nomi istalgan lotin harfi yoki harflaridan iborat bo‘lib, xizmatchi so‘zlar bilan bir xil bo‘lmasligi lozim.

Rasmiy parametrlar ro‘yxatida ishlatiladigan parametrlarga mos tipli o‘zgaruvchilar tiplari bilan alohida-alohida keltiriladi yoki bu soxa bo‘sh bo‘lishi ham mumkin. Eslatib o‘tish lozimki, funksiya aniqlashtirilayotganda nuqta vergul belgisi qo‘yilmaydi.

Funksiya tanasi o‘zining figurali qavslariga ega bo‘lib, o‘zida shu funksiyani tashkil etuvchi operatorlar yoki operatorlar blokini mujassamlashtiradi. Bir funksiya tanasi ichida boshqa funksiya aniqlanishi mumkin emas.

Funksiya tanasidan chiqish

return;

yoki

return ifoda;

ko‘rinishida bo‘ladi. Agar funksiya hech qanday qiymat qaytarmaydigan, ya’ni tipi void bo‘lsa, birinchi ko‘rinishdagi chiqish ishlatiladi. Agar funksiya uning tipiga mos biror qiymat qaytaradigan bo‘lsa, ikkinchi ko‘rinishdagi chiqish ishlatiladi. C++ tilida quyidagi ko‘rinishlar ekvivalent xisoblanadi, lekin birinchi ko‘rinish ko‘proq ishlatiladi:

```
double f (int n, float
x)
{
  funksiya tanasi;
  }
```

```
double f ( n, x)
int n;  float x;
{
funksiya tanasi;  }
```

Dasturda funksiya ishlatiladigan bo‘lsa, uni albatta e’lon qilish shart. Funksiyani e’lon qilishda uning tipi, nomi va qaytaradigan parametrlari haqida xabar beriladi. Dasturda biror funksiyani oldindan e’lon qilmasdan turib uni chaqirish mumkin emas. Funksiya asosiy funksiya main( ) dan oldin va keyin aniqlanishi mumkin. Agar funksiya asosiy funksiyadan oldin aniqlansa, u aniqlanishi bilan birga e’lon qilingan deb hisoblanadi va uni alohida main( ) ichida e’lon qilish shart bo‘lmay qoladi. Agar funksiya asosiy funksiyadan keyin aniqlanayotgan bo‘lsa, uni main( ) ichida albatta e’lon qilish shart bo‘ladi. Funksiya main() ichida e’lon qilinadigan bo‘lsa, uning nomi bilan birga ishlatiladigan parametrlarining faqatgina tiplari ko‘rsatilishi ham mumkin. Masalan:

```
int myFuncsion ( int, float);
double Area (float, float);
```

Funksiyaga murojaat qilishdan oldin uning rasmiy parametrlari aniqlangan bo‘lishi, ya’ni haqiqiy parametrlar berilgan bo‘lishi lozim. Funksiyaga murojaat qilish quyidagicha amalga oshiriladi:

funksiya_tipi funksiya_nomi (haqiqiy parametrlar ro‘yxati);

Masalan:

```
myFuncsion (78, 3.0+m);
Area (a, b);
g (6.4e-2, 5, 70);
```

Funksiyaning rasmiy va haqiqiy parametrlarining tipi, parametrlar soni va ularning kelish o‘rinlari albatta bir biriga mos kelishi shart!

Funksiyaga murojaat qilinganidan so‘ng aniqlangan funksiya tanasi bajariladi va mos tipli qiymat chaqirilgan joyga qaytib keladi.

Masalan: quyidagi funksiya chaqirilganida float tipli natija qaytaradi:

```
float ft (double x, int n)
{
 if (x < n) return x;
 else return n;  }
```

Funksiyalarga murojaat qilinganida unga uzatiladigan parametrlarga alohida e’tibor berish kerak. Parametrlarning uzatilishi quyidagi bosqichlardan iborat:

- Funksiyani tashkil etadigan rasmiy parametrlar uchun xotiradan joy ajratiladi. Agar parametrlar haqiqiy tipga ega bo‘lsa, ular double tipga, agar char va short int tipli bo‘lsalar, ular int tipi sifatida tashkil etiladilar. Agar parametrlar massiv shaklida bo‘lsalar, massiv boshiga ko‘rsatkich qo‘yiladi va u funksiya tanasi ichida massiv parametr bo‘lib xizmat qiladi.
- Funksiya chaqirilganida kerak bo‘ladigan ifodalar yoki haqiqiy parametrlar aniqlanadi va ular rasmiy parametrlar uchun ajratilgan joyga yoziladi;
- Funksiya chaqiriladi va aniqlangan haqiqiy parametrlar yordamida hisoblanadi. Bu yerda ham agar parametrlar haqiqiy tipga ega bo‘lsa, ular double tipga, agar char va short int tipli bo‘lsalar, ular int tipi sifatida tashkil etiladilar.
- Natija funksiya chaqirilgan joyga qaytariladi.
- Funksiyadan chiqishda rasmiy parametrlar uchun ajratilgan xotira qismi bo‘shatiladi.

Funksiyaga murojaat qilish ifodani tashkil etadi, lekin agar funksiyaning qaytaradigan qiymati bo‘sh (void) bo‘lsa, u ifoda bo‘lmasligi ham mumkin. Unda bunday funksiyalarga murojaat qilish quyidagicha bo‘ladi:

funksiya nomi (haqiqiy parametrlar);

Masalan:

```
void print (int gg, int mm, int dd)
{
 cout<< ”\n yil:”<< gg;
 cout << “ \n oy: ” << mm;
 cout << “ \n kun: ” << dd;}
```

ko‘rinishidagi funksiyaga print (1966, 11, 22); deb murojaat qilinsa, quyidagi natija chiqadi:

```
yil: 1966
oy: 11
kun: 22
```

Ba’zan umuman hech qanday parametrsiz funksiyalar ham ishlatiladi. Masalan:

```
void Real_Time (void)
{
 cout << “ Hozirgi vaqt: “ << TIME “(soat: min:
sek)”;}
```

funksiyasiga Real_Time ( ); deb murojaat qilinsa, ekranga

```
Hozirgi vaqt: 14: 16: 25 (soat: min: sek)
```

degan axborot chiqadi.

Funksiya - bu mantiqan to‘g‘ri tugatilgan dasturiy qismdir. Ular yordamida katta va murakkab hisoblashlarni qayta - qayta yozish mashaqqatidan xolos bo‘linadi va dastur bajarilishi yengillashadi. U bir marta tashkil etib yozib qo‘yiladi va unga dasturning istalgan yeridan murojaat qilish mumkin bo‘ladi. Funksiyani tashkil qilishda funksiyaning tipi, uning nomi va tashkil etuvchi parametrlari haqida axborot keltiriladi. Bu parametrlar rasmiy parametrlar deb yuritiladi.

Rasmiy va haqiqiy parametrlar soni, ularning tipi va kelish o‘rni bilan albatta bir biriga mos bo‘lishi shart! Rasmiy va haqiqiy parametrlar nomlari bir xil bo‘lishi mumkin. Funksiyani bosh funksiya ichida e’lon qilinganida haqiqiy parmetrlar nomlarini ko‘rsatmasdan, faqat ularning tiplarini keltirish ham mumkin [11].

Masalan: sonning kubini hisoblash uchun funksiya tashkil eting va undan foydalaning.

```
# include <iostream.h>
# include <conio.h>
void main ( )
{ int k, n, kw (int n); // kw - funksiya nomi
(ixtiyoriy)
cin>>n;          // n - berilayotgan son
k=kw(n);        // kw funksiyasiga murojaat
qilinmoqda
cout << «k=«<<k<<endl;
getch( );}
int kw (int a) // funksiya aniqlanmoqda. Bu yerda a
```

```
rasmiy parametr
{  int c;    // lokal o‘zgaruvchi
 c=a*a*a;   // hisoblash
 return c; }  // funksiyaga natijani qaytarish
```

Yuqoridagi *s* lokal o‘zgaruvchisini ishlatmasdan, to‘g‘ridan-to‘g‘ri return a*a*a; deb yozsa ham bo‘ladi.

Bu yerda funksiya bosh funksiyadan keyin aniqlandi, shuning uchun uni bosh funksiya ichida e’lon qildik. Dasturni yana quyidagicha yozsa ham bo‘ladi:

```
# include <iostream.h>
# include <conio.h>
int kw (int a)
{ return a*a*a;  }
void main ( )
{ int k, n ;
cin>>n;
k=kw(n);
cout << «k=«<<k<<endl;
getch( );}
```

2-misol. Ikkita sondan eng kattasini topish uchun funksiya tashkil qiling va undan foydalaning.

```
# include <iostream.h>
# include <conio.h>
void main( )
{ float a=7, b=9, c, max(float , float );
c = max(a, b);
cout << «c=«<<c<<endl;
getch( );
}float max ( float x, float y)
{  if  (x > y)  return x;  else  return y; }
Funksiyaga yana quyidagicha ham murojaat qilish
mumkin:
c = max( 7.23, 9.145);
c = max( a, 9.145);
```

3-misol. Uchburchak uchlarining koordinatalari berilgan. Shu koordinatalar yordamida uchburchak qursa bo‘ladimi? Agar mumkin bo‘lsa shu uchburchakning yuzini hisoblash dasturini tuzing.

Demak, berilgan koordinatalar yordamida uchburchak tomonini topish funksiyasini, shu tomonlar asosida uchburchak qurish mumkinmi yoki yo‘qligini va uning yuzini hisoblash funksiyalarini tuzing.

```
# include <iostream.h>
# include <math.h>
# include <conio.h>
// uchburchak tomonini topish funksiyasi
float line(float x1, float x2, float y1, float y2)
{ (float) p=sqrt((x1-x2)*(x1-x2)+ (y1-y2)*(y1-y2));
  return p; }
// uchburchak qurib bo‘ladimi? funksiyasi
int uch ( float a, float b, float c)
{ if ( a+b>c && b+c>a && c+a>b ) return 1;
else return 0; }
// uchburchakning yuzini topish funksiyasi
float s (float a, float b, float c)
{ float p, s ;
p=( a+b+c )/2; s = sqrt (p*(p-a)*(p-b)*(p-c));
return s; }
void main ( )
{ float x1,x2,x3,y1,y2,y3,p1,p2,p3; clrscr();
cin >> x1>> x2>> x3>> y1>> y2>> y3;
p1 = line (x1, x2, y1, y2);
p2 = line (x1, x3, y1, y3);
p3 = line (x2, x3, y2, y3);
t = uch (p1, p2, p3);
if ( t = = 1)
{ yuza=s(p1, p2, p3); cout <<“yuza = ”<<yuza<<endl;
else cout <<”uchburchak qurib bo‘lmaydi !!!”<<
endl;
}
getch( );
}
```

Bir funksiya ichida boshqa funksiya aniqlanishi mumkin emas, lekin funksiya ichida o‘zini-o‘zi chaqirishi mumkin. Bunday holat rekursiya holati deyiladi. Rekursiya 2 xil bo‘ladi: to‘g‘ri rekursiya va bilvosita rekursiya. Agar funksiya o‘zini-o‘zi chaqirsa, bu to‘g‘ri rekursiya deyiladi. To‘g‘ri rekursiyada funksiyaning nusxasi chaqiriladi.

Agarda funksiya boshqa bir funksiyani chaqirsa va u funksiya o‘z navbatida 1-sini chaqirsa, u holda bilvosita rekursiya deyiladi. Rekursiya 2 xil natija bilan yakunlanadi: biror natija qaytaradi yoki hech qachon tugallanmaydi va xatolik yuz beradi. Bunday holatlarda rekursiv funksiyalar uchun rekursiyani to‘xtatish shartini berish zarur, chunki rekursiyada xotira yetishmasligi xavfi bor.

4-misol. F = n! ni xisoblash uchun funksiya tashkil eting va undan foydalaning.

```
# include <iostream.h>
# include <conio.h>
int main( )
{ int n, f, fac(int);
cout << “sonni kiriting:”; cin >> n;
f = fac(n); cout << “sonning factoriali=”<<f<<endl;
getch( );
}
int fac(int i)
{ return i <=1 ? 1 : i * fac( i - 1); }
```

5-misol. Fibonachi sonlarini xosil qilish dasturini tuzing. Fibonachi sonlari quyidagicha topiladi:

$f_0 = 1; f_1 = 1; f_2 = f_1 + f_0$; ……

$f_n = f_{n-1} + f_{n-2}$;

Rekursiv jarayonni to‘xtatish sharti n < 2 deb olinadi. Masalan 9-o‘rindagi Fibonachi sonini topish kerak.

```
# include <iostream.h>
int main ( )
{ int n, f ; int fib ( int );
cout << “Raqamni kiriting =”;
cin >> n;
f = fib (n);
cout << “Fibonachi soni=”<< f << endl;
}
int fib ( int n )
{  if ( n < 2) return 1; else return ( fib (n-2) +
fib (n-1)); }
```

6-misol. Z = $\frac{a^5 + a^{-4}}{2a^n}$ hisoblash dasturi tuzilsin. Bu yerdagi darajani hisoblash funksiya sifatida tashkil etilsin. $y = x^n$ ni funksiya deb tashkil etamiz, bu yerda x, n - rasmiy parametrlar

```
# include <iostream.h>
float dar (float x, int n)
{ float y=1;
for (int i=0; i<=n; i++)
  y = y*x;
  return y; }
int main( )
{
int n=3 ;  float a, z;
cin>>a;
z = ( dar(a, 5) + dar(1/a, 4))/ (2* dar(a, n)) ;
 cout << «z=«<<z<<endl;  }
```

Bir xil nomdagi funksiyalarni har xil tipli o‘zgaruvchilar ro‘yxati bilan murojaat qilib chaqirish mumkin. Parametrlar soni ham har xil bo‘lishi mumkin. Bunday holatda parametrlar ro‘yxati va qiymatlarga qarab kompilyator o‘zi qaysi funksiyani chaqirish kerakligini aniqlaydi [1]. Masalan:

```
1. double multi (float x)
  {return x*x*x; }
2. double multi (float x, float y)
  { return x*y*y; }
3.    double multi (float x, float y, float z)
{ return x*y*z;  }
```

va quyidagi murojaatlarning hammasi to‘g‘ri yozilgan:

multi (0.5);
multi (1.45, 7);
multi (10, 39, 54);

Funksiyalarning bir xil nom bilan atalishi polimorfizm deb ataladi. Poli – ko‘p, morfe – shakl degan ma’noni bildiradi.

Masalan:

```
# include <iostream.h>
int max (int a, int b)
{ if (a>b) return a; else return b;}
float max (float a, float b)
{ if (a>b) return a; else return b;}
int main ( )
{
int a1, b1; float a2, b2;
```

```
cin >> a1>>b1;
cout << “butun max=”<<max(a1, b1)<<endl;
cin >>a2>>b2;
cout << “haqiqiy max=”<< max(a2, b2)<<endl;}
```

1-misol. B va C vektorlarining uzunliklarini hisoblash dasturini tuzing. Vektor uzunligini hisoblash uchun funksiyadan foydalaning.

```
# include < iostream. h>
float vector (int d[ ], int k)
{ float s=0; int i ;
for (i=0; i<k; i++)
s = s + d[i] * d[i];
s = sqrt (s);
return s; }
int main ( )
{
int b[3] = {10,20,30}, c[4] = {14,15,16,17};
float  s1, s2;
s1 = vector (b, 3);
s2 = vector (c, 4);
cout <<”s1=” << s1 <<” s2=” << s2 << endl;}
```

2-Misol. Butun sonli 4x5 matritsa berilgan. Aniq bir sondan kichik bo‘lgan hadlarining yig‘indisini topish dasturini tuzing. Matritsa elementlarini kiritish (tasodifiy sonlar yordamida), chiqarish va yig‘indini hisoblash jarayonlarini funksiya sifatida tashkil eting.

Funksiya ichida 2 o‘lchovli massivlardan foydalanilganda uning 1-parametrini, ya’ni satrlar sonini ko‘rsatmaslik ham mumkin, lekin 2-parametrini, ya’ni ustunlar sonini albatta ko‘rsatish shart.

```
# include <iostream.h>
# include <conio.h>
# include <stdlib.h>
# include <time.h>
void kir(int m[ ][5], int k);
void chiq(int m[ ][5], int k);
int summa(int m[ ][5], int k, int x);
int i, j ;
void main ( )
{ int matr[4][5]; int a, s; int b[ ][3];
cout<< “sonni kiriting=”; cin>>a;
```

```
  kir(matr, 4);  chiq(matr, 4);
  s = summa(matr, 4, a);
  cout<< “s=”<<s<< endl;
  getch( );  }

  void kir(int m[ ][5], int k)
  { srand(time(0));
  for (i=0;i<k;i++)
  for (j=0;j<5;j++)
  m[i][j]=rand( ) - 200; }

  void chiq(int m[ ][5], int k)
  { for (i=0;i<k;i++)
  for (j=0;j<5;j++)
  cout <<m[i][j]<<endl; }
  int summa(int m[ ][5], int k, int x)
  { int s1 = 0;
  for (i=0; i<k; i++)
  for (j=0; j<5; j++)
  if (m[i][j] < x) s1 = s1 + m[i][j];
  return s1; }
```

Funksiyalarga murojaat qilish quyidagi bosqichlardan iborat bo‘ladi:

1. Funksiya bajarilayotganda rasmiy parametrlar uchun xotiradan joy ajratiladi, ya’ni ular funksiyaning ichki parametrlariga aylantiriladi. Bunda parametr tipi float tipi double tipiga, char va shortint tiplari int tipiga aylantiriladi.
2. Haqiqiy parametrlar qiymatlari qabul qilinadi yoki hisoblanadi.
3. Haqiqiy parametrlar rasmiy parametrlar uchun ajratilgan xotira qismiga yoziladi.
4. Funksiya tanasi ichki parametrlar yordamida bajariladi va qiymat qaytarish joyiga yuboriladi.
5. Funksiyadan chiqishda rasmiy parametrlar uchun ajratilgan xotira qismi bo‘shatiladi.

Dasturdagi har bir o‘zgaruvchi – obyekt hisoblanadi. Uning nomi va qiymati bo‘ladi. Har bir obyekt xotiradan ma’lum joy egallaydi va ular ma’lum adresga ega bo‘ladi. Dasturlashning ma’lum etaplarida o‘zgaruvchining o‘ziga emas, balki uning adresiga murojaat qilishga to‘g‘ri keladi. Bunday paytlarda ko‘rsatkichlardan foydalaniladi.

Ko‘rsatkich - bu biror o‘zgaruvchining adresini o‘zida saqlovchi o‘zgaruvchidir. Adres - bu xotira yacheykasining tartib raqami. Umuman olganda adres 4 bayt joy oladi. Ko‘rsatkichlarni e’lon qilishda uning tipidan keyin * belgisi va o‘zgaruvchi nomi keltiriladi.

Masalan: int a; char *d; int *p;

Ko‘rsatkichlar ham inisalizasiya qilinishi mumkin. *r = 6; *d = ‘$‘;

cout <<*p bilan cout <<p ning farqi bor. *p da shu yerdagi qiymat chiqadi, p ning o‘zini yozsak, shu yerning adres raqami chiqadi. Masalan:

```
int a=10, b=5, e, *m;
e = a + b;
*m = e;
cout <<*m;   // deb yozilsa, m = 15 chiqadi;
cout <<m;    // deb yozilsa, m = 0xfff2
```

chiqadi, ya’ni shu yerning adres raqami chiqadi.

Ularning qiymatlarini “adresini ol!” (&) operatsiyasi orqali amalga oshirsa ham bo‘ladi, ya’ni *m=&e*; *cout <<*m;* deb yozish ham mumkin, u holda *m=15* chiqadi, ya’ni *e* ning adresidagi son qiymat chiqadi. Buni bilvosita murojaat operatori ham deyiladi.

Masalan:

```
int h;
int *p=35;
h = &p;
```

Natija: h = 35;

Adresi olish amali (&) son yoki ifodalarga qo‘llanilmaydi, ya’ni &3.14 va *&(a+b)* yozuvlari xatodir.

Ko‘rsatkichlar ustida quyidagi amallarni bajarish mumkin:

- Ko‘rsatkichlar ustida arifmetik amallar bajarish: **p1-*p2; *p1+*p2*
- Ko‘rsatkichlarga biror sonni qo‘shish yoki ayirish: **p1 - 25; *p1+3.45*
- Ko‘rsatkichlarni bittaga oshirish yoki kamaytirish: **p1++ yoki --*p1*

Misol.

```
# include <iostream.h>
# include <conio.h>
int main ( )
{
int x=10, y=10; int *xp, *yp;
```

```
*xp = &x; *yp = &y;
if (xp = = yp) cout << “ular teng!”<<endl;
else cout << “ular teng emas!”<<endl;
if (*xp = = *yp) cout << “ular teng!”<<endl;
else cout << “ular teng emas!”<<endl;
getch( ); }
```

1- if da ular teng emas chiqadi, chunki ularning adres qiymatlari har xil.

2- if da ular teng chiqadi, chunki ularning adreslaridagi son qiymatlari bir xil

2-misol.

```
# include <iostream.h>
int main ( )
{ int m = 5, *p = 0;
p = &m;
cout << m << endl;
cout << *p<<endl;
*p = 7;
cout << m << endl;
cout << *p << endl;
m = 9;
cout << m << endl;
cout << *p << endl;
}
```

Natija:

```
m = 5
*p = 5
m = 7
*p = 7
m = 9
*p = 9
```

Ba’zi masalalarda funksiya bilan ishlaganda funksiya tanasi ichida haqiqiy parametrlar qiymatlarini o‘zgartirish zaruriyati tug‘iladi, ya’ni natija bir emas, balki bir nechta hosil bo‘lishi kerak bo‘ladi. Bunday jarayonni protseduralar hosil qilish deyiladi va bu muammoni hal qilish uchun ko‘rsatkichlardan foydalaniladi. Funksiyani aniqlashtirishda rasmiy parametrlar bilan bir satrda natijalar nomlari ham ko‘rsatiladi. Shuning uchun protseduralar bilan ishlaganda funksiya tipini bo‘sh (void) deb olish maqsadga muvofiqdir, return shart bo‘lmay qoladi [10].

Masalan: tomonlari berilgan to‘rtburchakning perimetrini va yuzini hisoblash uchun funksiyani quyidagicha aniqlashtiriladi:

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

Bu yerda float a, float b beriladigan kattalik hisoblanadi, float* p, float* s lar esa natijalar hisoblanadi.

Bu funksiyaga murojaat qilish quyidagicha bo‘ladi:

tt (2.3, 4, &p, &s); ya’ni 2*(a+b); va a*b ning qiymatlari adreslar bo‘yicha olinadi.

{Protseduralarni beriladigan kattaliklarsiz ham tashkil etish mumkin. Bunda funksiya tanasi ichida ishlatilgan barcha kattaliklar beriladiganlar hisobiga o‘tadi. }

Hosil qilingan protseduralarga murojaat qilish adres (&) operatsiyasi orqali amalga oshiriladi.

Masalan: $Z = \frac{a^5 + a^{-4}}{2a^n}$ hisoblash dasturi tuzilsin. Bu yerdagi darajani hisoblash protsedura sifatida tashkil etilsin. $y = x^n$ ni protsedura deb tashkil etamiz, bu yerda x, n - rasmiy parametrlar

```
# include <iostream.h>
void dar1 (float x, int n, float *y)
{ *y=1;
for (int i=0; i<=n; i++)
  *y = y*x; }
void main( ) {
int n=3 ;  float a, z, z1, z2, z3;
cin>>a;
dar1(a, 5,&z1); dar2( 1/a, 4, &z2); dar1(a, n,
&z3);
z = ( z1+z2) / z3 ;
 cout << «z=«<<z<<endl;
}
```

2-misol. 2 ta vektor berilgan. Vektorlar orasidagi burchak quyidagi formula bilan hisoblanadi:

$$\varphi = \arccos \frac{(x, y)}{\sqrt{(x, x)(y, y)}}$$

bu yerda (x,u), (x,x), (u,u) - vektorlarning skalyar ko‘paytmasi. Vektorlarning skalyar ko‘paytmasini dasturda protsedura sifatida tashkil eting.

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

```
typedef float mm[4];
void vec(mm a, mm b, float* s)
{*s=0;
  for (int i=0; i<4; i++)
  *s=*s+a[i]*b[i];   }
int  main ( )
{float fi, f1, f2, f3; int i;
mm  x, y; // mm x={1,2,3,4}, y={5,6,7,8};
for (i=0; i<4; i++)
cin >>x[i] >> y[i];
vec (x, y, &f1); vec (x, x, &f2); vec (y,y,&f3);
fi = f1 / sqrt( f2*f3); fi = atan(sqrt (1-fi*fi) /
fi);
cout << «fi=«<<fi*180/3.1415<<endl;  // natija
gradusda chiqadi
getch( );
}
```

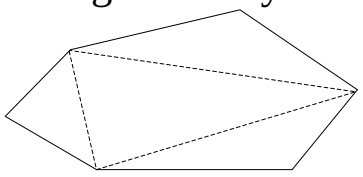
Protseduralarni tashkil etishda koʻrsatkichlardan tashqari yana ilovalardan ham foydalaniladi. Bu usul yanada qulay hisoblanadi. Unda (*) amalining oʻrniga toʻgʻridan-toʻgʻri adres olish (&) amali ishlatiladi va protseduraga murojaat qilish osonlashadi. Masalan:

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

Bu funksiyaga murojaat qilish quyidagicha boʻladi:

```
tt (2.3, 4, p, s);
```

Misol. Bir fermerning yer yuzasini va shu yerga toʻlaydigan yer soligʻini hisoblash dasturini tuzing. Yer maydoni quyidagi koʻrinishda:



```
# include <iostream.h>
# include <math.h>
# include <conio.h>
# define pi 3.1415
void yuza (int a, int b, int al, float& c, float&
s)
{ c = sqrt(a*a+b*b-2*a*b*cos(al*pi/180));
```

```
  s = a*b*sin(al*pi/180)/2;  }
int main ( )
{ int a1=10, b1=30, a2=40, b2=40,
a3=30,b3=50,al1=85, al2=145, al3=125;
float c1, c2, c3, s1, s2, s3, s4, s, sol, p;
yuza (a1, b1, al1, c1, s1); // yuza (10, 30, 85,
c1, s1) deb yozsa ham bo‘ladi
yuza (a2, b2, al2, c2, s2);
yuza (a3, b3, al3, c3, s3);
p = (c1+ c2 + c3) / 2;
s4 = sqrt ( p*(p - c1)*(p - c2)*(p - c3));
s = s1+s2+s3+s4;
s = s/100; sol = s * 8560; //    (som)
cout <<«er yuzasi=«<< s << endl;
cout << «soliq=«<<sol<<endl;
getch( ); }
```

2-misol. Kvadrat tenglamaning haqiqiy yechimlarini topish dasturini tuzing.

```
# include <iostream . h>
# include <math . h>
int kvad (float a, float b, float c, float &x1,
float &x2)
{ float d ;
d = b * b – 4*a*c;
if ( d < 0 ) return 0;
x1 = (-b + sqrt (d)) / (2*a);
x2 = (-b - sqrt (d)) / (2*a);
if ( x1 = = x2) return 1; else return 2;
}
int main ( )
{ float a, b, c, x1, x2;  int k;
cin >> a >> b >> c;
k = kvad (a, b, c, x1, x2);
switch ( k )
{
case 0 : cout << “echimi yo‘q”<< endl; break;
case 1 : cout << “x=”<< x1 << endl; break;
case 2 : cout << “x1=”<< x1 << “ x2=” << x2 <<
```

```
endl; break; }}
```

3-misol. 4x4 va 4x5 o‘lchamli matritsalar berilgan. Ulardagi juft ustunlari hadlari yig‘indisini topish dasturini tuzing. (natija vektor ko‘rinishida chiqadi)

```
# include <iostream. h>
typedef  float mmm[10][10];
typedef float mm[10]; int i, j;
void nodir (mmm a, int n, mm b)
{
for (j=0; j<4; j+=2)
{ b[j] = 0;
for (i=0; i<n; i++)
b[j] = b[j] + a[i][j]; } }
int main ( )
{ mmm d =
{{1,2,3,4},{1,2,3,4},{1,2,3,4},{1,2,3,4}};
mmm d1 =
{{1,2,3,4,5},{1,2,3,4,5},{1,2,3,4,5},{1,2,3,4,5}};
mm c, c1;
nodir (d, 4, 4, c);
nodir (d1, 4, 5, c1);
for (i=0; i<4; i++)
{ cout <<”c=”<< c[i];
cout << “c1=”<< c1[i] << endl;}
```

4 –misol. Ikki o‘zgaruvchining qiymatini o’zaro almashtirish dasturi tuzilsin.

```
# include <iostream.h>
int main( )
{
 float x, y;
 void aa( float *, float *);
 cout <<” x=”<< x <<endl; cin >> x;
 cout <<” y=” << y << endl;  cin >> y;
  aa ( &x, &y);
  cout << “\nNatija: \n”;
 cout <<”x=”<<x<<”y=”<<y;
 }
void aa (float *b, float *c)
```

```
   { float e;
    e = *b;
   *b = *c;
   *c = e; }
```

Asosiy dasturda x va y o‘zgaruvchilarining qiymatlari klaviaturadan kiritiladi. Masalada ikkita son o‘zaro o‘rin almashishi so‘ralmoqda. *aa ()* funksiyaning rasmiy parametrlari sifatida float * tavsiya etilgan. *aa ()* funksiyasiga murojaat qilinganida x va y larning son qiymatlari haqiqiy parametrlar sifatida qabul qilinadi. Bu dasturning ishlash jarayonida quyidagi natijalar olinadi:

x=33.3 y=66.6 qiymatlar kiritilsa

Natija:

x=66.600000 y=33.300000

5-misol. Uchburchakning perimetri va uning yuzasini hisoblash uchun dastur tuzilsin.

```
  # include <iostream.h>
  # include <math.h>
  int main ( )
  {
   float x, y, z, pp, ss;
   int tria (float, float, float, float *, float *);
   cout <<” x=”; cin >> x;
   cout <<“ y=”; cin >> y;
   cout <<” z=”; cin >> z;
   if (tria (x, y, z, &pp, &ss)==1)
   cout << “Uchburchak yuzasi=”<< ss << “va
  perimetri=”<< pp <<endl;
    else
   cout << “Ma’lumotlar noto‘g‘ri kiritilgan!” <<
  endl;
  }
   int tria ( float a, float b, float c, float *pp,
  float *ss)
  {
   float e;
   if (a+b<=c || a+c<=b || b+c<=a) return 0;
   else
    { *pp = a+b+c;
```

```
    e=*pp/2;
   *s=sqrt(e*(e-a)*(e-b)*(e-c));
    return 1;}
```

Dasturning natijasi:

```
x=3
y=4
z=5
pp=12.00000
ss=6.00000
```

Funksiyada parametrlar sifatida massivlar va satrlar ishlatilishi mumkin. Agar funksiyaning parametri sifatida massivlar ishlatilsa, funksiya ichida massiv boshlanishining adresi uzatiladi. Bunga misol tariqasida vektorlarning skalyar ko‘paytmasini hisoblovchi funksiya sarlavxasini ko‘rib chiqamiz:

float skalyar(int n, float a[ ], b[ ]) yoki

float skalyar (int n, float *a, float *b)

Bu yerda float a[ ] va float *a yozuvlari parametr sifatida bir xil ma’noni anglatadi.

Satrlar funksiya parametri sifatida.

Satrlar funksiya parametri sifatida ishlatiladigan bo‘lsa, char [ ] yoki char* tipli ko‘rsatkichlar kabi tavsiflanadi. Oddiy massiv parametridan farqli o‘laroq, satrning uzunligini ko‘rsatish shart emas. Bunda \0 belgisi satr oxirini avtomatik ravishda ko‘rsatadi. Misol tariqasida satrlarni qayta ishlovchi bir nechta dasturlarni ko‘rib chiqamiz. Bu dasturlarning o‘xshashi standart kutubxonalarda saqlanadi va ularni ishga tushirish uchun string.h, stdlib.h fayllaridan foydalanish kerak bo‘ladi.

1. Satr tipli o‘zgaruvchining uzunligini aniqlash uchun funksiya:

```
int len(char e[])
{ int m;
 for(m=0; e[m]!='\0'; m++)
 return m; }
```

yoki bu dasturdagi massivni ko‘rsatkichlar orqali quyidagicha ifoda etish mumkin:

```
int len(char *s)
{ int m;
 for(m=0; *s!='\0'; m++)
 return m; }
```

2. Satr tipli massiv elementlarini teskari tardibda ifoda etish uchun funksiya:

```
void invert (char e[ ])
{ char s; int i, j, m;
  for (m=0; e[m]!='\0'; m++)
  for(j=0, j=m-i; i<j; i++, j- -)
 { s=e[i]; e[i] = e[j]; e[j] =s; } }
```

Dasturdagi void tipdan ma'lumki, bu funksiya hech qanday qiymat qaytarmaydi.

Masalan:

```
# include <iostream.h>
int main( )
{ char ct[ ] =”0123456789”;
 void invert (char [ ]);
 invert(ct); cout << ct; }
```

Natija: 9876543210

3. Satrning chap tomonidan kiritilgan boshqa satrni qidirish funksiyasi:

```
int index(char *ct1, char *ct2)
{ int i, j, m1, m2;
 for(m1=0; ct1[m1]!='\0'; m1++)
 for(m2=0; ct2[m2]!='\0'; m2++)
 if (m2>m1) return -1;
 for(i=0; i<m1-m2; i++)
 { for(j=0; j<m2; j++)
  if (ct2[j] !=ct1[i+j] ) break;
  if (j==m2) return 1; }
  return -1; }
```

Funksiyaning ishlashiga misol:

```
# include <iostream.h>
int main ( )
{ char c1[ ] =”og'irlik_yig'indisi”;
 int index(char[ ], char[ ]);
 char c2[ ] = “non”;
 char c3[ ] = “olma”;
 cout<< index(c1,c2);
 cout<< index(c1,c3); }
```

Nazariy bilimlarni tekshirish uchun savollar

1. Funksiyaga ta’rif bering.
2. Funksiya prototipi nima?
3. Funksiya parametrlariga ta’rif bering?
4. Kelishuv bo‘yicha qiymat berish deganda nimani tushunasiz?
5. O‘zgaruvchining amal qilish sohasi deyilganda nima tushuniladi?
6. Funksiya bilan protsedurani farqi nimada?
7. Qiymat qaytarmaydigan funksiyalarni yana qanday nomlash mumkin?
8. Signatura deyilganda nima nazarda tutiladi?
9. Rekursiv funksiyaga ta’rif bering.
10. C++ dasturlash tilida kutubxona fayli qanday yaratiladi?

3.2.Massivlar

📖 M*assiv tushunchasi, uning turlari, elementlari, o‘lchamlari, elementlarga qiymat o‘zlashtirish va ularni qayat ishlash usullari haqida ma’lumotlar keltirilgan. Massiv elementlarini saralash usullari ko‘rib chiqilgan. Ko‘p o‘lchovli massivlar, ularning elementlariga murojaat, elmentlarning qiymatlarini chiqarish, elementlariga qiymat o‘zlashtirish va qayta ishlov berish yo‘llari haqaida ma’lumotlar keltirilgan.*

🖉 ***Kalit so‘zlar****: Massiv, index, manzil, element. ro‘yxat, manzil, nolinchi ko‘rchsatkich, tugun, adres olish &, bo‘shatish, ko‘rsatkich, virtual destruktor, xotira, xotira chiqishi, destruktor, tipni o‘zlashtirish, resurslar chiqishi, a’zo destruktori.*

REJA:

1. Massiv tushunchasi. Massiv elementlariga qiymat kiritish va chiqarish usullari.
2. Statik massivlar.
3. Massiv elementlarini saralash va qidiruv usullari.
4. Ko‘p o‘lchamli massivlarni tavsiflash, ularga ishlov berish;
5. Ko‘rsatgichli massivlar haqida. Funksiya va massivlar.

Biz bu bobni massiv ma’lumotlar turini tanishtirishdan boshlaymiz. Massivlar bir necha qiymatlarni yig‘ish uchun C++ da asosiy mexanizm hisoblanadi. Quyidagi bo‘limlarda siz massivlarni qanday aniqlashni va massiv elementlaridan qanday foydalanishni o‘rganib olasiz.

Deylik siz qiymatlar ketma-ketligini o‘quvchi va chop etuvchi dastur yozmoqchisiz, buning uchun siz eng katta qiymatni quyida

berilgan koʻrinishda belgilang:

```
32
54
67.5
29
34.5
80
115 <= eng katta qiymat
44.5
100
65
```

Bu qiymatlarni barchasini koʻrmasdan turib, siz qaysi birini eng katta qiymat deb belgilash kerakligini bilmaysiz. Oxir oqibat, oxirgi qiymat eng kattasi boʻlishi mumkin. Shuning uchun, dastur chop etishdan oldin birinchi navbatda barcha qiymatlarni saqlab olishi kerak. Har qaysi qiymatni alohida oʻzgaruvchida oddiygina saqlasangiz boʻlmasmidi? Oʻnta qiymatni ya'ni qiymat 1 (value1), qiymat 2 (value2), qiymat 3 (value3), …, qiymat 10 (value10)larni oʻnta oʻzgaruvchida saqlaydigan boʻlsak. Biroq, bunda oʻzgaruvchilarning ketma - ketligi foydalanish uchun noqulay. Siz, har qaysi oʻzgaruvchi uchun oddiy kodni oʻn marotaba yozishishingizga toʻgʻri keladi. Bu muammoni hal etish uchun massivdan foydalaning: qiymatlar ketma - ketligini saqlovchi struktura (tuzilma).

Xotirada ketma-ket (regulyar) joylashgan bir xil tipdagi qiymatlarga *massiv* deyiladi.

Massiv turlari. Odatda katta hajmdagi, lekin cheklangan miqdordagi va tartiblangan qiymatlarni qayta ishlash bilan bogʻliq masalalarni yechishda massivlarga ehtiyoj tugʻiladi. Faraz qilaylik, talabalar guruhining reyting ballari bilan ishlash masalasi qoʻyilgan. Unda guruhning oʻrtacha reytingini aniqlash, reytinglarni kamayishi boʻyicha tartiblash, aniq talabaning reytingi haqida ma'lumot berish va boshqa masalalarni yechish zarur boʻlsin. Qayd etilgan masalalarni yechish uchun ma'lumotlarning (reytinglarning) tartiblangan ketma-ketligi zarur boʻladi. Bu yerda tartiblanganlik ma'nosi shundaki, ketma-ketlikning har bir qiymati oʻz oʻrniga ega boʻladi (birinchi talabaning reytingi massivda birinchi oʻrinda, ikkinchi talabaniki - ikkinchi oʻrinda va hokazo). Ma'lumotlar ketma-ketligini ikki xil usulda hosil qilish mumkin. Birinchi yoʻl - har bir reyting uchun alohida oʻzgaruvchi aniqlash: Reyting1,…,ReytingN. Lekin,

guruhdagi talabalar soni yetarlicha katta bo‘lganda, bu o‘zgaruvchilar qatnashgan dasturni tuzish katta qiyinchiliklarni yuzaga keltiradi. Ikkinchi yo‘l - ma’lumotlar ketma-ketligini yagona nom bilan aniqlab, uning qiymatlariga murojaatni, shu qiymatlarning ketma-ketlikda joylashgan o‘rnining tartibi(indeksi) orqali amalga oshiriladi. Reytinglar ketma-ketligini Reyting deb nomlab, undagi qiymatlariga $Reyting_1$,…,$Reyting_N$ ko‘rinishida murojaat qilish mumkin. Odatda ma’lumotlarning bunday ko‘rinishiga massivlar deyiladi. Massivlarni matematikadagi sonlar vektoriga o‘xshatish mumkin, chunki vektor ham o‘zining individual nomiga ega va u fiksirlangan miqdordagi bir tipdagi qiymatlardan - sonlardan iboratdir.

Demak, massiv - bu fiksirlangan miqdordagi ayrim qiymatlarning (massiv elementlarining) tartiblangan majmuasidir. Barcha elementlar bir xil tipda bo‘lishi kerak va bu tip *elementlar tipi* yoki massiv uchun *asosiy tip* deb nomlanadi. Yuqoridagi keltirilgan misolda Reyting - haqiqiy tipdagi *vektor* deb nomlanadi.

Dasturda ishlatiladigan har bir massiv o‘zining individual nomiga ega bo‘lishi kerak. Bu nom *to‘liq o‘zgaruvchi* deyiladi, chunki uning qiymati massivning o‘zi bo‘ladi. Massivning har bir elementi massiv nomi, hamda kvadrat qavsga olingan va *element selektori* deb nomlanuvchi indeksni ko‘rsatish orqali oshkor ravishda belgilanadi. Murojaat sintaksisi:

<massiv nomi >[<indeks>]

Bu ko‘rinishga *xususiy o‘zgaruvchi* deyiladi, chunki uning qiymati massivning alohida elementidir. Bizning misolda Reyting massivining alohida komponentalariga Reyting[1],…,Reyting[N] xususiy o‘zgaruvchilar orqali murojaat qilish mumkin. Boshqachasiga bu o‘zgaruvchilar *indeksli o‘zgaruvchilar* deyiladi.

Massiv indeksi sifatida butun son qo‘llaniladi. Umuman olganda indeks sifatida butun son qiymatini qabul qiladigan ixtiyoriy ifoda ishlatilishi mumkin va uning qiymati massiv elementi tartibini aniqlaydi. Ifoda sifatida o‘zgaruvchi ham olinishi mumkinki, o‘zgaruvchining qiymati o‘zgarishi bilan murojaat qilinayotgan massiv elementini aniqlovchi indeks ham o‘zgaradi. Shunday qilib, dasturdagi bitta indeksli o‘zgaruvchi orqali massivning barcha elementlarini belgilash (aniqlash) mumkin bo‘ladi. Masalan, Reyting[I] o‘zgaruvchisi orqali I o‘zgaruvchining qiymatiga bog‘liq ravishda Reyting massivining ixtiyoriy elementiga murojaat qilish mumkin.

Haqiqiy tipdagi (float, double) qiymatlar to‘plami cheksiz bo‘lganligi sababli ular indeks sifatida ishlatilmaydi.

C++ tilida indeks doimo 0 dan boshlanadi va uning eng katta qiymati massiv e’lonidagi uzunlikdan bittaga kam bo‘ladi.

Massiv e’loni quyidagicha bo‘ladi:

<tip> <nom> [<uzunlik>]={boshlang‘ich qiymatlar}.

Bu yerda <uzunlik> - o‘zgarmas ifoda. Misollar:

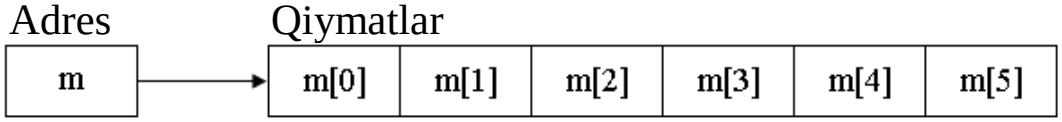
int m[6]={1,4,-5,2,10,3};

float a[4];

Massiv statik va dinamik bo‘lishi mumkin. Statik massivning uzunligi oldindan ma’lum bo‘lib, u xotirada ma’lum adresdan boshlab ketma-ket joylashadi. Dinamik massivni uzunligi dastur bajarilish jarayonida aniqlanib, u dinamik xotiradagi ayni paytda bo‘sh bo‘lgan adreslarga joylashadi. Masalan,

int m[6];

ko‘rinishida e’lon qilingan bir o‘lchamli massiv elementlari xotirada quyidagicha joylashadi:



7.1-rasm. Bir o‘lchamli massivning xotiradagi joylashuvi

Massivning *i*-elementiga *m[i]* yoki ***(m+i)** orqali murojaat qilish mumkin. Massiv uzunligini ***sizeof(m)*** amali orqali aniqladi.

Massiv e’lonida uning elementlariga boshlang‘ich qiymatlar berish mumkin va uning bir nechta variantlari mavjud.

1) o‘lchami ko‘rsatilgan massiv elementlarini to‘liq initsializatsiyalash:

int t[5]={-10,5,15,4,3};

Bunda 5 ta elementdan iborat bo‘lgan *t* nomli butun tipdagi bir o‘lchamli massiv e’lon qilingan va uning barcha elementlariga boshlang‘ich qiymatlar berilgan. Bu e’lon quyidagi e’lon bilan ekvivalent:

int t[5];

t[0]=-10; t[1]=5; t[2]=15; t[3]=4; t[4]=3;

2) o‘lchami ko‘rsatilgan massiv elementlarini to‘liqmas initsializatsiyalash:

int t[5]={-10,5,15};

Bu yerda faqat massiv boshidagi uchta elementga boshlang‘ich qiymatlar berilgan. Shuni aytib o‘tish kerakki, massivning boshidagi yoki o‘rtasidagi elementlariga qiymatlar bermasdan, uning oxiridagi elementlarga boshlang‘ich qiymat berish mumkin emas. Agarda massiv elementlariga boshlang‘ich qiymat berilmasa, unda kelishuv bo‘yicha

static va *extern* modifikatori bilan e’lon qilingan massiv uchun elementlarining qiymati 0 soniga teng deb olinadi va massiv elementlarining boshlang‘ich qiymatlari noma’lum hisoblanadi.

3) o‘lchami ko‘rsatilmagan massiv elementlarini to‘liq initsializatsiyalash:

int t[]={-10,5,15,4,3};

Bu misolda massivni barcha elementlariga qiymatlar berilgan hisoblanadi, massiv uzunligi kompilyator tomonidan boshlang‘ich qiymatlar soniga qarab aniqlanadi. Agarda massiv uzunligi berilmasa, boshlang‘ich qiymati berilishi shart.

Massivni e’lon qilishga misollar:

```
shar ch[4]={‘a’,‘b‘,‘c‘,‘d‘}; //belgilar massivi
int in[6] ={10,20,30,40};    // butun sonlar
massivi
char str[]=”abcd”;
//satr uzunligi 5 ga teng, chunki uning oxiriga
// ‘\0‘ belgisi qo‘shiladi
char str[]={‘a’,‘b‘,‘c‘,‘d‘};
// yuqoridagi satrning boshqacha yozilishi
```

Masala. Bir oy ichidagi kundalik haroratlar berilgan. Oy uchun o‘rtacha haroratni hisoblash dastursi tuzilsin.

Dastur matni:

```
int main()
{
const int n=30;
 int temp[n];
 int i,s,temp_urtacha;
 cout << ”Kunlik haroratni kiriting:\n”
 for (i=0;i<n;i++){
cout << ”\n temp[“<<i<<”]=”;
  cin >> temp[i]; }
 for (i=0,s=0; i<n;i++)s+=temp[i];
 temp_urtacha=s/n;
 cout << ”Kunlik harorat :\n”;
 for(i=0;i<n;i++)cout<< ”\t
temp[“<<i<<”]=”<<temp[i];
 cout<<”Oydagi o‘rtacha harorat= ”<<temp_urtacha;
 return 0; }
```

```
#include <iostream>
using namespace std;
int main(){
const int CAPACITY = 1000;
double values[CAPACITY];
int current_size = 0;
cout << "Please enter values, Q to quit:" << endl;
double input;
while (cin >> input){
 if (current_size < CAPACITY){
values[current_size] = input;
current_size++; }
}
double largest = values[0];
for (int i = 1; i < current_size; i++){
if (values[i] > largest){
largest = values[i]; }}
for (int i = 0; i < current_size; i++){
cout << values[i];
f (values[i] == largest){
cout << " <== largest value";  }
cout << endl; }
return 0;}
```

Dastur natijasi:

```
Please enter values, Q to quit:
34.5 80 115 44.5 Q
34.5
80
11
<== largest value  44.5
```

Massiv elementlarini tartiblash. Agar standard algoritmlardan hech biri sizning masalangizni yechish uchun yetarli bo‘lmasa nima qilasiz? Bu bo‘limda siz fizik predmetlarni manipulyasiya qilish orqali algoritmlarni tuzish texnikalarini o‘rganasiz. Faraz qiling sizga jadval berilgan siz uning birinchi va ikkinchi yarmini boshqa holatga keltirishinngiz kerak. Masalan, agar jadval 8 ta raqamdan iborat bo‘lsa,

9 13 21 4 11 7 1 3

Siz uni shunday o‘zgartirishingiz kerak

11 7 1 3 9 13 21 4

Massivlarni initsializatsiyalash quyidagi misollarda ko‘rsatilgan:

```
int a[2][3]={0,1,2,10,11,12};
int b[3][3]={{0,1,2},{10,11,12},{20,21,22}};
int c[3][3][3]={{{0}},{{100,101},{110}},
{{200,201,202},{210,211,212},{220,221,222}};
```

Birinchi operatorda boshlang‘ich qiymatlar ketma-ket yozilgan, ikkinchi operatorda qiymatlar guruhlashgan, uchinchi operatorda ham guruhlashgan, lekin ba’zi guruhlarda oxirgi qiymatlar berilmagan.

Misol uchun, matritsalar va vektor ko‘paytmasini - $C = A \times b$ hisoblash masalasini ko‘raylik. Bu yerda $A = \{a_{ij}\}$, $b = \{b_j\}$, $c = \{c_i\}$, $0 \le i < m, 0 \le j < n.$ Hisoblash formulasi - $c_i = \sum_{j=0}^{n-1} a_{ij} b_j$.

Mos dastur matni:

```
int main(){
 const int n=4,m=5;
 float a[m][n],b[n],c[m];
 int i,j; float s;
 for(i=0;i<m;i++)
  for(j=0;j<n;i++)cin>>a[i][j];
 for(i=0;i<m;i++)cin>>b[i];
 for(i=0;i<m;i++) {
  for (j=0,s=0;j<n;j++)s+=a[i,j]*b[j];
  c[i]=s;  }
 for (i=0;i<m;i++)cout<<”\t c[“<<i<<”]=”<<c[i];
 return 0; }
```

```
#include <iostream>
using namespace std;
int main(){
const int CAPACITY = 1000;
double values[CAPACITY];
int current_size = 0;
cout << "Please enter values, Q to quit:" << endl;
double input;
while (cin >> input) {
if (current_size < CAPACITY){
```

```
values[current_size] = input;
current_size++;}
}
double largest = values[0];
for (int i = 1; i < current_size; i++){
if (values[i] > largest){
largest = values[i]; }}
for (int i = 0; i < current_size; i++){
cout << values[i];
if (values[i] == largest){
cout << " <== largest value"; 37 }
cout << endl; }
return 0; }
```

Umuman olganda saralashning maqsadi berilgan obyektlar to‘plamini aniq bir tartibda guruhlab chiqish jarayoni ta’riflanadi.

a_0, a_1... a_n ketma – ketlik berilgan bo‘lsin.

Ketma – ketlikning elementlarini saralash (masalan: $a_i <= a_{i+1}$, i = 0 dan n-1 gacha) masalasi qo‘yilgan bo‘lsin. Bu masalani ishlash algoritmini tanlaganda quyidagilarni baholash zarur: *Saralash vaqti* – algoritmni baholaydigan asasiy parametr hisoblanadi. *Xotira* – algoritm ishlashi uchun ketadigan ko‘shimcha xotira hajmi. Bunda ma’lumotlar va dastur kodi uchun ketadigan xotira hajmi hisobga olinmaydi. *Turg‘unlik* – dastur(ketma-ketlik)ning boshqa qiymatlarda ham turg‘un ishlashi tushuniladi.

Saralash algoritmlari sinfifikasiyasi. Berilgan ketma-ketlikni saralashda ketma-ketlikning xarakteristikasi mos ravishda u yoki bu saralash algoritmi olinadi. Aks holda algoritmlar kerakli natijani bermaydi.

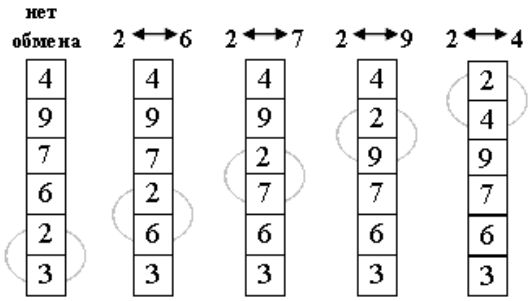
“Tez” saralash algoritmi. “Tez” saralash algoritmi bosqichlari:

1. Massivning o‘rta elementini tanlab olamiz.
2. O‘rta element chap tomoniga o‘rta elementdan kichik elementlarni joylashtiramiz, o‘rta elementning o‘ng tomoniga esa o‘rta elementdan katta elementlarni joylashtiramiz.

```
int i=quyi;
      int j=yuqori;
      int x=A[(quyi+yuqori)/2];
      do {
          while(A[i]<x) ++i;
          while(A[j]>x) --j;
```

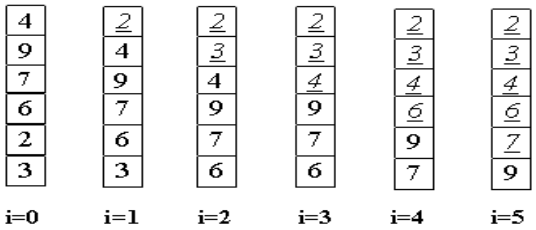
```
            if(i<=j){
                int temp=A[i];
                A[i]=A[j];
                A[j]=temp;
                i++; j--;
            }
        } while(i<=j);
```

Sharsimon saralash algoritmi. Massiv elementlarini tepadan pastga qarab saralaymiz. Bunda faqat juft elementlar $a_i <= a_{i+1}$ (i = 0 dan n-1 gacha) shart bilan tekshiriladi, agar shart bajarilmasa ular o‘zaro o‘rin almashtiriladi.



Bu jarayon oxirgi element qolguncha bajariladi. Natijada massiv elementlari o‘sish tartibida saralanadi.

Dasturdagi bosqichlar:




```
    long i, j,
    float x, a[];
    for( i=0; i < size; i++)
  for( j = size-1; j > i; j-- )
  {  if ( a[j-1] > a[j] )
  { x=a[j-1]; a[j-1]=a[j]; a[j]=x;  }  }
```

Saralashning maqsadi keyinchalik, saralashgan to‘plamni qidirilayotgan elementini topishdan iborat. Bu qariyb unversal, fundamental jarayon. Biz bu jarayon bilan har kuni uchrashamiz–telefon daftaridagi saralash, kitoblar sarlavhasida, kutubxonalarda, lug‘atlarda, pochtada va h.k. Hatto yosh bolalar ham o‘z narsalarini tartiblashga o‘rganadi. Saralashning juda ko‘p usullari mavjud. Ular turli to‘plamlar uchun turlicha bo‘lishi mumkin. Massivlarni saralash uchun

ishlatiladigan usul unga berilgan xotirani ixcham holda ishlatish lozim. Boshqacha qilib aytganda, saralanayotgan massiv xuddi shu massivni o‘zida amalga oshirilishi lozim. Biz quyidagi saralash bo‘yicha bir nechta sodda va ma’lum usullarni qaraymiz. Ular to‘g‘ri usullar deb aytiladi. Saralash usullari to‘g‘risida quyidagi fikrlarni bildirish mumkin:

1. To‘g‘ri usullar ko‘plab saralashning asosiy tamoyillarining xarakterini ochib berishi uchun qulay.

2. Bu usullarni dasturchilar oson tushunadi va ular qisqa. Eslatib o‘tamiz, dasturning o‘zi ham xotira egallaydi.

3. Murakkab usullar ko‘p sondagi amallarni talab qiladi, lekin bu amallarning o‘zlari yetarlicha murakkab bo‘lganlari uchun, kichik n larda tez va katta n larda sekin ishlaydi. Ammo ularni katta n larda ishlatib bo‘lmaydi.

Bitta massivni o‘zida saralashni ularni mos aniqlangan tamoyillari bilan uch kategoriyaga ajratish mumkin:

1. Qo‘shish orqali saralash (by insertion);
2. Ayirish orqali saralash (by selection);
3. Almashish orqali saralash (by exchange).

To‘g‘ridan-to‘g‘ri qo‘shish orqali saralash

Bu usul karta o‘yinida ko‘p qo‘llaniladi.

Kartaning elementlari fikran tayyor holdagi ketma-ketlik qismlarga bo‘linadi.

Har qadamda *i=2* dan boshlab *i* ta element ketma-ketlikdan chiqariladi va tayyor ketma-ketlikka qo‘yiladi. Bunda u har doim kerakli joyga qo‘yiladi. *i* ning qiymati har doim bittaga oshirib boriladi.

To‘g‘ridan to‘g‘ri tanlash yordamida saralash

- Eng kichik kalitli element tanlanadi.
- Uni birinchi element a_1 bilan o‘rinlari almashtiriladi.

So‘ng bu jarayon qolgan n-1 element bilan, so‘ngra n-2 element bilan va h.k. bitta eng katta element qolmaguncha davom ettiriladi.

C/C++ algoritmik tilida faqat bir o‘lchamli massivlar bilan emas, balki ko‘p o‘lchamli massivlar bilan ham ishlash mumkin. Agar massiv o‘z navbatida yana massivdan iborat bo‘lsa, demak ikki o‘lchamli massiv, ya’ni matritsa deyiladi. Massivlarning o‘lchovi kompyuterda ishlashga to‘sqinlik qilmaydi, chunki ular xotirada chiziqli ketma-ket elementlar sifatida saqlanadi. Ko‘p o‘lchamli massivlarni xuddi 1 o‘lchamli massivga o‘xshab e’lon qilinadi, faqat indeks tipi sifatida massivning satrlari (qatorlari) va ustunlari tipi ko‘rsatiladi va ular

alohida [ ][ ] qavslarda ko‘rsatiladi. Masalan: A nomli butun sonlardan iborat 2 o‘lchamli massiv berilgan bo‘lsa va satrlar soni **n** ta, ustunlar soni **m** ta bo‘lsa: *int a[n][m]*

Ikki o’lchovli massiv elementlarini kiritish-chiqarish, ular ustida amallar bajarish ichma-ich joylashgan parametrli sikllar ichida bo‘ladi, ya’ni 1-sikl satrlar uchun, 2-sikl ustunlar uchun. Masalan:

```
for  ( i=0; i<=3; i++)
for ( j=0; j<=3; j++)
cin >>a[i][j];
```

Agar ularni klaviaturadan kiritish kerak bo‘lsa, *cin* operatori yordamida tashkil etilsa, quyidagicha kiritiladi:

1 2 3
4 5 6
7 8 9

Bundan tashqari massiv elementlarini e’lon qilish bilan birga ularni initsializatsiya ham qilish mumkin:

int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}};

Natijalar chiroyli ko‘rinishda bo‘lishi uchun chiqarish operatorini quyidagicha qilib tashkil etish kerak:

```
for  (int i=0; i<3; i++)
{ for (int j=0; j<3; j++)
 cout <<”a[“<<i<<”,”<<j<<”]=”<<a[i][j];
cout <<endl; }
getch ( );}
```

1-misol. *A* va *B* matritsalari berilgan. Quyidagi formula orqali yangi *S* matritsasini hosil qiling: $S_{ij} = A_{ij} + B_{ij}$; bu yerda i=1,3; j=1,2;

$$A = \begin{Bmatrix} 24.3 & -4.15 \\ 0 & 18.4 \\ -8.8 & -15.75 \end{Bmatrix}, B = \begin{Bmatrix} 0.1 & -4.8 \\ 6.8 & 7.1 \\ -2.8 & 0.4 \end{Bmatrix}$$

```
# include <iostream.h>
# include <conio.h>
void main ( )
{ float a[3][2]={{24.3,-4.15},{0,18.4},{-8.8,-
15.75}},
b[3][2]={{0.1,-4.8},{6.8,7.1},{-2.8,0.40}};
float  c[3][2];
int i, j;
for (i = 0; i < 3; i++)
```

```
{ for (j = 0; j < 3; j++)
{ c[i][j] = a[i][j] + b[i][j];
cout <<”c[“<<i<<”,”<<j<<”]=”<<c[i][j]; }
cout <<endl; }
getch ( );}
```

Massiv elementlariga son qiymat berishda kompyuter xotirasidagi tasodifiy butun sonlardan foydalanish ham mumkin. Buning uchun standart kutubxonaning rand ( ) funksiyasini ishga tushirish kerak. rand ( ) funksiyasi yordamida 0 ÷ 32767 oraliqdagi ixtiyoriy sonlarni olish mumkin. Bu qiymatlar umuman tasodifiydir. (psevdo – tasodifiy degani).

Agar dastur qayta-qayta ishlatilsa, ayni tasodifiy qiymatlar takrorlanaveradi. Ularni yangi tasodifiy qiymatlar qilish uchun ***srand()*** funksiyasini dasturda bir marta e’lon qilish kerak. Dastur ishlashi jarayonida ehtiyojga qarab ***rand()*** funksiyasi chaqirilaveradi. Tasodifiy qiymatlar bilan ishlash uchun <stdlib.h> faylini e’lon qilish zarur. ***srand( )*** funksiyasidagi qiymatni avtomatik ravishda o‘zgaradigan holatga keltirish uchun ***srand (time (NULL))*** yozish ma’qul, shunda kompyuter ichidagi soatning qiymati ***time ( )*** funksiyasi yordamida o‘rnatiladi va srand ga parametr sifatida beriladi. NULL yoki 0 deb yozilsa, qiymat sekundlar ko‘rinishida beriladi. Vaqt bilan ishlash uchun <time.h> ni e’lon qilish kerak. Misol:

```
# include <iostream.h>
# include <conio.h>
# include <stdlib.h>
# include <time.h>
int main ( )
{ srand ( time (0));
int a[5], b[5], i;
for (i = 0; i < 5;  i++)     a[i] = rand ( );
for (i = 0; i < 5;  i++)
{  b[i] = a[i] + 64;
cout << “b=”<<b[i]<<endl; }    getch ( ); }
```

Izoh: tasodifiy sonlar ichida manfiy sonlarning ham qatnashishini ixtiyor etsak,

a[i] = 1050 - rand ( ); yoki a[i] = rand ( )-1000; deb yozish ham mumkin.

2-misol. 2ta matritsa berilgan. Ularni o‘zaro ko‘paytirib yangi matritsa hosil qiling. Bu yerda 1-matritsaning ustunlar soni 2-matritsaning satrlar soniga teng bo‘lishi kerak.

```
# include <iostream.h>
# include <conio.h>
# include <stdlib.h>
# include <time.h>
int main ( )
{
{ srand ( time (0));
int a[3][3], b[3][3],c[3][3], i, j, k;
for (i=0; i<3; i++)
for (j=0; j<3; j++)
a[i][j] = rand ( );
for (i=0; i<3; i++)
for (j=0; j<3; j++)
b[i][j] = rand ( );
for (i=0; i<3; i++)
{ for (j=0; j<3; j++)
{  c[i][j] = 0;
for (k=0; k<3; k++)
c[i][j] = c[i][j] + a[i][k]*b[k][j];
cout <<"c="<<c[i][j]<<"\t"; }
 cout << endl; }
getch ( );}
```

3-misol. A matritsani B vektorga ko‘paytirish algoritmi.

$S_i = \sum_{\substack{i=1 \\ j=1}}^{n} a_{ij} * b_j$

Izoh: matritsaning satrlari soni vektorning satrlariga teng bo‘lishi kerak.

Masalan: $A = \begin{Bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{Bmatrix}, B = \begin{Bmatrix} 1 \\ 3 \\ 6 \end{Bmatrix}$

$C_1 = 1*1+2*3+3*6 = 25$

$C_2 = 4*1+5*3+6*6 = 55$

$C_3 = 7*1+8*3+9*6 = 85$

```
# include <iostream.h>
# include <conio.h>
int main ( ){
int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}}, b[3] =
{1,3,6}, c[3], i, j;
```

```
for (i=0; i<3; i++)
{  c[i] = 0;
    for (j=0; j<3; j++)
    c[i] = c[i] + a[i][j] * b[j];
    cout <<"c="<<c[i]<<endl; }
getch ( );}
```

4-misol. Matritsani transponerlash algoritmini tuzing. Matritsani transponerlash deb, ustun va satr elementlarini o‘zaro o‘rin almashtirishga aytiladi, ya’ni $A_{ij} = B_{ji}$

```
# include <iostream.h>
# include <conio.h>
int main ( ){
int  a[3][3] = {{1,2,3},{4,5,6},{7,8,9}},
b[3][3],
 i, j;
for ( i=0; i<3; i++)
{  for  ( j=0; j<3; j++)
 {   b[i][j] = a[j][i];
    cout <<"b["<<i<<","<<j<<"]="<<b[i][j];   }
cout << endl;   }
getch ( );}
```

5-misol. 3 ta qator va 4 ta ustunga ega A matritsa berilgan. Undagi eng kichik elementni va uning indeksini topish, hamda o‘sha qatorni massiv shaklida chiqarish dasturini tuzing.

```
# include <iostream.h>
# include <conio.h>
int main ( ){
int  a[3][4] = {{1,2,3,4},{4,5,6,7},{7,8,9,10}},
i, j, k, h,  min;
int b[4];
min = a[0][0];
for (i=0; i<3; i++)
for (j=0; j<4; j++)
{ if ( a[i][j] > min)  { min = a[i][j];  k = i;
h = j; }  }
cout << "min="<<min<<"  k="<<k<<"
h="<<h<<endl;
for  ( j=0; j<4; j++)
```

```
{ b[j] = a[k][j];
   cout <<”b=”<<b[j];  }
getch ( ); }
```

6-misol. Saralash masalasi. Massiv elementlarini o‘sib borish tartibida saralash algoritmini tuzing. (пузырковый метод)

Avval 1 o‘lchovli massiv elementlarini saralashni ko‘rib o‘tamiz.

```
# include <iostream.h>
# include <conio.h>
# include <stdlib.h>
# include <time.h>
int main ( ){
srand (time (0));
float   a[10] , b;  int  i,  j;
for (i = 0; i<10; i ++)
a[i] = rand( ) /33.;
for( j = 0; j<10; j++)
for ( i = 0; i < 0;  i ++){
 if  (a[i] < a[i+1])
{  b = a[i];
a[i] = a[i+1];
a[i+1] = b; }  }
cout. precision (3);
for (i = 0; i  < 10; i++)
cout << a[i]<<endl;  getch ( ); }
```

Endi 2 o‘lchamli massiv elementlarini saralashni ko‘ramiz:

```
# include <iostream.h>
# include <conio.h>
int main ( ){
float   a[3][3] = {{.....},{.....},{.....}}, b;
int i, j, k;
for ( k=0; k<3; k++)
for ( i=0; i<3; i++)
for ( j=0; j<2; j++)
{  if  (a[i][j] > a[i][j+1] )
{  b = a[i][j];  a[i][j] = a[i][j+1];  a[i][j+1]
= b;  }  }
for ( i=0; i<3; i++)
```

```
{  for ( j=0; j<3; j++)
cout <<"a="<<a[i][j];  cout << endl;  } getch (
); }
```

Yuqoridagi dastur saralashni qator bo‘yicha olib borish uchun. Agar saralashni ustun bo‘yicha qilish kerak bo‘lsa, quyidagicha yozish kerak bo‘ladi:

```
for  ( i=0; i< 2; i++)
for  ( j=0; j<3; j++)
{ if  ( a[i,j] > a[i+1, j] ) {   b:= a[i, j];
a[i, j]:= a[i+1, j];  a[i+1, j]:= b;  }
```

Agar saralashni o‘sib borish tartibida qilish kerak bo‘lsa, if operatoridagi solishtirish belgisi > bo‘lishi kerak, agar kamayish tartibida kerak bo‘lsa, solishtirish belgisi < ko‘rinishida bo‘lishi kerak.

7-misol. Matritsaning izini hisoblash dasturini tuzing. Matritsaning izi deb bosh diagonal elementlarining yig‘indisiga aytiladi. Shu dasturda teskari (qo‘shimcha) diagonal elementlarining yig‘indisini ham hisoblashni ko‘rib o‘ting.

```
# include <iostream.h>
# include <conio.h>
int main ( ){
float   a[3][3] = {{.....},{.....},{.....}},
s1=0, s2=0;
int i, j;
for  ( i=0; i<3; i++)
s1 = s1 + a[i][i];
for ( i=0; i<3; i++)
for ( j=0; j<3; j++)
if  ( i+j = = 2)    s2 = s2 + a[i][j];
cout <<"s1="<<s1<<"   s2="<< s2<< endl;
getch ( );  }
```

8-misol. Har bir hadi $a_n = \frac{n!}{(2n)!}$ formulasi orqali hisoblanadigan satr yig‘indisini 0,0001 aniqlikda hisoblash dasturini tuzing.

```
# include <iostream.h>
# include <conio.h>
int  main ( )
{  int  n =1;  float  s1 = 0, s2 = 0;
float  p1 =1,  p2 = 1;
```

```
while  (s2 > 0.0001)
{ p1 = p1 * n;                                  //
p1*=n;
p2 = p2 * 2*n*(2*n-1);              // p2*=
2*n*(2*n-1);
s2 = p1 / p2;
s1 = s1 + s2;                                  // s1+ =
s2;
n ++; }
cout<<precision(3);
cout << “s1=”<<s1 <<” n=” << n << endl;}
```

Ko‘rsatgichli massivlar bilan ishlash. Statik massivlarning kamchiliklari shundaki, ularning o‘lchamlari oldindan ma’lum bo‘lishi kerak, bundan tashqari bu o‘lchamlar ma’lumotlarga ajratilgan xotira segmentining o‘lchami bilan chegaralangan. Ikkinchi tomondan, yetarlicha katta o‘lchamdagi massiv e’lon qilib, aniq masala yechilishida ajratilgan xotira to‘liq ishlatilmasligi mumkin. Bu kamchiliklar dinamik massivlardan foydalanish orqali bartaraf etiladi, chunki ular dastur ishlashi jarayonida kerak bo‘lgan o‘lchamdagi massivlarni yaratish va zarurat qolmaganda yo‘qotish imkoniyatini beradi.

Dinamik massivlarga xotira ajratish uchun malloc(), calloc() funksiyalaridan yoki new operatoridan foydalanish mumkin. Dina-mik obyektga ajratilgan xotirani bo‘shatish uchun free() funksiyasi yoki delete operatori ishlatiladi.

Yuqorida qayd qilingan funksiyalar «alloc.h» kutubxonasida joylashgan. malloc() funksiyasining sintaksisi **void * malloc(size_t size);** ko‘rinishida bo‘lib, u xotiraning uyum qismidan size bayt o‘lchamidagi uzluksiz sohani ajratadi. Agar xotira ajratish muvaffaqiyatli bo‘lsa, malloc() funksiyasi ajratilgan sohaning boshlanish adresini qaytaradi. Talab qilingan xotirani ajratish muvaffaqiyatsiz bo‘lsa, funksiya NULL qiymatini qaytaradi.

Sintaksisdan ko‘rinib turibdiki, funksiya void turidagi qiymat qaytaradi. Amalda esa aniq tipdagi obyekt uchun xotira ajratish zarur bo‘ladi. Buning uchun void tipini aniq tipga keltirish texnologiyasidan foydalaniladi. Masalan, butun turdagi uzunligi 3 ga teng massivga joy ajratishni quyidagicha amalga oshirish mumkin:

int * pInt=(int*)malloc(3*sizeof(int));

calloc() funksiyasi malloc() funksiyasidan farqli ravishda massiv uchun joy ajratishdan tashqari massiv elementlarini 0 qiymati bilan initsializatsiya qiladi. Bu funksiya sintaksisi

void * calloc(size_t num, size_t size);

ko‘rinishda bo‘lib, num parametri ajratilgan sohada nechta element borligini, size har bir element o‘lchamini bildiradi.

free() xotirani bo‘shatish funksiyasi o‘chiriladigan xotira bo‘lagiga ko‘rsatkich bo‘lgan yagona parametrga ega bo‘ladi:

void free(void * block);

free() funksiyasi parametrining void tipida bo‘lishi ixtiyoriy turdagi xotira bo‘lagini o‘chirish imkonini beradi.

Quyidagi dasturda 10 ta butun sondan iborat dinamik massiv yaratish, unga qiymat berish va o‘chirish amallari bajarilgan.

```
#include <iostream.h>
#include <alloc.h>
int main(){
 int * pVector;
 if
((pVector=(int*)malloc(10*sizeof(int)))==NULL)  {
  cout<<”Xotira yetarli emas!!!”;
  return 1;  }
 // ajratilgan xotira sohasini to‘ldirish
 for(int i=0;i<10;i++) *(pVector+i)=i;
 // vektor elementlarini chop etish
 for(int i=0; i<10; i++)
cout<<*(pVector+i)<<endl;
 // ajratilgan xotira bo‘lagini qaytarish
(o‘chirish)
 free(pVector);
 return 0; }
```

Keyingi dasturda $n \times n$ o‘lchamli haqiqiy sonlar massivining bosh diagonalidan yuqorida joylashgan elementlar yig‘indisini hisoblash masalasi yechilgan.

```
#include <iostream.h>
#include <alloc.h>
int main(){
 int n;
 float * pMatr, s=0;
 cout<<”A(n,n): n=”;
```

```
   cin>>n;

  if((pMatr=(float*)malloc(n*n*sizeof(float)))==NULL)
  {
    cout<<”Xotira yetarli emas!!!”;
    return 1;  }
   for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)cin>>*(pMatr+i*n+j);
   for(int i=0;i<n;i++)
    for(int j=i+1;j<n;j++)s+=*(pMatr+i*n+j);
   cout<<”Matritsa bosh diagonalidan yuqoridagi ”;
   cout<<”elementlar yig‘indisi S=”<<s<<endl;
   return 0;}
```

new operatori yordamida, massivga xotira ajratishda obyekt tipidan keyin kvadrat qavs ichida obyektlar soni ko‘rsatiladi. Masalan, butun tipdagi 10 ta sondan iborat massivga joy ajratish uchun

pVector=new int[10];

ifodasi yozilishi kerak. Bunga qarama-qarshi ravishda, bu usulda ajratilgan xotirani bo‘shatish uchun

delete [] pVector;

ko‘rsatmasini berish kerak bo‘ladi.

Ikki o‘lchamli dinamik massivni tashkil qilish uchun

int **a;

ko‘rinishidagi «ko‘rsatkichga ko‘rsatkich» ishlatiladi.

Boshda massiv satrlari soniga qarab ko‘rsatkichlar massiviga dinamik xotiradan joy ajratish kerak:

a=new int *[m] // bu yerda m massiv satrlari soni

Keyin, har bir satr uchun takrorlash operatori yordamida xotira ajratish va ularning boshlang‘ich adreslarini a massiv elementlariga joylashtirish zarur bo‘ladi:

for(int i=0;i<m;i++)a[i]=new int[n];//n ustunlar soni

Shuni qayd etish kerakki, dinamik massivning har bir satri xotiraning turli joylarida joylashishi mumkin.

Ikki o‘lchamli massivni o‘chirishda oldin massivning har bir elementi (satri), so‘ngra massivning o‘zi yo‘qotiladi:

for(i=0;i<m;i++) delete[]a[i];

delete[]a;

Matritsani vektorga ko‘paytirish masalasi uchun dinamik massivlardan foydalanishga misol:

```
int main (){
 int n,m;
 int i,j; float s;
cout<<"\n n="; cin>>n; // matritsa satrlari
soni
cout<<"\n m="; cin>>m; //matritsa ustunlari
soni
 float *b=new float[m];
 float *c=new float[n];
// ko‘rsatkichlar massiviga xotira ajratish
 float **a=new float *[n] ;
 for(i=0;i<n;i++)  // har bir satr uchun
a[i]=new float[m];    //dinamik xotira
ajratish
 for(j=0;j<m;j++)cin>>b[j];
 for(i=0;i<n;i++)
  for(j=0;j<m;j++)cin>>a[i][j];
 for(i=0;i<n;i++) {
  for(j=0,s=0;j<m;j++)s+=a[i,j]*b[j];
  c[i]=s; }
 for(i=0;i<n;i++)cout<<"\t
c["<<i<<"]="<<c[i];
 delete[]b;
 delete[]c;
 for (i=0;i<n;i++) delete[]a[i];
 delete[]a;
 return;}
```

Funksiya va massivlar. Funksiyalar massivni parametr sifatida ishlatishi va uni funksiyaning natijasi sifatida qaytarishi mumkin.

Agar massiv parametr orqali funksiyaga uzatilsa, elementlar sonini aniqlash muammosi tug‘iladi, chunki massiv nomidan uning uzunligini aniqlashning iloji yo‘q. Ayrim hollarda, masalan, belgilar massivi sifatida aniqlangan satr (ASCII Z satrlar) bilan ishlaganda massiv uzunligini aniqlash mumkin, chunki satrlar ‘\0‘ belgisi bilan tugaydi.

Misol uchun:

```
#include <iostream.h>
int len(char s[])   //massivni parametr sifatida
ishlatish
```

```
{
 int m=0;
 while(s[m++]);
 return m-1; }
int main (){
 char z[]=”Ushbu satr uzunligi = ”;
 cout<<z<<len(z); }
```

Funksiya parametri satr bo‘lmagan hollarda fiksirlangan uzunlikdagi massivlar ishlatiladi. Agar turli uzunlikdagi massivlarni uzatish zarur bo‘lsa, massiv o‘lchamlarini parametr sifatida uzatish mumkin yoki bu maqsadda global o‘zgaruvchidan foydalanishga to‘g‘ri keladi.

Misol:

```
#include <iostream.h>
float sum(int n,float *x) //bu ikkinchi usul
{
 float s=0;
 for (int i=0;i<n;i++)s+=x[i];
 return s;}
int main(){
 float E[]={1.2,2.0,3.0,4.5,-4.0};
 cout<<sum(5,E);}
```

Massiv nomi ko‘rsatkich bo‘lganligi sababli massiv elementlarini funksiyada o‘zgartirish mumkin va bu o‘zgartirishlar funksiyadan chiqqandan keyin ham saqlanib qoladi.

```
#include <iostream.h>
void vector_01(int n,int*x,int * y) {
//bu ikkinchi usul
 for (int i=0;i<n;i++)
  y[i]=x[i]>0?1:0; }
int main(){
 int a[]=(1,2,-4,3,-5,0,4};
 int c[7];
 vector_01(7,a,c);
 for(int i=0;i<7;i++) cout<<‘\t‘<<c[i]; }
```

Masala. Butun tipdagi va elementlari kamaymaydigan holda tartiblangan bir o‘lchamli ikkita massivlarni yagona massivga, tartiblanish saqlangan holda birlashtirish amalga oshirilsin.

Dastur matni:

```
#include <iostream.h>
\\butun turdagi massivga ko‘rsatkich
qaytaradigan. funksiya
int * massiv_ulash(int,int*,int,int*);
     int main(){
 int c[]={-1,2,5,10},d[]={1,7,8};
 int * h;
 h=massiv_ulash(5,c,3,d);
for(int i=0;i<8;i++) cout<<‘\t‘<<h[i];
 delete[]h; }
     int * massiv_ulash(int n,int *a ,int m,int
*b){
 int * x=new int[n+m];
 int ia=0,ib=0,ix=0;
 while (ia<n && ib<m)
  a[ia]>b[ib]?x[ix++]=b[ib++]:x[ix++]=a[ia++];
 while(ib<m)x[ix++]=b[ib++];
 while(ia<n)x[ix++]=a[ia++];
 return x; }
```

Ko‘p o‘lchamli massivlar bilan ishlash ma’lum bir murakkablikka ega, chunki massivlar xotirada joylash tartibi turli variantda bo‘lishi mumkin. Masalan, funksiya parametrlar ro‘yxatida $n \times n$ o‘lchamdagi haqiqiy tipdagi x[n][n] massivga mos keluvchi parametrni

float sum(float x[n][n])

ko‘rinishda yozib bo‘lmaydi. Muammo yechimi - bu massiv o‘lchamini parametr sifatida uzatish va funksiya sarlavhasini quyidagicha yozish kerak:

float sum(int n,float x[][]);

Ko‘p o‘lchamli massivlarni parametr sifatida ishlatishda bir nechta usullardan foydalanish mumkin.

1-usul. Massivning ikkinchi o‘lchamini o‘zgarmas ifoda (son) bilan ko‘rsatish:

```
float sum(int n,float x[][10])
{float s=0.0;
 for(int i=0;i<n;i++)
 for(int j=0;j<n;j++)
 s+=x[i][j];
 return s;}
```

2-usul. Ikki o‘lchamli massiv ko‘rsatkichlar massivi ko‘rinishida aniqlangan holatlar uchun ko‘rsatkichlar massivini (matritsa satrlar adreslarini) berish orqali:

```
float sum(int n,float *p[]){
 float s=0.0;
 for(int i=0;i<n;i++)
 for(int j=0;j<n;j++)
 s+=p[i][j];\\”*p[i][j]” emas,chunki massivga
murojaat
 return s; }
      int main()  {
      float x[][4]={{11,-12,13,14},{21,22,23,24},
       {31,32,33,34},{41,42,43,44}};
 float *ptr[4];
 for(int i=0;i<4;i++) ptr[i]=(float *)&x[i];
 cout<<sum(4,ptr)<<endl; }
3-usul. Ko‘rsatkichlarga ko‘rsatkich ko‘rinishida
aniqlangan dinamik massivlarni ishlatish bilan:
float sum(int n,float **x) {
 float s=0.0;
 for(int i=0;i<n;i++)for(int
j=0;j<n;j++)s+=x[i][j];    return s;
}
    int main(){
     float **ptr;
     int n;
     cin>>n;
     ptr=new float *[n];
for(int i=0;i<n;i++){
      ptr[i]=new float [n];
      for(int j=0;j<n;j++)
       ptr[i][j]=(float)((i+1)*10+j);
     }
     cout<<sum(n,ptr);
for(int i=0; i<n;i++) delete ptr[i];
delete[]ptr;}
```

Navbatdagi dasturda funksiya tomonidan natija sifatida ikki o‘lchamli massivni qaytarishiga misol keltirilgan. Massiv elementlarning qiymatlari tasodifiy sonlardan tashkil topadi. Tasoddifiy sonlar «math.h» kutubxonasidagi random() funksiya yordamida hosil qilinadi:

```
#include <iostream.h>
#include <math.h>
int **rmatr(int n,int m){
 int ** ptr;
       ptr=new int *[n];
       for(int i=0;i<n;i++)  {
       ptr[i]=new int[m];
for(int j=0;j<m;j++) ptr[i][j]=random(100); }
       return ptr;}
int sum(int n,int m,int **ix){
 float s=0;
 for(int i=0;i<n;i++)
 for(int j=0;j<m;j++) s+=ix[i][j];
 return s; }
int main(){
 int n,m;
       cin>>n>>m;
       int **matr;
       randomize();
       matr=rmatr(n,m);
       for(int i=0;i<n;i++) {
  cout<<endl<<i<<" - satr:"
  for (int j=0;j<m;j++) cout<<‘\t‘<<matr[i][j];
       }
 cout<<endl<<"Summa="<<sum(n,m,matr);
       for(int i=0;i<n;i++) delete matr[i];
 delete[]matr;  }
```

Nazariy bilimlarni tekshirish uchun savollar.

1. C++da Massiv qanday ishlaydi?
2. Rekursiv funksiya nima? Rekursiv funksiya parametrlari.
3. Massivlarni qabday turlari mavjud? Massivlar nima uchun qo‘llaniladi? Ikki o‘lchamli massivlar.

4. Bir o‘lchovli massiv nima? Ikki o‘lchovli massiv nima? Statik bir o‘lchovli massivlar.
5. Dinamik bir o‘lchovli massivlar. Statik ko‘p o‘lchovli massivlar. Dinamik ko‘p o‘lchovli massivlar.
6. Statik massivlar indeksli o‘zgaruvchilar. Massiv e’loni sintaksisi.
7. O‘lchami ko‘rsatilgan massiv elementlarini to‘liq initsializatsiyalash.
8. O‘lchami ko‘rsatilgan massiv elementlarini to‘liqmas initsializatsiyalash.
9. Ikki o‘lchamli massivning e’loni sintaksisi. Ikki o‘lchamli massiv elementlarining xotiradagi joylashuvi.
10. Uch o‘lchamli massivning e’loni sintaksisi. Uch o‘lchamli massiv elementlarining xotiradagi joylashuvi.
11. Ko‘rsatkichlar va bir o‘lchovli massivlar. Ko‘rsatkichlar va ikki o‘lchovli massivlar. Ko‘rsatkichlar va uch o‘lchovli massivlar.
12. Satr massivi e’lon qilinishi. Satr terminatori deb nimaga aytiladi? Satr massivi e’lon qilinishida nimalarga e’tibor berish kerak?

3.3. Ko‘rsatkichlar va dinamik xotira bilan ishlash.

📖 *Ko‘rsatgichlar orqali dinamik xotira bilan ishlash, xotiraga qanday murojaat qilish, berilgan o‘zgaruvchining adresi orqali qanday o‘zlashtirish va ular ustida amallar bajarish, ko‘rsatkichlarni e’lon qilishda unga boshlang‘ich qiymatlar berish vazifalari ko‘rib chiqiladi. Ko‘rsatkich tiplari, birorta obyektga, xususan o‘zgaruvchiga ko‘rsatkich, funksiyaga ko‘rsatkich, void ko‘rsatkich, o‘zidan keyin ko‘rsatkich nomi yoziladigan delete operatori yordamida qanday ishlanishi keltirib o‘tilgan.*

✎ ***Kalit so‘zlar**: Xotira, dinamik, ko‘rsatgich, obyekt, void, main, delete, new, operator, funksiya, address, parameter, matritsa, pointer, massiv.*

REJA:

1. Ko‘rsatgichlar. Obyektga ko‘rsatkich. void ko‘rsatkich.
2. Dinamik xotira bilan ishlash. Ko‘rsatkich ustida amallar.
3. Ko‘rsatkichlar va adres oluvchi o‘zgaruvchilar funksiya parametri sifatida.
4. Dinamik massiv va ularni funksiya parametri sifatida qo‘llanilishi.

Ko‘rsatkich – bu kompyuter xotirasi yacheykasining adresi yozilgan o‘zgaruvchidir. Kompyuter xotirasi raqamlangan yacheykalar ketma-ketligidan iboratdir. Har bir o‘zgaruvchining qiymati uning adresi deb ataluvchi alohida xotira yacheykasida saqlanadi.

Dasturdagi o‘zgarmaslar, o‘zgaruvchilar, funksiyalar va sinf obyektlari adreslarini xotiraning alohida joyida saqlash va ular ustida amallar bajarish mumkin.

Ko‘rsatkich uch xil turda bo‘lishi mumkin:

- birorta obyektga, xususan o‘zgaruvchiga ko‘rsatkich;
- funksiyaga ko‘rsatkich;
- void ko‘rsatkich.

Ko‘rsatkichning bu xususiyatlari uning qabul qilishi mumkin bo‘lgan qiymatlari bilan farqlanadi. Ko‘rsatkich albatta birorta tipga bog‘langan bo‘lishi kerak, ya’ni u ko‘rsatilgan adresda qandaydir qiymat joylanishi mumkin va bu qiymatning xotirada qancha joy egallashi oldindan ma’lum bo‘lishi shart.

Ko‘rsatgichlar

E’lon:

```
char *p;   //ixtiyoriy simvol yoki satrni adresi
int *pI;   // Butun sonni adresi
float *pF; // Xaqiqiy sonni adresi
```

Butun o‘zgaruvchilar va massivlar:

```
int n = 6, A[5] = {0, 1, 2, 3, 4};
int *p;     // Butun songa ko‘rsatgich
p = &n;     // n manzilini yozish
*p = 20;    // n = 20
p = A + 2;  // A[2] (&A[2]) adresni yozish
*p = 99;    // A[2] o‘zgartirish
p ++;       // A[3] ga o‘tish
printf(“Adres: %p, qiymat %d", p, *p);
```

Obyektga ko‘rsatkich. Agar bir tipda bir nechta ko‘rsatkichlar e’lon qilinadigan bo‘lsa, har bir ko‘rsatkich uchun **‘*‘** belgisi qo‘yilishi shart:

```
int *i, j,*k;
float x,*y,*z;
```

Keltirilgan misolda **i** va **k** - butun tipdagi ko‘rsatkichlar va j - butun tipdagi o‘zgaruvchi, ikkinchi operatorda **x** - haqiqiy o‘zgaruvchi va **y, z** - haqiqiy tipdagi ko‘rsatkichlar e’lon qilingan.

void ko‘rsatkich. Bu ko‘rsatkich obyekt tipi oldindan noma’lum bo‘lganda ishlatiladi. void ko‘rsatkichining muhim afzalliklaridan biri - unga har qanday tipdagi ko‘rsatkich qiymatini yuklash mumkinligidir. void ko‘rsatkich adresidagi qiymatni ishlatishdan oldin, uni aniq bir tipga oshkor ravishda keltirish kerak bo‘ladi. void ko‘rsatkichni e’lon qilish quyidagicha bo‘ladi:

void ko‘rsatkich

- **int i;** // butun o‘zgaruvchi
- **const int ci=1;** // butun o‘zgarmas
- **int * pi;** // butun o‘zgaruvchiga ko‘rsatkich
- **const int *pci;** *// butun o‘zgarmasga ko‘rsatkich*
- **nt *const cp=&i;** //butun o‘zgaruvchiga o‘zgarmas ko‘rsatkich
- **const int*const cpc=&ci;** *// butun o‘zgarmasga o‘zgarmas ko‘rsatkich*

Dinamik xotira bilan ishlash. Ko‘rsatkichlar ko‘pincha **dinamik xotira** (boshqacha nomi «uyum» yoki «heap») bilan bog‘liq holda ishlatiladi. **Xotiraning dinamik deyilishiga sabab**, bu sohadagi bo‘sh xotira dastur ishlash jarayonida, kerakli paytida ajratib olinadi va zarurat qolmaganida qaytariladi (bo‘shatiladi).

Dinamik xotiraga faqat ko‘rsatkichlar yordamida murojaat qilish mumkin. Bunday o‘zgaruvchilar *dinamik o‘zgaruvchilar* deyiladi va ularni yashash vaqti yaratilgan nuqtadan boshlab dastur oxirigacha yoki oshkor ravishda yo‘qotilgan (bog‘langan xotira bo‘shatilgan) joygacha bo‘ladi.

Ko‘rsatgichlarga dastlabki qiymat kiritish. Ko‘rsatkichlarni e’lon qilishda unga boshlang‘ich qiymatlar berish mumkin. Boshlang‘ich qiymat (initsializator) ko‘rsatkich nomidan so‘ng yoki qavs ichida yoki ‘=‘ belgidan keyin beriladi. Boshlang‘ich qiymatlar quyidagi usullar bilan berilishi mumkin:

- *Ko‘rsatkichga mavjud bo‘lgan obyektning adresini berish:*
- *Oshkor ravishda xotiraning absolyut adresini berish:*
- *Bo‘sh qiymat berish:*
- *Dinamik xotirada new amali bilan joy ajratish va uni adresini ko‘rsatkichga berish:*

Ko‘rsatkichning adreslarni saqlash vositasi sifatida qo‘llanilishi.

Ko‘rsatkichga mavjud bo‘lgan obyektning adresini berish:

a) adresni olish amal orqali:

```
int i=5,k=4;      // butun o‘zgaruvchilar
int *p=&i;        // p ko‘rsatkichga i o‘zgaruvchining
```

// adresi yoziladi
*int *p1(&k); // p1 ko‘rsatkichga k o‘zgaruvchining*
// adresi yoziladi
b) boshqa initsializatsiyalangan ko‘rsatkich qiymatini berish:
*int * r=p; // p oldin e’lon qilingan va qiymatga ega*
// bo‘lgan ko‘rsatkich
v) massiv yoki funksiya nomini berish:
int b[10]; *// massivni e’lon qilish*
int *t=b; *// massivning boshlang‘ich adresini berish*
void f(int a){/* … */} *// funksiyani aniqlash*
void (*pf)(int); *// funksiyaga ko‘rsatkichni e’lon qilish*
pf=f; *// funksiya adresini ko‘rsatkichga berish*

- **Oshkor ravishda xotiraning absolyut adresini berish:**

char *vp = (char *)0xB8000000;

Bunda **0xB8000000** - o‘n oltilik o‘zgarmas son va (char*) - tipiga keltirish amali bo‘lib, u **vp** o‘zgaruvchisini xotiraning absolyut adresidagi baytlarni **char** sifatida qayta ishlovchi ko‘rsatkich tipiga aylantirilishini anglatadi.

- **Bo‘sh qiymat berish:**

int *suxx=NULL;
int *r=0;

Birinchi satrda maxsus **NULL** o‘zgarmasi ishlatilgan, ikkinchi satrda **0** qiymat ishlatilgan. Ikkala holda ham ko‘rsatkich hech qanday obyektga murojaat qilmaydi. Bo‘sh ko‘rsatkich asosan ko‘rsatkichni aniq bir obyektni ko‘rsatayotgan yoki yo‘qligini aniqlash uchun ishlatiladi.

New operatori. Xotiraning obyektlar o‘rtasidan dinamik taqsimlanuvchi sohasidan joy ajratish uchun new operatori ishlatiladi. new operatoridan keyin xotiraga joylashtiriladigan obyekt tipini ko‘rsatish lozim. Bu obyektni saqlash uchun talab etiladigan xotira sohasi o‘lchovini aniqlash uchun kerak bo‘ladi. Masalan, new unsigned short int deb yozish orqali biz dinamik taqsimlanuvchi xotiradan ikki bayt joy ajratamiz. Xuddi shuningdek, new long satri orqali to‘rt bayt joy obyektlar o‘trasida dinamik taqsimlanuvchi sohadan ajratiladi.

new operatori natija sifatida belgilangan xotira yacheykasining adresini qaytaradi. Bu addres ko‘rsatkichga o‘zlashtirilishi lozim. Masalan, unsigned short tipidagi o‘zgaruvchi uchun dinamik sohadan joy ajratish uchun quyidagi dastur kodi yoziladi:

```
unsigned short int *pPointer;
```

```
pPointer = new unsigned short int;
```

yoki xuddi shu amalni bitta satrda ham yozish mumkin.

```
unsigned short int *pPoiner=new unsigned short int;
```

Ikkala holatda ham pPointer ko‘rsatkichi unsigned short int tipidagi qiymatni saqlovchi dinamik soha xotirasining yacheykasini ko‘rsatib turadi. Endi pPointer ko‘rsatkichini shu tipdagi ixtiyoriy o‘zgaruvchiga ko‘rsatkich sifatida qo‘llash mumkin. Ajratilgan xotira sohasiga biror bir qiymat joylashtirish uchun quyidagicha yozuv yoziladi:

```
* pPointer = 72 ;
```

Bu satr quyidagi ma’noni anglatadi: «pPointer ko‘rsatkichida adresi saqlanayotgan xotiraga 72 sonini yozing». Dinamik xotira sohasi albatta chegaralangan bo‘ladi. U to‘lib qolganda new operatori orqali xotiradan joy ajratishga urinsak xatolik yuz beradi.

Delete operatori. Agarda o‘zgaruvchi uchun ajratilgan xotira kerak bo‘lmasa uni bo‘shatish zarur. Bu o‘zidan keyin ko‘rsatkich nomi yoziladigan delete operatori yordamida amalga oshiriladi. delete operatori ko‘rsatkich orqali aniqlangan xotira sohasini bo‘shatadi. Shuni esda saqlash lozimki, dinamik xotira sohasidagi adresni o‘zida saqlovchi ko‘rsatkich lokal o‘zgaruvchi bo‘lishi mumkin. Shuning uchun bu ko‘rsatkich e’lon qilingan funksiyadan chiqishimiz bilan ko‘rsatkich ham xotiradan o‘chiriladi. Lekin new operatori orqali bu ko‘rsatkichga dinamik xotiradan ajratilgan joy bo‘shatilmaydi. Natijada xotiraning bu qismi kirishga imkonsiz bo‘lib qoladi. Dasturchilar bu holatni xotiraning sirqib ketishi, yoki yo‘qolishi (утечка памяти) deb tavsiflaydilar. Bu tavsif haqiqatga butunlay mos keladi, chunki dastur ishini yakunlaguncha xotirani bu qismidan foydalanib bo‘lmaydi.

Xotirani ajratilgan qismini bo‘shatish uchun delete kalit so‘zidan foydalaniladi. Masalan:

```
delete pPointer;
```

Bunda ko‘rsatkich o‘chirilmaydi, balki unda saqlanayotgan adresdagi xotira sohasi bo‘shatiladi. Belgilangan xotirani bo‘shatilishi ko‘rsatkichga ta’sir qilmaydi, unga boshqa adresni o‘zlashtirish ham mumkin.

Dinamik xotirada new amali bilan joy ajratish va uni adresini ko‘rsatkichga berish:

```
int * n=new int;  // birinchi operator
int * m=new int(10);  // ikkinchi operator
int * q=new int[5];   // uchinchi operator
```

Birinchi operatorda new amali yordamida dinamik xotirada **int** uchun yetarli joy ajratib olinib, uning adresi **n** koʻrsatkichga yuklanadi. Koʻrsatkichning oʻzi uchun joy kompilyasiya vaqtida ajratiladi.

delete amali

- Ikkinchi operatorda joy ajratishdan tashqari **m** adresiga boshlangʻich qiymat - **10** sonini joylashtiradi.
- Uchinchi operatorda **int** tipidagi **5 ta element** uchun joy ajratilgan va uning boshlangʻich adresi **q** koʻrsatkichga berilyapti.
- Xotira **new** amali bilan ajratilgan boʻlsa, u delete amali bilan boʻshatilishi kerak. Yuqoridagi **dinamik oʻzgaruvchilar** bilan bogʻlangan xotira quyidagicha boʻshatiladi: **delete n; delete m; delete[]q;**
- Agarda xotira **new[]** amali bilan ajratilgan boʻlsa, uni boʻshatish uchun **delete []** amalini oʻlchovi koʻrsatilmagan holda qoʻllash kerak.
- Xotira boʻshatilganligiga qaramasdan koʻrsatkichni oʻzini keyinchalik qayta ishlatish mumkin.

Koʻrsatkich ustida amallar. Koʻrsatkich ustida quyidagi amallar bajarilishi mumkin:

- obyektga vositali murojaat qilish amali;
- qiymat berish amali;
- koʻrsatkichga oʻzgarmas qiymatni qoʻshish amali;
- ayirish amali;
- inkrement va dekrement amallari;
- solishtirish amali;
- tipga keltirish amali.

Vositali murojaat qilish. Vositali murojaat qilish amali koʻrsatkichdagi adres boʻyicha joylashgan qiymatni olish yoki qiymat berish uchun ishlatiladi:

```
char a;   // char tipidagi oʻzgaruvchi eʼloni.
     char *p=new char;   // Koʻrsatkichni eʼlon
qilib, unga
        // dinamik xotiradan ajratilgan
        // xotiraning adresini berish
     *p=ʻbʻ;  // p adresiga qiymat joylashtirish
           a=*p; // a oʻzgaruvchisiga p adresidagi
qiymatni berish
```

Shuni qayd etib oʻtish kerakki, xotiraning aniq bir joyidagi adresni bir paytning oʻzida bir nechta va har xil tipdagi koʻrsatkichlarga berish

mumkin va ular orqali murojaat qilinganda ma’lumotlarning har xil tipdagi qiymatlarini olish mumkin:

```
unsigned long int A=0Xcc77ffaa;
  unsigned short int * pint=(unsigned short
int*)&A;
  unsigned char* pchar=(unsigned char*)&A;
cout<<hex<<A<<‘ ‘<<hex<<*pint<<‘
‘<<hex<<(int)*pchar;
Ekranga har xil qiymatlar chop etiladi:
cc77ffaa ffaa aa
```

O‘zgaruvchilar bitta adresda joylashgan holda yaxlit qiymatning turli bo‘laklarini o‘zlashtiradi. Bunda, bir baytdan katta joy egallagan son qiymatining xotirada «**teskari**» joylashishi inobatga olinishi kerak. Agar har xil tipdagi ko‘rsatkichlarga qiymatlar berilsa, albatta tipga keltirish amalidan foydalanish kerak:

```
int n=5;
float x=1.0;
int *pi=&n;
float *px=&x;
void *p;
int *r,*r1;
px=(float *)&n;
p=px;
r=(int *)p;
r1=pi;
```

Arifmetik amallar

-Ko‘rsatkich tipini **void** tipiga keltirish amalda ma’noga ega emas. Xuddi shunday, tiplari bir xil bo‘lgan ko‘rsatkichlar uchun tipni keltirish amalini bajarishga hojat yo‘q.

-Ko‘rsatkich ustida bajariladigan arifmetik amallarda avtomatik ravishda tiplarning o‘lchami hisobga olinadi.

-Arifmetik amallar faqat bir xil tipdagi ko‘rsatkichlar ustida bajariladi va ular asosan, massiv tuzilmalariga ko‘rsatkichlar ustida bajariladi.

-Inkrement amali ko‘rsatkichni massivning keyingi elementiga, dekrement esa aksincha, bitta oldingi elementining adresiga ko‘chiradi. Bunda ko‘rsatkichning qiymati sizeof(<massiv elementining tipi>) qiymatiga o‘zgaradi. Agar ko‘rsatkich k o‘zgarmas qiymatga oshirilsa

yoki kamaytirilsa, uning qiymati k*sizeof(<massiv elementining tipi>) kattalikka oʻzgaradi.

Masalan:

```
short int *p=new short[5];
long * q=new long [5];
p++;  // p qiymati 2 ga oshadi
q++;  // q qiymati 4 ga oshadi
q+=3; // q qiymati 3*4=12 oshadi
```

Koʻrsatkichlarning ayirmasi deb, ular ayirmasining tip oʻlchamiga boʻlinishiga aytiladi. Koʻrsatkichlarni oʻzaro qoʻshish mumkin emas.

Adresni olish amali. Turli kompyuterlarda xotirani adreslash turlicha qoida asosida tashkil etiladi. Koʻp hollarda dasturchilar uchun biror bir oʻzgaruvchini aniq adresini bilish zarur emas. Zarurat tugʻilganda bunday axborotni adres operatori (&) yordamida olish mumkin. Dasturning har bir oʻzgaruvchisi oʻzining adresiga egadir. Bu adresni saqlash uchun esa oʻzgaruvchiga koʻrsatkich e'lon qilish kerak. Adresning oʻzining qiymatini bilish esa unchalik shart emas.

Adresni olish quyidagicha e'lon qilinadi:

<tip> & <nom>;

Bu yerda <**tip**> - adresi olinadigan qiymatning tipi, <nom>- adres oluvchi oʻzgaruvchi nomi. Oʻrtadagi **'&'** belgisiga ***adresni olish amali*** deyiladi.

Bu koʻrinishda e'lon qilingan oʻzgaruvchi shu tipdagi oʻzgaruvchining sinonimi deb qaraladi. Adresni olish amali orqali bitta oʻzgaruvchiga har xil nom bilan murojaat qilish mumkin boʻladi.

Misol: int kol;

```
int & pal=kol;    // pal murojaati, u kol
                                  //
oʻzgaruvchisining alternativ nomi
const char & cr='\n'; // cr - oʻzgarmasga
murojaat
```

Adresni olish amalini ishlatishda qoidalarga rioya qilish

- Adresni olish amalini ishlatishda quyidagi qoidalarga rioya qilish kerak: adres oluvchi oʻzgaruvchi funksiya parametri sifatida ishlatilgan yoki **extern** bilan tavsiflangan yoki sinf maydoniga murojaat qilingan hollardan tashqari barcha holatlarda boshlangʻich qiymatga ega boʻlishi kerak.
- Adresni olish amali asosan funksiyalarda adres orqali uzatiluvchi parametrlar sifatida ishlatiladi.

- Adres oluvchi o‘zgaruvchining ko‘rsatkichdan farqi shundaki, u alohida xotirani egallamaydi va faqat o‘z qiymati bo‘lgan o‘zgaruvchining boshqa nomi sifatida ishlatiladi.

Ko‘rsatkichlar va adres oluvchi o‘zgaruvchilar funksiya parametri sifatida

- Funksiya prototipida yoki aniqlanish sarlavhasida ko‘rsatilgan parametrlar ***formal parametrlar*** deyiladi, funksiya chaqirilishida ko‘rsatilgan argumentlarga ***faktik parametrlar*** deyiladi.
- Funksiya chaqirilishida faktik parametrning tipi mos o‘rindagi formal parametr tipiga to‘g‘ri kelmasa yoki shu tipga keltirishning iloji bo‘lmasa kompilyasiya xatosi ro‘y beradi.
- Faktik parametrlarni funksiyaga ikki xil usul bilan uzatish mumkin: ***qiymati*** yoki ***adresi*** bilan.

Faktik parametrlarni funksiyaga qiymat bilan uzatish. Funksiya chaqirilishida argument qiymat bilan uzatilganda, argument yoki uning o‘rnida kelgan ifoda qiymati va boshqa argumentlarning nusxasi (qiymatlari) stek xotirasiga yoziladi. Funksiya faqat shu nusxalar bilan bajariladi, kerak bo‘lsa bu nusxalarga o‘zgartirishlar qilinishi mumkin, lekin bu o‘zgarishlar argumentning o‘ziga ta’sir qilmaydi, chunki funksiya o‘z ishini tugatishi bilan nusxalar o‘chiriladi (stek tozalanadi).

- Agar parametr adres bilan uzatilsa, stekka adres nusxasi yoziladi va xuddi shu adres bo‘yicha qiymatlar o‘qiladi (yoziladi). Funksiya o‘z ishini tugatgandan keyin shu adres bo‘yicha qilingan o‘zgarishlar saqlanib qolinadi va bu qiymatlarni boshqa funksiyalar ishlatishi mumkin.
- Argument qiymat bilan uzatilishi uchun mos formal parametr sifatida o‘zgaruvchini tipi va nomi yoziladi. Funksiya chaqirilishida mos argument sifatida o‘zgaruvchining nomi yoki ifoda bo‘lishi mumkin.
- Faktik parametr adres bilan uzatilganda unga mos keluvchi formal parametrni ikki xil usul bilan yozish mumkin: ***ko‘rsatkich orqali*** **yoki** ***adresni oluvchi parametrlar orqali*****.**

Misol:

```
#include <iostream>
using namespace std;
void f(int, int*, int &);
int main()
{
```

```
 int i=1, j=2, k=3;   cout<<i<<" "<<j<<"
"<<k<<endl;
 f(i, &j, k);     cout<<i<<" "<<j<<" "<<k;
}
void f(int i, int *j, int &k)
{
 i++;
 (*j)++;
 k++;
 *j=i+k;
 k=*j+i;  }
```

Misol: $ax^2+bx+c=0$ ko‘rinishidagi kvadrat tenglama ildizlarini funksiya parametrlari vositasida olish masalasi

```
#include "iostream"
#include "math.h"
using namespace std;
int kvadrat_ildiz(float a,float b,float c,
float & x1, float & x2){
float d; d=b*b-4*a*c;
if(d<0) return 0;
if(d==0){ x1=x2=-b/(2*a); return 1;
} else { x1=(-b+sqrt (d))/(2*a);
x2=(-b-sqrt(d))/(2*a); return 2;
     }}
int main(){
float a,b,c,d,x1,x2;
cout<<"ax^2+bx+c=0 tenglama ildizini topish. ";
cout<<"\n a-koeffisiyentini kiriting="; cin>>a;
cout<<"\n b-koeffisiyentini kiriting="; cin>>b;
cout<<"\n c-koeffisiyentini kiriting="; cin>>c;
switch (kvadrat_ildiz(a, b,c, x1,  x2))
{
case 0: cout<<" Tenglama haqiqiy ildizga  ega
emas!";break;
case 1: cout<<" Tenglama yagona ildizga ega:";
cout<<"\n x="<<x1;break;
default: cout<<" Tenglama ikkita ildizga ega :
";
```

```
cout<<"\nx1="<<x1; cout<<"\nx2="<<x2;}
return 0;
}
```

Parametrlar soni noma’lum bo‘lgan funksiyalar. Bunday funksiyalar sarlavhasi quyidagi formatda yoziladi:**<funksiya tipi> <funksiya nomi> (<oshkor parametrlar ro‘yxati>, …)**

Bu yerda <**oshkor parametrlar ro‘yxati**> - oshkor ravishda yozilgan parametrlar nomi va tipi. Bu parametrlar ***majburiy parametrlar*** deyiladi. Bunday parametrlardan kamida bittasi bo‘lishi shart. Qolgan parametrlar soni va tipi noma’lum hisoblanadi. Ularni aniqlash va ishlatish to‘la ravishda dastur tuzuvchi zimmasiga yuklanadi.

O‘zgaruvchan sondagi parametrlarni tashkil qilish usuli:

1-usul. Parametrlar ro‘yxati oxirida yana bir maxsus parametr yoziladi va uning qiymati parametrlar tugaganligini bildiradi. Kompilyator tomonidan funksiya tanasida parametrlar birma-bir aniqlashtiriladi. Barcha parametrlar tipi oxirgi maxsus parametr tipi bilan ustma-ust tushadi deb hisobla nadi;

2-usul. Birorta maxsus parametr sifatida noma’lum parametrlar soni kiritiladi va unga qarab parametrlar soni aniqlanadi.

Ikkala usulda ham parametrlarga murojaat qilish uchun ko‘rsatkichlar ishlatiladi.

Misol:

```
#include "iostream"
using namespace std;
float summa (int k,...)
{ float p=0;
int *prt=&k;
if(*prt==0.0) cout<<" 0 ";
for(; *prt; prt++) { p+=*prt; }
return p; }
int main(){
cout<<"\n "<<summa(10,20,30,40,0.0);
cout<<"\n "<<summa(1,2,3,4,0.0);
}
```

Har xil tipdagi parametrlarni ishlatish uchun tipni aniqlaydigan funksiya

```
#include "iostream"
int Summa(char ,int,...);
using namespace std;
int main(){
```

```
cout<<"1="<<Summa(‘i‘,3,10,20,30)<<endl;
cout<<"2="<<Summa(‘f‘,3,10.0,20.0,5.0)<<endl;
cout<<"3="<<Summa(‘d‘,3,10,20,30)<<endl;
}
int Summa(char z, int k,...){
switch(z){
case ‘i‘:{
int * ptr=&k+1;
int s=0;
for(;k--;ptr++) s+=*(ptr);
return(int)s;
}
case ‘f‘:{
float*ptr=(float *)(&k+1); float s=0.0;
for(;k--;ptr++) s+=*(ptr);
return s;
}
default:{
cout<<"\n parametr hato berilgan ";
return 9999999.0;
break;
}}}
```

Dinamik xotira bilan ishlash. Dinamik massiv va ularni funksiya parametri sifatida qo‘llanilishi. Statistik massivlarning kamchiliklari shundaki, ularning o‘lchami oldindan ma’lum bo‘lishi kerak, undan tashqari bu o‘lcham ma’lumotlarga ajratilgan xotira segmentining o‘lchami bilan chegaralangan. Ikkinchi tomondan, yetarlicha katta o‘lchamdagi massiv e’lon qilib, aniq masala yechilishida ajratilgan xotira to‘liq ishlatilmasligi mumkin. Bu kamchiliklar dinamik massivlardan foydalanish orqali bartaraf etiladi, chunki ular dastur ishlashi jarayonida zarur bo‘lganda kerak o‘lchamdagi massivlarni yaratish va zarurat qolmaganda yo‘qotish imkoniyatini beradi.

Funksiyaga ko‘rsatkich dastur joylashgan xotiradagi funksiya kodining boshlang‘ich adresini ko‘rsatadi, ya’ni funksiya chaqirilganda boshqaruv ayni shu adresga uzatiladi. Ko‘rsatkich orqali funksiyani oddiy yoki vositali chaqirish amalga oshirish mumkin. Bunda funksiya uning nomi bo‘yicha emas, balki funksiyaga ko‘rsatuvchi o‘zgaruvchi orqali chaqiriladi. Funksiyani boshqa funksiyaga argument sifatida

uzatish ham funksiya koʻrsatkichi orqali bajariladi. Funksiyaga koʻrsatkichning yozilish sintaksisi quyidagicha:

<tip> (* <nom>) (<parametrlar roʻyxati>);

Bunda <**tip**>- funksiya qaytaruvchi qiymat tipi; ***<nom>** - koʻrsatkich oʻzgaruvchining nomi; <parametrlar roʻyxati> - funksiya parametrlarining yoki ularning tiplarining roʻyxati.

Masalan: int (*fun)(float,float);

Funksiya va massivlar. Koʻp oʻlchamli massivlarni parametr sifatida ishlatishda bir nechta usullardan foydalanish mumkin:

1-usul. Massivning ikkinchi oʻlchamini oʻzgarmas ifoda (son) bilan koʻrsatish:

```
float sum (int n, float  x[][10])
{float s=0.0;
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
s+=x[i][j];
return s;}
```

2-usul. Ikki oʻlchamli massiv koʻrsatkichlar massivi koʻrinishida aniqlangan holatlar uchun koʻrsatkichlar massivini (matritsa satrlar adreslarini) berish orqali:

```
float sum (int n, float  x[][10])
{float s=0.0;
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
s+=x[i][j];
return s;}
int main()
{ float x[][4]={{11,-
12,13,14},{21,22,23,24},{31,32,33,34},{41,42,43
,44}};
float *ptr[4];
for(int i=0;i<4;i++)
ptr[i]=(float*)&x[i];
cout<<sum(4,ptr)<<endl;}
```

3-usul. Koʻrsatkichlarga koʻrsatkich koʻrinishida aniqlangan dinamik massivlarni ishlatish bilan:

```
float sum(int n,float **x)
{ float s=0.0;
for(int i=0;i<n;i++)
```

```
for(int j=0;j<n;j++)
s+=x[i][j];
return s;}
int main()
{float **ptr;
int n;
cin>>n;
ptr=new float * [n];
for(int i=0;i<n;i++)
{
ptr[i]=new float [n];
for(int j=0;j<n;j++)
ptr[i][j]=(float)((i+1)*10+j;
}
cout<<sum(n,ptr);
for(int i=0;i<n;i++)
delete ptr[i];
delete []ptr;}
```

Dinamik matritsalar. Masala: Matritsani o‘lchamini kiriting va unga dastur davomida xotiradan joy ajrating.

Muammo : Matritsa o‘lchami oldindan ma’lum emas.

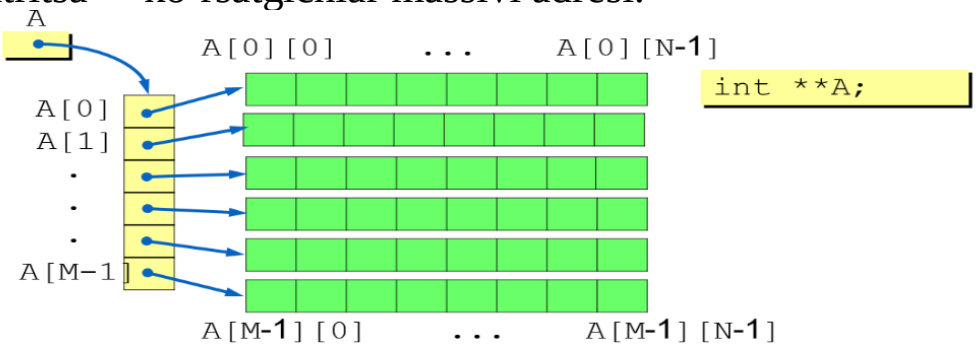
Yechish usullari:

- Har bir satr uchun aloxida xotira blokini ajratish;
- Butun matritsa uchun bir yo‘la xotira ajratish.

Har bir satr uchun xotira bloki

Matritsa adresi:

- Matritsa = satr massivi;
- Matritsa adresi = massiv adresi, bunda satrlar adresi saqlanadi;
- Satr adresi = ko‘rsatgich;
- Matritsa = ko‘rsatgichlar massivi adresi.



typedef int *pInt;

```
int main()
{
int M, N, i;
pInt *A;
A = new pInt[M];// Ko‘rsatgichlar massivini
ajratish
for ( i = 0; i < M; i ++ )
A[i] = new int[N];// Massivning har- bir satr
uchun
for ( i = 0; i < M; i ++ )
delete A[i];// Har – bir satr o‘chiriladi
delete A; // Massiv o‘chiriladi
}
```

Nazariy bilimlarni tekshirish uchun savollar.

1. Ko‘rsatkich nima? Ko‘rsatkichlar qanday e’lon qilinadi va qo‘llaniladi?
2. Obyektga ko‘rsatkich nima vazifani bajaradi? void ko‘rsatkich nima vazifani bajaradi?
3. Dinamik xotira bilan qanday ishlanadi? Dinamik massiv va ularni funksiya parametri sifatida qo‘llanilishi.
4. Xotiraning obyektlar o‘rtasidan dinamik taqsimlanuvchi sohasidan joy ajratish uchun qanday operator ishlatiladi?
5. new unsigned short int deb yozish orqali biz dinamik taqsimlanuvchi xotiradan qancha joy ajratamiz?
6. Dinamik xotirada new amali bilan joy ajratish va uni adresini ko‘rsatkichga berish qanday amalga oshiriladi?
7. Xotirani ajratilgan qismini bo‘shatish uchun qaysi operatordan foydalanish mumkin?
8. Ko‘rsatkichdagi adres bo‘yicha joylashgan qiymatni olish yoki qiymat berish uchun qanday amal ishlatiladi?
9. C++ dasturlash tilida cout<<&b; qatorining vazifasi nima? Kompyuter xotirasi yacheykasining adresi yozilgan o‘zgaruvchi bu..?
10. Funksiya prototipida yoki aniqlanish sarlavhasida ko‘rsatilgan parametrlar qanday parametrlar deyiladi?

4-BOB. OBYEKTGA YO‘NALTIRILGAN DASTURLASH ASOSLARI

4.1.obyektga yo‘naltirilgan dasturlash asoslari

📖 *Obyektga yo‘naltirilgan dasturlash tamoyillari (inkapsulyatsiya, polimorfizm, vorislik).*

🖉 ***Kalit so‘zlar:*** *inkapsulyatsiya, sinf, OYD, konstruktor va destruktor, ro‘yxat, manzil, tugun, adres olish &, bo‘shatish, ko‘rsatkich.*

REJA:

1. Obyektga yo‘naltirilgan dasturlash asoslari (Inkapsulyasiya, vorislik, polimorfizm)
2. Sinflar va obyektlar;
3. Tuzilma va birlashmalar;
4. Nusxalash konstruktori.

Obyektga yo‘naltirilgan dasturlash (OYD) – bu dasturlashga yangi bir yondashuvdir. Hisoblash texnikasining rivojlanishi va yechilayotgan masalalarni tobora murakkablashuvi dasturlashning turli modellarini (paradigmalarini) yuzaga kelishiga sabab bo‘lmoqda. Birinchi kompilyatorlarda (masalan, FORTRAN tili) dasturlashning funksiyalardan foydalanishga asoslangan protsedura modelini qo‘llab quvvatlagan. Bu model yordamida dastur tuzuvchi bir necha ming satrli dasturlarni yozishi mumkin edi. Rivojlanishning keyingi bosqichida dasturlarning strukturali modeli paydo bo‘ldi va ALGOL, Pascal va C tillari kompilyatorlarida o‘z aksini topdi. Strukturali dasturlashning mohiyati – dasturni o‘zaro bog‘langan protseduralar (bloklar) va ular qayta ishlaydigan ma’lumotlarning majmuasi deb qarashdan iborat. Ushbu model dastur bloklarini keng qo‘llashga, GOTO operatoridan imkon qadar kam foydalanishga tayangan va unda dastur tuzuvchi o‘n ming satrdan ortiq dasturlarni yarata olgan. Yaratilgan dasturni protsedurali modelga nisbatan sozlash va nazorat qilish oson kechgan.

Murakkab masalalarni yechish uchun dasturlashning yangi uslubiga zarurat paydo bo‘ldiki, u OYD modelida amalga oshirildi. OYD modeli bir nechta asosiy konsepsiyalarga asoslanadi.

Ma’lumotlarni abstraksiyalash – ma’lumotlarni yangi tipini yaratish imkoniyati bo‘lib, bu tiplar bilan xuddi ma’lumotlarning asosiy tiplari bilan ishlagandek ishlash mumkin. Odatda yangi tip ma’lumotlarning abstrakt tipi deyiladi, garchi ularni soddaroq qilib “foydalanuvchi tomonidan aniqlangan tip” deb atash mumkin.

Inkapsulyasiya – bu ma’lumotlar va ularni qayta ishlovchi kodni birlashtirish mexanizmidir. Inkapsulyasiya ma’lumotlar va kodni tashqi ta’sirdan saqlash imkonini beradi.

Obyektga yo‘naltirilgan dasturlash tushunchasi. Yuqoridagi ikkita konsepsiyani amalga oshirish uchun C++ tilida *sinflar* ishlatiladi. *Class* atamasi bilan obyektlar tipi aniqlanadi. Sinfning har bir vakili (nusxasi) *obyekt* deb nomlanadi. Har bir obyekt o‘zining alohida holatiga ega bo‘ladi. Obyekt holati uning *ma’lumotlar-a’zolarning* ayni paytdagi qiymati bilan aniqlanadi. Sinf vazifasi uning *funksiya-a’zolarining* sinf obyektlari ustida bajaradigan 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 ma’lumotlariga murojaatni cheklab qo‘yishi mumkin. Cheklov ma’lumotlarni ochiq (public), yopiq (private) va himoyalangan (protected) deb aniqlash bilan tayinlanadi.

Sinf, shu tipdagi obyektning tashqi dunyo bilan o‘zaro bog‘lanishi uchun qat’iy muloqot shartlarini aniqlaydi. Yopiq ma’lumotlarga yoki kodga faqat shu obyekt ichida murojaat qilish mumkin. Boshqa tomondan, ochiq ma’lumotlarga va kodlarga, garchi ular obyekt ichida aniqlangan bo‘lsa ham, dasturning ixtiyoriy joyidan murojaat qilish mumkin va ular obyektni tashqi olam bilan muloqot yaratishiga xizmat qiladi. Yaratilgan obyektlarni, ularni funksiya–a’zolariga oddiygina murojaat orqali amalga oshiriluvchi *xabarlar* (yoki *so‘rovlar*) yordamida boshqarish mumkin. Keyinchalik Windows xabarlari bilan adashtirmaslik uchun so‘rov atamasi 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 (*sinf–avlod*) o‘zining ona sinfidan (*sinf-ajdod*) ma’lumotlar va funksiyalarni vorislik bo‘yicha oladi, hamda ular satriga faqat o‘ziga xos bo‘lgan qirralarni amalga oshirishga imkon beruvchi ma’lumot va funksiyalarni qo‘shadi. Ajdod sinfdagi himoyalangan berilgan-a’zolarga va funksiya-a’zolarga 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. *Asosiy sinf* atamasi sinflar shajarasidagi ona sinf sinonimi sifatida ishlatiladi. Agar obyekt o‘z atributlarini

(ma’lumot-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 paytida yuzaga keladigan holatga bog‘liq ravishda uning turli xil amallarni bajarish xususiyatidir. Polimorfizm – bu faqat obyektlar xususiyati bo‘lmasdan, balki funksiya-a’zolar xususiyatidir va ular xususan, bitta nomdagi funksiya-a’zoni, har xil tipdagi argumentlarga ega va bajaridagan amali unga uzatiladigan argumentlar tipiga bog‘liq bo‘lgan funksiyalardan foydalanish imkoniyatini beradi. Bu holatga *funksiyalarni qayta yuklash* deyiladi. Polimorfizm amallarga ham qo‘llanishi mumkin, ya’ni amal mazmuni (natijasi) operand (ma’lumot) tipiga bog‘liq bo‘ladi. Polimorfizmning bunday tipiga *amallarni qayta yuklash* deyiladi.

Polimorfizmning yana bir ta’rifi quyidagicha: polimorfizm – bu asosiy sinfga ko‘rsatgichlarning (murojaatlarning), ularni virtual funksiyalarini chaqirishdagi turli qiymatlarni qabul qilish imkoniyatidir. C++ tilining bunday imkoniyati *kechiktirilgan bog‘lanish* natijasidir. Kechiktirilgan bog‘lanishda chaqiriladigan funksiya-a’zolar 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‘rinli.

C++ tili obyektga yo‘naltirilgan dasturlash prinsiplarini qo‘llab quvvatlaydi. Bu prinsiplar quyidagilardan iborat:

1. Inkapsulyasiya
2. Vorislik
3. Polimorfizm

Inkapsulyasiya. Agar muxandis ishlab chiqarishda diod, triod yoki rezistorni ishlatsa, u bu elementlarni yangitdan ixtiro qilmaydi, balki do‘kondan sotib oladi [9]. Demak, muxandis ularning qanday tuzilganligiga e’tiborini qaratmaydi, bu elementlar yaxshi ishlasa yetarli. Aynan shu tashqi konstruksiyada ishlaydigan yashirinlik yoki avtonomlik xossasi inkapsulyasiya deyiladi.

Vorislik. Yangi obyekt yaratilayotgan bo‘lsa, ikkita variantdan biri tanlanadi: mutlaqo yangisini yaratish yoki mavjud modelning konstruksiyasini takomillashtirishdir. Ko‘pincha 2-variant tanlanadi, demak, ba’zi xususiyatlari o‘zgartiriladi xolos. Bu narsa vorislik prinsipiga asos soladi. Yangi sinf oldin mavjud bo‘lgan sinfni

kengaytirishdan hosil bo‘ladi. Bunda yangi sinf oldingi sinfning merosxo‘ri deb ataladi.

Polimorfizm. Poli – ko‘p, morfe – shakl degan ma’noni bildiradi. C++ tili bir xil nomdagi funksiya turli obyektlar tomonidan ishlatilganda turli amallarni bajarish imkoniyatini ta’minlaydi. Polimorfizm – shaklning ko‘p xilligidir.

Dasturda ishlatiladigan har bir o‘zgaruvchi o‘z tipiga ega va u quyidagilarni aniqlaydi:

1. Xotiradagi o‘lchovini;
2. Unda saqlanayotgan ma’lumotlarni;
3. Uning yordamida bajarilishi mumkin bo’lgan amallarni.

C++ tilida dasturchi o‘ziga kerakli ixtiyoriy tipni hosil qilishi mumkin. Bu yangi tip ichki tiplarning xossalari va ularning funksional imkoniyatlarini o‘zida ifodalaydi. Yangi sinf e’lon qilish orqali tuziladi. Sinf bu – bir-biri bilan funksional bog‘angan o‘zgaruvchilar va funksiyalar to‘plamidir.

Masalan: Mushuk nomli sinf tuzmoqchimiz. Bu yerda uning yoshi, og‘irligi kabi o‘zgaruvchilar va miyovlash, sichqon tutish kabi funksiyalardan ishdatiladi. Yoki Mashina sinfi g‘ildirak, eshik, o‘rindiq, oyna kabi o‘zgaruvchilar va haydash, to‘xtatish kabi funksiyalardan iborat.

Sinfdagi o‘zgaruvchilar – sinf a’zolari yoki sinf xossalari deyiladi. Sinfdagi funksiyalar odatda o‘zgaruvchilar ustida biror bir amal bajaradi. Ularni sinf funksiyalari (metodlari) deb ham ataladi.

Sinfni e’lon qilish uchun class so‘zi , { } ichida esa shu sinfning a’zolari va funksiyalari keltiriladi. Masalan:

```
class non
{ int ogirlik ;
  int baho ;
  void yasash ( );
  void yopish ( );
  void eyish ( );}
```

Sinfni e’lon qilishda xotira ajratilmaydi. Sinf e’lon qilinganda kompilyator faqat shunday (non) sinf borligini, hamda unda qanday qiymatlar (ogirlik, baho) saqlanishi mumkinligini, ular yordamida qanday amallarni (yasash, yopish, yeyish) bajarish mumkinligi haqida xabar beradi. Bu sinf obyekti hammasi bo‘lib 4 bayt joy egallaydi (2 ta int).

Obyekt sinfning nusxasi hisoblanadi. C++ tilida tiplarga qiymat o‘zlashtirilmaydi, balki o‘zgaruvchiga o‘zlashtiriladi. Shuning uchun

to‘g‘ridan-to‘g‘ri int = 55 deb yozib bo‘lmaganidek non.baho=1200 deb ham bo‘lmaydi. O‘zlashtirishda xatolikka yo‘l qo‘ymaslik uchun oldin non sinfiga tegishli patir obyektini hosil qilamiz keyin esa unga kerakli qiymatlarni beramiz.

Masalan:

```
int a;     // butun tipli a o‘zgaruvchisi, obyekti
non patir; //
```

Endi non sinfining real obyekti aniqlanganidan so‘ng uning a’zolariga murojaat qilinadi

```
patir.baho = 1200;
patir.ogirlik = 500;
patir.yasash ( ) ;
Sinfni e’lon qilishda quyidagilardan foydalaniladi:
public - ochiq
         private – yopiq
```

Sinfning barcha funksiya va a’zolari boshlang‘ich holda avtomatik ravishda yopiq bo‘ladi. Yopiq a’zolarga esa faqat shu sinfning funksiyalari orqaligina murojaat qilish mumkin. Obyektning ochiq a’zolariga esa dasturdagi barcha funksiyalar murojaat qilishi mumkin. Lekin sinf a’zolariga murojaat qilish ancha mushkul ish hisoblanadi. Agar to‘g‘ridan to‘g‘ri:

non patir;
patir.baho = 1200;
patir.og‘irlik = 500; deb yozsak xato bo‘ladi.

A’zolarga murojaat qilishdan oldin uni ochiq deb e’lon qilish kerak:

```
# include < iostream.h >
class non
{ public :
  int baho;
  int ogirlik;
  void yasash ( );  };
int main ( ){
 non patir;
patir.baho = 1200;  patir.ogirlik = 500;
cout <<“men olgan patir” <<patir.baho
<<“so‘m”<<endl;
cout <<“uning og‘irligi =”<<patir.og‘irlik <<endl ;
}
```

Obyektlar trassirovkasi. Foydalanuvchi ma’lumotlarni kiritmaguncha menyu kutib turadi. Agarda foydalanuvchi to‘g‘ri qiymatni kiritmasa, menyu yangilanadi, foydalanuvchi ma’lumotlarni boshqatdan kiritishi mumkin.

O‘z obyektlaringizning majburiyatini belgilovchi ro‘yxat tuzing. Faqat sizning topshirig‘ingizni yechish uchun zarur bo‘lgan funksiyalarni amalga oshiring. Real narsalar, masalan kassa aparati yoki bank hisob-raqami funksiyasini amalga oshirish uchun o‘n ikkilik funksiyasidan foydalaniladi. Biroq, sizning vazifangiz real dunyoning modelini yaratishdan iborat emas. Sizning topshirig‘ingizni yechish uchun zarur bo‘ladigan vazifalarni aniqlashtirib olishingiz lozim.

Display the menu.(Menyu kiritish)

Get user input. (Foydalanuvchidan kirish ma’lumotlarini olish). Obyekt qanday yaratiladi? Qanday oddiy faoliyatlar ro‘y berishi kerak, har bir savdoni boshlanishida kassa aparatini tozalashga o‘xshash? Menyuni tuzishni menyu yaratish misolida ko‘rib chiqing. Dasturchi menyuning bo‘sh obyektini yaratadi va undan so‘ng “Yangi akkaunt ochish”, “Yordam ” opsiyasini qo‘shadi. Bu yerda yashirin majburiyat bor:

```
Menu main_menu;
main_menu.add_option("Open new account");
// Add more options
int input = main_menu.get_input();
```

Endi biz o‘ziga xos metodlar ro‘yxatiga egamiz

- void add_option(string option)
- int get_input() const

Menyu chiqarish masalasichi? Menyuni foydalanuvchidan ma’lumot kirtishni so‘ramasdan ko‘rsatishning ma’nosi yo‘q. Agar foydalanuvchi xato ma’lumot kiritsa *get_input* menyuni bir martadan ortiq kiritadi. Shunday qilib *display* xususiy metod uchun yaxshi nomzoddir. Ijtimoiy interfeysni yakunlash uchun siz konstruktorlarni aniqlashingiz kerak. O‘zingizdan so‘rang obyekt yaratish uchun sizga nima kerak. Ba’zan siz 2 ta konstruktorga ehtiyoj sezasiz: biri hamma elementlar uchun ko‘zda tutilgan(default) qiymatlarni va ikkinchisi esa foydalanuvchi tomonidan kiritilgan qiymatlarni o‘rnatuvchi. Menyu misolida biz bo‘sh menyu yaratuvchi yagona konstruktor bilan kifoyalanamiz.

Nusxalash konstruktori. Vektorlar obyektida paralel vektorlarni yarating**.** Ba’zida, bir xil uzunlikdagi vektorlarni ishlatayotganingizni

anglaysiz, har bir saqlaydigan qismi obyekt hisoblanadi. Bu vaziyatda, dasturingizni qayta yaratish va elementlari obyekt hisoblangan yagona vektordan foydalanish yaxshi fikrdir.

Faraz qilaylik hisob raqami bir qator tavsif va narxlardan iborat bo‘lsin. Yagona yechim ikkita vektorni saqlab turishdir:

```
vector<string> descriptions;
vector<double> prices;
```

Vektorlarning har biri bir xil uzunlik va bo‘lakka ega bo‘lib, unga consisting of descriptions[i] va [i] narxlar maydonlaridan iborat birga ishlanadigan ma’lumotlar kiradi. Bu vektorlar paralel vektrlar deb aytiladi.

```
class Item {
public:
 ...
private:
 string description;
 double price;};
```

Fayldagi manba’

• Komponentlik funksiyalari tavsifi

• Komponentsiz funksiyalar tavsifi.

Nazariy bilimlarni tekshirish uchun savollar

1. Obyektga yo‘naltirilgan dasturlash tamoyillari.
2. Sinflar va ob‘yektlar
3. Joylashtiriladigan (inline) funksiya–a’zolar
4. Inkapsulyasiya tushunchasi
5. Vorislik tushunchasi
6. Polimorfizm tushunchasi
7. Inkapsulyasiya nima?
8. Polimorfizm haqida tushuncha.
9. Vorislikning qo‘llanishi.
10. Sinfdagi o‘zgaruvchilar – sinf a’zolarining qo‘llanishi.

4.2.Konstruktorlar va destruktorlar.

📖 *Konstruktorlar bu sinf funksiyasi bo‘lib, obyektlarni avtomatik initsializatsiya qilish uchun ishlatilishini bilib olamiz. Konstruktordan foydalanib obyekt yaratilganda parametr uzatish mumkin. Konstruktorlarda ko‘zda tutilgan qiymatlardan ham foydalanish mumkindir*

✎ ***Kalit so‘zlar.*** Friend, do‘stona (friend) sinflar, sinfning statik ma’lumotlari, destruktorlar, konstruktorlar.

REJA:

1. Konstruktorlar va destruktorlar.
2. Munosabat turlari.
3. Friend funksiyalar va sinflar.
4. Do‘stona (friend) sinflar, sinfning statik ma’lumotlari;
5. Ko‘rsatkichlar va sinf metodlari.
6. Obyektlar massivi. Dinamik ob‘etlarni boshqarish.

Konstruktorlar bu sinf funksiyasi bo‘lib, obyektlarni avtomatik initsializatsiya qilish uchun ishlatiladi. Konstruktorlar ko‘rinishi quyidagicha bo‘lishi mumkin:

Sinf nomi (formal parametrlar ro‘yxati)
{konstruktor tanasi}

Bu konstruktor nomi sinf nomi bilan bir xil bo‘lishi lozim.

Misol uchun complex sinfi uchun konstruktorni quyidagicha kiritish mumkin :

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

Konstruktorlar uchun qaytariluvchi tip ko‘rsatilmaydi, hatto void tipi ham. Dasturchi tomonidan qiymatlar ko‘rsatilmagan holda obyekt yaratilganda ham konstruktor avtomatik ravishda chaqiriladi.

Masalan obyekt *complex ss*; shaklida aniqlangan bo‘lsa, konstruktor avtomatik chaqirilib *real* va *imag* parametrlari avtomatik ravishda 0.0 qiymatlariga ega bo‘ladi.

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

Misol uchun

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

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

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

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

Sinf_nomi.obyekt_nomi(konstruktor_haqiqiy_parametlari)

Sinf_nomi(konstruktor_haqiqiy_parametlari)

Birinchi shakl ishlatilganda haqiqiy parametrlar ro‘yxati bo‘sh bo‘lmasligi lozim. Bu shakldan yangi obyekt yaratilganda foydalaniladi:

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

Konstruktorni ikkinchi shaklda chaqirish nomsiz obyekt yaratilishiga olib keladi. Bu nomsiz obyektdan ifodalarda foydalanish mumkin.

Misol uchun :

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

Bu ifoda orqali ZZ obyekti yaratilib, unga nomsiz obyekt qiymatlari(real= 4.0; imag= 5.0) beriladi;

Siz employee sinfdan foydalansangiz, konstruktor ham employee nomga ega bo‘ladi. Agar dasturda konstruktor berilgan bo‘lsa obyekt yaratilganda avtomatik chaqiriladi. Quyidagi dasturda employee nomli sinf kiritilgandir:

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

```
  salary = 0.0;
  }
```

Shu sinfdan foydalanilgan dastur:

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

Konstruktordan foydalanib obyekt yaratilganda parametr uzatish mumkin: *employee worker(101, 10101.0);*

Agar dasturda *employee* tipidagi obyektlar mavjud bo‘lsa har birini quyidagicha initsializatsiya qilish mumkin

```
  employee worker(101, 10101.0);
```

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

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

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

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

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

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

Obyektlar massivlari. Obyektlar massivini yaratish uchun sinf ko‘zda tutilgan (parametrsiz) konstruktorga ega bo‘lishi kerak.

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

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

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

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

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

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

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

```
  arr.input_array(matt,2);
  arr.show_array(matt,2);
  return 0;}
```

Initsializatorlar ro‘yxati. Konstruktor yordamida obyekt ma’lumotlarini initsiyalizatsiyalashni ikkita usuli mavjud.

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

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

Destruktorlar. Sinfning biror obyekti uchun ajratilgan xotira obyekt yo‘qotilganidan so‘ng bo‘shatilishi lozimdir.

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

Destruktorni standart shakli quyidagicha :

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

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

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

Dastur obyektni o‘chirganda destruktor avtomatik chaqiriladi.

Misol:

```
  #include <iostream>
  using namespace std;
  class Person{
  public:
  Person (){
  cout<<"Yaratidi"<<endl;
  }
  ~Person (){
  cout<<"O‘chirldi"<<endl;
  }};
  int main() {{
  Person work;
  }
  int kk;cin>>kk;
  return 0;
  }
```

Natija

```
Yaratidi
O‘chirldi
```

Maydon qiymatlaridan birgalikda foydalanish. Odatda, ma’lum sinf obyektlari yaratilayotganda, har bir obyekt o‘zining maydon qiymatlari to‘plamini oladi. Biroq shunday hollar ham yuzaga keladiki, unda bir xil sinflar obyektlariga bir yoki bir nechta maydon qiymatlaridan (statik maydon qiymatlaridan) birgalikda foydalanish kerak bo‘lib qoladi. Bunday hollarda maydon qiymatlari umumiy yoki turg‘un (statik) deb e’lon qilinadi va tip oldidan *static* kalit-so‘zi ko‘rsatilganidek ishlatiladi:

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

Sinf e’lon qilingach, elementni sinfdan tashqaridagi global o‘zgaruvchi sifatida e’lon qilish kerak.

int class_name::shared_value;

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

```
#include <iostream>
using namespace std;
class book_series{
public:
  book_series(float);
  void show_book(void);
  void set_pages(int) ;
private:
  static int page_count;
  float price;
};
int book_series::page_count;
void book_series::set_pages(int pages){
  page_count = pages;
}
book_series::book_series(float price){
  book_series::price = price;
}
void book_series:: show_book (void){
```

```
    cout << "Narx: " << price << endl;
    cout << "Betlar: " << page_count << endl;
  }
  int main() {
    book_series programming(213.95);
    book_series word(19.95);
    word.set_pages(256);
    programming.show_book ();
    word.show_book() ;
    cout << endl << "page_count ning o‘zgarishi " <<
  endl;
    programming.set_pages(512);
    programming.show_book();
    word.show_book();
  return 0;
  }
```

Ko‘rinib turganidek, sinf *page_count* ni *static int* sifatida e’lon qiladi. Sinfni aniqlagandan so‘ng, dastur shu vaqtning o‘zida page_count elementini global o‘zgaruvchi sifatida e’lon qiladi. Dastur page_count elementini o‘zgartirganda, o‘zgarish shu vaqtning o‘zidayoq book_series sinfining barcha obyektlarida namoyon bo‘ladi.

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

```
  #include <iostream>
  using namespace std;
  class book_series{
  public:
  static int page_count;
  private:
  float price;
  };
  int book_series::page_count;
```

```
int main(){
book_series::page_count = 256;
cout << "page_count ning joriy qiymati " <<
book_series::page_count <<"ga teng"<<endl;
return 0;
}
```

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

Statik funksiya elementlaridan foydalanish. Avvalgi dasturda ma’lumotlarning *statik* elementlaridan foydalanish ko‘rsatib berilgan edi. C++ tili xuddi shunday usul bilan *statik* funksiya-a’zolarini ham aniqlash imkonini beradi. Agar *statik* funksiya yaratilayotgan bo‘lsa, dastur bunday funksiyani, hatto uning obyektlari yaratilmagan holda ham, chaqirib olinishi mumkin. Masalan, agar sinf sinfdan tashqari ma’lumotlar uchun qo‘llanishi mumkin bo‘lgan funksiyaga ega bo‘lsa, siz bu funksiyani statik qila olishingiz mumkin bo‘ladi. Funksiyadan foydalanish uchun dasturda uni *public* va *static* sifatida e’lon qilishi kerak. Masalan, quyidagi dasturda, hatto book_series sinfidagi obyektlar mavjud bo‘lmasa ham, bu sinfning show_count() funksiyasidan foydalaniladi:

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

Friend funksiyalar. Sinfning private va protected qismiga sinfga tegishli bo‘lmagan **friend** funksiyaga murojaat qilishi mumkin. Friend funksiyalar sinfning ichida friend kalit so‘zi bilan yoziladi.

E’lon qilinishi:

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

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

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

Do‘stona (friend) sinflar. Bir sinf boshqa bir sinfga do‘stona bo‘lishi mumkin. Bunda sinflar bir – birining a’zolaridan foydalanish imkoniyatiga ega bo‘ladi. Bunda shu narsaga e’tibor berish lozimki, biror sinfga do‘stona bo‘ladigan sinf (ya’ni friend kalit so‘zi orqali e’lon qilinadigan sinf), mazkur sinfning a’zolaridan foydalanish imkoniyatini yaratadi

E’lon qilinishi:

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

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

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

Sinfning statik ma’lumotlari. Sinf o‘zgaruvchisini static deb e’lon qilinganda kompilyator uni obyektlar uchun bitta nusxa ko‘rinishida yaratadi. Ya’ni bir nechta obyekt bitta o‘zgaruvchidan foydalanadi. Statik o‘zgaruvchi 0 ga inisalizatsiya qilinadi. Sinf statik a’zolariga **ClassName::static_member** ko‘rinishida murojaat qilinadi. Obyekt orqali ham murojaat qilsa bo‘ladi.

Statik o‘zgaruvchi static kalit so‘zi bilan e’lon qilinadi.

E’lon qilish:

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

Static maydonlardan foydalanish

```
class Proper {
  public:
      static int ob_counter;};
int Proper::ob_counter;
int main() {
  Proper a;
```

```
    cout<<Proper::ob_counter++<<endl;
    cout<<a.ob_counter<<endl;
        return 0;}
```

Statik metodlar. Sinf metodlarini statik o‘zgaruvchilar kabi e’lon qilsa bo‘ladi. Statik metodlar statik a’zolarga murojaat qiladi.

E’lon qilish:

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

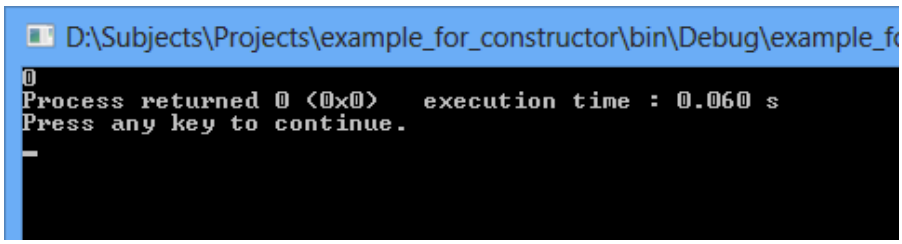
Statik metodlardan foydalanish

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

:: operatoridan foydalanish. Biz bilamizki :: operatori sinf a’zolariga murojaat qilish uchun ishlatiladi. Quyidagi holat berilgan:

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

Ekranga nima chiqadi?



:: operatori orqali global o‘zgaruvchiga murojaat qilish quyidagicha amalga oshiriladi.

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

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

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

Misol asosida ko‘rib o‘tamiz

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

Funksiya obyekt qaytaradi. Funksiya obyekt qaytarishi uchun qaytariladigan obyektning tipi funksiyaning qaytarish tipiga ko‘rsatilishi kerak.

Misol:

```
class XY { … };
XY f() { … return object; }
XY a = f();
Misol-7
class Sum {
  int a, b;
  public: int plus() { cout<<a + b;}
          void set_nums(int x, int y) { a = x; b =
y; }
};
Sum f() {
  x.set_nums(4,5);
  return x; // Funksiya Sum tipdagi obyekt
qaytaradi
}
int main() {
  Sum a;
  a = f(); // a obyektga f() dan qaytgan obyekt
o‘zlashtirildi
  a.plus();
     return 0;
}
```

Obyektlarni o‘zlashtirish

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

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

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

Quyidagi misolda obyektla rmassivi yaratiladi va
ushbu obyektlar orqali set_num() funksiyasiga
qiymat jo‘natiladi. Bunda har obyektda alohida
alohida qiymatlar saqlanadi.
class c1 {
  int a;
  public: int get_num() { return a;}
          void set_num(int x) { a = x; }
};
int main() {
  c1 a[3];        // a obyektdan 3 ta yaratildi,
yani massiv hosil bo‘ldi
  for(int i=0; i<3; i++)
      a[i].set_num(i);    // obyekt a’zolariga
murojaat qilish
    return 0;
}
```

Obyektlar massivini inisalizatsiya qilish. Agar parametrli konstruktor mavjud bo‘lsa obyektlarni inisalizatsiya qilsa bo‘ladi.

```
class c1 {
  int a, b;
  public: int get_num() { return a;}
          c1(int x, int y){ a = x; b = y;}
          c1(){a=0;}
          c1(int x){ a = x; }
```

```
};
c1 a[3] = {1,2,3};        |= {c1(1), c1(2), c1(3)}
       // c1(int x)
c1 a[2] = {{1,2}, {3,4}};  |= {c1(1,2), c1(3,4)}
  // c1(int x, int y);
c1 a[5];                                  // c1()
```

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

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

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

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

Obyekt massiviga ko‘rsatkich quyidagi misoldagi kabi amalga oshiriladi. Bunda c1 sinf tipida a obyekt massivi e’lon qilingan va p sinf tipidagi ko‘rsatkichga obyekt massivi o‘zlashtirilgan.

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

Obyekt a’zolarini ko‘rsatkichga olish.

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

this ko‘rsatkichi. this joriy obyektga ko‘rsatkich.

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

a o‘zgaruvchi joriy obyektga tegishli

Nasl olingan tipga koʻrsatkich. Bir tipdagi koʻrsatkich boshqa bir obyekt tipiga koʻrsata olmaydi. Faqatgina bir holat istisno. Bu ona sinf voris sinflarga koʻrsatkich boʻla oladi.

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

Nasl olingan tipga koʻrsatkich.

```
base *bp;     // ona bp koʻrsatkich
derived d;    // voris d obyekt
bp = &d;      // base koʻrsatkich derived ga
koʻrsatadi
//derived obyektiga base koʻrsatkich orqali
murojaat
bp->set_a(5);
cout<<bp->get_a();
//errorrrr, chunki bp base pointer orqali derived
obyektning a’zolariga
   murojaat qila olmaymiz
bp->set_b(5);
//derived obyektning a’zolariga murojaat qilish
uchun bp base koʻrsatkichni derived koʻrsatkich
tipiga oʻzgartirish kerak
((derived*)bp)->set_b(10);
cout<<((derived*)bp)->get_b();
```

Sinf a’zolariga koʻrsatkich. C++ da shunday koʻrsatkich qilsa boʻladiku bu koʻrsatkich sinf a’zosini koʻrsatib turadi. Bunday koʻrsatkichlarni **pointer-to-member** deb ataladi. Sinf a’zosiga koʻrsatkichda maxsus .* va ->* operatorlar ishlatiladi.

E’lon qilinishi:

```
int c1::*data;    //tipga koʻrsatkich
int (c1::*func)();    //funksiyaga koʻrsatkich
data = &c1::val;      //val joyini data’ga olish
```

```
func = &c1::get_num;  //get_num joyini func‘ga
olish
ob.*data;          // val ga murojaat
(ob.*func)();      // get_num() ga murojaat
```

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

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

qilinmoqda
cout<<(a.*func)();<<(b.*func)(); // obyekt
a’zolariga ko‘rsatkichlar orqali murojaat
qilinmoqda
```

Obyektga ko‘rsatkich bo‘lgan holat.

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

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

Nazariy bilimlarni tekshirish uchun savollar

1. Sinflar va obyektlar tushunchasi. Sinf xususiyatlari va metodlari.
2. Sinf elementlariga murojaat huquqlari. Konstruktor nima?
3. Konstruktorni destruktordan nima ajratib turadi?
4. Obyektlar massivlarini tushuntiring. Sinf statik xususiyat va metodlari.
5. Obyektlarga ko‘rsatkich qanday yaratiladi? Oddiy sinf bilan nasl olingan sinfga ko‘rsatkich qanday farqlanadi?
6. Do‘st funksiyalar qanday e’lon qilinadi? Do‘st sinflar qanday e’lon qilinadi?
7. Obyektlar massivini inisalizatsiya qilish.
8. Inline funksiya va oddiy funksiyaning nima farqi bor? Sinf ichidagi inline funksiyalar.
9. Agarda sinfning ikkita obyektini e’lon qilsak, ularning o‘zgaruvchi a’zolari qiymati turlicha bo‘lishi mumkinmi?
10. Sinf obyektini hosil qilishda qanday funksiya chaqiriladi? Sinf obyektiga qanday murojaat qilinadi?

5-BOB. SATRLAR VA FAYLLAR BILAN ISHLASH USULLARI.

5.1.Satrlar va kengaytirilgan belgilar

📖 *Char tipidagi massivlarning elementlariga va string tipidagi o‘zgaruvchilarga qayta ishlov beruvchi maxsus funksiyalar.*

✎ **Kalit so‘zlar**: *char tipidagi massiv, string, strlen(), sizeof(), strcpy(), strcat(), strstr(), strchr(), assign(), append(), resize(), insert(), delete(), add().*

REJA:

1. Satrlarga ishlov berish standart funksiyalari – satrlarni ulash, solishtirish.
2. Belgilarni va satr qismlarni izlash.
3. O‘zgartirish va o‘chirish.
4. 16-bitli belgilarni boshqarish funksiyalari.

C++ tilida standart satr tipiga qo‘shimcha sifatida string tipi kiritilgan va u string sinfi ko‘rinishida amalga oshirilgan. Bu tipdagi satr uchun ‘\0‘ belgisi tugash belgisi hisoblanmaydi va u oddiygina belgilar massivi sifatida qaraladi. string tipida satrlar uzunligining bajariladigan amallar natijasida dinamik ravishda o‘zgarib turishi, uning tarkibida bir qator funksiyalar aniqlanganligi bu tip bilan ishlashda ma’lum bir qulayliklar yaratadi. *string* tipidagi o‘zgaruvchilar quyidagicha e’lon qilinishi mumkin:

string s1,s2,s3;

Bu tipdagi satrlar uchun maxsus amallar va funksiyalar aniqlangan. string satrga boshlang‘ich qiymatlar har xil usullar orqali berish mumkin:

```
string s1=”birinchi usul”;
string s2(”ikkinchi usul”);
string s3(s2);
string s4=s2;
```

Xuddi shunday, string tipidagi o‘zgaruvchilar ustida qiymat berish amallari ham har xil:

```
string s1,s2,s3; char *str=”misol”;
//satrli o‘zgarmas qiymati berish
s1=”Qiymat berish 1-usul”;
s2=str;       // char tipidagi satr yuklanmoqda
s3=‘A’;       // bitta belgi qiymat sifatida berish
s3=s3+s1+s2+”0123abc”; //qiymat sifatida satr ifoda
```

Ushbu jadvalda string tipidagi satrlar ustida bajariladigan amallar keltirilgan.

Satr elementiga indeks vositasidan tashqari at() funksiyasi orqali murojaat qilish mumkin:

```
string s1=”satr misoli“;
cout<<s.at(3) // natijada ‘r‘ belgisi ekranga
chiqadi
```

Shuni aytib o‘tish kerakki, string sinfda shu tipdagi o‘zgaruvchilar bilan ishlaydigan funksiyalar aniqlangan. Boshqacha aytganda, string tipida e’lon qilingan o‘zgaruvchilar (obyektlar) o‘z funksiyalariga ega hisoblanadi va ularni chaqirish uchun oldin o‘zgaruvchi nomi, keyin ‘.‘ (nuqta) va zarur funksiya nomi (argumentlari bilan) yoziladi.

5.1-jadval. string tipidagi satrlar ustidan amallar

Amal	**Mazmuni**	**Misol**
=, +=	Qiymat berish amali	s=”satr01234” s+=”2satr000”
+	Satrlar ulash amali(konkantenatsiya)	s1+s2
==, !=, <, <=, >, >=	Satrlarni solishtirish amallari	s1==s2 || s1>s2 && s1!=s2
[]	Indeks berish	s[4]
<<	Oqimga chiqarish	cout << s
>>	Oqimdan o‘qish	cin >> s (probelgacha)

Satr qismini boshqa satrga nusxalash funksiyasi. Bir satr qismini boshqa satrga yuklash uchun quyidagi funksiyalarni ishlatish mumkin, ularni prototipi quyidagicha:

```
assign(const string &str);
assign(const string &str,unsigned int pos,unsigned
int n);
assign(const char *str, int n);
```

Birinchi funksiya qiymat berish amal bilan ekvivalentdir: string tipidagi str satr o‘zgaruvchi yoki satr o‘zgarmasni amalni chaqiruvchi satrga beradi:

```
string s1,s2;
s1=”birinchi satr”;
s2.assign(s1); // s2=s1 amalga ekvivalent
```

Ikkinchi funksiya chaqiruvchi satrga argumentdagi str satrning pos o‘rnidan n ta belgidan iborat bo‘lgan satr qismini nusxalaydi. Agarda

pos qiymati str satr uzunligidan katta bo‘lsa, xatolik haqida ogohlantiriladi, agar pos + n ifoda qiymati str satr uzunligidan katta bo‘lsa, str satrining pos o‘rnidan boshlab satr oxirigacha bo‘lgan belgilar nusxalanadi. Bu qoida barcha funksiyalar uchun tegishlidir.

Misol:

```
string s1,s2,s3;
s1=”0123456789”;
s2.assign(s1,4,5);     // s2=”45678”
s3.assign(s1,2,20);    // s3=”23456789”
```

Uchinchi funksiya argumentdagi char tipidagi str satrini string tipiga aylantirib, funksiyani chaqiruvchi satrga o‘zlashtiradi:

```
char * strold;
cin.getline(strold,100);//”0123456789” kiritiladi
string s1,s2;
s2.assign(strold,6);  // s2=”012345”
s3.assign(strold,20); // s3=”0123456789”
```

Satr qismini boshqa satrga qo‘shish funksiyasi. Satr qismini boshqa satrga qo‘shish funksiyalari quyidagicha:

```
append(const string &str);
append(const string & str,unsigned int pos,
            unsigned int n);
append(const char *str, int n);
```

Bu funksiyalarni yuqorida keltirilgan mos assign funksiya-lardan farqi - funksiyani chaqiruvchi satr oxiriga str satrni o‘zini yoki uning qismini qo‘shadi.

```
char * sc;
cin.getline(sc,100);  //”0123456789” kiritiladi
string s1,s,s2;
s2=sc; s1=”misol”;
s=”aaa”;              //s2=”0123456789”
s2.append(”abcdef”); //s2+=”abcdef” amali
   //va s2=”0123456789abcdef”
s1.append(s2,4,5);   //s1=”misol45678”
s.append(sc,5);      // s=”aaa012345”
```

Bir satrga ikkinchi satr qismini joylashtirish uchun quyidagi funksiyalar ishlatiladi:

```
insert(unsigned int pos1,const string &str);
insert(unsigned int pos1,const string & str,
    unsigned int pos2,unsigned int n);
```

```
insert(unsigned int pos1,const char *str, int n);
```

Bu fuksiyalar append kabi ishlaydi, farqi shundaki, str satrini yoki uning qismini funksiyani chaqiruvchi satrning koʻrsatilgan pos1 oʻrnidan boshlab joylashtiradi. Bunda amal chaqiruvchi satrning pos1 oʻrindan keyin joylashgan belgilar oʻnga suriladi.

Misol:

```
char * sc;
cin.getline (sc,100); //”0123456789” satri
kiritiladi
unsigned int i=3;
string s1,s,s2;
s2=sc; s1=”misollar”; s=”xyz”; // s2=”0123456789”
s2.insert(i,“abcdef”); // s2=”012abcdef3456789”
s1.insert(i-1,s2,4,5); // s1=”mi45678sollar”
s.insert(i-2,sc,5); // s=”x01234yz”
```

Satr qismini oʻchirish uchun quyidagi funksiyani ishlatish mumkin:

erase(unsigned int pos=0,unsigned int n=npos);

Bu funksiya, uni chaqiruvchi satrning pos oʻrnidan boshlab n ta belgini oʻchiradi. Agarda pos koʻrsatilmasa, satr boshidan boshlab oʻchiriladi. Agar n koʻrsatilmasa, satrni oxirigacha boʻlgan belgilar oʻchiriladi:

```
string s1,s2,s3;
s1=”0123456789”;
s2=s1;s3=s1;
s1.erase(4,5); // s1=”01239”
s2.erase(3);   // s2=”012”
s3.erase();    // s3=””
void clear() funksiyasi, uni chaqiruvchi satrni
toʻliq tozalaydi.
Masalan:
s1.clear(); //satr boʻsh hisoblanadi (s1=””)
```

Bir satr qismining oʻrniga boshqa satr qismini qoʻyish uchun quyidagi funksiyalardan foydalanish mumkin:

```
replace(unsigned int pos1,unsigned int n1,
    const string & str);
replace(unsigned int pos1,unsigned int n1,
    const string & str,unsigned int pos2,
    unsigned int n2);
```

```
replace(unsigned int pos1,unsigned int n1,
   const char *str, int n);
```

Bu funksiyalar insert kabi ishlaydi, undan farqli ravishda amal chaqiruvchi satrning ko‘rsatilgan o‘rnidan (pos1) n1 belgilar o‘rniga str satrini yoki uning pos2 o‘rindan boshlangan n2 belgidan iborat qismini qo‘yadi (almashtiradi).

Misol:

```
char * sc=”0123456789”;
unsigned int i=3,j=2;
string s1,s,s2;
s2=sc; s1=”misollar”; s=”xyz”; // s2=”0123456789”
s2.replace(i,j,“abcdef”); // s2=”012abcdef56789”
s1.replace(i-1,j+1,s2,4,5); // s1=”mi45678lar”
s.replace(i-2,j+2,sc,5); // s=”x012345”
```

swap(string & str) funksiyasi ikkita satrlarni o‘zaro almashti rish uchun ishlatiladi. Masalan:

```
string s1,s2;
s1=”01234”;
s2=”98765432”;
s1.swap(s2); // s2=”01234” va s1=”98765432”
bo‘ladi.
```

Funksiya prototipi quyidagicha:

string substr(unsigned int pos=0,
unsigned int n=npos)const;

Bu funksiya, uni chaqiruvchi satrning pos o‘rnidan boshlab n belgini natija sifatida qaytaradi. Agarda pos ko‘rsatilmasa, satr boshidan boshlab ajratib olinadi, agar n ko‘rsatilmasa, satr oxirigacha bo‘lgan belgilar natija sifatida qaytariladi:

```
string s1,s2,s3;
s1=”0123456789”;
s2=s1; s3=s1;
s2=s1.substr(4,5); // s2=”45678”
s3=s1.substr(3);   // s3=”3456789”
// “30123456789” satr ekranga chiqadi
cout<<s1.substr(1,3)+s1.substr();
```

string tipidagi satrni char tipiga o‘tkazish uchun **const char * c_str()const;**

funksiyani ishlatish kerak. Bu funksiya char tipdagi ‘\0‘ belgisi bilan tugaydigan satrga o‘zgarmas ko‘rsatkichni qaytaradi:

```
shar *s1; string s2=”0123456789”;
s1=s2.c_str();
```

Xuddi shu maqsadda **const char * data()const;** funksiyasidan ham foydalanish mumkin. Lekin bu funksiya satr oxiriga ‘\0‘ belgisini qo‘shmaydi.

string sinfida satr qismini izlash uchun har xil variantdagi funksiyalar aniqlangan. Quyida ulardan asosiylarining tavsifini keltiramiz.

unsigned int find(const string &str,
unsigned int pos=0)const;

Funksiya, uni chaqirgan satrning ko‘rsatilgan joydan (pos) boshlab str satrni qidiradi va birinchi mos keluvchi satr qismining boshlanish indeksini javob sifatida qaytaradi, aks holda maksimal musbat butun npos sonni qaytaradi (npos=4294967295), agar izlash o‘rni (pos) berilmasa, satr boshidan boshlab izlanadi.

unsigned int find(char c.unsigned int pos=0)const;

Bu funksiya oldingidan farqi ravishda satrdan c belgisini izlaydi.

unsigned int rfind(const string &str, unsigned int pos=npos)const;

Funksiya, uni chaqirgan satrning ko‘rsatilgan pos o‘rnigacha str satr ning birinchi uchragan joyini indeksini qaytaradi, aks holda npos qiymatini qaytaradi, agar pos ko‘rsatilmasa satr oxirigacha izlaydi.

unsigned int rfind(char c.unsigned int pos=npos) const;

Bu funksiyaning oldingidan farqi - satrdan c belgisi izlanadi.

unsigned int find_first_of(const string &str, unsigned int pos=0)const;

Funksiya, uni chaqirgan satrning ko‘rsatilgan (pos) joyidan boshlab str satrining ixtiyoriy birorta belgisini qidiradi va birinchi uchraganining indeksini, aks holda npos sonini qaytaradi.

unsigned int find_first_of(char c, unsigned int pos=0)const;

Bu funksiyaning oldingidan farqi - satrdan c belgisini izlaydi;

unsigned int find_last_of(const string &str, unsigned int pos=npos)const;

Funksiya, uni chaqirgan satrning ko‘rsatilgan (pos) joydan boshlab str satrni ixtiyoriy birorta belgisini qidiradi va o‘ng tomondan bi rinchi uchraganining indeksini, aks holda npos sonini qaytaradi.

unsigned int find_last_of(char c, unsigned int pos=npos) const;

Bu funksiya oldingidan farqi - satrdan c belgisini izlaydi;

unsigned int find_first_not_of(const string &str, unsigned int pos=0) const;

Funksiya, uni chaqirgan satrning ko‘rsatilgan (pos) joydan boshlab str satrning birorta ham belgisi kirmaydigan satr qismini qidiradi va chap tomondan birinchi uchraganining indeksini, aks holda npos sonini qaytariladi.

unsigned int find_first_not_of(char c, unsigned int pos=0)const;

Bu funksiyaning oldingidan farqi - satrdan c belgisidan farqli birinchi belgini izlaydi;

unsigned int find_last_not_of(const string &str, unsigned int pos=npos)const;

Funksiya, uni chaqiruvchi satrning ko‘rsatilgan joydan boshlab str satrini tashkil etuvchi belgilar to‘plamiga kirmagan belgini qidi-radi va eng o‘ng tomondan birinchi topilgan belgining indeksini, aks holda npos sonini qaytaradi.

unsigned int find_last_not_of(char c, unsigned int pos=npos)const;

Bu funksiyaning oldingidan farqi - satr oxiridan boshlab c belgisiga o‘xshamagan belgini izlaydi.

Izlash funksiyalarini qo‘llashga misol:

```
#include <iostream.h>
#include <conio.h>
int main(){
 string s1=”01234567893456ab2csef”,
        s2=”456”,s3=”ghk2”;
 int i,j;
 i=s1.find(s2);
 j=s1.rfind(s2);
 cout<<i; // i=4
 cout<<j; // j=11
 cout<<s1.find(‘3‘) <<endl; // natija 3
 cout<<s1.rfind(‘3‘) <<endl;// natija 10
 cout<<s1.find_first_of(s3)<<endl; // natija 2
 cout<<s1.find_last_of(s3)<<endl; // natija 16
 cout<<s1.find_first_not_of(s2)<<endl; // natija
14
 cout<<s1.find_last_not_of(s2)<<endl; // natija
20
}
```

Satrlarni solishtirish. Satrlar qismlarini solishtirish uchun compare funksiyasi ishlatiladi:

```
int compare(const string &str)const;
int compare(unsigned int pos1,unsigned int n1,
const string & str)const;
int compare(unsigned int pos1,unsigned int n1,
const string & str,unsigned int pos2,
             unsigned int n2)const;
```

Funksiyaning birinchi shaklida ikkita satrlar to‘la solishtiri-ladi: funksiya manfiy son qaytaradi, agar funksiyani chaqiruvchi satr str satrdan kichik bo‘lsa, 0 qaytaradi agar ular teng bo‘lsa va musbat son qaytaradi, agar funksiya chaqiruvchi satr str satrdan katta bo‘lsa.

Ikkinchi shaklda xuddi birinchidek amallar bajariladi, faqat funksiya chaqiruvchi satrning pos1 o‘rnidan boshlab n1 ta belgili satr osti str satr bilan solishtiriladi.

Uchinchi ko‘rinishda funksiya chaqiruvchi satrning pos1 o‘rnidan boshlab n1 ta belgili satr qismi va str satrdan ros2 o‘rnidan boshlab n2 ta belgili satr qismlari o‘zaro solishtiriladi.

Misol:

```
#include <iostream.h>
int main() {
 String s1=”01234567893456ab2csef”, s2=”456”,
        s3=”ghk”;
 cout<<“s1=”<<s1<<endl;
 cout<<“s2=”<<s2<<endl;
 cout<<“s3=”<<s3<<endl;
 if(s2.compare(s3)>0)cout<<“s2>s3”<<endl;
 if(s2.compare(s3)==0)cout<<“s2=s3”<<endl;
 if(s2.compare(s3)<0)cout<<“s2<s3”<<endl;
 if(s1.compare(4,6,s2)>0)cout<<“s1[4-9]>s2”<<endl;
 if(s1.compare(5,2,s2,1,2)==0)
   cout<<“s1[5-6]=s2[1-2]”<<endl;
 }
```

Masala. Familiya, ismi va shariflari bilan talabalar ro‘yxati berilgan. Ro‘yxat alfavit bo‘yicha tartiblansin.

Dastur matni:

```
#include <iostream.h>
#include <alloc.h>
int main(int argc, char* argv[]){
```

```
   const int FISh_uzunligi=50;
   string * Talaba;
   char * Satr=(char*)malloc(FISh_uzunligi);
   unsigned int talabalar_soni;
   char son[3];
   do {
    cout<<"Talabalar sonini kiriting: ";
    cin>>son; }
    while((talabalar_soni=atoi(son))<=0);
   Talaba =new string[talabalar_soni];
   cin.ignore();
   for(int i=0; i<talabalar_soni; i++) {
    cout<<i+1<<"-talabaning Familya ismi sharifi: ";
    cin.getline(Satr,50);
    Talaba[i].assign(Satr);  }
   bool almashdi=true;
   for(int i=0; i<talabalar_soni-1 && almashdi; i++)
  {
  almashdi=false;
    for(int j=i; j<talabalar_soni-1; j++)
     if(Talaba[j].compare(Talaba[j+1])>0)   {
      almashdi=true;
      strcpy(Satr,Talaba[j].data());
      Talaba[j].assign(Talaba[j+1]);
      Talaba[j+1].assign(Satr);    }  }
   cout<<"Alfavit bo‘yicha tartiblangan ro‘yxat:\n";
   for(int i=0; i<talabalar_soni; i++)
    cout<<Talaba[i]<<endl;
   delete [] Talaba; free(Satr); return 0; }
```

Dasturda talabalar ro‘yxati string tipidagi Talaba dinamik massiv ko‘rinishida e’lon qilingan va uning o‘lchami foydalanuvchi tomonidan kiritilgan talabar_soni bilan aniqlanadi. Talabalar sonini kiritishda nazorat qilinadi: klaviaturadan satr o‘qiladi va u atoi() funksiyasi yordamida songa aylantiriladi. Agar hosil bo‘lgan son noldan katta son bo‘lmasa, sonni kiritish jarayoni takrorlanadi. Talabalar soni aniq bo‘lgandan keyin har bir talabaning familiya, ismi va sharifi bitta satr sifatida oqimdan o‘qiladi. Keyin, string tipida aniqlangan compare() funksiyasi yordamida massivdagi satrlar o‘zaro solishtiriladi va mos o‘rindagi belgilar kodlarini o‘sishi bo‘yicha «pufakchali saralash» orqali

tartiblanadi. Dastur oxirida hosil bo‘lgan massiv chop etiladi, hamda dinamik massivlar yo‘qotiladi.

Satr xossalarini aniqlash funksiyalari. string sinfida satr uzunligi, uning bo‘shligini yoki egallagan xotira hajmini aniqlaydigan funksiyalar bor:

```
unsigned int size()const; // satr o‘lchami
unsigned int length()const; // satr elementlar soni
unsigned int max_size()const; // satrning maksimal uzunligi(4294967295)
unsigned int capacity()const;// satr egallagan xotira hajmi
bool empty()const; // true, agar satr bo‘sh bo‘lsa
```

Nazariy bilimlarni tekshirish uchun savollar

1. Satr simvolli massivdan qanday farq qiladi ?
2. Bir o‘lchovli massivlarni initsializatsiya qilish usullarini ko‘rsating.
3. Ko‘p o‘lchovli massiv ta’rifi xususiyatlarini ko‘rsating.
4. Ko‘p o‘lchovli massivlar initsializatsiyasi xususiyatlari.
5. Satrlarni initsializatsiya qilish usullarini ko‘rsating.
6. So‘zlar massivi qanday kiritiladi?
7. Qanday qilib bir o‘lchovli massivlar formal parametrlar sifatida ishlatilishi mumkin?
8. Qanday qilib ko‘p o‘lchovli massivlar formal parametrlar sifatida ishlatilishi mumkin?

5.2. Fayllar va fayllar bilan ishlash

📖 *Dasturchi fayllar ishini tashkil qilar ekan, faqat dastur va uning natijasi haqida qayg‘uribgina qolmasdan, balki ko‘plab qo‘shimcha dasturlar yordamida fayllar yaratish, faylda saqlanayotgan ma’lumotlarni boshqarish, tahlil qilish, tartiblash, ehtiyojga qarab displey yoki qog‘ozda akslantirish kabi masalalarni ham hal qilishi kerak. Yana ilgari ko‘zda tutilmagan yangi ehtiyojlar uchun qo‘shimcha dasturlar yaratish haqida ham o‘ylashi kerak.*

✎ ***Kalit so‘zlar****: Fayl, axborotlarni saqlash, matnli ma’lumotlar, nom va kengaytma, tiplashgan fayllar, ofstream, ifstream, fstream, fizik fayllar, faylni ochish, oqim, oqim obyekti, mode, open, ikkilik rejim, matnli fayl, binar fayl, faylni yopish, EOF, o‘qish, seekg, seekp, tellg,*

tellp, streamoff, streampos, sinxronizatsiya, buffer, istisno, exception, xatoliklar, try, catch, throw.

REJA:

1. Fayllar va oqimlar. Diskdagi fayllar bilan ishlash.
2. Fayllar almashinuvi uchun binary rejim.
3. Matnli fayllar.
4. Binar fayllar.
5. C++ tilining fayllar bilan ishlash funksiyalari.
6. Istisno (exception) larni qayta ishlash (throw, try va catch).

C++ tilidagi standart va foydalanuvchi tomonidan aniqlangan tiplarning muhim xususiyati shundan iboratki, ular oldindan berilgan chekli komponentalardan iborat yoki dinamik aniqlanganda operativ xotiraning cheklanganligidadir. Ma’lum bir sinf masalalari uchun oldindan komponentalar sonini aniqlash imkoni yo‘q, ular masalani yechish jarayonida aniqlanishi va yetarlicha katta hajmda bo‘lishi mumkin.

O‘tilganlardan bizga ma’lumki, massivlar yuzlab, hatto minglab elementdan iborat bo‘lishi mumkin. Buncha ma’lumotni klaviatura orqali kiritish uchun qancha vaqt behuda sarf bo‘lishini tushunish qiyin emas. Shuning uchun dasturlashda, odatda, katta hajmdagi ma’lumotlar matnli fayldan o‘qib olinadi. Bunday ma’lumotlar matnli ma’lumotlar sifatida turli usullar bilan hosil qilinadi. Masalan, ba’zi qurilmalarni nazorat testidan o‘tkazish vaqtida olingan natijalar maxsus qurilmalar yordamida matnli faylga yozib boriladi.

Ma’lumotlarni kompyuterning xotira qurilmalaridan birida saqlashning eng qulay shakli fayllar hisoblanadi. Axborotlarni saqlashning boshqa variantlari (masalan, ma’lumotlar bazasi) ham fayllarga asoslanadi.

Fayl — bu kompyuter xotira qurilmalaridan birida saqlanayotgan va o‘z nomiga ega bo‘lgan ma’lumotlar to‘plamidir.

Fayl bo‘sh bo‘lishi ham mumkin. Fayllar ma’lumot saqlashning eng qulay usuli ekanligining sababi quyidagilardan iborat:

1)odatda dasturni bajarib, olingan natijalar dastur o‘z ishini tugatgandan so‘ng, EHM xotirasidan o‘chib ketadi. Bu ma’lumotlarga yana ehtiyoj paydo bo‘lsa, dasturni yangidan ishga tushirishga to‘g‘ri keladi. Buning oldini olish uchun hosil qilingan natijalarni fayllarga yozib qo‘yish mumkin;

2)faylda saqlanayotgan ma’lumotlar ko‘plab masala va dasturlar uchun yangi asos bo‘lishi mumkin, ya’ni dastur natijalari saqlab qo‘yilsa, bu ma’lumotlardan foydalanib boshqa masalalarni yechish mumkin;

3)ma’lumotlar soni EHMning operativ xotirasiga sig‘maydigan darajada ko‘p bo‘lishi mumkin. Bunday vaqtda ma’lumotlarning bir qismini biror faylda vaqtincha saqlab qo‘yish mumkin;

4)fayllardan ulardagi ma’lumotlar doirasidagi ixtiyoriy maqsad va masalalar uchun foydalanish mumkin.

Fayllar o‘zining manzili hamda nomiga ega bo‘ladi. Faylning nomi odatda ikkita qismdan iborat bo‘lishi mumkin: nom va kengaytma.

Masalan: **D:/Dasturlar/tuit.cpp**

yozuvi, tuit.cpp faylni anglatadi. Bu yerda tuit – faylning nomi, .cpp esa uning kengaytmasi. Bu faylning manzili - D diskdagi Dasturlar papkasi.

Dasturchi fayllar ishini tashkil qilar ekan, faqat dastur va uning natijasi haqida qayg‘uribgina qolmasdan, balki ko‘plab qo‘shimcha dasturlar yordamida fayllar yaratish, faylda saqlanayotgan ma’lumotlarni boshqarish, tahlil qilish, tartiblash, ehtiyojga qarab display yoki qog‘ozda akslantirish kabi masalalarni ham hal qilishi kerak.

Yana ilgari ko‘zda tutilmagan yangi ehtiyojlar uchun qo‘shimcha dasturlar yaratish haqida ham o‘ylashi kerak.

C++ tilida fayllar deb, EHMda saqlanayotgan mansub bo‘lgan ma’lumotlar (komponentalar) to‘plamiga aytiladi.

Faylda saqlanayotgan ma’lumotlardan foydalanish uchun ularni o‘qish va o‘zgaruvchilarga qiymat qilib berish talab qilinadi.

Ixtiyoriy vaqtda faylning faqat bitta komponentasi bilan ishlash mumkin, xolos. Bu ma’lumotni ko‘rsatkich (kursor) ko‘rsatib turadi. Ko‘rsatkich birinchi komponentadan boshlab, har bir ma’lumot o‘qilgandan keyin, navbatdagi o‘qish kerak bo‘lgan ma’lumotni ko‘rsatib turadi. (Boshlang‘ich sinfdagi xatcho‘plarni eslab ko‘ring.)

Fayldagi ma’lumotlar soni o‘zgarib turadi va u dastlab nolga teng bo‘ladi. Bu son keyinchalik faylga yangi ma’lumotlar qo‘shilganda ortishi yoki o‘chirilganda nolgacha kamayishi mumkin. Yangi ma’lumotlar odat bo‘yicha doim faylning oxiriga qo‘shiladi.

Dastur yordamida qayta ishlashga mo‘ljallangan fayllar odatda tiplashgan va tiplashmagan bo‘ladi.

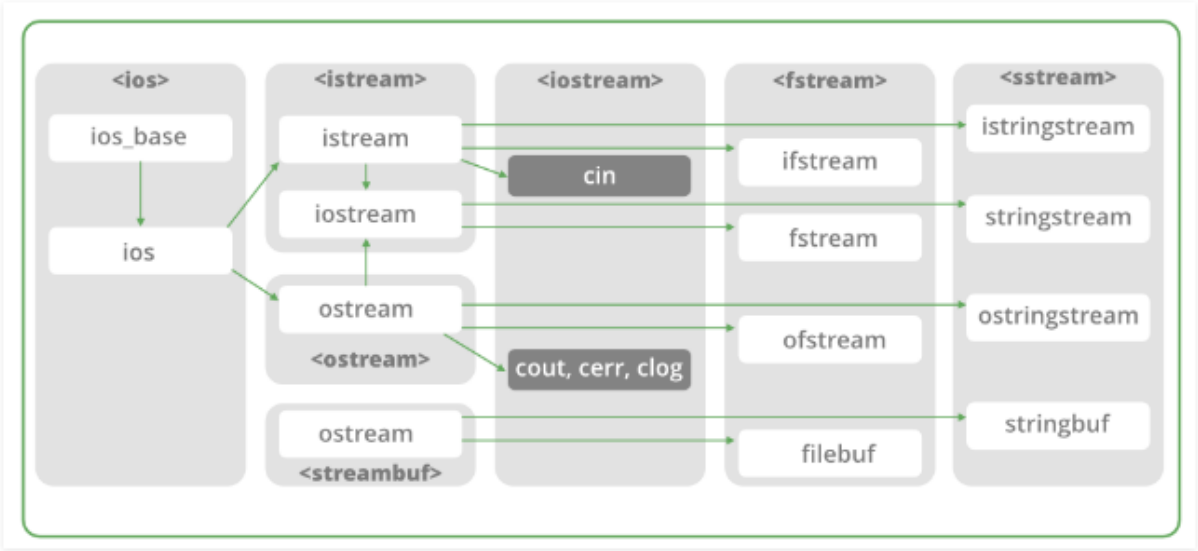
Tiplashgan fayllar faqat ma’lum bir tipdagi ma’lumotlarni saqlaydi.

Ma’lum bir tipga mansub bo‘lgan va fayllarda saqlanayotgan ma’lumotlar yozuv deb ataladi. Faylning yozuvlari baytlar bilan o‘lchanadigan chekli hajmga ega va bu hajm barcha yozuvlar uchun bir xil. Har bir yozuvning faylda turgan o‘rnini doimo aniqlash mumkin.

Tiplashmagan fayllar ma’nosi dasturchi tomonidan aniqlanadigan baytlarning chiziqli ketma-ketligini o‘z ichiga oladi.

Fayllarni C ++ sinflari orqali ishlash. C ++ da fayllar, asosan, fstream sarlavha faylida mavjud bo‘lgan uchta ofstream, ifstream, fstream oqimlari yordamida ko‘rib chiqiladi. Ular:

- **ofstream:** fayllarga ma’lumotlarni yozish uchun sinf oqimi;
- **ifstream:** fayllardan ma’lumotlarni o‘qish uchun sinf oqimi;
- **fstream:** fayllardan ma’lumotlarni o‘qish va fayllarga yozish uchun sinf oqimi.



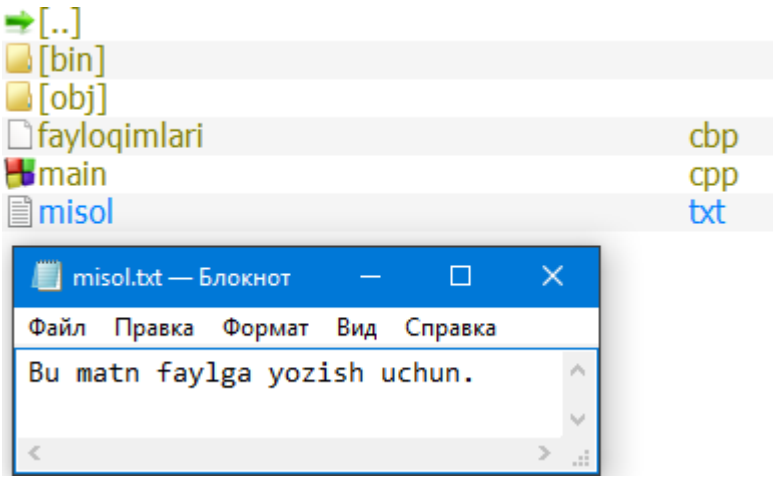
Bu sinflar to‘g‘ridan-to‘g‘ri yoki bilvosita istream va ostream sinflaridan olinadi. Biz allaqachon bu sinflar turidan bo‘lgan istream sinfining obyekti hisoblangan *cin* va ostream sinfining obyekti hisoblangan *cout* lardan foydalanganmiz. Shuning uchun, biz avvaldan fayl oqimlari bilan bog‘liq bo‘lgan sinflardan foydalanib kelganmiz. Aslida, biz fayl oqimlarimizdan avvalgidek foydalana olamiz, faqat cin va coutdan yagona farqi bu fayllarni fizik fayllar bilan bog‘lashimiz kerak. Quyidagi misolni ko‘rib chiqaylik:

```
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream meningfaylim;
```

```
    meningfaylim.open ("misol.txt");
    meningfaylim << "Bu matn faylga yozish uchun.\n";
    meningfaylim.close();
    return 0; }
```

Natija:



Ushbu kod *misol.txt* nomli fayl yaratadi va unga biz odatdagi ya’ni coutdan foydalanganimiz kabi, lekin cout o‘rniga fayl oqimidan foydalanib *meningfaylim* jumlalarini qo‘shamiz.

Faylni ochish. Odatda ushbu sinflar orqali hosil qilingan obyekt yordamida bajariladigan birinchi operatsiya uni haqiqiy faylga bog‘lashdir. Ushbu jarayon faylni ochish sifatida ma’lum. Ochiq fayl dastur ichida oqim bilan taqdim etiladi (masalan, ushbu sinflardan birining obyekti; oldingi misolda bu meningfaylim deb nomlangan edi) va ushbu oqim obyekti ustida bajarilgan har qanday kirish yoki chiqish operatsiyalari bog‘langan fizik faylga qo‘llaniladi, ya’ni har qanday bajargan kodlarimizning natijasi biz yaratgan fayl ichida namoyon bo‘ladi.

Oqim obyekti yordamida faylni ochish uchun biz uning a’zo funksiyasi bo‘lgan *open* dan foydalanamiz : *open (faylnomi, mode);*

Bu yerda *faylnomi* - ochiladigan fayl nomini ko‘rsatadigan satr va *mode*(rejim) - bu parametr quyidagi jadvalda keltirilgan belgilarning ixtiyoriy bittasi:

ios::in	Kirish operatsiyalari ochish uchun.
ios::out	Chiqish operatsiyalari ochish uchun.
ios::binary	Ikkilik rejimda ochish.
ios::ate	Fayl oxirida boshlang‘ich pozitsiyasini o‘rnatish. Agar ushbu belgi o‘rnatilmagan bo‘lsa, boshlang‘ich pozitsiya faylning boshidir.
ios::app	Barcha chiqish operatsiyalari kontentni faylning amaldagi tarkibiga qo‘shib, fayl oxirida amalga oshiriladi.

ios::trunc	Agar fayl chiqish operatsiyalari uchun ochilgan bo‘lsa va u avvaldan mavjud bo‘lsa, avvalgi tarkib o‘chiriladi va yangisi bilan almashtiriladi.

Ushbu belgilarning barchasini OR (|) buyrug‘i yordamida birlashtirish mumkin . Masalan, agar biz misol.bin ma’lumotlarni qo‘shish uchun ikkilik rejimda faylni ochishni xohlasak, uni a’zo funktsiyasiga quyidagicha murojaat qilish orqali amalga oshirishimiz mumkin:

ofstream meningfaylim;
meningfaylim.open("misol.bin", ios::out | ios::app | ios::binary);

ofstream, ifstream va fstream sinflari agar fayl ikkinchi argumentsiz ochilgan bo‘lsa har birining *open* a’zo funktsiyalari ishlatiladigan odatiy rejimiga ega:

Sinf	odatiy rejim parametri
ofstream	ios :: out
ifstream	ios :: in
fstream	ios :: in | ios::out

ifstream va ***ofstream*** sinflari uchun ios :: in va ios :: out avtomatik ravishda va mos ravishda qabul qilinadi, hatto ular mode(rejim)ni o‘z ichiga olmasa ham ***mode open*** a’zo funktsiyasining ikkinchi parametri sifatida beriladi (belgilar birlashtirilgan).

fstream uchun odatiy(default) qiymati faqat funksiya mode(rejim) parametri uchun hech qanday qiymat ko‘rsatmasdan murojaat qilinganda qo‘llaniladi. Agar funksiya ushbu parametrda biron bir qiymat bilan chaqirilsa, odatiy mode(rejim) bekor qilinadi, birlashtirilmaydi.

Ikkilik rejimda ochilgan fayl oqimlari kirish va chiqish operatsiyalarini har qanday format nuqtai nazaridan mustaqil ravishda amalga oshiradi. Ikkilik bo‘lmagan fayllar *matnli fayllar* deb nomlanadi va ba’zi bir maxsus belgilar (masalan, yangi qator va yetkazadigan qaytaruvchi belgilar)ni formatlash tufayli ba’zi o‘zgartirishlar yuzaga kelishi mumkin.

Fayl oqimida bajariladigan birinchi vazifa odatda faylni ochish bo‘lganligi sababli, ushbu uchta sinf avtomatik ravishda ochiq a’zo funktsiyasini chaqiradigan va ushbu a’zo bilan bir xil parametrlarga ega bo‘lgan konstruktorni o‘z ichiga oladi. Shuning uchun biz oldin *meningfaylim* obyektini e’lon qilishimiz va avvalgi misolimizdagi bilan bir xil bo‘lgan ochilish operatsiyasini yozish orqali amalga oshirishimiz mumkin edi:

```
ofstream meningfaylim("misol.bin", ios::out |
ios::app | ios::binary);
```

Obyektning yaratilishi va oqim ochilishini bitta ifodaga birlashtirish. Faylni ochish uchun ikkala holat ham to‘g‘ri va ikkalasini ham qo‘llash mumkin.

Fayl oqimi faylni muvaffaqiyatli ochganligini tekshirish uchun a’zo funksiya is_openga murojaat qilib buni amalga oshirish mumkin. Ushbu funktsiya agar oqim obyekti ochiq fayl bilan bog‘langan bo‘lsa yoki aksi bo‘lsa, mantiqiy qiymat qaytaradi:

```
if (meningfaylim.is_open()){ /* OK, chiqish bilan
davom etish */}
```

Faylni yopish. Faylni kiritish va chiqarish bo‘yicha amallarimizni tugatgandan so‘ng, biz uni yopamiz, shunda operatsion tizim xabardor qilinadi va fayl yana mavjud bo‘ladi(dasturda foydalanilayotgan fayl xotiradan o‘qib olingan bo‘ladi va yopish buyrug‘i bilan o‘zgartirilgan fayl xotiraga qayta yoziladi). Buning uchun biz oqimning *close* deb nomlangan a’zo funktsiyasidan foydalanamiz. Ushbu a’zo funktsiya xotiraning band qilib turilgan buferlarini tozalaydi va faylni yopadi:

```
meningfaylim.close();
```

Ushbu funksiya chaqirilganda, oqim obyekti boshqa faylni ochishda qayta ishlatilishi mumkin va fayl boshqa jarayonlar uchun yana ochib ishlatilishi mumkin bo‘ladi.

Agar obyekt ochiq fayl bilan bog‘langan paytda o‘chirilsa yoki qaysidir sabab bilan yo‘q qilinsa, destruktor avtomatik ravishda a’zo funktsiyasi bo‘lgan ***close*** ni chaqiradi.

Matnli fayllar. Ushbu fayllar matnni saqlash uchun mo‘ljallangan va shuning uchun ularga kirish yoki chiqish qiymatlari bazi formatlash o‘zgarishlariga duch kelishi mumkin, lekin bu ularning asl ikkilik qiymatiga to‘g‘ri kelmaydi. Boshqacha qilib aytganda, matnli fayllar - bu odam o‘qiydigan belgilarni matnli hujjat sifatida saqlash uchun foydalaniladigan ikkilik fayllarning maxsus to‘plami. Matnli fayllar ham ma’lumotlarni ketma-ket baytlarda saqlaydi.

Matnli fayllar buzilishlarga kamroq moyil bo‘ladi, chunki istalmagan o‘zgarishlar shunchaki fayl ochilganda paydo bo‘lishi mumkin va keyin ularni osongina olib tashlash mumkin.

Matnli fayllar ikki xil bo‘ladi:

- Oddiy matnli fayllar: Ushbu fayllar har bir satr oxirida satr uzilishini va fayl oxirida faylning oxiri(End of File-EOF)ni ifodalash uchun End of Line (EOL) markerini saqlaydi.
- Boyitilgan matnli fayllar: Ushbu fayllar oddiy matnli fayllar bilan bir xil sxema bo‘yicha ishlaydi, lekin boyitilgan matnli fayllar matn rangi, matn uslubi, shrift uslubi va boshqalar kabi matn bilan bog‘liq ma’lumotlarni saqlashi mumkin.

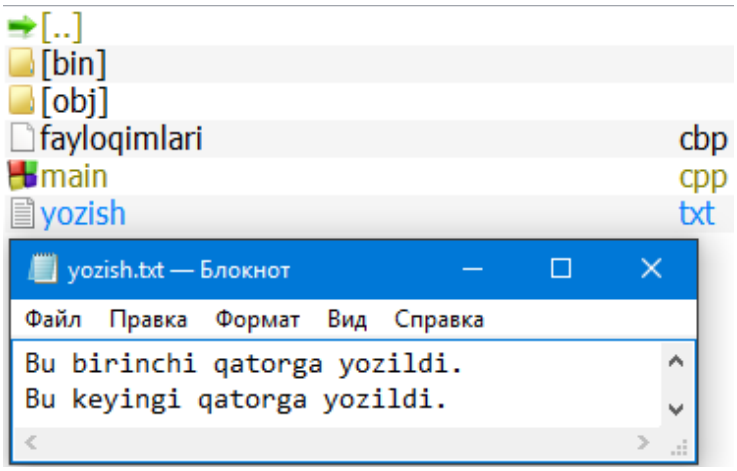
Ma’lumotni saqlash uchun oddiy va standart format tufayli, matnli fayllar matnli ma’lumotlarni saqlash uchun eng ko‘p ishlatiladigan fayl formatlaridan biri bo‘lib, ko‘plab dasturlarda qo‘llab-quvvatlanadi.

Matn fayllariga yozish operatsiyalari biz ishlatgan cout operatori bilan bir xil usulda bajariladi:

```
// Matnli faylga yozish
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream meningfaylim ("misol.txt");
  if (meningfaylim.is_open())
  {
    meningfaylim<< "Bu birinchi qatorga
yozildi.\n";
    meningfaylim<< "Bu keyingi qatorga yozildi.
\n";
    meningfaylim.close();
  }
  else cout << "Faylni ochib bo‘lmadi!";
  return 0;
}
```

Natija:



Fayldan ma’lumotni o‘qish, xuddi cin operatori qo‘llanilish jarayonidek amalga oshiriladi:

```
// Matnli fayldan o‘qish
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string uq_matn;
  ifstream meningfaylim ("yozish.txt");
  if (meningfaylim.is_open())
  {
    cout<<"Matnli fayldan o‘qilgan
ma’lumotlar:"<<endl;
    while ( getline (meningfaylim, uq_matn) )
    {
      cout << uq_matn << ‘\n‘;
    }
    meningfaylim.close();
  }

  else cout << " Faylni ochib bo‘lmadi!";

  return 0;
}
```

Ushbu oxirgi misol matnli faylni o‘qiydi va o‘qilgan ma’lumotlarni ekranga chiqaradi. Biz getline() funksiyasidan foydalanib fayl satrini ketma-ket o‘qiydigan while siklini yaratdik.

Getline tomonidan qaytarilgan qiymat oqim obyektining havolasi bo‘lib, agar u mantiqiy ifoda sifatida baholanganda (ushbu while sikli kabi) oqim ko‘proq operatsiyalarga tayyor bo‘lsa va u faylning oxiriga yetgan bo‘lsa yoki boshqa biron bir xato bo‘lsa, yolg‘on(false) qiymat qaytaradi.

Masala. Binar fayldan haqiqiy sonlarni o‘qish (agar fayl mavjud bo‘lmasa uni hosil qilish va haqiqiy sonlar bilan to‘ldirish) va o‘qilgan sonlarning o‘rta arifmetigini hisoblash, hamda ushbu sonlar orasidan

hisoblangan o‘rta arifmetikdan kichiklari miqdorini aniqlash dasturi tuzilsin.

Izoh: Faylni yaratish va undagi o‘rta arifmetikdan kichik sonlar miqdorini aniqlash, alohida funksiyalar ko‘rinishida tasvirlanishi mumkin.

```
#include <iostream.h>
#include <stdio.h>
# include <string.h>
// Yangi fayl yaratish va unga sonlarni yozish
funksiyasi;
int Fayl_tuzish()
{
 FILE *f;
 double x;
 // «f» fayl yangidan hosil qilish uchun
ochilmoqda;
 if((f=fopen("Haqiqiy.son", "wb+"))==NULL) return
0;
 char *satr=new char[10];
 int n=l;
 do {
  cout<<"Haqiqiy sonni kiriting: ";
  gets(satr);
  if(strlen(satr))
{ x=atof(satr);
   fwrite (&x,sizeof(double),n,f);  } }
while(strlen(satr));
// kiritilgan satr bo‘sh bo‘lmasa, takrorlanish
davom etadi;
  fclose(f);
  return  1; }
// O‘rta arifmetikdan kichik sonlar miqdorini
hisoblash funksiyasi;
 int Kichiklar_soni()
 {
  FILE*f;
double x;
 f=fopen("Haqiqiy.son", "rb+");
```

```
 double s=0; // s - f fayl elementlari yig‘indisi;
  while(!feof(f))
  {
   if (fread(&x,sizeof(double),1,f)) s+=x;
  }
  long sonlar_miqdori=ftell(f)/sizeof (double);
  s/=sonlar_miqdori;    // s- o‘rta arifmetik;
  cout<<"Fayldagi sonlar o‘rta
arifmetigi="<<s<<‘endl‘;
  fseek(f,SEEK_SET,0); // fayl boshiga kelinsin;
  int k=0;
  while (fread(&x,sizeof(x),1,f))
  {
   k+=(x<s); //o‘rta arifmetikdan kichik elementlar
soni;
  }
   fclose(f);
   return k;
  }
  int main()
  {
   if(Fayl_tuzish())
   {
    cout<<”Haqiqiy.son faylidagi \n";
    int Kichik=Kichiklar_soni();
    cout<<‘‘O‘rta arifmetikdan kichik  sonlar
miqdori=";       cout<<Kichik;
   }
   else   // f faylini yaratish muvafaqiyatsiz
bo‘ldi.
   cout<<"Haqiqiy.son faylini ochish imkoni
bo‘lmadi!!!‘‘;
return 0;
}
```

Dasturda bosh funksiyadan tashqari ikkita funksiya aniqlangan:

Int Fayl_tuzish() - diskda “*Haqiqiy.son*” nomli faylni yaratadi. Agar faylni yaratish muvaffaqiyatli bo‘lsa, funksiya *1* qiymatini, aks holda *0* qiymatini qaytaradi. Faylni yaratishda klaviaturadan sonlarning satr

koʻrinishi oʻqiladi va haqiqiy songa aylantirilib, faylga yoziladi. Agar boʻsh satr kiritilsa, sonlarni kiritish jarayoni toʻxtatiladi va fayl yopiladi;

int Kichiklar_soni() funksiyasi diskdagi "*Haqiqiy.son*" nomli faylni oʻqish uchun ochadi va fayl elementlarining oʻrta arifmetigi *s* hisoblanadi soʻngra oʻrta arifmetikdan kichik boʻlgan elementlar miqdori *k* hisoblanib, funksiya natijasi sifatida qaytariladi.

Bosh funksiyada faylning yaratilishi tekshiriladi va unga mos xabar beriladi.

Holat belgilarini tekshirish. Oqimning muayyan holatini tekshirish uchun quyidagi funktsiyalar mavjud (ularning barchasi bool qiymatni qaytaradi):

bad() – agar oʻqish yoki yozish jarayoni muvaffaqiyatsiz boʻlsa funksiya true qaytaradi . Masalan, biz yozishga tayyor boʻlmagan faylga yozishga harakat qilsak yoki biz yozmoqchi boʻlgan qurilmada boʻsh joy qolmasa.

fail() – true ba'zi bir xil hollarda ***bad()*** qiymat qaytaradi, ammo bu holatda format xatosi yuz berishi ham mumkin, masalan, butun sonni oʻqishga harakat qilayotganimizda son emas matn olinganda.

eof() **–** agar oʻqish uchun ochilgan fayl oxiriga yetgan boʻlsa, true(rost) qiymatni qaytaradi.

good() - bu eng umumiy holat belgisi: oldingi funksiyalardan birini chaqirgan paytda rostan true(rost) boʻlgan holatlarda u false qaytaradi. E'tibor bering, yaxshi va yomon aniq bir-biriga zid emas (good bir vaqtning oʻzida koʻproq holat belgilarini tekshiradi). ***clear()*** a'zo funksiyasi holat belgilarini tiklash uchun ishlatilishi mumkin.

Oqim holatida get(olish) va put(joylashtirish).

ifstream istream kabi keyingi kirish jarayonida oʻqilishi kerak boʻlgan elementning joylashuvi bilan ichki kirish holatini saqlab turadi.

ofstream ostream kabi keyingi element yozilishi kerak boʻlgan joylashuv bilan ichki joylashuv pozitsiyasini saqlab turadi. Va nihoyat, fstream, iostream kabi, ikkalasini ham olish va qoʻyish pozitsiyasi(holati)ni saqlaydi.

Oqimning ichki pozitsiyalari keyingi oʻqish yoki yozish jarayoni amalga oshiriladigan oqim ichidagi joylarga ishora qiladi. Ushbu pozitsiyalar quyidagi a'zo funktsiyalari yordamida kuzatilishi va oʻzgartirilishi mumkin:

tellg() and tellp()

Parametrlari bo‘lmagan ushbu ikki a’zo funktsiyalari mavjud get pozitsiyasini (tellg holatida) yoki joylashish pozitsiyasini (tellp holatida) ifodalaydigan streampos turdagi qiymatini qaytaradi.

seekg() and seekp()

Ushbu funksiyalar get(olish) va put(joylashtirish) pozitsiyalarini o‘zgartirishga imkon beradi. Ikkala funksiya ikki xil prototip bilan qayta yuklangan. Birinchi shakl:

```
seekg ( position );
seekp ( position );
```

Ushbu prototipdan foydalanib, oqim ko‘rsatkichi mutlaq pozitsiya holatiga o‘zgartiriladi (fayl boshidan hisoblab chiqiladi). Ushbu parametr uchun tip - bu streampos, bu esa tellg va tellp funksiyalari tomonidan qaytariladigan turga o‘xshaydi.

Ushbu funktsiyalar uchun boshqa shakl:

```
seekg ( offset, direction );
seekp ( offset, direction );
```

Ushbu prototipdan foydalanib get yoki put pozitsiyasi parametr yo‘nalishi bilan aniqlangan ba’zi bir aniq nuqtaga nisbatan ofset qiymatiga o‘rnatiladi. offset bu streamoff(oqim)ning tipidir. Va oqim *seekdir* turiga kiradi, bu sanab o‘tilgan tip bo‘lib u offset hisoblanadigan joyni belgilaydi va quyidagi qiymatlardan birini qabul qilishi mumkin:

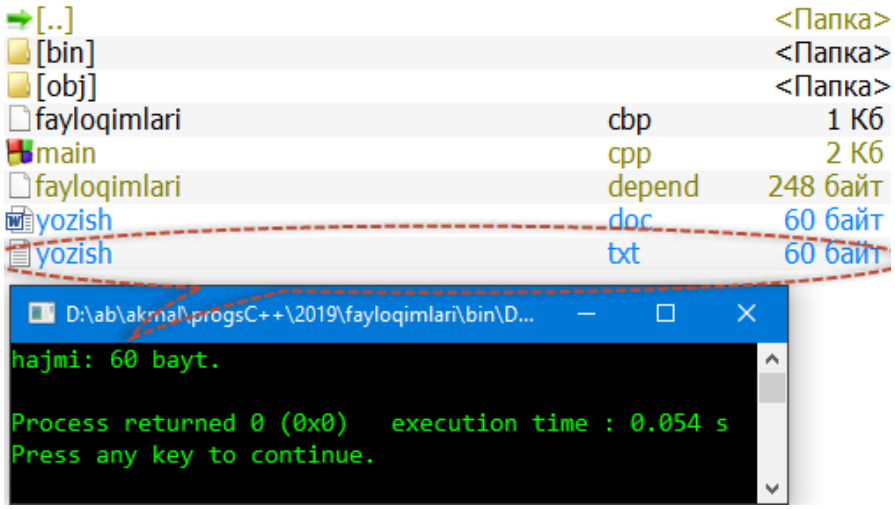
ios::beg	oqim boshidan hisoblangan offset
ios::cur	joriy holatdan hisoblangan offset
ios::end	oqim oxiridan hisoblangan offset

Quyidagi misolda faylning hajmini olish uchun biz ko‘rgan a’zo funksiyalardan foydalaniladi:

```
// fayl hajmini olish
#include <iostream>
#include <fstream>
using namespace std;
int main () {
  streampos begin,end;
  ifstream meningfaylim ("yozish.txt", ios::binary);
  begin = meningfaylim.tellg();
  meningfaylim.seekg (0, ios::end);
  end = meningfaylim.tellg();
```

```
  meningfaylim.close();
  cout << "hajmi: " << (end-begin) << " bayt.\n";
  return 0;
}
```

Natija:



O‘zgaruvchilar uchun foydalangan turimizga e’tibor bering *begin* va *end*:

```
streampos size;
```

streampos - bufer va faylni joylashishi uchun ishlatiladigan o‘ziga xos tip va file.tellg() tomonidan qaytarilgan tip. Ushbu tipdagi qiymatlarni boshqa tipdagi qiymatlardan xavfsiz ajratib olish mumkin va shuningdek, fayl hajmini o‘z ichiga oladigan butun son tipiga o‘tkazish mumkin.

Oqimni aniqlashning ushbu funktsiyalari ikkita o‘ziga xos turdan foydalanadi: *streampos* va *streamoff*. Ushbu turlar, shuningdek, oqim sinfining a’zo turlari sifatida aniqlanadi:

Turi	A’zo turi	Ta’rif
streampos	ios::pos_type	fpos<mbstate_t> sifatida belgilangan. U streamoff-ga / dan o‘zgartirilishi mumkin va ushbu turdagi qiymatlarni qo‘shish yoki ayirish mumkin.
streamoff	ios::off_type	Bu asosiy integral turlaridan biriga(masalan int yoki long long) biri kabi o‘xshashdir .

Yuqoridagi a’zolar turlarining har biri uning a’zosi bo‘lmagan ekvivalenti o‘xshashidir (ular aynan bir xil). Qaysi biri ishlatilganligi muhim emas.

Fayldan o‘qish va yozishga misol. Quyidagi C++ dasturi faylni o‘qish va yozish rejimida ochiladi. Foydalanuvchi tomonidan kiritilgan

ma’lumotni sinov.dat nomli faylga yozgandan so‘ng, dastur fayldan ma’lumotlarni o‘qiydi va uni ekranga chiqaradi.

```
#include <fstream>
#include <iostream>
using namespace std;

int main () {
   char data[100];

   // faylni yozish rejimida ochish.
   ofstream outfile;
   outfile.open("sinov.dat");

   cout << "Faylga yozish" << endl;
   cout << "Ismingizni kiriting: ";
   cin.getline(data, 100);

   // kiritilgan ma’lumotlarni faylga yozish.
   outfile << data << endl;

   cout << "Yoshingizni kiriting: ";
   cin >> data;
   cin.ignore();

   // faylga yana kiritilgan ma’lumotlarni
yozing.
   outfile << data << endl;

   // ochilgan faylni yopish.
   outfile.close();

   // faylni o‘qish rejimida ochish.
   ifstream infile;
   infile.open("sinov.dat");

   cout << "Fayldan o‘qish" << endl;
   infile >> data;
```

```
    // ekranda ma’lumotlarni chop qilish.
    cout << data << endl;

    // fayldan ma’lumotlarni qayta o‘qish va uni
  namoyish etish.
    infile >> data;
    cout << data << endl;

    // ochilgan faylni yopish.
    infile.close();

    return 0;
  }
```

Binar(ikkilik) fayllar. *Binar(ikkilik) fayllar*−bu oddiygina baytlar ketma-ketligidir. Binar fayllardan ma’lumotlarni foydalanuvchi tomonidan bevosita ko‘rish zarur bo‘lmagan hollarda foydalaniladi. Binar fayllaridan o‘qish-yozishda baytlar ustida hech qanday konvertatsiya amallari bajarilmaydi.

Ikkilik fayl sakkiz yoki ba’zan o‘n olti bitga guruhlangan baytlar ketma-ketligi ko‘rinishida ma’lumotlarni saqlaydigan tipik fayllar. Ushbu bitlar maxsus ma’lumotlarni anglatadi va bunday fayllar bir nechta ma’lumotlarni (rasmlar, audio, matn va h.k.) bitta fayl ostida saqlashi mumkin.

Ikkilik faylning eng keng tarqalgan namunalaridan biri bu rasmli fayl .PNG yoki .JPG. Agar kimdir ushbu fayllarni matn muharriri yordamida ochishga harakat qilsa, u tanib bo‘lmaydigan belgilarga ega bo‘lishi mumkin, ammo rasmlarni ko‘rishni qo‘llab-quvvatlaydigan vositasi yordamida ochilganda fayl bitta rasm sifatida ko‘rsatiladi. Buning sababi fayl ikkilik formatda va baytlar ketma-ketligi ko‘rinishidagi ma’lumotlarni o‘z ichiga oladi. Matn muharriri ushbu baytlarni o‘qishga harakat qilganda, bitlarni belgilarga aylantirib chiqadi va ular keraksiz maxsus belgilarni foydalanuvchiga ko‘rsatadi.

Ikkilik fayllar shuningdek, fayl nomi, fayl formati va boshqalar kabi fayl ma’lumotlarini saqlaydi. Ular faylga sarlavha sifatida kiritilishi mumkin va hatto fayl matn muharririda ochilganda ham ko‘rinadi.

Ikkilik fayllar ma’lumotlarni ketma-ket baytlarda saqlaganligi sababli, fayldagi kichik o‘zgarish faylni buzishi va unlarni o‘qiydigan dasturda ham o‘qib bo‘lmaydigan holga keltirishi mumkin.

Ikkilik fayllardan ma’lumotlarni o‘qish va yozish uchun ***getline*** hamda kiritish va chiqarish operatorlari (<< va >>) kabi funktsiyalardan foydalanish samarasiz, chunki biz biron-bir ma’lumotni formatlashimiz shart emas va ma’lumotlar satrlarda formatlanmagan bo‘lishi mumkin.

Endi fayllar bilan ishlashda kerak bo‘ladigan bir qator tushunchalar bilan tanishamiz. Oqim tushunchasi−bu ma’lumotlarni faylga o‘qish-yozishda ularni belgilar ketma-ketligi yoki oqimi ko‘rinishida tasavvur qilishdan kelib chiqqan. Oqim ustida quyidagi amallarni bajarish mumkin:

- *oqimdan ma’lumotlar blokini operativ xotiraga o‘qish;*
- *operativ xotiradagi ma’lumotlar blokini oqimga yozish (chiqarish);*
- *oqimdagi ma’lumotlar blokini yangilash;*
- *oqimdan yozuvni o‘qish;*
- *oqimga yozuvni chiqarish.*

Oqim bilan ishlaydigan barcha funksiyalar buferli, formatlashgan yoki formatlashmagan o‘qish-yozishni ta’minlaydi.

Dastur ishga tushganda o‘qish-yozishning quyidagi standart oqimlari ochiladi:

stdin - o‘qishning standart vositasi;
stdout - yozishning standart vositasi;
stderr- xatolik haqida xabar berishning standart vositasi;
stdprn - qog‘ozga chop qilishning standart vositasi;
stdaux - standart yordamchi qurilma.

Kelishuv bo‘yicha *stdin*-foydalanuvchi klaviaturasi, *stdout* va *stderr* - terminal (monitor), *stdprn* - printer bilan, hamda *stdaux* - kompyuter yordamchi portlariga bog‘lanish hisoblanadi. Ma’lumotlarni o‘qish-yozishda *stderr* va *stdaux* oqimidan boshqa oqimlar buferlanadi, ya’ni belgilar ketma-ketligi operativ xotiraning bufer deb nomlanuvchi sohasida vaqtincha jamlanadi.

Fayl oqimlari ikkilik(binary) ma’lumotlarini ketma-ket o‘qish va yozish uchun maxsus yaratilgan ikkita a’zo funktsiyasini o‘z ichiga oladi: ***write*** (yozish) va ***read*** (o‘qish). Birinchisi (yozish) ***ostream***ning a’zo funktsiyasi (meros orqali meros qilib olingan). Va o‘qish ***istream*** a’zoligi funktsiyasi (***ifstream*** tomonidan meros qilib olingan). ***fstream*** sinf obyektlari ikkalasiga ham ega. Ularning prototiplari:

```
write(xotira_block,size);
read(xotira_block,size);
```

xotira_block char * (pointer to char) tipiga kiradi va o‘qilgan

ma’lumotlar elementlari saqlanadigan yoki yoziladigan ma’lumot elementlari olinadigan baytlarning manzilini bildiradi. ***size*** parametri - bu xotira blokidan o‘qiladigan yoki yoziladigan belgilar sonini aniqlaydigan butun son.

```
// to‘liq ikkilik faylni o‘qish
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  streampos hajm;
  char * xotirablock;

  ifstream fayl ("misol.bin",
ios::in|ios::binary|ios::ate);
  if (fayl.is_open())
  {
    hajm = fayl.tellg();
    xotirablock = new char [hajm];
    fayl.seekg (0, ios::beg);
    fayl.read (xotirablock, hajm);
    fayl.close();

    cout << " butun fayl tarkibi xotirada";

    delete[]xotirablock;
  }
  else cout << " Faylni ochib bo‘lmadi!";
  return 0;
}
```

Ushbu misolda, fayl to‘lig‘icha o‘qiladi va xotira blokida saqlanadi. Buni qanday amalga oshirilishini ko‘rib chiqamiz:

Birinchidan, fayl ios::ate belgisi bilan ochilgan, ya’ni *get pointer* fayl oxirida joylashishini anglatadi. Shu tarzda, tellg() a’zoga murojaat qilganimizda , biz fayl hajmini to‘g‘ridan-to‘g‘ri olamiz.

Fayl hajmini olgandan so‘ng, butun faylni ushlab turish uchun katta hajmdagi xotira blokini ajratishni so‘raymiz:

```
xotirablock = new char[hajm];
```

Shundan so‘ng, biz faylning boshida *get manzili*ni o‘rnatamiz (oxirida faylni ushbu ko‘rsatgich bilan ochganimizni eslang), so‘ngra faylni to‘liq o‘qib chiqdik va nihoyat uni yopamiz:

```
file.seekg (0, ios::beg);
file.read (xotirablock, size);
file.close();
```

Bu vatda biz fayldan olingan ma’lumotlar bilan ishlashimiz mumkin edi. Ammo bizning dasturimiz shunchaki faylning mazmuni xotirada ekanligini va keyin tugashini e’lon qiladi.

Qisqacha qilib aytganda ma’lumotlar ikkilik formatdagi faylda saqlansa, ma’lumotlarni o‘qish va yozish tezroq amalga oshiriladi, chunki ma’lumotlarni bir formatdan boshqa formatga o‘tkazish uchun vaqt yo‘qotilmaydi. Bunday fayllarga ***ikkilik fayllar*** deyiladi.

Matnli va binar fayllarni ochish turlari(mode o‘rniga yozish mumkin bo‘lgan qiymatlar) quyidagilar:

mode	**Ma’nosi**
r	Fayl o‘qish uchun ochiladi(ingl.read – o‘qish)
w	Fayl yozish uchun hosil qiladi(ingl.write - yozish)
a	Fayl davomiga qo‘shish uchun ochadi(ingl.append – oxiriga qo‘shish)
rb	Ikkilik faylini o‘qish uchun ochadi
wb	Ikkilik faylini yozish uchun hosil qiladi
ab	Ikkilik faylini oxiriga qo‘shish uchun ochadi
r+	Faylni o‘qish va yozish uchun ochadi
w+	O‘qish va yozish uchun fayl hosil qiladi
a+	Faylni o‘qish va davomiga qo‘shish uchun ochadi
r+b	Ikkilik faylini o‘qish va yozish uchun ochadi
w+b	Ikkilik faylini o‘qish va yozish uchun hosil qiladi
a+b	Ikkilik faylini o‘qish va oxiriga yozish uchun ochadi

Bufferlar va sinxronizatsiya. Fayl oqimlari bilan ishlaganda, ular streambuf tipidagi ichki bufer obyekti bilan bog‘lanadi. Ushbu bufer obyekti oqim va fizik fayl o‘rtasida vositachi sifatida ishlaydigan xotira blokini ifodalaydi. Masalan, ofstream bilan har safar a’zo funktsiyasi put (bitta belgi yozadi) chaqirilganda, oqim to‘g‘ridan-to‘g‘ri bog‘liq bo‘lgan fizik faylga yozilmasdan, belgi shu oraliq buferga joylashtirilishi mumkin.

Operatsion tizim fayllarni o‘qish va yozish uchun buferlashning boshqa qatlamlarini ham belgilashi mumkin.

Bufer tozalanganda, undagi barcha ma’lumotlar fizik muhitga yoziladi (agar u chiqish oqimi bo‘lsa). Ushbu jarayon ***sinxronizatsiya*** deb ataladi va quyidagi holatlardan birida sodir bo‘ladi:

- **Fayl yopilganda:** faylni yopishdan oldin, hali tozalanmagan barcha buferlar sinxronlashtiriladi va barcha kutilayotgan ma’lumotlar fizik vositaga yoziladi yoki o‘qiladi.
- Bufer to‘lganda **:** Buferlar ma’lum hajmga ega. Bufer to‘lgandan keyin u avtomatik ravishda sinxronlashtiriladi.
- **Aniq, manipulyatorlar bilan:** Ba’zi manipulyatorlardan oqimlarda foydalanilganda, aniq sinxronizatsiya sodir bo‘ladi. Ushbu manipulyatorlar quyidagilar: flush va endl.
- **Shubhasiz, sync() a’zo funktsiyasi bilan:** Oqimning a’zo funktsiyasi sync() ni chaqirish darhol sinxronizatsiyaga olib keladi. Agar oqim buferga tegishli bo‘lmasa yoki ishlamay qolsa, bu funksiya -1 ga teng int qiymatini qaytaradi. Aks holda (agar oqim buferi muvaffaqiyatli sinxronlangan bo‘lsa) 0 ni qaytaradi.

5.3-jadval. Matnli Fayl va Binar Fayl o‘rtasidagi farq

Matnli fayl	**Ikkilik fayl**
Bitlar xarakterini anglatadi.	Bitlar odatiy ma’lumotlarni anglatadi.
Fayl ochilishi bilanoq o‘zgartirishlar aks etadi va osongina qaytarib olinishi mumkinligi sababli buzilish xavfi kamroq.	Osonlik bilan buzilib ketishi mumkin, hatto bitta bit o‘zgarishi ham faylni buzishi mumkin.
Faqat oddiy matnni faylda saqlash mumkin.	Turli xil ma’lumotlarni (rasm, audio, matn) bitta faylda saqlashi mumkin.
Keng tarqalgan fayl formati va har qanday oddiy matn muharriri yordamida ochilishi mumkin.	Aniq dastur uchun ishlab chiqilgan va uni boshqa ilovalar tushunmasligi mumkin.
Ko‘pincha .txt va .rtf matnli	Ha bir ilova o‘zining aniqlangan

fayllarga kengaytma sifatida ishlatiladi.	kengaytmasiga ega bo‘lishi mumkin.

C ++ da ikkilik fayllar ustida asosiy operatsiyalar. Ushbu dastur ikkilik fayllarni qanday yaratishni va shuningdek, ikkilik fayllardan ma’lumotlarni qanday o‘qish, yozish, qidirish, o‘chirish va o‘zgartirish usullarini tushuntiradi.

```
#include<iostream>
#include<fstream>
#include<cstdio>
using namespace std;
class Talaba{
    int admno;
    char name[50];
public:
    void setData()    {
        cout << "\nKirish raqamini kiriting ";
        cin >> admno;
        cin.ignore(1000,‘\n‘);
        cout << "Talaba ismini kiriting ";
        cin.getline(name,50);
    }
    void showData()    {
        cout << "\nKirish raqami : " << admno;
        cout << "\nTalaba ismi : " << name;
    }
    int retAdmno()    {
        return admno;
    }};
// ikkilik faylga yozish funksiyasi.
void write_record(){
    ofstream outFile;
    outFile.open("talaba.dat", ios::binary |
ios::app);
    Talaba obj;
    obj.setData();
    outFile.write((char*)&obj, sizeof(obj));
    outFile.close();
}
// fayl yozuvlarini ko‘rsatish funksiyasi
void display(){
```

```
    ifstream inFile;
    inFile.open("talaba.dat", ios::binary);
    Talaba obj;
    while(inFile.read((char*)&obj, sizeof(obj)))
{
        obj.showData();
    }
    inFile.close();
}
//ikkilik fayldan qidirish va namoyish qilish
funktsiyasi
void search(int n){
    ifstream inFile;
    inFile.open("talaba.dat", ios::binary);
    Talaba obj;
    while(inFile.read((char*)&obj, sizeof(obj)))
{
        if(obj.retAdmno() == n)        {
            obj.showData();
        }    }
    inFile.close();
}
// yozuvni o‘chirish funktsiyasi
void delete_record(int n){
    Talaba obj;
    ifstream inFile;
    inFile.open("talaba.dat", ios::binary);
    ofstream outFile;
    outFile.open("vaqtincha.dat", ios::out |
ios::binary);
    while(inFile.read((char*)&obj, sizeof(obj)))
{
        if(obj.retAdmno() != n)        {
            outFile.write((char*)&obj, sizeof(obj));
        }   }
    inFile.close();
    outFile.close();
    remove("talaba.dat");
    rename("vaqtincha.dat", "talaba.dat");
}
// yozuvni o‘zgartirish uchun funksiya
void modify_record(int n){
```

```
    fstream file;
    file.open("talaba.dat",ios::in | ios::out);
    Talaba obj;

    while(file.read((char*)&obj, sizeof(obj)))    {
        if(obj.retAdmno() == n)        {
            cout << "\nTalaba haqidagi yangi
ma’lumotlarni kiriting";
            obj.setData();
            int pos = -1 * sizeof(obj);
            file.seekp(pos, ios::cur);
            file.write((char*)&obj, sizeof(obj));
        }    }
    file.close();}
int main(){
    //4 ta yozuvni faylda saqlash
    for(int i = 1; i <= 4; i++)
       write_record();
    //Barcha yozuvlarni koʻrsatish
    cout << "\nYozuvlar roʻyxati";
    display();
    //Yozuvni qidirish
    cout << "\nQidiruv natijasi";
   search(100);
    //Yozuvni oʻchirish
    delete_record(100);
    cout << "\nYozuv oʻchirildi";
    //Yozuvni oʻzgartirish
    cout << "\n101 yozuvini oʻzgartirish ";
    modify_record(101);
    return 0;
}
```

Fayl ustida amal bajaruvchi turli misollardan namunalar: C ++ tilidagi matnli fayllar ustida ishlash uchun misollar

```
//Matn faylidan oʻqish va uni namoyish etish
uchun dastur
#include<fstream>
#include<iostream>
using namespace std;

int main(){
```

```
    ifstream fin;
    fin.open("yozish.txt");
    char ch;
    while(!fin.eof())
    {
         fin.get(ch);
         cout << ch;
    }
    fin.close();
    return 0;
}
```

```
//Fayl tarkibini boshqa faylga nusxalash uchun
dastur.
#include<iostream>
#include<fstream>
using namespace std;
int main(){
    ifstream fin;
    fin.open("yozish.txt");
    ofstream fout;
    fout.open("nusxafayl.txt");
    char ch;
    while(!fin.eof())    {
        fin.get(ch);
        fout << ch;
    }
    fin.close();
    fout.close();
    cout<<"Nusxa olish jarayoni tugadi!"<<endl;
    return 0;
}
```

Istisno qilinadigan holatlarni qayta ishlash. Xatoliklarni qayta ishlashning umumiy mexanizmi. Ayrim hollarda dasturda kutilmagan bir qator vaziyatlar roʻy berib oʻz vazifasini bajara olmay qolishi mumkin. Bunga nolga boʻlinish, mavjud boʻlmagan xotira qismiga murojaat qilish kabi mumkin boʻlmagan holatlarni misol qilib keltirish mumkin.

Istisno qilinadigan holatlar(sodda qilib xatoliklar deb aytish mumkin)dan foydalanishdan maqsad bajarilishi jarayonida oldindan kutilmagan holatlar yuzaga kelganda dastur o‘z ishini davom ettirishini ta’minlashdan iborat.

C++ tili istisno qilinadigan holatlar sodir bo‘lganda dastur o‘zini qanday tutishini belgilab qo‘yish uchun dasturchilarga bir qator vositalarni taklif qiladi.

Ma’lumki, ***xatoliklar ikki turga*** bo‘linadi: ***kompilyatsiya vaqtidagi*** va ***dasturni bajarish vaqtidagi*** xatoliklar. Odatda 1-turdagi xatoliklarni kompilyator aniqlab beradi, 2-tur xatoliklarni esa faqat dastur bajarilayotgan vaqtda aniqlash mumkin, xolos. 2-tur xatoliklar yuzaga kelganda dastur o‘z ishini to‘xtatib qo‘yadi. Dasturchi buning oldini olishi, ya’ni dastur har qanday holda ham o‘z ishini davom ettirib, kutilgan natijani berishini ta’minlashi lozim. Boshqacha aytganda, bajarish vaqtida yuzaga kelishi mumkin bo‘lgan turli xatoliklarni nazarda tutishi va ularning har biri sodir bo‘lganda dasturning javob reaksiyasini belgilab qo‘yishi zarur.

Dastur ishlab chiqish jarayonida dasturchi xatoliklar yuzaga kelishi mumkin bo‘lgan barcha holatlarni nazorat qilishi lozim. Bunday holatlar *try* bilan boshlanadigan nazorat bloklarida qayta ishlanadi.

Istisno qilinadigan holatlarni qayta ishlash xato yuzaga kelganidan keyin boshlanadi. Bu jarayonni tashkil qilish uchun *throw* operatoridan foydalaniladi.

catch - dasturdagi muammoni hal qilmoqchi bo‘lgan joyda dastur istisno tutish vositasi yordamida istisnoni tutadi. *catch* kalit so‘zi istisnolarni tutishni anglatadi.

Dasturning javob reaksiyasi xatolikni qayta ishlagichlar yordamida ta’minlanadi. Agar dasturning ularga mos javob reaksiyasi belgilanmagan bo‘lsa, standart *terminate* funksiyasi ishga tushadi va u hisoblash jarayonini to‘xtatish uchun *abort* funksiyasini chaqiradi.

Dasturchi mana shunday hollarda jarayonni to‘xtatish uchun shaxsiy funksiyalarini ishlab chiqishi mumkin.

Istisno qilinadigan holatlar sintaksisi. Istisno qilinadigan holatlarni nazorat qiluvchi blok *try* so‘zi bilan boshlanadi va figurali qavslar orasida yoziladi.

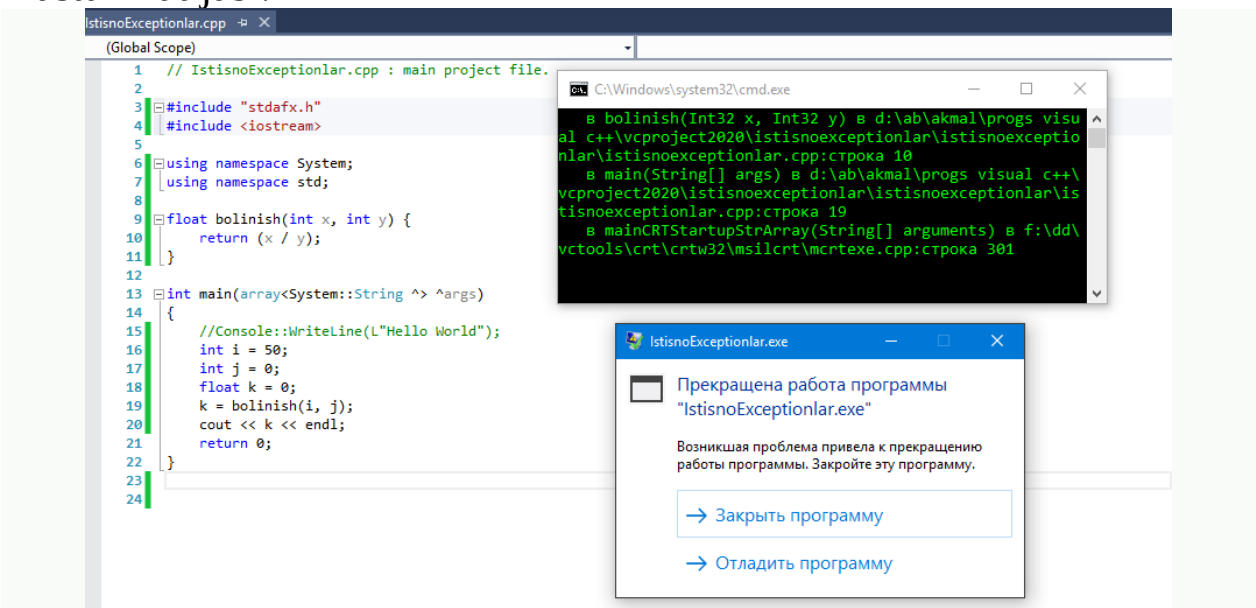
```
try{
...
}
```

C++ try/catch. C ++ dasturlash tilida istisnolarni ishlatish *try/catch* ifodasi yordamida amalga oshiriladi. Istisno yuzaga kelishi mumkin bo‘lgan kodni joylashtirish uchun C++ *try* blokidan foydalaniladi. *Catch* blok istisnolarni bajarish uchun ishlatiladi.

C ++ *try/catch* dan foydalanilmagan holatga misol.

```
#include <iostream>
using namespace std;
float bolinish(int x, int y) {
   return (x/y);
}
int main () {
   int i = 50;
   int j = 0;
   float k = 0;
      k = bolinish (i, j);
      cout << k << endl;
   return 0;
}
```

Dastun natijasi:



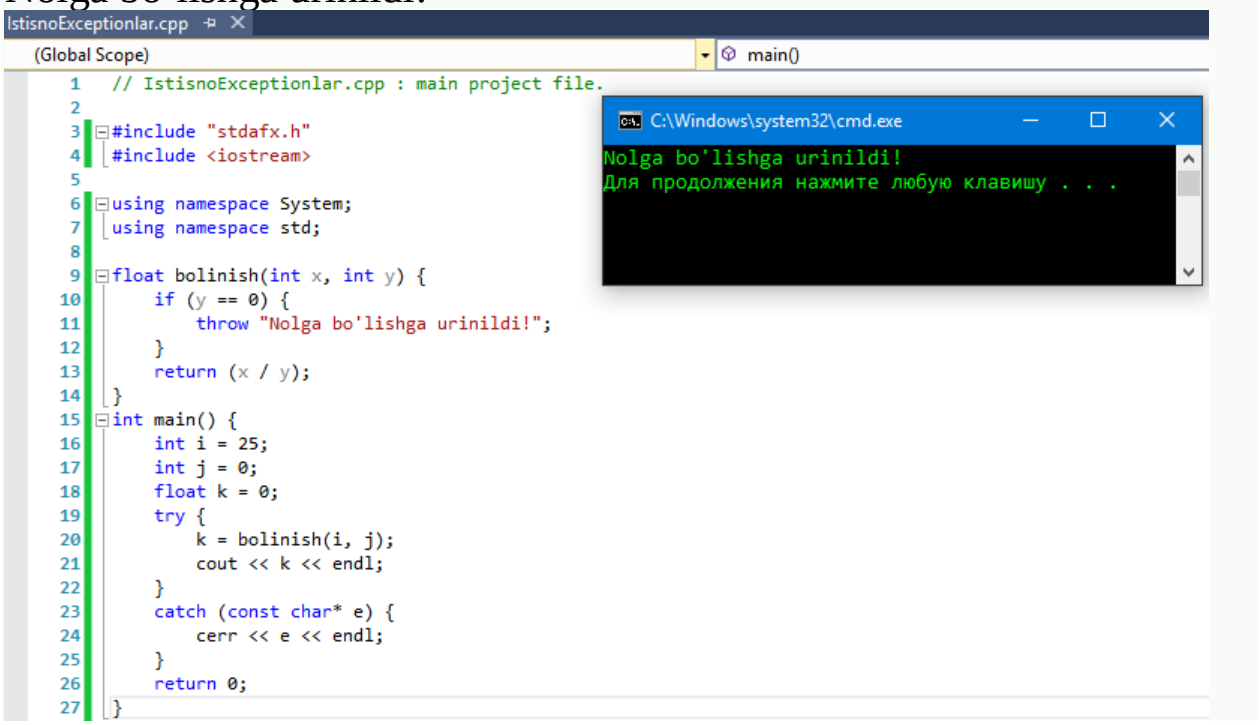
C ++ tilida try/catch dan foydalanishga misol.

```
#include <iostream>
using namespace std;
float bolinish (int x, int y) {
   if( y == 0 ) {
throw "Nolga bo‘lishga urinildi!";
   }
```

```
    return (x/y);
}
int main () {
    int i = 25;
    int j = 0;
    float k = 0;
    try {
        k = bolinish (i, j);
        cout << k << endl;
    }catch (const char* e) {
        cerr << e << endl;
    }
    return 0;
}
```

Dastur natijasi:

Nolga bo‘lishga urinildi!



Istisno qilinadigan holatlarni aniqlash *throw* xizmatchi so‘zi yordamida amalga oshiriladi:

throw [ifoda];

t*hrow* dan keyin ko‘rsatilgan ifodaning tipi xatolik tipini aniqlab beradi. Xatolikni qayd etish vaqtida joriy blokni bajarish jarayoni to‘xtatiladi va boshqaruv unga mos qayta ishlagichga uzatiladi.

Yuzaga kelgan xatolikni to‘g‘ri qayta ishlash har doim ham mumkin bo‘lavermaydi. Ayrim hollarda bir nazorat blogi ikkinchisining

ichida kelishi mumkin. Bunday holda boshqaruv parametrisiz *throw* yordamida tashqi qayta ishlagichlarga uzatiladi.

Xatoliklarni qayta ishlagichlar *catch* xizmatchi so‘zi bilan boshlanib, qavslar ichida istisno qilinadigan holatlar tipi ko‘rsatiladi. Ular bevosita *try* blogidan keyin yoziladi. Qayta ishlanayotgan xatoliklarning tipiga mos ravishda bir yoki bir nechta qayta ishlagichlardan foydalanish mumkin.

Qayta ishlagichlarni umumiy holda quyidagi ko‘rinishlardan birida yozish mumkin:

```
catch (tip nom){ ... /* qayta ishlagich*/ }
catch(tip){... /* qayta ishlagich */}
catch(...){... /* qayta ishlagich */}
```

1-shakl parametr nomi qayta ishlagichda qandaydir amallarni bajarishni tashkil qilishga to‘g‘ri kelganda (masalan, xatolik haqida axborotni ekranga chiqarilganda) qo‘llanadi. Ikkinchi shakl esa xatolik haqida axborot berishni nazarda tutmaydi. Uchinchi shakldagi uch nuqta qayta ishlagich hamma xatoliklarni tutib qolishini anglatadi. Masalan:

```
catch (int i){
... // int tipidagi xatolikni qayta ishlash
}
catch (const char *){
... // const char* tipidagi xatolikni qayta ishlash
}
catch (overflow){
... // Overflow sinfidagi xatoliklarni qayta
ishlash
}
catch(...){
... // ko‘zda tutilmagan barcha xatoliklarni qayta
ishlash
}
```

Qayta ishlash tugaganidan so‘ng boshqaruv bevosita qayta ishlagichdan keyin ko‘rsatilgan birinchi operatorga uzatiladi. *try* blogida ko‘rsatilgan xatoliklar ro‘y bermaganda ham boshqaruv aynan shu operatorga o‘tadi.

Xatoliklarni tutib qolish. *throw* operatori yordamida istisno qilinadigan holatlar qayd qilinganda C++ quyidagi amallarni bajaradi:

1) *throw* parametrini statik obyekt sifatida nusxasini oladi va istisno qilinadigan holatlar qayta ishlanmagunicha saqlab turadi;

2) mos qayta ishlagichni qidirib, steklarni aylantirib koʻrib chiqadi va amal qilish doirasidan chetga chiqqan lokal obyektlarning destruktorlarini chaqiradi;

3) boshqaruvni shu obyekt bilan tipi bir xil boʻlgan parametrli qayta ishlagichga uzatadi.

Qayta ishlagich quyidagi hollarda topilgan hisoblanadi, agar *throw* dan keyin koʻrsatilgan obyektning tipi:

a) *catch* ning parametrida koʻrsatilgan boʻlsa (parametr T. const T, T& yoki const T& shaklida yozilgan boʻlishi mumkin. Bu yerda T - xatolik tipi);

b) *catch* parametrining hosilasi boʻlsa (agar vorislik public kaliti bilan hosil qilingan boʻlsa);

c) tipini standart tiplarni almashtirish qoidalari yordamida *catch* parametridagi koʻrsatkich tipiga keltirish mumkin boʻlgan koʻrsatkichlar.

Koʻrinib turibdiki, sinf hosilalari qayta ishlagichlarini bazaviy qayta ishlagichlardan oldinroq joylashtirish lozim, aks holda boshqaruv hech qachon ularga oʻtmaydi. Void tipidagi koʻrsatkichlarni qayta ishlagichlar toʻgʻridan-toʻgʻri boshqa tipdagi koʻrsatkichlarni toʻsib qoʻyadi va shu sababli uni barcha aniq tipli qayta ishlagichlardan keyin koʻrsatish lozim.

Quyidagi dasturga e’tibor bering.

```
#include <iostream.h>
class Hello{
// o ‘zining yo ‘qotilgani haqida axborot beruvchi
sinf
public:
Hello(){cout « "Hello!" « endl:}
~Hello(){cout « “Bye!” « endl;}
};
Void fl(){
ifstream ifs(“\\INVALID\ FILE‘ ‘NAME "); //Faylni
ochamiz
if (!ifs) {
cout « "xatolikni qayd etamiz « endl;
throw "Fayli ochishdagi xatolik ";}
}
void f2(){
Hello H; // Lokal obyekt yaratilmoqda
```

```
  fl(); //xatolikni yuzaga keltiruvchi funksiya
  chaqirilmoqda
  }
  int та in (){
  try{
  cout« " try-blokka kirish ” « endl;
  f2();
  cout « “try-blokdan chiqish “ « endl;
  }
  catch(int i){
  cout« “int istisnoni qayta ishlagich chaqirildi- “
  «i« endl;
  return -1;
  }
  catch (const char * p){
  cout «“const char* istisnoni qayta ishlagich
  chaqirildi-" « p « endl;
  return -1;
  }
  catch(...){
  cout « "Barcha istisnolarni qayta ishlagich
  chaqirildi –“« endl;
  return -1;}
  return 0; // Hammasi yaxshilik bilan tugadi
  }
```

Ushbu dastur quyidagi natijani beradi:

```
  Try - blokka kirish
  Hello!
  Istisnoni qayd qilamiz
  Bye!
  Const char* istisnoni qayta ishlagich chaqirildi
  Faylni ochishdagi xatolik
```

E’tibor bering, xatolik yuz berganidan keyin lokal obyektning destruktori chaqirildi, ammo bu vaqtda boshqaruv *fl* dan *main* funksiyasida turgan qayta ishlagichga uzatildi. “Try-blokdan chiqish” axboroti ekranga chiqarilmadi. Dasturda fayllar bilan ishlash uchun oqimlardan foydalanildi.

Shunday qilib, istisnolarni qayta ishlash mexanizmi xatoliklar yuz berganda obyektlarni yo‘qotishi mumkin. Shuning uchun resurslarni ajratish va bo‘shatish amalini sinflar ko‘rinishida (konstruktor tashkil qiladi, destruktor esa bo‘shatadi) tashkil qilish maqsadga muvofiq hisoblanadi. Misol tariqasida fayllar bilan ishlash sinfini keltirish mumkin. Bu sinfning konstruktori faylni ochadi, destruktor esa yopadi. Albatta bu holda xatolik yuz berganda faylni yopish to‘g‘ri tashkil qilinadi va undagi ma’lumotlar yo‘qolmaydi.

Ta’kidlab o‘tilganidek, istisno qilinadigan holatlar standart tipda ham, foydalanuvchi aniqlagan tipda ham bo‘lishi mumkin. Bunday hotlarda bu tipni global e’lon qilish shart emas va xatolikni qayd qilish hamda ularni qayta ishlash vaqtida ma’lum bo‘lsa yetarli.

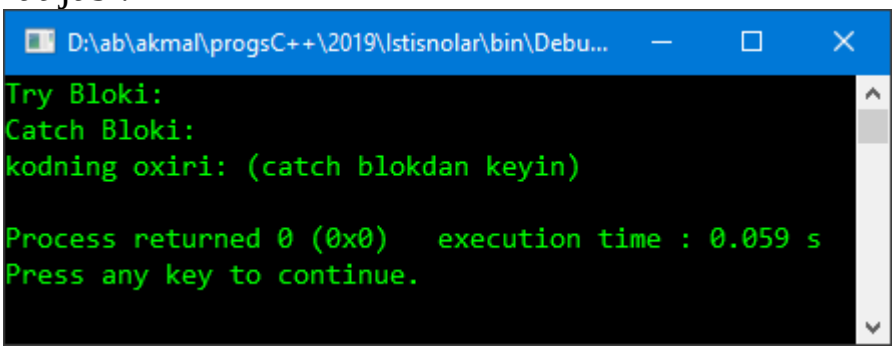
Istisno qilinadigan holatlarni ifodalovchi sinflarni istisnolarni qayta ishlash vaqtida yuzaga kelishi mumkin bo‘lgan sinflar ichida e’lon qilish mumkin. Bu sinfning ko‘chirish konstruktori public tarzida e’lon qilinishi shart, aks holda xatolikni qayd qilish vaqtida obyektning nusxasini yaratish mumkin bo‘lmay qoladi.

Sintaksis. ***try*** bloki ichidagi kod bajariladi. Agar biron bir xato yuzaga kelsa, unda ***throw*** kalit so‘zi istisnoni qayta ishlov beruvchiga, ya’ni ***catch*** blokiga tashlaydi. Keyinchalik ***catch*** bloki blokni ichida joylashgan kodni bajaradi va shu bilan istisnolarni ko‘rib chiqadi.

C ++ da exception ishlashi uchun namunaviy kodni ko‘rib chiqamiz:

```
#include <iostream>
using namespace std;
int main(){
int x = 1;
try{
cout << "Try Bloki: "<<endl;
if(x < 10){
throw x;
}}
catch (int x ) {
cout << "Catch Bloki: "<<endl;
}
cout<<"kodning oxiri: (catch blokdan keyin)
"<<endl;
return 0;
}
```

Dastur natijasi:



Izoh. Ushbu dasturda istisno bilan ishlash ko‘rsatilgan. Bizda *x* o‘zgaruvchisi mavjud bo‘lib, unga 1 qiymati berilgan, keying qatorda ***try*** bloki boshlangan. Ushbu blokda *x <10* shartni tekshiruvchi if operatorimiz bor. Bizning holatimizda shart rost, chunki *x=1*. Keyin dastur istisno qaytaradi va boshqaruv catch blokiga o‘tadi. Biz shartni catch qismida bajaramiz va blokdan chiqamiz.

```
catch (...) {
cout << "Odatiy Istisno"<<endl;
}
```

Mumkin bo‘lgan istisnolar soniga qarab, catchning bir nechta versiyasi bo‘lishi mumkin.

C ++ tilida ushbu istisnolarga oid holatlarni ko‘rib chiqamiz.

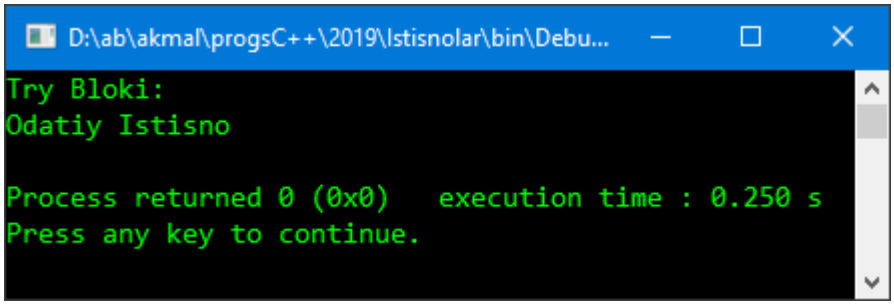
***catch* blokining bajarilmay qolishi.** Oldingi dasturni ko‘rib chiqamiz, agar *x* qiymati o‘rniga “ABC” so‘zi qo‘yilsa, *catch* funktsiyasi uni bajara olmaydi. Bunday holda ekranda xato xabari ko‘rsatilishi mumkin.

Bunday muammolarni hal qilish uchun biz kodga *odatiy(default) catch* funktsiyasini qo‘shishimiz kerak.

```
#include <iostream>
using namespace std;
int main(){
int x = 1;
try {
cout << "Try Bloki: "<<endl;
if(x < 10){
throw "ABC";
}}
catch (int x ) {
cout << "Catch Bloki: n"<<endl;
}
catch(...){//Odatiy (default) catch
cout << "Odatiy Istisno"<<endl;
```

```
}
return 0;
}
```

Natija:



Izoh:

Ushbu kod avvalgisiga o‘xshash. Faqatgina o‘zgarish shundaki, istisnolar *char* tipiga tegishli. Bu bizning catch funktsiyamiz foydasiz bo‘lishiga olib keladi. Shunday qilib, biz odatiy *catch* funktsiyasini qo‘shdik.

Agar catch iboralarining hech biri mos kelmasa, u holda odatiy *catch* bajariladi.

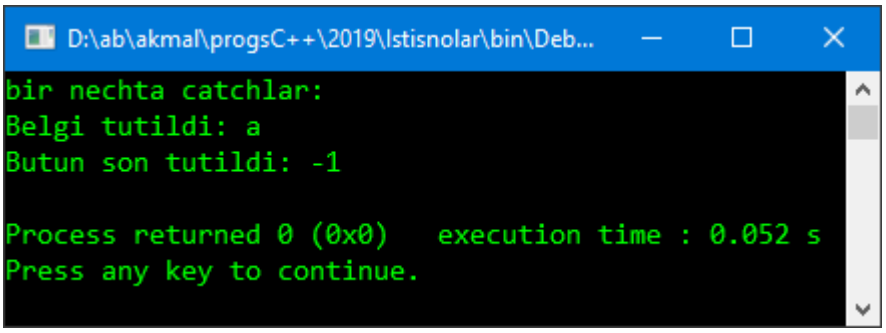
Bir nechta catch bloklari

Bitta *try* blokining bir nechta *catch* bloklari bo‘lishi mumkin.

Mana, misol,

```
#include <iostream>
using namespace std;
int test(int a) {
try{
if(a < 0)
throw a;
else
throw ‘a’;
}catch(int a){
cout<<"Butun son tutildi: " << a<<endl;
}catch (char a){
cout<<"Belgi tutildi: " << a<<endl;
}
return 0;
}
int main() {
cout<<"bir nechta catchlar:"<<endl;
test(10);
test(-1);
return 0;
}
```

Natija:

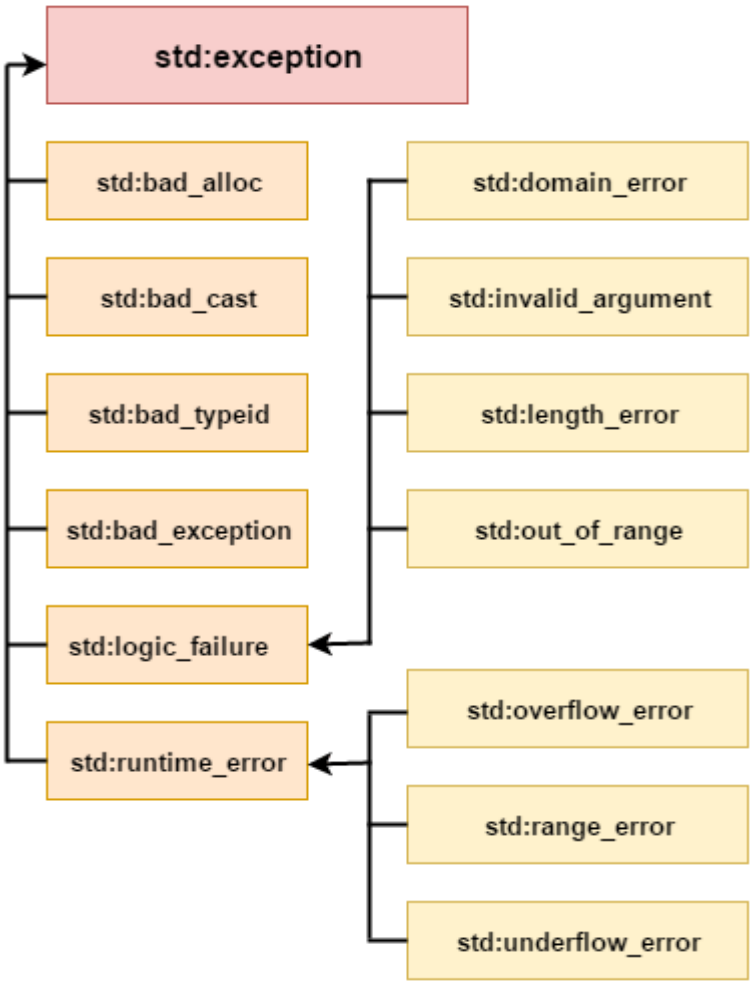


Izoh:

Yuqoridagi kodda biz *catch*ning bir nechta variantlaridan foydalanamiz. Bizda istisnoni keltirib chiqaradigan test funksiyasi mavjud. Birinchi test holatida qiymat 10 ga teng, ‘a’ belgi tashlanadi, chunki 10 noldan katta va ikkinchi *catch* funksiyasi bilan ushlanadi.

Ikkinchi holda, qiymat 0 dan kichik, shuning uchun -1 qiymati tashlanadi va u butun istisno bilan tutiladi.

C ++ istisno(exception) sinflar. C ++ tilida standart istisnolar <exception> sinfida aniqlanadi, biz ularni o‘z dasturlarimiz ichida ishlatishimiz mumkin. Asosiy(parent)-avlod(child) sinflari ierarxiyasining tartibi quyida ko‘rsatilgan:



C ++ tilidagi barcha istisnolar sinflari *std :: exception* sinfidan meros qilib olingan. C ++ istisno sinflarining umumiy ro‘yxatini ko‘rib chiqamiz.

Istisno	**Ta’rif**
std :: exception	Bu barcha standart C ++ istisnolarining istisno va asosiy sinfidir.
std :: logic_failure	Kodni o‘qish orqali aniqlanishi mumkin bo‘lgan istisno.
std :: runtime_error	Kodni o‘qish orqali aniqlab bo‘lmaydigan istisno.
std :: bad_exception	U C++ dasturida kutilmagan istisnolarni qayta ishlash uchun ishlatiladi.
std :: bad_cast	Ushbu istisno odatda **dynamic_cast** tomonidan sodir bo‘ladi **.**
std :: bad_typeid	Ushbu istisno odatda **typid** tomonidan sodir bo‘ladi **.**
std :: bad_alloc	Ushbu istisno odatda **new** tomonidan sodir bo‘ladi **.**

Konstruktor va destruktordagi istisnolar. C++ tili konstruktor va destruktordan qiymat qaytarishda foydalanishga ruxsat bermaydi. Istisnolarni qayta ishlash mexanizmi obyektning konstruktori yoki destruktorida yuzaga kelgan xatolik haqida axborot berishi mumkin. Bu fikrni namoyish qilish uchun Vector sinfini tashkil qilamiz. Unda so‘raladigan xotira hajmi cheklanadi.

```
Class Vector{
public:
class Size{}; // istisno sinfi
enum {max = 32000} : // vektorning maksimal
uzunligi
Vector(int n) //konstruktor
{if (n<0 || n>max) throw Size():... }
};
Vector sinfidan foydalanganda Size tipidagi
```

```
xatoliklarni kuzatish mumkin:
try{
Vector *p = new Vector(i):
}
Catch( Vector: :Size)(
... // vector o ‘lchami bilan bog ‘liq xatolikni
qayta ishlash
}
```

Qayta ishlagichda xatolik haqida axborot berish va qayta tiklashning asosiy usullaridan foydalanish mumkin. Istisnoni aniqlovchi sinf ichida qayta ishlagichga uzatiladigan istisno haqidagi axborotni ham saqlashga ruxsat beriladi. Buning ma’nosi istisno qilinadigan holat aniqlangan nuqtadan xatolik haqidagi axborotni qayta ishlagich yetarlicha imkoniyatga ega bo‘lgan joyga uzatishni ta’minlashdan iborat.

Agar obyekt konstruktorida istisno qayd qilinsa, avtomatik tarzda joriy vaqtgacha shu blokda to‘la yaratilgan obyektlar hamda joriy obyektning maydonlari uchun destruktor chaqiriladi. Masalan, agar istisno obyektlar massivini yaratishda yuzaga kelsa, destruktorlar faqat muvaffaqiyatli yaratilgan elementlar uchun ishga tushadi.

Agar obyekt dinamik xotirada *new* yordamida yaratilayotgan bo‘lib, konstruktorda xatolik ro‘y bersa, bu obyekt band qilgan xotira qismi bo‘shatiladi.

Standart tipdagi istisnolarga qaraganda shaxsiy tipdagi istisnolarni qayta ishlash afzal sanaladi. Istisnolarni qayta ishlashga qaraganda sinflar yordamida istisnolar haqidagi axborotlarni uzatish osonroq. Bundan tashqari, sinflar shajarasidan foydalanishga imkon paydo bo‘ladi.

Istisnolarni boshqarish mexanizmi bazaviy sinflar uchun shaxsiy qayta ishlagichlarni yaratishga imkon beradi, qardosh (bir-biriga yaqin) istisnolarni ko‘pincha shajaralar ko‘rinishida yaratish mumkin bo‘ladi. Umumiy bazaviy sinfdan istisnolarni keltirib chiqarishda polimorfizm prinsipidan foydalanib qayta ishlagichda bazaviy sinfga ko‘rsatkichlarni tutib qolish mumkin bo‘ladi. Masalan, matematik kutubxonada sinflarni quyidagicha tashkil qilish mumkin:

```
class Matherr{};
class Overflow: public Matherr{}: //xotiraning to
‘lib ketishi
class Underflow: public Matherr{};// tartibning yo
‘qolishi
```

```
class ZeroDivide: public Matherr{};   // nolga bo
‘linish
Ma’lumotlarni kiritish va chiqarish bilan bog‘liq
xatoliklarni ifodalash uchun quyidagi sinflardan
foydalanish mumkin:
class Ioegg{}:
class Readerr: public loerg{};//o‘qishdagi xatolik
class Writerr: public Ioegg{}://yozishdagi xatolik
class Seekerr: public Ioegg{}; // qidirishdagi
xatolik
```

Vaziyatga bog‘liq ravishda yoki hosila istisnolarni ham tutib qola oladigan bazaviy sinf qayta ishlagichidan yoki shaxsiy hosila sinf qayta ishlagichlaridan foydalanish mumkin.

Afzalligi. Ilovaning normal oqimini saqlaydi. Bunday holda, kodning qolgan qismi istisnolardan keyin ham bajariladi.

C ++ foydalanuvchi belgilaydigan istisnolar. Istisno sinfi funktsional imkoniyatlarini bekor qilish va meros qilib olish bilan yangi istisno aniqlanishi mumkin .

C ++ foydalanuvchi tomonidan belgilangan istisno misoli. Istisnoni aniqlash uchun std::exception sinfi foydalanadigan foydalanuvchi tomonidan belgilangan istisnoning oddiy misolini ko‘rib chiqamiz .

```
#include <iostream>
#include <exception>
using namespace std;
class UzIstisno : public exception{
    public:
        const char * what() const throw()  {
             return " Nolga bo‘lishga
urinish!\n";
        }
};
int main(){
    try
    {
        int x, y;
        cout << "Ikkita raqam kiriting : \n";
        cin >> x >> y;
        if (y == 0)
        {
```

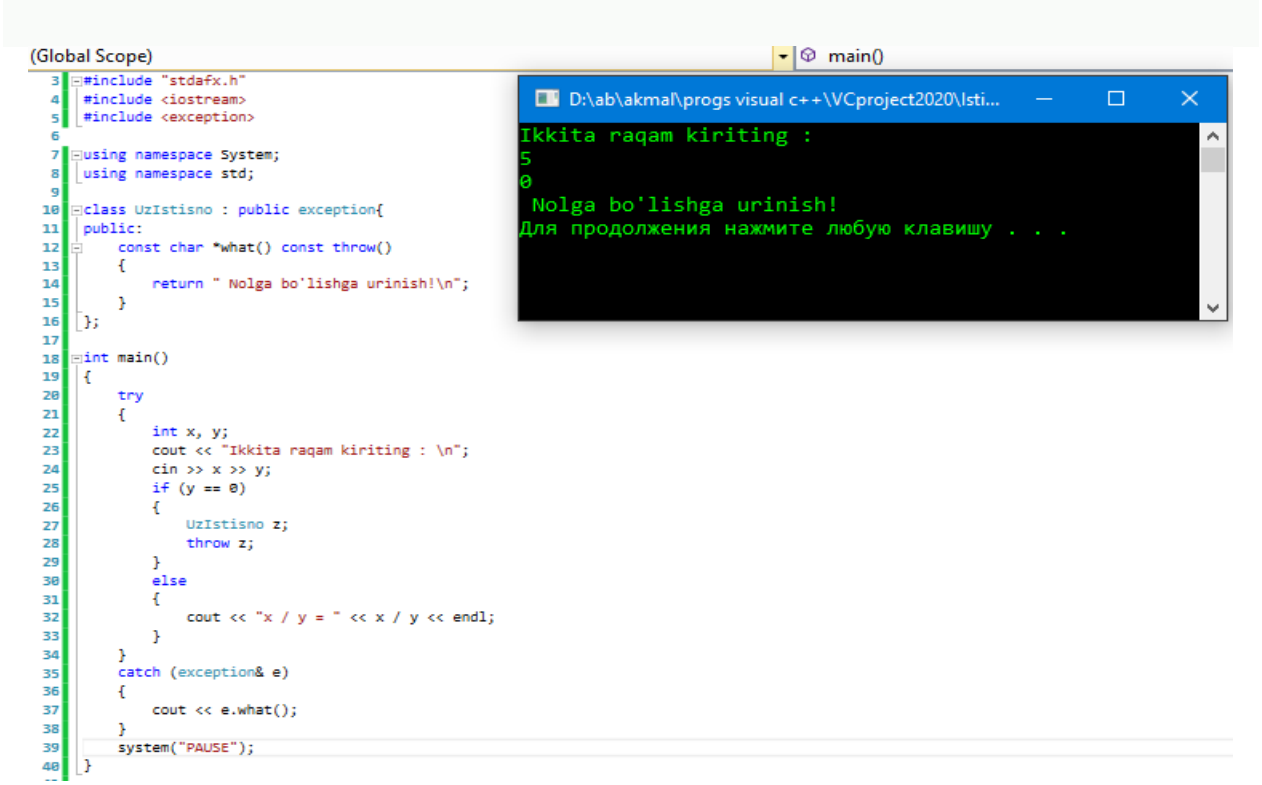
```
                UzIstisno z;
                throw z;
            }
            else
            {
                cout << "x / y = " << x/y << endl;
            }
        }
        catch(exception& e)
        {
            cout << e.what();
        }
    }
```

Dastur natijasi:

```
Ikkita raqam kiriting:
   10
   2
   x / y = 5
```

Dastur natijasi:

```
Ikkita raqam kiriting:
   10
   0
   Nolga bo‘lishga urinish!
```



Eslatma: Yuqoridagi misolda *what() exception* sinfi tomonidan taqdim etilgan public metod. Istisno sababini qaytarish uchun ishlatiladi.

Nazariy bilimlarni tekshirish uchun savollar

1. Fayl nima? Faylning qanday turlarini bilasiz?
2. Fayllar qanday maqsadlarda ochiladi? Matnli fayllar qanday fayllar?
3. Binar fayllar qanday fayllar? Ma’lumotlar oqimini qanday tashkil qilish mumkin?
4. Faylga maiumotlar oqimi qanday ko‘rsatma bilan yoziladi?
5. Faylning oxirini aniqlash mumkinmi?
6. Fayl bilan ishlovchi qanday sinflarni bilasiz?
7. fstream sinfi qanday sinf? ostream sinfi qanday sinf? ifstream sinfi qanday sinf?
8. Oqim argumentlari haqida nimalarni bilasiz?
9. ios :: ate ofstream uchun argument sifatida nimani anglatadi?
10. eof()funksiyasi qanday funksiya? bad()funksiyasi qanday funksiya?
11. ios::beg qiymati nima?ios::cur qiymati nima? ios::end qiymati nima?
12. Istisnolar nima?Istisnolarni tutishning afzalligi nimada?

6-BOB. INKAPSULYATSIYA, MEROSXO‘RLIK, POLIMORFIZM VA SHABLONLAR BILAN ISHLASH.

6.1.Inkapsulyatsiya va merosxo‘rlik

📖 *Obyektga yo‘naltirilgan dasturlashning asosiy tushunchalariga qisqacha to‘xtalib o‘tilgan bo‘lib, asosiy urg‘u inkapsulyatsiya va merosho‘rlik (vorislik) mavzulariga qaratilgan. Inkapsulyasiya – bu ma’lumotlar va ularni qayta ishlovchi kodni birlashtirish mexanizmi hisoblaib, u ma’lumotlar va kodni tashqi ta’sirdan saqlash imkonini berishini nazariy ma’lumotlar va amaliy misollar yordamida yoritilgan. Vorislik – ham OYD ning asosiy ustunlaridan biri ekanligi va vorislik sinflarda ierarxik ko‘rinishdagi sinflanishni ta’minlashi nazariy ma’lumotlar va amaliy misollar bilan asoslangan.*

🖋 ***Kalit so‘zlar:*** *Inkapsulyatsiya, merosxo‘rlik, vorislik, public, private, protected, ma’lumotlarni abstraksiyalash, konstruktor, destruktor, asos sinf, voris sinf, yakka vorislik, to‘plamli vorislik*

REJA:

1. Inkapsulyatsiya.
2. Merosxo‘lik va uning turlari.
3. Himoyalangan merosxo‘rlik.
4. Bazaviy sinf a’zolariga murojaatni boshqarish.

Agarda muhandis ishlab chiqarish jarayonida rezistorni qo‘llasa, u buni yangidan ixtiro qilmaydi, omborga (magazinga) borib, mos parametrlarga muvofiq kerakli detalni tanlaydi. Bu holda muhandis joriy rezistor qanday tuzilganligiga e’tiborini qaratmaydi, rezistor faqatgina zavod xarakteristikalariga muvofiq ishlasa yetarlidir. Aynan, shu tashqi konstruksiyada qo‘llaniladigan yashirinlik yoki obyektni yashirinligi yoki avtonomligi xossasi inkapsulyatsiya deyiladi. Inkapsulyatsiya yordamida ma’lumotlarni yashirish ta’minlanadi. Bu juda yaxshi xarakteristika bo‘lib foydalanuvchi o‘zi ishlatayotgan obyektning ichki ishlari haqida umuman o‘ylamaydi. Yaxshi ishlab chiqilgan dastur obyektini qo‘llashda uning ichki o‘zgaruvchilarining o‘zaro munosabati haqida qayg‘urish zarur emas.

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

C++ tilida inkapsulyatsiya tamoyili sinf deb ataluvchi nostandart

tiplarni(foydalanuvchi tiplarini) hosil qilish orqali himoya qilinadi.

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

Sinf elementini e’lon qilishda bir nechta kalit so‘zlardan foydalaniladi: **public, private, protected**.

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

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

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

Ulardan eng muhimlari **public** (ochiq) va **private** (yopiq) kalit so‘zlari bo‘lib, ular orqali obyektning a’zolariga murojaat qilish imkoniyati chegaralanadi. Sinfning barcha funksiyalari va xossalari boshlang‘ich holda yopiq deb e’lon qilinadi. Yopiq a’zolarga faqatgina shu sinfning funksiyalari orqaligina murojaat qilish mumkin. Obyektning ochiq a’zolariga esa dasturdagi barcha funksiyalar murojaat qilishlari mumkin. Sinf a’zolariga murojaat qilish imkonini belgilash juda muhim xususiyat bo‘lib, bu masalani yechishda uncha katta tajribaga ega bo‘lmagan dasturlarchilar ko‘pincha qiyinchiliklarga duch keladilar. Bu holatni batafsilroq tushuntirish uchun mavzuni boshida keltirilgan masalamizga qaytamiz.

```
Class Mushuk {
 unsigned int itsYosh;
 unsigned int itsOgirlik;
 void Miyovlash();  }
```

Bu tarzda sinfni e’lon qilishda itsYosh va itsOgirlik maydonlari ham, Miyovlash () funksiyasi ham yopiq a’zo sifatida aniqlanadi. Dasturda yuqoridagi tartibda Mushuk sinfi e’lon qilingan bo‘lsa va bu sinf nusxasi bo‘lgan obyektning itsYosh a’zosiga main() funksiyasi tanasidan turib murojaat qilsak, kompilyator xatolik ro‘y berganligi haqida xabar beradi.

Mushuk Baroq;

Baroq.itsYosh = 5 // Xatolik!
// Yopiq a’zoga murojaat qilish mumkin emas.

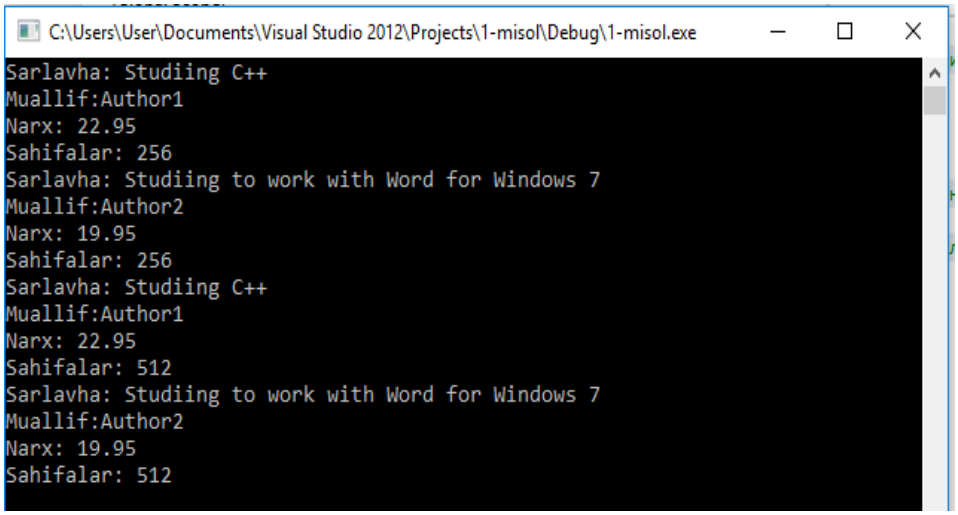
Statik elementlar hamda funksiyalar. Shu paytgacha, har bir yaratilgan element o‘zining xususiy ma’lumotlar elementiga ega bo‘lar edi. Lekin, shunday holat bo‘ladiki, bitta sinf doirasidagi obyektlarning ba’zi elementlari o‘zaro bog‘langan bo‘ladi. Masalan, ish vaqti bir xil bo‘lgan 1000 ta ishchining oylik maoshini hisoblaydigan dastur tuzish taklif qilinayotgan bo‘lsin. Soliq stavkasini aniqlash uchun dastur har bir ishchining sharoitini bilishi kerak. Buning uchun aytaylik, state_of_employee nomli sinfdan foydalanamiz. Agar, ishchilar bir xil sharoitda ishlasa, demak, dastur barcha employee tipidagi obyektlar uchun (barcha ishchilar uchun) ushbu elementlardan o‘zaro moslikda foydalanadi. Ushbu holatda dastur, bitta axborotning 999 ta nusxasidan foydalanish bilan xotiradan foydalanish hajmini kamaytiradi.

Sinfning elementidan o‘zaro moslikda foydalanish uchun, ushbu element **static** (statik) deb e’lon qilinishi zarur. Agar, dastur ushbu elementga yangi qiymat o‘zlashtirsa, hamma obyekt elementi ushbu yangi qiymatni qabul qiladi. Sinf elementi statik deb e’lon qilinganidan so‘ng, u umumiy (global) o‘zgaruvchi sifatida e’lon qilinishi zarur.

```
#include "stdafx.h"
#include <string.h> //strcpy() uchun
#include <stdio.h> //printf() uchun
#include <conio.h> //_getch() uchun
using namespace std;
class book_series{
public:
book_series(char *, char *, float);
void show_book(void); void set_pages(int);
private:
static int page_count; /*bu umumiy element
hisoblanadi*/
char title[64]; char author[64];
float price; };
int book_series::page_count; /*sinfdan
tashaoidaumumiy o‘zgaruvchini e’lon qilish*/
void book_series::set_pages(int pages){
page_count = pages; }
book_series::book_series(char *title, char *author,
float price){ /*Sinfning konstruktori*/
strcpy(book_series::title, title); /*string sinfiga
ulanish uchun zarur bo‘lgan, strcpy() funksiyasi*/
```

```
  strcpy(book_series::author, author);
  book_series::price = price; }
  void book_series:: show_book (void){
  printf("Sarlavha: %s\n",title);
  printf("Muallif:%s\n",author);
  printf("Narx: %.2f\n",price);
  printf("Sahifalar: %d\n",page_count); }
  void main(){
  book_series programming("Studiing C++", "Author1",
  22.95);
  /*programming obyektini konstruktor yordamida
  yaratish*/
  book_series word( "Studiing to work with Word for
  Windows 7", "Author2", 19.95); /*word obyektini
  konstruktor yordamida yaratish*/
  word.set_pages(256); /*Word o‘ektining sahifalari
  soni beriladi, bu programmingga ham ta’sir qiladi
  */
  programming.show_book ();
  word.show_book() ;
  programming.set_pages(512); /*page_countni
  o‘zgartirish*/
  programming.show_book(); /*obyekt ma’lumotlarini
  ekranga chiqarish*/
  word.show_book(); /*obyekt ma’lumotlarini ekranga
  chiqarish*/
  _getch(); }
```

Natijasi:



Obyekt mavjud bo‘lmaganda, public static atributli elementlardan foydalanish. Sinfning barcha obyektlarida o‘zaro moslikda foydalaniladigan, elementi ***static*** sifatida e’lon qilinishi

tushunarli bo‘ldi, lekin, shunday holat bo‘lishi mumkin: hech qanday obyekt yaratilmagan, ammo, ushbu elementdan foydalanish zarur. Dasturda bu elementdan foydalanish uchun, uni ***public*** hamda ***static*** deb e’lon qilish zarur. Ushbu dasturda xuddi shu holatga e’tibor qaratilgan.

Bu holatni ifodalaydigan dasturning kodi quyida ifodalangan:

```
#include "stdafx.h"
#include <string.h> //strcpy() uchun
#include <stdio.h> //printf() uchun
#include <conio.h> //_getch() uchun
using namespace std;
class book_series{
book_series();
public:
static void show_book(void); /*Funksiyani statis
elementini chop etish uchun, ushbu atribut
qo‘shiladi*/
static int page_count;
private:
char title [64];
char author[64];
float price; };
int book_series::page_count; /*O‘zgaruvchini global
o‘zgaruvchi sifatida e’lon qilish*/
void book_series::show_book (void){
printf("Sahifalar soni=%d\n",page_count); }
int main(void){
book_series::page_count = 256;
book_series::show_book(); _getch(); }
```

Natija: Sahifalar soni = 256

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

Hosila sinfni e’lon qilishning umumlashgan sintaksisi:

```
class <sinf nomi>: [<kirish huquqini beruvchi
sertifikator >] <ajdod sinf nomi> {…}
```

Sinf o‘zining bazaviy sinfidan yuzaga kelayotganida, uning barcha

a’zolari hosila sinfda avtomatik tarzda yashirin **private** bo‘lib qoladi. Ammo uni, bazaviy sinfning quyidagi kirish spertifikatorlarini ko‘rsatgan holda osongina o‘zgartirish mumkin:

private. Bazaviy sinfning meros bo‘lib o‘tayotgan (ya’ni himoyalangan va ommaviy) nomlari hosila sinf nusxalarida kirib bo‘lmaydigan bo‘lib qoladi.

public. Bazaviy sinf va uning ajdodlarining nomlari hosila sinf nusxalarida kirib bo‘ladigan bo‘ladi, barcha himoyalangan nomlar esa himoyalangan bo‘lib qolaveradi.

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

Me‘roslikda ko‘rsatilmagan kirish statusini asosiy(bazaviy) sinf ismini oldidan ko‘rsatilgan **private, protected** va **public** kirish atributlari yordamida o‘zgartirish mumkin. Agarda B sinf quyidagicha aniqlangan bo‘lsa:

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

unda quyidagi hosila sinflarni kiritish mumkin:

```
class M: protected B { ... }; //t va u protected
sifatida merosxo‘r.
class P: public B { ... }; // protected, va u-
public sifatida merosxo‘r.
class D: private B { ... }; //t va u private
sifatida merosxo‘r.
struct F: private B { ... }; //t i u private
sifatida merosxo‘r.
struct G: public B { ... }; t - protected va u –
public sifatida merosxo‘r.
```

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

Masalan.

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

Sinf obyektlari pastdan tepaga qarab konstruktorlanadi: avvalo asosiy(bazaviy), keyin esa komponent – obyektlar (agarda ular mavjud bo‘lsa), undan keyin esa hosila sinfning o‘zi. Shunday qilib, hosila sinfning obyekti quyi obyekt sifatida asosiy (bazaviy) sinf obyektini o‘z ichiga oladi. Obyektlar teskari tartibda o‘chiriladi: avvalo hosila, keyin uning komponent – obyektlari, undan keyin esa asosiy(bazaviy) obyekt.

Shunday qilib, obyektni o‘chirish tartibi uning konstruktorlash tartibiga nisbatan teskari bo‘ladi.

Ko‘plikdagi vorislik va virtual sinflar. Bu sinf ketma-ket (to‘g‘ridan-to‘g‘ri) baza sinfidir, agar u boshqa sinflarni aniqlashda ishlatilsa, baza ro‘yxatidan chiqariladi. Ba’zi hollarda A sinf B sinfning bazasini ifodalasa va C uchun B baza bor bo‘lsa, u holda B sinf C uchun to‘g‘ridan-to‘g‘ri baza hisoblanadi, natijada A sinf C sinf uchun to‘g‘ri bo‘lmagan baza bo‘lib hisoblanadi. Quyida keltirilgan sinflarni tasvirlashda bazalar ishlab chiqilgan. Xuddi shu tartibda yangi baza sinflarini kompilyator e’lon qiladi.

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

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

Bir necha to‘g‘ri baza sinflari mavjud bo‘lib, ular ko‘plik vorislari deb nomlanadi. Ko‘plik vorislarida ketma-ket bazada hech qanday sinf bittadan ortiq ishlatilishi mumkin emas. Bitta sinf to‘g‘ri bo‘lmagan sinfda bir necha marta ishlatilishi mumkin:

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

Bu misolda X va Z sinflari D sinfiga voris bo‘ladi. Bir xil nomdagi

obyektlarni bartaraf qilishda to‘g‘ri bo‘lmagan sinf bazalarining ko‘plik vorislari virtual deb e’lon qilinadi. Buning uchun sinf bazalari ro‘yxatida oldingi sinf nomini **virtual** kalit so‘zini ishlatish kerak. Misol uchun X sinfi virtual baza sinfi bo‘lsa u quyidagi ko‘rinishda yoziladi:

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

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

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

Abstrakt sinf hosila sinf uchun asosiy (bazaviy) sinf sifatida ishlatilishi mumkin. Abstrakt sinflarning mexanizmi keyinchalik aniqlashtirilanadigan umumiy tushunchalarni tavsiflash uchun ishlab chiqilgan. Bu holda, sinflar ierarxiyasini yaratish quyidagi sxema bo‘yicha bajariladi. Ierarxiya asosida abstrakt bazaviy sinf turadi. U interfeysni meros qilib olish uchun foydalaniladi. Hosila sinflar bu interfeysni aniqlaydi va amalga oshiradi. Abstrakt sinfda sof virtual funksiyalar e’lon etilgan, ular aslida **abstrakt funksiyalar.**

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

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

Abstrakt sinfni faqat boshqa sinf ajdodi sifatida ishlatish mumkin:

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

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

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

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

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

Drayverlar character_device sinfining ajdodlari sifatida kiritilishi mumkin.

Nazariy bilimlarni tekshirish uchun savollar

1. Inkapsulyatsiya nima? Inkapsulyatsiyadan foydalanishning maqsadi nimadan iborat?
2. Inkapsulyatsiyaning qanday afzallik tomonlari bor?
3. Vorislik nima uchun kerak? Vorislikdan foydalanishning qanday afzalliklari mavjud? Vorislikning qanday turlari mavjud?
4. Public ruxsat modifikatorini hayotiy misollar bilan tushuntirib bering. Private ruxsat modifikatorini misollar bilan tushuntirib bering.
5. Protected ruxsat modifikatorini hayotiy misollar bilan tushuntirib bering.
6. Himoyalangan protected va xususiy private huquqlari orasida qanday farq bor? Protected va public huquqlari orasida qanday farq bor?
7. private va public huquqlari orasida qanday farq bor?
8. Public murojaat huquqi orqali voris olishni misollar orqali tushuntiring.
9. Private murojaat huquqi orqali voris olishni misollar orqali tushuntiring.
10. Protected murojaat huquqi orqali voris olishni misollar orqali tushuntiring.
11. Nima uchun protected kalit so‘zi asosiy sinfda ishlatiladi?
12. Nima uchun avval ajdod sinf konstruktorlari chaqirilib, so‘ngra avlod sinf konstruktori chaqiriladi?

6.2.Polimorfizm.

📖 *Obyektga yo‘naltirilgan dasturlash tamoyillaridan polimorfizmva uning turlari hamda qo‘llanilish usullari.*

🖋 ***Kalit so‘zlar:*** *sinf, polimorfizm, virtual funksiyalar, novirtual funksiyalar, abstrakt sinflar, dinamik polimorfizm, virtual destructor.*

REJA:

1. Polimorfizm va uning turlari.
2. Virtual funksiya.
3. Abstrakt sinf va funksiyalar
4. Vaqtli va kechiktirilgan bog‘lanishlar, virtual va novirtual funksiyalar;
5. Dinamik polimorfizmni qo‘llash.
6. Vertual destructorlar;
7. Abstrakt sinflar va sof virtual funksiyalar.

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 polimorfizmni amalga oshirishning asosi C++ tilidagi asosiy sinfga ko‘rsatkichni aniqlanishidir. O‘z navbatida, bu ko‘rsatkich nafaqat asosiy sinfga, balki shu sinfning vorisi bo‘lgan ixtiyoriy sinf obyektiga ko‘rsatishi mumkin. Sinflarning bu xossasi vorislikdan kelib chiqadi, chunki har qanday voris sinf obyekti asosiy sinf turida bo‘ladi. Dasturni yig‘ish paytida (komponovka paytida) asosiy sinfga ko‘rsatkich egasi bo‘lgan foydalanuvchi tomonidan qaysi sinf obyekti yaratilishi noma’lum bo‘ladi. Shu sababli, ko‘rsatkich o‘z obyekti 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 ma’lumotlar 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, keyin jadvaldan mos funksiya adresini oladi. Keyin ko‘rsatilgan adresga o‘tish bilan funksiya chaqirilishi ro‘y beradi.

Jadval 1. Sinf virtual funksiyalar jadvali

Sinf shajarasi va ko‘rsatgichlar	**Ko‘rsatgich qiymatining turi**	**Sinflardagi virtual funksiyalarning adreslari**		
		f1()	**f2()**	**…**
Asosiy * asosiy;	Asosiy	$Adres_{11}$	$Adres_{21}$	…
Voris1 * voris1; // *Asosiy vorisi*	Voris1	$Adres_{12}$	$Adres_{22}$	…
Voris2 * voris2; // *Asosiy vorisi*	Voris2	$Adres_{13}$	$Adres_{23}$	…
…	…	…	…	…

Ma’lumki, hosilaviy sinf obyekti yaratilishida asosiy 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 obyekti 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 taqoza etadi va undan oqilona foydalanish zarur bo‘ladi.

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 aniq 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, asosiy 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 asosiy sinfga ko‘rsatkich yoki murojaat ishlatilgan bo‘lsa ham.

```
class Asosiy{
 public:
  Asosiy(int _x) {x=_x;}
  virtual int Qiymat_X(){return x;}
  virtual void Chop_X();
 private:
  int x;};
void Asosiy:: Chop_X()
{cout<<"Asosiy::x="<<Qiymat_X()<<‘\n‘; }
class Hosila1: public Asosiy{
 public:
  Hosila1(int _x): Asosiy(_x){}
  void Chop_X();};
void Hosila1:: Chop_X()
{cout<<"Hosila1::x="<<Qiymat_X()<<‘\n‘;}
class Hosila2: public Asosiy{
 public:
  Hosila2(int _x): Asosiy(_x){}
  void Chop_X();
};
void Hosila2:: Chop_X()
{cout<<"Hosila2::x="<<Qiymat_X()<<‘\n‘; }
int main(int argc, char* argv[]){
 Asosiy * asosiy=new Asosiy(10);
 Hosila1 * hos1=new Hosila1(20);
 Hosila2 * hos2=new Hosila2(30);
 asosiy->Chop_X();
 asosiy=hos1;
 asosiy->Chop_X();
 asosiy=hos2;
 asosiy->Chop_X();
 return 0;}
```

Hosilaviy sinflardagi Chop_X() funksiyalari virtual hisoblanadi, chunki u Asosiy asosiy sinfida virtual deb e’lon qilingan. Virtual funksiyalarni chaqirish uchun quyidagi kodlar ishlatilgan:

```
asosiy=hos1;
asosiy->Chop_X();
asosiy=hos2;
asosiy->Chop_X();
```

Sinflardagi Chop_X() funksiyalari virtual bo‘lganligi uchun har bir obyektning o‘z funksiyasi chaqiriladi. Hosila1 va Hosila2 sinflarida Chop_X() funksiyalari asosiy sinfidagi Chop_X() funksiyasini qayta aniqlaydi. Agar hosilaviy sinfda Chop_X() funksiyasi qayta aniqlanmasa, kelishuv bo‘yicha asosiy sinfdagi Chop_X() funksiyasi bajariladi.

Yuqoridagi dastur ishlashi natijasida ekranga quyidagi natijalar chop etiladi:

```
Asosiy::x=10
Hosila1::x=20
Hosila2::x=30
```

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 majburiy emas.

Asosiy sinfdagi virtual funksiyalar shu sinfda aniqlanishi kerak, agar ular sof virtual deb e’lon qilingan bo‘lmasa. Hosilaviy sinfda e’lon qilingan funksiya asosiy sinfdagi virtual funksiyani qayta aniqlaydi, agar uning nomi virtual funksiya nomi bilan mos tushsa, ular bir xil miqdordagi va tiplari 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 asosiy sinfdagi o‘zi bilan bir xil prototipga ega virtual funksiyani qayta aniqlaydi.

```
#include <iostream.h>
class Asosiy{
  int x;
 public:
  virtual void Qiymat(int _x)  {  x=_x;
```

```
  cout<<"Asosiy::x = "<<x<<‘\n‘;  }
    virtual void Chop_Qilish(Asosiy * pOb)  {
  Qiymat(10); }
  };
  class Hosila: public Asosiy{
    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(Asosiy * pOb)  {
  Qiymat(15,20); }
  };
  int main(){
   Asosiy * pOb1=new Asosiy;
   Asosiy * pOb2=new Hosila;
   // Asosiy 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:

```
  Asosiy::x = 10
  Hosila::x =15 Hosila::y = 20
  Hosila::x =15 Hosila::y = 20
```

Keltirilgan misolda asosiy 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 asosiy sinfdagi mos virtual funksiyani qayta aniqlanishi deb qaraladi.

Virtual va novirtual funksiyalar. Quyidagi misol, ko‘rsatkich orqali chaqirilganda virtual va novirtual funksiyalar o‘zini qanday tutishi ko‘rsatilgan:

```
#include <iostream.h>
class Asosiy{
 public:
  virtual void
Virtual_Fun(){cout<<"Asosiy::Virtual_Fun()\n"; }
  void
NoVirtual_Fun(){cout<<"Asosiy::NoVirtual_Fun()\n";}
};
class Hosila: public Asosiy{
 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;
 Asosiy * pAsosiy = &hosila;
  // Hosila sinfidan Virtual_Fun() funksiyasini
chaqirish
 pAsosiy->Virtual_Fun();
  //Asosiy sinfidan NoVirtual_Fun() funksiyasini
chaqirish
 pAsosiy->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 natija chiqadi:

```
Hosila::Virtual_Fun()
Asosiy::NoVirtual_Fun()
Hosila::Virtual_Fun()
Hosila::NoVirtual_Fun()
```

Shunga e’tibor berish kerakki, Virtual_Fun() funksiyasi qaysi sinfga -Asosiy yoki Hosila ko‘rsatkich orqali chaqirilishidan qat’iy nazar Hosila sinfidan Virtual_Fun() funksiyasi chaqiriladi. Bunga sabab - Virtual_Fun() funksiyasi virtual va pAsosiy hamda pHosila ko‘rsatkichlari Hosila turidagi obyektga ko‘rsatadi. Ikkinchi tomondan, asosiy sinfga ko‘rsatkichi pAsosiy garchi novirtual funksiyalarga ega hosilaviy sinf obyektiga ko‘rsatsa ham, asosiy sinfdagi mos funksiyani chaqiradi.

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

```
#include <iostream.h>
class Asosiy{
 public:
  virtual void
Virtual_Fun(){cout<<"Asosiy::Virtual_Fun()\n";}
};
class Hosila: public Asosiy{
 public:
  virtual void
Virtual_Fun(){cout<<"Hosila::Virtual_Fun()\n";}
};
int main(){
Asosiy * pAsosiy =new Hosila;
 // Hosila sinfidan Virtual_Fun() funksiyasini
chaqirish
 pAsosiy->Virtual_Fun();
 //Asosiy sinfidan Virtual_Fun() funksiyasini
chaqirish
 pAsosiy->Asosiy::Virtual_Fun();
 return 0;}
```

Dastur ekranga quyidagilarni chop etadi.

```
Hosila::Virtual_Fun()
Asosiy::Virtual_Fun()
```

Ko‘rinib turibdiki,

pAsosiy->Asosiy::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 Asosiy{
 public:
  Asosiy(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 Asosiy{
 public:
  Hosila1(char * nom):Asosiy(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(){
   Asosiy asosiy("Asosiy");
   Hosila1 hosila1("Hosila1");
   Hosila11 hosila11("Hosila11");
   asosiy.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 Asosiy::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. Asosiy sinfida Fun(char) virtual funksiyasi e’lon qilingan. Hosila1 sinfi Fun(char) virtual funksiyasining o‘z variantini va ikkita qayta yuklanuvchi novirtual Fun(const char*) va Fun(int) funksiyalarini 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) funksiyalarini 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 funksiyalari mavjudligi sababli yashiringan bo‘ladi.

Agar voris sinflardagi virtual funksiyalar e’lonlari o‘chirilsa, belgi argumentli funksiya chaqirishda, amalda Hosila1 sinfidigi Fun(int) funksiyasini chaqirish ro‘y beradi. Asosiy sinfidagi virtual funksiyani chaqirish zarur bo‘lsa, ko‘rish sohasiga ruxsat berish amalidan foydalanish mumkin:

Hosila1.Asosiy::Fun(‘Y‘);

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 ma’lumotlarni saqlash va qayta ishlashni amalga oshiruvchi tuzilmalar hisoblanadi. Dasturda bog‘langan ro‘yxatni yaratish, unga qiymat joylashtirish va o‘chirishni amalga oshiruvchi *Ruyxat* asosiy 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<<"Joylshtirish? 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 chiqariladi. Dinamik polimorfizm *Ruyxat* ko‘rsatkichi *Navbat* yoki *Stek* obyektlariga ko‘rsatishiga mos ravishda virtual *Olish()* va *Joylash()* funksiyalarini chaqirishida namoyon bo‘ladi.

Virtual destruktorlar. Konstruktorlar virtual bo‘lmaydi, lekin destruktorlar virtual bo‘lishi mumkin va aksariyat holatlarda shunday bo‘ladi. Asosiy sinfiga ko‘rsatkich hosilaviy sinf obyektiga ko‘rsatib turganda, agar destruktor virtual qilib e’lon qilingan bo‘lsa, hosilaviy sinf destruktori chaqiriladi. Hosilaviy sinf destruktori o‘z navbatida asosiy sinf destruktorini chaqiradi va obyekt to‘g‘ri (to‘laligicha) o‘chiriladi. Aks holda ko‘rsatkich turiga mos ravishda asosiy sinf destruktori 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 Asosiy{
  int * px;
```

```
 public:
  Asosiy(int _x) {px=new int; *px=_x; }
 /*virtual*/~Asosiy() { cout<<"Asosiy sinf
destruktori ishladi!\n"; delete px; }
};
class Hosila: public Asosiy{
  int * pxx;
 public:
  Hosila (int n):Asosiy(n) { pxx=new int;
*pxx=n*n;  }
 ~Hosila() { cout<<"Hosila sinf destruktori
ishladi!\n";  delete pxx; }
};
int main(){
  Asosiy * pAsosiy=new Hosila(5);
  delete pAsosiy;
  return 0;}
```

Hosilaviy sinf - Hosila sinfida destruktor virtual deb e’lon qilinmagan va delete pAsosiy;

til ko‘rsatmasi bajarilishi natijasida ekranga

Asosiy sinf destruktori ishladi!

xabari chiqadi. Bu holat xotiradagi Hosila sinf obyekti uchun ajratilgan xotira bo‘shatilmay qolganligini bildiradi. Agar asosiy sinf destruktorini virtual deb e’lon qilinsa, obyektni o‘chirish to‘g‘ri ro‘y beradi - oldin hosilaviy sinf destruktori, keyin asosiy sinf destruktori bajariladi. Dasturning ekranga chiqaradigan xabarlari buni isbotlaydi.

```
Hosila sinf destruktori ishladi!
Asosiy sinf destruktori ishladi!
```

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. Asosiy 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 shajarisini yaratish zarur bo‘lsin va asosiy sinf umumiy funksional imkoniyatlarni ta’minlashi kerak. Lekin, asosiy sinfi shu darajada umumlashgan bo‘lib, natijada undagi ayrim funksiyalarni aniqlashtirish imkoni bo‘lmasligi mumkin.

Misol uchun jonivorlar shajarasini tavsiflovchi Jonivor abstrakt asosiy 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 aniqlashtirishning imkoni yo‘q, chunki Jonivor sinfi hayvonlar shajarasini aniqlab beruvchi, umumlashtiruvchi sinf va aniq hayvon aniqlanmaguncha uning ovozining qanday bo‘lishi va nima bilan oziqlanishini bilib bo‘lmaydi. Lekin, aksariyat hayvonlar ovoz chiqaradi va albatta oziqlanadi. Shu sababli, ovozi() va oziqasi() funksiyalar umumiy bo‘lib, u Jonivor sinfida mavhum holda e’lon qilingan. Bu funksiyalar Kuchuk va Mushuk sinflarida aniqlashtirilgan (majburiy ravishda).

Dastur ishlashi natijasida ekranga quyidagilar chiqariladi:

```
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, balkim 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 Asosiy{
 public:
  Asosiy(){};
  virtual ~Asosiy()=0;    // Sof virtual destruktor
};
Asosiy::~Asosiy(){}   // Destruktorni aniqlash
class Hosila: public Asosiy{
 public:
  Hosila(){};
 ~Hosila(){};
};
void main(){
 Hosila * pHosila = new Hosila;
 delete pHosila;}
```

Ma’lumki, destruktor virtual bo‘lganda, oldin hosilaviy sinf destruktori, keyin asosiy sinf destruktori bajariladi. Sof virtual destruktorining aynan asosiy sinfda aniqlanishi, uning qandaydir amalga oshirilgan variantini yaratadiki, u destruktorlar 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 tipini oshkor ravishda abstrakt sinf tipiga keltirish mumkin emas;
- abstrakt sinf obyektini yaratib bo‘lmaydi;
- abstrakt sinfga ko‘rsatkich yoki adres olish amalini e’lon qilish mumkin.

Nazariy bilimlarni tekshirish uchun savollar

1. Kompilyatsiya jarayonidagi vaqtli va kechiktirilgan bog‘lanishlar tushunchalarini izohlab bering.
2. C++ tilida dinamik polimorfizmni amalga oshirish mexanizmi qanday? Sinflar shajarasida virtual va novirtual funksiyalar ishlatilishini tushuntiring.
3. Sinf destruktorini virtual qilib aniqlanishiga sabab nima? Sof virtual funksiya vazifasi nimadan iborat?
4. Inkapsulyatsiya nima? Inkapsulyatsiyadan foydalanishning maqsadi nimadan iborat? Inkapsulyatsiyaning qanday afzallik tomonlari bor?
5. Vorislik nima uchun kerak? Vorislikdan foydalanishning qanday afzalliklari mavjud? Vorislikning qanday turlari mavjud?
6. Voris olishning umumiy formasi qanday?
7. Yakka vorislikni misollar bilan tushuntiring. To‘plamli vorislikni misollar bilan tushuntiring.
8. Qanday ruxsat modifikatorlarini bilasiz?
9. Public ruxsat modifikatorini hayotiy misol bilan tushuntirib bering.
10. Private ruxsat modifikatorini hayotiy misol bilan tushuntirib bering.
11. Protected ruxsat modifikatorini hayotiy misollar bilan tushuntirib bering.
12. Himoyalangan protected va xususiy private huquqlari orasida qanday farq bor?

6.3.Operatorlarni qayta yuklash

📖 *C++ tilida operatorlarni qayta yuklash imkoniyati mavjud. Operatorlar global ravishda yoki sinf doirasida qayta yuklanishi mumkin. Qayta yuklangan operatorlar operator kalit so‘zi yordamida funksiya ko‘rinishida amalga oshiriladi.*

🖉 ***Kalit so‘zlar.*** *Preprotsessor belgilari, shart amali, sinf doirasi, adressni olish, Unar plyus, inkriment, binar operatorlari.*

REJA:

1. Operatorlarni qayta yuklash, nusxa olish konstruktorlari va argumentlari.
2. Kiritish va chiqarish operatorlarini qayta yuklash. Kiritish va chiqarish universial funksiyalari.
3. Formatli kiritish va chiqarish. Kiritish va chiqarish manipulyatorlari.
4. Sinf a’zolariga murojaat operatorlarini qayta yuklash

C++ tilida operatorlarni qayta yuklash imkoniyati mavjud. Operatorlar global ravishda yoki sinf doirasida qayta yuklanishi mumkin. Qayta yuklangan operatorlar operator kalit so‘zi yordamida funksiya ko‘rinishida amalga oshiriladi. Qayta yuklanuvchi funksiya *operator funksiya* nomlanadi va nomi operatorX ko‘rinishida bo‘lishi kerak, bu yerda 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 chaqirish
nuqta3=nuqta1.operator+(nuqta2);
```

6.1–jadval. Qayta yuklanuvchi operatorlar

Operator	Tavsifi	Tipi
,	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‘lish qiymat 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 kiluvchi YOKI qiymat berish bilan	Binar
|	Razryadli YOKI	Binar
|=	Razryadli YOKI qiymat berish bilan	Binar
||	Mantiqiy YOKI	Binar
~	Bitli mantiqiy INKOR	Binar
delete	Dinamik obyektni yo‘qotish	–

new	Dinamik obyektni yaratish	–

6.2–jadvalda keltirilgan operatorlar qayta yuklanmaydigan operatorlar hisoblanadi.

6.2–jadval. Qayta yuklanmaydigan operatorlar

Operator	**Tavsifi**
.	A’zoni tanlash
::	Ko‘rinish sohasiga ruxsat berish operatori
.*	Ko‘rsatkich bo‘yicha a’zoni tanlash
?:	Shart amali
#	Preprotsessor belgilari
##	Preprotsessor belgilari

Qayta yuklanadigan operatorlarning operator funksiyalari, new va delete operatorlaridan tashqari, quyidagi qoidalardan biriga bo‘ysunishi kerak:

- operator funksiya sinfning nostatik funksiya–a’zosi bo‘lishi;
- operator funksiya sinf yoki sanab o‘tiladigan tipdagi argument qabul qilishi;
- operator funksiya sinf yoki sanab o‘tiladigan tipga ko‘rsatkich yoki murojaat bo‘lgan argumentlarni qabul qilishi.

Masalan,

```
class Nuqta{
  public:
  //«kichik» operatori uchun operator funksiya-
a’zoni
  // e’lon qilish
  Nuqta operator<(Point&);
  …
  // Qo‘shish operatorlarini e’lon qilish
  friend Nuqta operator+(Point&, int);
  friend Nuqta operator+(int, Point&);
};
```

Bu misolda «kichik» 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 parametrga ega bo‘lmasligi kerak. Agar operator funksiya global funksiya bo‘lsa, u faqat bitta parametrga ega bo‘ladi;

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

5) operator funksiya kelishuv bo‘yicha parametrlarga ega bo‘lmasligi kerak;

6) sinf funksiya a’zosi sifatida e’lon qilingan operator funksiyaning birinchi parametri (agar u 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 nostatik funksiya a’zolari bo‘lishi kerak (va ular global funksiya bo‘la olmaydi).

Operatorlarni qayta yuklash orqali, sinf doirasida 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.

Binar operatorlarni qayta yuklash. Binar operatorning operator funksiyasi sinfning nostatik funksiya–a’zosi sifatida e’lon qilinganda u quyidagi sintaksisga ega bo‘lishi kerak:

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

Bu yerda <qaytariladigan qiymat tipi > – funksiya qaytaradigan qiymat tipi, X– qayta yuklanadigan operator, <parametr tipi > –parametr tipi va <parametr> – funksiya parametri.

Funksiya parametriga operatorning o‘ng tomonidagi obyekt uzatiladi, operatorning chap tomonidagi obyekt esa nooshkor ravishda this ko‘rsatkichi bilan uzatiladi.

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

<qaytariladigan qiymat tipi>operatorX(<parametr tipi$_1$> <parametr$_1$>, <parametr tipi$_2$><parametr$_2$>);

Bu yerda funksiya parametrlarining kamida bittasi operator qayta yuklanayotgan sinf tipida bo‘lishi kerak.

Garchi operator funksiya qaytaradigan qiymat tipiga hech qanday cheklov bo‘lmasa ham, u sinf tipida yoki sinfga ko‘rsatkich bo‘ladi.

Operator funksiyalarni 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: "<<"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: "<<"A.x="<<x<<" A.y="<<y<<endl;
    return 0;}
```

Dastur ishlashi natijasida ekranga quyidagi koʻrinishidagi natijalar chiqariladi:

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

Dasturda shu narsaga e'tibor berish kerakki, operator funksiya parametri sinf obyektga murojaat koʻrinishida aniqlangan. Umuman olganda argument sifatida obyektni oʻzini ham chaqirish mumkin, lekin funksiyadan chiqishda bu obyekt destruktor yordamida yoʻqotiladi. Funksiya parametri sinf obyektga murojaat koʻrinishida boʻlishining afzalligi shundaki, funksiya chaqirilganda unga obyekt emas, balki obyektga koʻrsatkich uzatiladi va sinf nusxasi uchun chaqiriladigan destruktorni ishlatilmaydi. Operator funksiyalarning qaytaruvchi qiymati ayni shu sinf tipida va hol obyektlarni nisbatan murakkab ifodalarda qoʻllash imkonini beradi. Masalan, quyidagi amallar dastur uchun ruxsat etilgan til koʻrsatmasi hisoblanadi:

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

Ikkinchi tomondan, quyidagi ifoda ham oʻrinli:

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

Bu ifodada qoʻshish operatorining operator funksiyasidagi vaqtincha (OraliqOb) obyektining Nuqta_Qiymati() funksiyasi ishlatiladi.

Keyingi misol operator funksiya parametri sifatida sanab oʻtiladigan tipdagi berilgan kelgan holatini namoyon qiladi. Bu berilgan operatorning oʻng 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 sinf turidagi obyekt bo‘lgan
      // qayta yuklangan qo‘shish operatori amal
qiladi
   C.Nuqta_Qiymati(x,y);
   cout<<" C=A+B: "<<"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: "<<"C.x="<<x<<" C.y="<<y<<endl;
   return 0;
}
```

Dastur ishlashi natijasida ekranga quyidagi ko‘rinishidagi natijalar chiqariladi:

```
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: "<<"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.

Taqqoslash va mantiqiy operatorlarni qayta yuklash. Taqqoslash va mantiqiy operatorlar garchi binar operatorlar bo‘lsa ham ular alohida-alohida qaraladi. Chunki ularga mos keluvchi operator-funksiyalar o‘zlari aniqlangan sinf tipini 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 chiqariladi:

```
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.

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 funksiyasi 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, bunday qiymat berish operatori ishlamaydi.

Qayd etish kerakki, qiymat berish operatori bajarilganidan keyin chap tomondagi operand o‘zgaradi, chunki unga yangi qiymat beriladi. Shuning uchun qiymat berish operatorining operator funksiyasi chaqirilgan obyektga ko‘rsatkichni qaytarishi shart. Buning uchun funksiya nooshkor ravishda sinf funksiyalariga birinchi parametr sifatida uzatiladigan this ko‘rsatkichini qaytarishi yetarli. 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 quyidagi natija chiqariladi.

```
Nuqta2 va Nuqta3 o‘zaro teng.
Nuqta1 va Nuqta3 o‘zaro teng.
Nuqta3.x=10 Nuqta3.y=20
```

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 joylashtiradi. 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.

Unar operatorlarni qayta yuklash. Unar operatorlar uchun faqat bitta operand kerak bo‘ladi. Unar operatorni sinfning funksiya–a’zosi ko‘rinishida qayta yuklashda bu yagona operand bu amalni chaqirgan obyektning o‘zi hisoblanadi. Shu sababli, unar operatorning operator funksiyasi nostatik funksiya-a’zo sifatida e’lon qilinadi va u quyidagi ko‘rinishga ega bo‘ladi:

<qaytaruvchi qiymat tipi > operatorX();

bu yerda <qaytaruvchi qiymat tipi > funksiya qaytaradigan qiymat tipi, X– qayta yuklanayotgan unar operator.

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

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

bu yerda <parametr tipi> – parametr tipi va <parametr> – funksiya parametri.

Plyus va minus unar operatorlarni 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+();
  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 plyus operatorini
chaqirish
 N1.Qiymat_xy(a,b);
 cout<<"N1.x=”<<a<<” N1.y=”<<b<<endl;
 N1=-N1;      //qayta yuklangan plyus operatorini
chaqirish
 N1.Qiymat_xy(a,b);
 cout<<"N1.x=”<<a<<” N.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.

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 tipidagi 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 operatorning postfiks ko‘rinishi 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<<” (++N1).y=”<<b<<endl;
 N1++;    //postfiks inkrement operatorini
chaqirish
 N1.Qiymat_xy(a,b);
 cout<<"(N1++).x=”<<a<<” (N1++).y=”<<b<<endl;
 N3=--N2; //prefiks dekrement operatorini chaqirish
 N3.Qiymat_xy(a,b);
 cout<<"N3=--N2; => N3.x="<<a<<" N3.y="<<b<<endl;
 //postfiks dekrement operatorini chaqirish
 (N3=N1--).Qiymat_xy(a,b);
 cout<<"(N3=N1--).x="<<a<<" (N3=N1--).y="<<b<<endl;
 N1.Qiymat_xy(a,b);
 cout<<"N1.x="<<a<<" N1.y="<<b<<endl;
 N2.Qiymat_xy(a,b);
 cout<<"N2.x="<<a<<" N2.y="<<b<<endl;
 return 0;
}
```

Dastur ishlashi natijasida ekranga quyidagi natijalar chiqadi.

```
(++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
```

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 obyektni 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 murojaat 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
  //postfiks inkrement
  friend Nuqta operator++(Nuqta&,int);
  friend Nuqta & operator--(Nuqta&); // prefiks
dekrement
  // postfiks dekrement
  friend Nuqta operator--(Nuqta&int);
};
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.x--;
 ob.y--;
 return ob;
}
Nuqta operator--(Nuqta &,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<<” (++N1).y=”<<b<<endl;
 N1++;    //postfiks inkrement operatorini
chaqirish
 N1.Qiymat_xy(a,b);
 cout<<"(N1++).x=”<<a<<” (N1++).y=”<<b<<endl;
```

```
   N3=--N2; //prefiks dekrement operatorini chaqirish
   N3.Qiymat_xy(a,b);
   cout<<"N3=--N2; => N3.x="<<a<<" N3.y="<<b<<endl;
   //postfiks dekrement operatorini chaqirish
   (N3=N1--).Qiymat_xy(a,b);
   cout<<"(N3=N1--).x="<<a<<" (N3=N1--).y="<<b<<endl;
   N1.Qiymat_xy(a,b);
   cout<<"N1.x="<<a<<" N1.y="<<b<<endl;
   N2.Qiymat_xy(a,b);
   cout<<"N2.x="<<a<<" N2.y="<<b<<endl;
   return 0;}
```

Dastur ishlashi natijasida ekranga xuddi oldingi misoldagidek natija chiqariladi.

Yuqorida qayd qilinganidek, int tipidagi argument odatda ishlatilmaydi, lekin zarur bo‘lganda ishlatilishi mumkin. Bu argumentni ishlatishga 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);

ifodasini operatorgacha bo‘lgan qismini alohida ajratib

N1++;

ko‘rsatma deb tushunadi.

Indekslash operatorini qayta yuklash. Kvadrat qavslar (‘[‘,’]‘) bilan yoziladigan indekslash operatori binar operator hisoblanadi va u qayta yuklanishida operator funksiya sinfning bitta argumentli nostatik funksiya–a’zosi sifatida aniqlanishi kerak. Funksiya argumenti ixtiyoriy tipda bo‘lishi mumkin va u sinf obyektlari massivining indeksi deb qabul qilinadi. Quyidagi misol buni namoyon qiladi:

```
#include <iostream.h>
class BS_Massiv{
  int MaxIndex;
  int * kButun;
 public:
  BS_Massiv(int Elem_Soni)
 ~BS_Massiv(){delete kButun;}
  int & operator[](int index);
};
BS_Massiv::BS_Massiv(int Elem_Soni){
 kInt=new int(Elem_Soni);
 MaxIndex=Elem_Soni;
}
int & BS_Massiv::operator[](int index){
 static int iXato=-1;
 if(index>=0 && index<MaxIndex) return kInt[index];
 else  {
  cout<<”Xato: Massiv chegarasidan chiqish ro‘y
berdi!”;
  cout<<endl;
  return iXato; }
}
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 tipidagi elementlardan tashkil topgan vector massivi BS_Massiv sinfining obyekti 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 chiqish ro‘y berdi!
vector[5]=-1
```

Shunga e’tibor berish kerakki, operator[] funksiyasi murojaat qaytaradi (son qiymatini emas) va shu sababli bu funksiyani qiymat berish operatorining ikki tomonida ham qo‘llash imkoni yuzaga keladi.

Funksiyalarni qayta yuklash. Qavslar vositasida amalga oshiriladigan funksiyani chaqirish operatori quyidagi sintaksisga ega bo‘lgan binar operator hisoblanadi:

<ifoda>(<ifodalar ro‘yxati>)

Bu yerda <ifoda> – birinchi operand, hamda <ifodalar ro‘yxati> – majburiy bo‘lmagan ikkinchi operanddir. Funksiyani chaqirish operatorining operator funksiyasi sinfning nostatik funksiya–a’zo ko‘rinishida e’lon qilinishi kerak. Funksiyani chaqirish operatorini qayta yuklashga zarurat, odatda ko‘p parametrni 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 chaqirilish 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;
 // Nuqta sinfining qayta yuklangan funksiyani
chaqirish
 // operatorining operator funksiyasini N1 obyekt
uchun
 // chaqirish.
 N2=N1(5,10);
 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);** ifodalarini chalkashtirmaslik kerak. Birinchi ifoda sinfning parametrli konstruktoriga murojaatni anglatsa, ikkinchisi – sinfdagi funksiyani chaqirish operatori qayta yuklangan operator funksiyani chaqirish kerakligini bildiradi. Misol shuni ko‘rsatadiki, funksiyani chaqirish operatori qayta yuklangan sinf obyektiga funksiya vositasida murojaat qilish mumkin.

Sinf a’zolariga murojaat operatorlarini qayta yuklash. Sinf a’zolariga murojaat operatorlarini qayta yuklash a’zolarni tanlash operatorini (->) qayta yuklash orqali amalga oshiriladi («.» operatori qayta yuklanmaydi). Bu operator unar hisoblanadi va uning operator funksiyasi sinfning nostatik a’zosi qilib e’lon qilinishi kerak. Mos operator funksiyaning ko‘rinishi quyidagicha:

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

Bu yerda <sinf nomi> – operator tegishli bo‘lgan sinf nomi. Sinf a’zolarini tanlash operatorini qayta yuklash, odatda qo‘llanishi ayni

paytda o‘rinlimi yoki yo‘qligini nazorat qiluvchi «aqlli» 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<<”Ob‘ect elementiga murojaat: ”;
 return this;
}
int main(){
 Nuqta N(5,10);
 // Nuqta sinfining qayta yuklangan a’zolarini
tanlash
 // operatorining operator funksiyasini N obyekt
uchun
 // chaqirish.
 cout<<”N->x = ”<<N->Qiymat_X()<<endl;
 cout<<”N->y = ”<<N->Qiymat_Y()<<endl;
 return 0;}
```

Dastur bajarilganda, ekranga quyidagi satrlar chop etiladi:

Ob‘yekt elementiga murojaat: N->x = 5
Ob‘yekt elementiga murojaat: N->y = 10

Kompilyator tomonidan “N->Qiymat_X()” ko‘rsatmasini

(N.operator->())->Qiymar_X()

ko‘rinishida talqin qilinishi operator funksiyani qanday bajarilishining ichki mohiyatini ochib beradi.

New va delete operatorlarini qayta yuklash. Xotirani dinamik ravishda ajratish va tozalash vazifasini bajaruvchi *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);

<sinf nomi>:: operator new() - sinf funksiyasi;

<sinf nomi>:: operator delete() - sinf funksiyasi.

Sinfda aniqlangan *operator new()* operator funksiya sinfning 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* tipidagi parametrga ega bo‘lishi kerak. Oxirgi parametr «stddef.h» sarlavha faylida *unsigned int* ko‘rinishida aniqlangan. *new* operatorini qayta yuklash uchun quyidagi prototipdan foydalaniladi:

```
void*operator new(size_t size); // obyektlar uchun
void*operator new[](size_t size);//obyektlar massivi uchun
```

Bu prototiplar «*new.h*» sarlavha faylida joylashgan. Shu sababli, agar *new* va *delete* operatorlarni qayta yuklash zarur bo‘lganda bu faylni dasturga qo‘shish kerak.

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 asosiy 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* foydalanishi mumkin. Mos operator funksiya sintaksisi quyidagi ko‘rinishiga ega:

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

Odatda *new* operatorining joylashuvchi shaklidan global obyektlar uchun, tipiga keltirish amali bajarilgan holda qo‘llaniladi. Bu variantda absolyut adres bo‘yicha oldindan ajratilgan joyga obyektni joylashtirish amalga oshiriladi. Ko‘rsatilgan adres uyumdan bo‘lishi shart emas. Joylashadigan obyekt o‘lchami ko‘rsatilgan soha 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 <tip uzunligi> < tip nomi> <(<initsializator>)> yoki

<::> new < tip uzunligi> (<tip nomi>) <(<initsializator>)>

Bu yerda

<::> - shart bo‘lmagan, ko‘rinish sohasiga ruxsat berish operatori;

< tip uzunligi> - operator *new()* funksiyasining *size* parametri, ko‘rsatilmasligi mumkin;

< tip nomi> - xotira ajratiladigan ma’lumotning tipi;

<(<initsializator>)> - < tip nomi> tipining konstruktori uchun uzatiladigan boshlang‘ich qiymatlar ro‘yxati. Ro‘yxat ko‘rsatilmasligi 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:

```
#include <iostream.h>
#include <stdio.h>
// Global new operatorini qayta aniqlash
void* operator new(size_t size) {
 cout<<”Xotiradan “<<size;
```

```
   cout<<” 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;
   // Global new operatorini chaqirish
     long * pInt = new long;
     // Global delete operatorini chaqirish
     delete pInt;
     // Qayta yuklangan new operatorini chaqirish
     pInt = new(fayl_nomi, qator) long;
     // Global delete operatorini chaqirish
     delete pInt;
     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 *new* operatorining joylashuvchi shaklidan foydalanish ko‘rsatilgan.

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

```
  // Global new operatorini qayta aniqlash
  void* operator new(size_t sizeb 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”;
        cout<<” initsializatsiyalandi.\n“;
   cout<<“ x=”<<x<<” y=”<<y<<”\n”;
    }
       Nuqta(){
     x=0; y=0;
        cout<<” Obyektlarni kelishuv bo‘yicha”;
        cout<<” initsializatsiyalandi.\n“;
   cout<<“ x=”<<x<<” y=”<<y<<”\n”;
    }
    ~Nuqta() {cout<<”Nuqta::~Nuqta() ishladi\n”;}
  }
  // Global obyekt
  Nuqta nuqta;
  int main(){
   // Qandaydir ishni bajarish
   Tizimni_tekshirish();
   // Endi obyektni initsializatsiyalash mumkin!
   Nuqta * krst=new(nuqta) Nuqta(10,20);
   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 *new* operatorini qayta yuklashni sinf ichida joylashuvchi shaklidan foydalangan holda amalga oshirishga misol:

```
#include <iostream.h>
class Satr  {
  union   {
   char ch;
   char buf[81];
  };
 public:
  Satr(char c=‘\0‘): ch(c) {
      cout<<”Satr sinfining belgili
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(){
 // Satrni buferga joylashtirish
 Satr * krst=new(satr_buffer) Satr(“C++”);
 cout<<”satr_buffer”<<satr_buffer<<endl;
 // Destruktorni oshkor ravishda chaqirish
 krst->Satr::~Satr();
```

```
    // ‘c‘ belgisini satr boshiga joylashtirish
  krst=new(satr_buffer)Satr(‘c‘);
    cout<<”satr_buffer[0]=”<<satr_buffer[0]<<endl;
    cout<<”Buferning yangi qiymati”<<endl;
    cout<<”satr_buffer=”<<satr_buffer<<endl;
    // Destruktorni oshkor ravishda chaqirish
    krst->Satr::~Satr();
    return 0;  }
```

Dasturda global obyekt satr_buffer satri yaratiladi va shu xotira sohasiga Satr sinfi obyekti “C++” qiymati bilan joylashtiriladi. Oshkor ravishda sinf destruktori chaqirish orqali sinf obyekti yo‘qotiladi. Keyin, xuddi shu adresda Satr sinfining ikkinchi obyektni ‘c‘ belgisi bilan yaratiladi, xotiraning satr_buffer adresli sohasida satr qiymat chop etiladi va sinf obyekti yo‘qotiladi.

Dastur ishlashi natijasida ekranda quyidagi xabarlar chop etiladi.

```
Satr sinfining satrli konstruktori
satr_buffer=C++
Satr::~Satr()
Satr sinfining belgili konstruktori
satr_buffer[0]=c
Bufferning yangi qiymati
satr_buffer=c++
Satr::~Satr()
```

Nazariy bilimlarni tekshirish uchun savollar.

1. Operatorlarni qayta yuklash nima? Operatorlarni qayta yuklash afzalliklari nimada? Operatorlarni qayta yuklash qanday amalga oshiradi va nimaga?
2. Qayta yuklash nima? Operatorlarni qayta yuklash qaysi operatorlar uchun mumkin emas?
3. Operatorlarni qayta yuklash kamchilklari va yutuqlari nimada?
4. Operatorlarni qayta yuklash nima? Operatorlarni qayta yuklash afzalliklari nimada?
5. Operatorlarni qayta yuklash qanday amalga oshiradi va nimaga?
6. Funksiyalarni qayta yuklash nima uchun kerak?
7. Qayta yuklash jarayoni nimadan tashkil topadi? Qanday operator turlarini bilasiz?
8. Sinfdan olingan obyektga qayta yuklashni nima dahli bor? Kiritish chiqarish operatorlarni ham qayta yuklab bo‘ladimi?

9. Qanday operatorlarni qayta yuklab bo‘lmaydi? Eng ko‘p qaysi operator qayta yuklanadi?

10. Bitta sinfdan olingan obyektlar uchun operatorlarni qayta yuklash shartmi?

6.4. Shablonlar bilan ishlash

📖 *C++ da shablonlar yaratish usullariga bag‘ishlangan. Shablonlar va ularning qo‘llanilishi, funksiya shabloni va sinf shabloni hamda ularning qo‘llanilish usullari, muammolari, murakkabliklari batavfsil yoritilgan. Funksiya va sinf shablonlari ko‘pgina holatlarni inobatga olgan holda amaliy masalalar yordamida aniq va to‘liq tushuntirilgan.*

✎ ***Kalit so‘zlar.*** *Template, template class, template function, template prefix, type paramete*

REJA:

1. Shablon (template) tushunchasi va ularning qo‘llanilishi.
2. Funksiya shablonlarini yaratish usullari.
3. Sinf shablonlari yaratish usullari.
4. Funksiya va sinf shablonlarining bog‘liqlik tomonlari.

C++ da umumiy tiplardan foydalangan holda, shablon funksiyalar va sinflarni aniqlashimiz mumkin. C++ tili qayta foydalaniluvchi dasturiy ta’minotni ishlab chiqish uchun shablon funksiyalar va sinflar bilan ta’minlaydi. Shablonlar funksiyalar va sinflarda tiplarni muvofiqlashtirish (sozlash) qobiliyatini taqdim etadi. Bunday qobiliyat bilan, kompilyator aniq bir tip o‘rnida qabul qila oladigan umumiy tip sifatida bitta funksiya yoki bitta sinfni aniqlashimiz mumkin. Masalan, biz umumiy tipdagi ikkita sondan kattasini topish uchun bitta funksiyani aniqlashimiz mumkin. Agar bu funksiyani ikkita **int** argumentlar orqali chaqirsak, umumiy tip **int** tipi bilan almashadi. Agar bu funksiyani ikkita **double** argumentlar orqali chaqirsak, umumiy tip **double** tipi bilan almashadi.

Mazkur va bundan keying mavzularda shablonlar tushunchasi yoritib beriladi va siz qanday qilib funksiya shablonlari yoki sinf shablonlarini aniqlashni hamda ularni aniq tiplar bilan ishlatishni o‘rganib olishingiz mumkin. Shuningdek, ko‘p qo‘llaniluvchi, massivlarni almashtirishda qo‘llashingiz mumkin bo‘lgan umumiy **vector** shablonlarini ham o‘raganishingiz mumkin.

Shablonlar nazariyasi. Shablonlar sinflar va funksiyalarda tiplarni muvofiqlashtirish imkonini taqdim etadi. Biz funksiyalarni yoki sinflarni

kompilyator tomonidan aniq bir tip o‘rnida qabul qilinuvchi umumiy tip bilan aniqlashimiz mumkin.

Keling, shablonlarga bo‘lgan ehtiyojni ko‘rsatib berish uchun, oddiy bir misoldan boshlaymiz. Faraz qilaylik, biz ikkita butun sonlar, ikkita haqiqiy sonlar, ikkita belgilar va ikkita satrlardan kattasini topmoqchimiz. Shu kungacha o‘rgangan bilimlarimiz asosida, biz quyidagicha ko‘rinishdagi to‘rtta ko‘p yuklanuvchi funksiyalarni aniqlashimiz mumkin:

```
int maxValue(int value1, int value2){
if (value1 > value2)
return value1;
else
return value2;
}
double maxValue(double value1, double value2){
if (value1 > value2)
return value1;
else
return value2;
}
char maxValue(char value1, char value2){
if (value1 > value2)
return value1;
else
return value2;
}
string maxValue(string value1, string value2){
if (value1 > value2)
return value1;
else
return value2;}
```

Bu funksiyalarning to‘rtalasi ham ulardagi qo‘llanilgan tiplarning har xil ekanliklari inobatga olinmasa, deyarli bir xil. Birinchi funksiya **int** tipini, ikkinchi funksiya **double** tipini, uchinchi funksiya **char** tipini va to‘rtinchi funksiya **string** tipini qo‘llaydi. Agar biz quyidagicha ko‘rinishda, umumiy tip bilan, bittagina, oddiy funksiyani aniqlasak, bir funksiyaning o‘zida barcha tiplarning saqlanib qolishi, ortiqcha bo‘sh joyning paydo bo‘lishi va dasturning osonlik bilan qayta sozlanishiga erishishimiz mumkin:

```
GenericType maxValue(GenericType value1,
GenericType value2){
```

```
  if (value1 > value2)
  return value1;
  else
  return value2;}
```

Mazkur **GenericType** (UmumiyTip) **int**, **double**, **char**, **string** kabi tiplarning barchasini qo‘llay oladi. C++ funksiya shablonlarini umumiy tip bilan aniqlash imkonini beradi. 6.4.1-kodli ro‘yxat umumiy tipdagi ikkita qiymatning kattasini topish uchun funksiya shablonini aniqlaydi.

6.4.1-kodli ro‘yxat. GenericMaxValue.cpp

```
1.  #include <iostream>
2.  #include <string>
3.  using namespace std;
4.  template <typename T>
5.  T maxValue(T value1, T value2){
6.  if (value1 > value2)
7.  return value1;
8.  else
9.  return value2;
10. }
11. int main()
12. {
13. cout << "1 va 3 ning kattasi: " << maxValue(1, 3)
    << endl;
14. cout << "1.5 va 0.3 ning kattasi: " "
15. << maxValue (1.5, 0.3) << endl;
16. cout << "‘A’ va ‘N‘ ning kattasi: "
17. << maxValue (‘A’, ‘N‘) << endl;
18. cout << " \"NBC\" va \"ABC\" ning kattasi: "
19. << maxValue (string("NBC"), string("ABC")) <<
    endl;
20. return 0;
21. }
```

Natija:

```
  1 va 3 ning kattasi: 3
  1.5 va 0.3 ning kattasi: 1.5
  ‘A’ va ‘N‘ ning kattasi: N
  "NBC" va "ABC" ning kattasi: NBC
```

Funksiya shablonining aniqlanishi parametrlar roʻyxati tomonidan berilgan **template** – kalit soʻzi bilan boshlanadi. Har bir parametr dastlab oʻzaro teng kuchli boʻlgan **typename** yoki **class** kalit soʻzi orqali, **<typename typeParameter>** yoki **<class typeParameter>** koʻrinishida beriladi. Masalan, 4-satrdagi

template<**typename** T>

maxValue uchun funksiya shablonining aniqlanishini boshlaydi. Shuningdek, bu satr *prefiks shablon* deb ham yuritiladi. Bu yerda **T** – parametr tipi. Katta **T** harfining faqat parametr tipini ifodalashda ishlatilishi kelishilgan.

maxValue funksiyasi 5-10 qatorlarda aniqlangan. Undan funksiya qaytaruvchi qiymat tipi, funksiya parametrlari yoki funksiyada e'lon qilingan oʻzgaruvchilarning tiplarini aniqlashda foydalanish mumkin. Kodning 13-19-qatorlarida **int**, **double**, **char** va **string** tiplari boʻyicha katta qiymatlilarni qaytarish uchun **maxValue** funksiyasi chaqirilgan. Funksiya **maxValue(1, 3)** koʻrinishda chaqirilganda, kompilyator argument tipi **int** ekanligini aniqlaydi va funksiyani aniq **int** tipida chaqirish uchun, **T –** parametr tipini **int** ga oʻzgartiradi. Funksiya **maxValue(string("NBC"), string("ABC"))** koʻrinishda chaqirilganda, kompilyator argument tipi **string** ekanligini aniqlaydi va funksiyani aniq **string** tipida chaqirish uchun, **T –** parametr tipini **string** ga oʻzgartiradi.

Agar 19-qatorda berilgan **maxValue(string("NBC"), string("ABC"))** ni **maxValue("NBC", "ABC")** deb oʻzgartirsak nima boʻladi? Unda biz funksiya **ABC** ni qaytarishini koʻrish uchun "syurpriz" tayyorlagan boʻlamiz. Nima uchun? Chunki **"NBC"** va **"ABC"** lar – C-satrlardir. **maxValue("NBC", "ABC")** ning chaqirilishi **"NBC"** va **"ABC"** larning manzillarini funksiya parametriga uzatadi. **value1 > value2** taqqoslash vaqtida ikkita massivning manzillari taqqoslanadi, tarkiblari emas.

Ogohlantirish. Umumiy **maxValue** funksiyasi ixtiyoriy tipdagi ikkita qiymatning kattasini topish uchun moʻljallangan boʻlib, quyidagicha shartlar asosida ishlaydi.

Masalan, agar siz bir qiymatni **int** tipida, ikkinchisini esa, **double** tipida bersangiz, kompilyatsion xatolik yuz beradi. Chunki kompilyator funksiyani chaqirishda mos tipni aniqlay olmaydi. Agar siz funksiyani **maxValue(Circle(1), Circle(2))** koʻrinishda chaqirsangiz, kompilyatsion xatolik yuz beradi. Chunki **Circle** sinfida > operatori aniqlanmagan.

Maslahat. Parametr tipini belgilashda **<typename T>** yoki **<class T>** dan foydalanishimiz mumkin. **<typename T>** dan foydalangan ma’qulroq, chunki **<typename T>** – tushunarlidir. **<class T>** ni esa, sinf aniqlanishi bilan adashtirib yuborish mumkin.

Eslatma. Ba’zi hollarda funksiya shabloni bittadan ko‘p parametrlarga ega bo‘lishi mumkin. Bunday vaziyatda parametrlarni **<typename T1, typename T2, typename T3>** kabi, barchasini bitta uchburchak qavslar oralig‘iga, vergullar bilan ajratilgan holda joylashtiriladi.

6.3-kodli ro‘yxatdagi asosiy funksiya parametrlari qiymat qabul qilib oluvchi sifatida aniqlangan. Uni havola qabul qilib oladigan qilib, 6.4.2-kodli ro‘yxatdagi kabi o‘zgartirishimiz mumkin.

6.3-kodli ro‘yxat. GenericMaxValuePassByReference.cpp

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
T maxValue(const T& value1, const T& value2)
{
if (value1 > value2)
return value1;
else
return value2;
}
int main(){
cout << "1 va 3 ning kattasi: " << maxValue(1, 3)
<< endl;
cout << "1.5 va 0.3 ning kattasi: "
<< maxValue(1.5, 0.3) << endl;
cout << "‘A’ va ‘N‘ ning kattasi: "
<< maxValue(‘A’, ‘N‘) << endl;
cout << " \"NBC\" va \"ABC\" ning kattasi: "
<< maxValue(string("NBC"), string("ABC")) << endl;
return 0;}
```

Natija:

```
1 va 3 ning kattasi: 3
1.5 va 0.3 ning kattasi: 1.5
‘A’ va ‘N‘ ning kattasi: N
"NBC" va "ABC" ning kattasi: NBC
```

Misol: Umumiy tip. Qismda umumiy tipli funksiya aniqlanadi.

O‘tgan mavzularda ko‘rib chiqilgan kodli ro‘yxatdagi *TanlabSaralash.cpp* dasturida **double** tipidagi elementlardan tashkil topgan massivni saralovchi funksiya berilgan edi. Bu yerda o‘sha funksiya nusxasi keltirilgan:

```
1.  void tanlabSaralash(double list[], int listHajm)
2.  {
3.  for (int i = 0; i < listHajm - 1; i++){
4.  // list[i..listHajm-1] dagi minimumni topish
5.  double joriyMinimum = list[i];
6.  int joriyMinimumIndeks = i;
7.  for (int j = i + 1; j < listHajm; j++) {
8.  if (joriyMinimum > list[j]){
9.  joriyMinimum = list[j];
10. joriyMinimumIndeks = j;
11. }
12. }
13. // list[i] ni list[joriyMinimumIndeks] bilan
    almashtirish, agar zarur bo‘lsa;
14. if (joriyMinimumIndeks != i) {
15. list[joriyMinimumIndeks] = list[i];
16. list[i] = joriyMinimum;
17. } }
18. }
```

Bu funksiyani **int**, **char**, **string** va hokazo tiplardagi qiymatlardan iborat massivlarni saralashga mo‘ljallangan yangi funksiya hosil qilish uchun qayta sozlash oson. Bu tiplarning har biri uchun saralashni bajarish uchun koddagi **double** kalit so‘zini **int**, **char** yoki **string** kalit so‘zlari bilan almashtirish kerak (1- va 6-qator).

Bir qancha saralash funksiyalarini yozish o‘rniga, shunchaki, barcha tiplar uchun o‘rinli bo‘lgan, bir dona funksiya shablonini yozishimiz mumkin. 6.4.3-kodli ro‘yxatda massiv elementlarini saralash funksiyasi aniqlangan.

6.4.3-kodli ro‘yxat. GenericSort.cpp

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4. template <typename T>
5. void sort(T list[], int listSize) {
```

```
6. for (int i = 0; i < listSize; i++) {
7. // list[i..listHajm-1] dagi minimumni topish
8. T currentMin = list[i];
9. int currentMinIndex = i;
10.for(int j = i + 1; j < listSize; j++) {
11.if (currentMin > list[j]) {
12.currentMin = list[j];
13.currentMinIndex = j;
14.}}
15.// list[i] ni list[joriyMinimumIndeks] bilan
   almashtirish, agar zarur bo‘lsa;
16.if(currentMinIndex != i)
17.{list[currentMinIndex] = list[i];
18.list[i] = currentMin;}} }
19.template <typename T>
20.void printArray(const T list[], int listSize)
21.{for (int i = 0; i < listSize; i++)
22.{cout << list[i] << " ";}
23.cout << endl;}
24.int main(){int list1[] = {3, 5, 1, 0, 2, 8, 7};
25.sort(list1, 7);
26.printArray(list1, 7);
27.double list2[] = {3.5, 0.5, 1.4, 0.4, 2.5, 1.8,
   4.7};
28.sort(list2, 7);
29.printArray(list2, 7);
30.string list3[]={"Atlanta", "Denver", "Chicago",
   "Dallas"};
31.sort(list3, 4);
32.printArray(list3, 4);
33.return 0;}
```

Natija:

```
0 1 2 3 5 7 8
0.4 0.5 1.4 1.8 2.5 3.5 4.7
Atlanta Chicago Dallas Denver
```

Bu dasturda ikkita funksiya shaboni aniqlangan. **sort** funksiya shabloni (5-18-qatorlar) massiv elementlari tipini belgilab olish uchun **T** parametridan foydalanadi. **double** parametri umumiy **T** parametri bilan almashtirilganligi hisobga olinmaganda, bu funksiya *tanlabSaralash* funksiyasining o‘zginasi.

printArray funksiya shabloni (20-23-qatorlar) massiv elementlari tipini aniqlash uchun **T** – tip parametrdan foydalanadi. Bu funksiya massivdagi barcha elementlarni konsol oynada chop etadi.

main funksiya **int**, **double** va **string** qiymatlardagi massivlarni saralash uchun **sort** funksiyasini chaqiradi (25-,28-,31-qatorlar) va bu massivlartni chop etish uchun **printArray** funksiyasini chaqiradi (26-, 29-, 32-qatorlar).

Maslahat. Umumiy funksiyani aniqlaganingizdan avval, unga mos bo‘lgan oddiy funksiyani yozishdan boshlab, keyin uni umumiy funksiyaga o‘tkazsangiz yaxshiroq bo‘ladi.

Funksiya shablonlari (function template). Shablonlar yordamida umumiy funksiyalar va umumiy sinflar yaratish imkoniyati mavjud. Umumiy funksiya va umumiy sinflar har xil ma’lumot tiplaridan ularni **overload** qilmasdan (ko‘p kod yozmasdan) foydalanish imkoniyatini beradi. Ya’ni bunda biz har bir tip uchun alohida funksiya yozishimiz shart bo‘lmaydi.

Shablonlar ikki xil bo‘ladi:

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

Funksiya shablonini (function template) yaratish.

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

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

Bu yerda **TIP** funksiya tomonidan joriy holda ishlatiladigan ma’lumot tipi. Ushbu tipni kompilyator avtomatik ravishda funksiyaga kelayotgan ma’lumot tipi bilan almashtirib qo‘yadi. Bu yerda **class** va **template** lar funksiya shablonini yaratish uchun ishlatiladigan kalit so‘zlardir. Lekin ba’zi hollarda **class** kalit so‘zi o‘rniga **typename** kalit so‘zini ishlatishimiz mumkin. Quyidagi misol xohlagan tipda berilgan ikkita o‘zgaruvchi o‘rnini almashtirib berish uchun xizmat qiladi va bunda har bir tip uchun alohida funksiya yozishimizga zaruriyat qolmaydi.

Funksiya shabloniga misol

```
#include <iostream>
using namespace std;
// Funksiya shabloni e’lon qilinishi...
```

```
template <class X> void swapargs(X &a, X &b){
X temp;
temp = a;
a = b;
b = temp;}
int main()
{int i=10, j=20;
double x=10.1, y=23.3;
char a=‘x‘, b=‘z‘;
cout << "Original i, j: " << i << ‘ ‘ << j << ‘\n‘;
cout << "Original x, y: " << x << ‘ ‘ << y << ‘\n‘;
cout << "Original a, b: " << a << ‘ ‘ << b << ‘\n‘;
swapargs(i, j); // swap funksiyasi butun tip uchun
(int)
swapargs(x, y); // swap funksiyasi haqiqiy tip
uchun (float)
swapargs(a, b); // swap funksiyasi simvol tip uchun
(char)
cout << "Swapped i, j: " << i << ‘ ‘ << j << ‘\n‘;
cout << "Swapped x, y: " << x << ‘ ‘ << y << ‘\n‘;
cout << "Swapped a, b: " << a << ‘ ‘ << b << ‘\n‘;
return 0;}
```

Umumiy funksiyaning boshqacha ko‘rinishi.

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

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

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

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

```
  a = b;
  b = temp;}
```

Funksiya shablonini override (qayta yozish) qilish.

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

Dastur natijasi:

```
  Original i, j: 10 20
  Original x, y: 10.1 23.3
  Original a, b: x z
```

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

Funksiya shablonini Override qilishning yangi usuli:

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

Funksiya shablonini overload qilish:

// Oddiy funksiyalardek, funksiya shablonini ham overload qilish mumkin.

```
#include <iostream>
using namespace std;
// f() funksiya shablonining birinchi tipi.
template <class X> void f(X a)
{
cout << "Inside f(X a)\n";
}
// f() funksiya shablonining ikkinchi tipi.
template <class X, class Y> void f(X a, Y b)
{cout << "Inside f(X a, Y b)\n";}
int main()
Original i, j: 10 20
Original x, y: 10.1 23.3
Original a, b: x z
int tipi uchun maxsus swapargs funksiyasi.
swapargs funksiya shabloni chaqirildi.
swapargs funksiya shabloni chaqirildi.
Swapped i, j: 20 10
Swapped x, y: 23.3 10.1
Swapped a, b: z x
{f(10); // calls f(X)
```

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

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

Umumiy sinflar (sinf shabloni)

Sinf shablonini e’lon qilishning umumiy formasi:

template <class TIP> class sinf_nomi{
}

Yoki quyidagi ko‘rinishda e’lon qilish mumkin

template <class TIP>
class sinf_nomi {
...
}

Sinf shablonini ishlatish

sinf_nomi <tip> obyekt;

Sinf shabloni uchun oddiy misol.

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

```
  return 0;}
```

Sinf shablonida ikki xil tipdan foydalanish

```
  #include <iostream>
  using namespace std;
  template <class Type1, class Type2> class myclass
  {
  Type1 i;
  Type2 j;
  public:
  myclass(Type1 a, Type2 b) { i = a; j = b; }
  void show() { cout << i << ‘ ‘ << j << ‘\n‘; }
  };
  int main()
  {
  myclass<int, double> ob1(10, 0.23);
  myclass<char, char *> ob2(‘X‘, "Templates add
  power.");
  ob1.show(); // show int, double
  ob2.show(); // show char, char *
  return 0;}
```

Dastur natijasi:

10 0.23

X Templates add power.

Maxsuslashtirilgan sinf shabloni

template<> konstruktori maxsusashtirilgan sinf shabloni uchun ishlatiladi.

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

```
  myclass(int a) {
  cout << "Inside myclass<int> specialization\n";
  x = a * a;
  }int getx() { return x; }};
```

Shablonning asosiy xususiyatlari:

• qayta-qayta foydalanish mumkin bo'lgan kod yozish imkonini beradi.

• Shablonlar yordamida framework lar yaratish mumkin

X Templates add power.

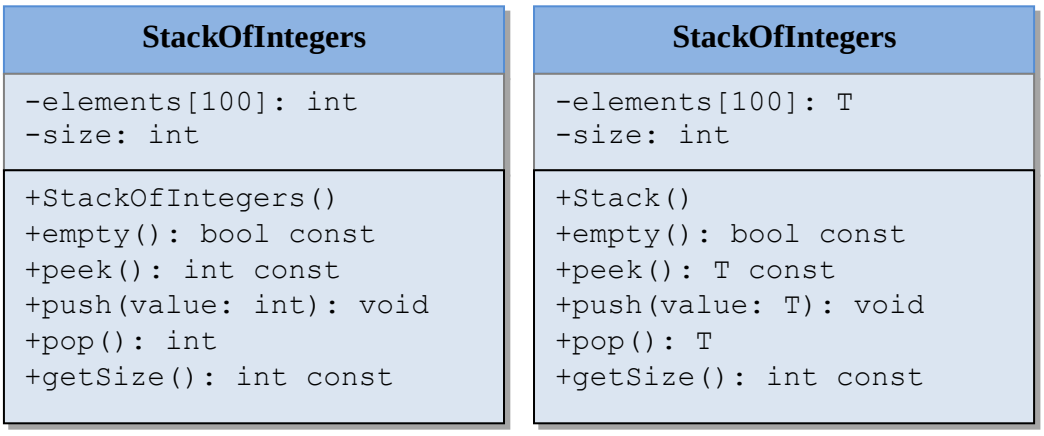
• Dastur kodi egiluvchanlik xususiyatiga ega bo'ladi.

• Turli xil tipdagi ma'lumotlar ustida ishlash uchun kod yozishda vaqtni tejash.

• C++ dagi STL lar (Standard Shablon Kutubxonalar), nomidan ko'rinib turibdiki, shablonlar yordamida yaratilgan

Sinf shablonlari (class template). Sinf uchun umumiy tipni aniqlash mumkin. Oldingi qismda funksiya uchun mo'ljallangan tip parametrga ega funksiya shabloni aniqlandi. Biz sinf uchun mo'ljallangan tip parametrga ega sinfni ham aniqlashimiz mumkin. Tip parametrlar sinfning tip shakllantiriladigan ixtiyoriy qismida foydalanilishi mumkin.

int qiymatlar uchun stek hosil qilish mumkin. Quyida, 6.4a-rasmdagi kabi, o'sha sinf nusxasi va uning UML sinf diagrammasi.



(a) (b)

6.4-rasm. Stack<T> – Stack sinfining umumiy versiyasi

```
  #ifndef STACK_H
  #define STACK_H
  class StackOfIntegers{
  public:
  StackOfIntegers();
```

```
  bool empty() const;
  int peek() const;
  void push(int value);
  int pop();
  int getSize() const;
  private:
  int elements[100];
  int size;};
  StackOfIntegers::StackOfIntegers()
  {size = 0;}
  bool StackOfIntegers::empty() const
  {return size == 0;}
  int StackOfIntegers::peek() const
  {return elements[size - 1];}
  void StackOfIntegers::push(int value)
  {elements[size++] = value;}
  int StackOfIntegers::pop()
  {return elements[--size];}
  int StackOfIntegers::getSize() const
  {return size;}
  #endif
```

Yuqoridagi koddan **int** kalit so‘zlarini **double**, **char**, yoki **string** bilan almashtirib, **double**, **char** va **string** qiymatlar uchun **StackOfDouble**, **StackOfChar** va **StackOfString** kabi sinflarni aniqlash uchun ushbu sinfni osongina o‘zgartirishlar kiritishimiz mumkin. Lekin, deyarli bir xil bo‘lgan bir nechta sinflarni aniqlamasdan, shunchaki ixtiyoriy tipdagi elemen uchun ishlaydigan bitta shablon sinfni aniqlashimiz mumkin. Yangi umumiy **Stack** sinfi uchun UML diagramma 6.4b-rasmda berilgan. 6.4-kodli ro‘yxatda umumiy tip elementlarini saqlashga mo‘ljallangan umumiy stek sinfi aniqlangan.

6.5-kodli ro‘yxat. GenericStack.h

```
1. #ifndef STACK_H
2. #define STACK_H
3. template <typename T>
4. class Stack
5. {public:
6. Stack();
7. bool empty() const;
```

```
8. T peek() const;
9. void push(T value);
10. T pop();
11. int getSize() const;
12. private:
13. T elements[100];
14. int size;};
15. template <typename T>
16. Stack<T>::Stack()
17. {size = 0;}
18. template <typename T>
19. bool Stack<T>::empty() const
20. {return size == 0;}
21. template <typename T>
22. T Stack<T>::peek() const
23. {return elements[size - 1];}
24. template <typename T>
25. void Stack<T>::push(T value)
26. {elements[size++] = value;}
27. template <typename T>
28. T Stack<T>::pop()
29. {return elements[--size];}
30. template <typename T>
31. int Stack<T>::getSize() const
32. {return size;}
33. #endif
```

Sinf shablonlari sintaksisi asosan funksiya shablonlari bilan bir xil. Xuddi funksiya shablonidagi kabi sinf aniqlanishidan oldin *template* qo‘shimchasi yozilishi kerak.

template <typename T>

Xuddi oddiy ma’lumot tipiga o‘xshab, sinfda tip parametri qo‘llanilishi mumkin. Bu yerda **T** tip **peek()**(8-qator), **push(T value)** (9-qator), va **pop()** (10-qator) funksiyani aniqlash uchun ishlatilgan. Shuningdek, **T** tip 16-qatorda, massiv elementlarini e’lon qilishda ham qo‘llanilgan.

Konstruktorlar va funksiyalar ularning o‘zlari shablonlar ekanliklari inobatga olinmaganda, oddiy sinflardagi kabi, bir xil yo‘l bilan aniqlanadi. Buni amalga oshirish uchun, *template* qo‘shimchasini

konstruktor yoki funksiyaning bosh qismidan avval joylashtirish kerak bo‘ladi. Masalan:

```
template <typename T>
Stack<T>::Stack()
{size = 0;}
template <typename T>
bool Stack<T>::empty()
{return size == 0;}
template <typename T>
T Stack<T>::peek()
{return elements[size - 1];}
```

Shu o‘rinda e’tibor qaratishimiz lozimki, **::** qarashlilik operatoridan oldin keladigan sinf nomi **Stack<T>**, **Stack** emas.

Maslahat. GenericStack.h sinf aniqlanishini va tadbiqini bitta faylga o‘tkazadi. Sinf aniqlanishini va tadbiqini ikkita turli fayllarga joylashtirish yaxshi ish albatta. Biroq, shunday bo‘lsa ham, sinf shablonlari uchun ularni birlashtirish bir muncha xavfsizroq, chunki, ba’zi kompilyatorlar ularni tarqoq holda kompilyatsiya qilmaydi.

6.6-kodli ro‘yxatda, uning qatorida **int** qiymatlar uchun va 14-qatorda satrlar uchun stek hosil qiluvchi sinovchi dastur berilgan.

6.6-kodli ro‘yxat. TestGenericStack.cpp

```
1.  #include <iostream>
2.  #include <string>
3.  #include "GenericStack.h"
4.  using namespace std;
5.  int main()
6.  {// int qiymatlar uchun yangi stek yaratish
7.  Stack<int> intStack;
8.  for (int i = 0; i < 10; i++)
9.  intStack.push(i);
10. while (!intStack.empty())
11. cout << intStack.pop() << " ";
12. cout << endl;
13. // Satrlar uchun yangi stek yaratish
14. Stack<string> stringStack;
15. stringStack.push("Chicago");
16. stringStack.push("Denver");
17. stringStack.push("London");
```

```
18. while(!stringStack.empty())
19. cout << stringStack.pop() << " ";
20. cout << endl;
21. return 0;}
```

Natija:

```
  9 8 7 6 5 4 3 2 1 0
  London Denver Chicago
```

Shablon sinfda ob‘yekt yaratishda **T** - tip parametri uchun aniq bir tipni belgilab olishimiz zarur. Masalan:

*Stack<**int**> intStack;*

Bu e’lon qilinish **T** - tip parametrini **int** bilan almashtiradi. Shu sababli ham **intStack** steki **int** qiymatlar steki hisoblanadi. **intStack** ob‘yekti ham xuddi ixtiyoriy boshqa ob‘yektlarga o‘xshaydi. Dastur **intStack** stekka **int** qiymatlarni qo‘shish uchun **push** funksiyasini chaqiradi (9-qator) va stekdan elementlarni chop etadi (11-12-qatorlar).

Dastur 14-qatorda satrlarni yozish uchun stek ob‘yektni e’lon qiladi, stekda uchta satrni yozadi (15-17-qatorlar) va stekdagi satrlarni chop etadi (19-qator).

Quyidagi kodli qismlarga e’tibor beraylik:

```
while (!intStack.empty())
  cout << intStack.pop() << " ";
cout << endl;
while (!stringStack.empty())
  cout << stringStack.pop() << " ";
cout << endl;
```

Bu kodli qismlar deyarli bir xil. Ular o‘rtasidagi farq **intStack** va **stringStack** yozuvlaridagi shakllantirish operatsiyalarida. Stekdagi elementlarni chop etish uchun stek parametrli funksiya aniqlashimiz mumkin. Yangi dastur 6.7-kodli ro‘yxatda keltirilgan.

6.7-kodli ro‘yxat. TestGenericStackWithTemplateFunction.cpp

```
#include <iostream>
#include <string>
#include "GenericStack.h"
using namespace std;
template <typename T>
void printStack(Stack<T>& stack)
{while(!stack.empty())
cout << stack.pop() << " ";
cout << endl;}
```

```
int main()
{// int qiymatlar uchun yangi stek yaratish
Stack<int> intStack;
for (int i = 0; i < 10; i++)
intStack.push(i);
printStack(intStack);
// Satrlar uchun yangi stek yaratish
Stack<string> stringStack;
stringStack.push("Chicago");
stringStack.push("Denver");
stringStack.push("London");
printStack(stringStack);
return 0;}
```

Shablon funksiyada umumiy sinf nomi **Stack<T>** tip parametr sifatida qo‘llanilgan.

Eslatma. C++ sinf shablonida tip parametri uchun *jimlik qoidasiga ko‘ra* tip ni ta’minlashga ruxsat beradi. Masalan, jimlik qoidasiga ko‘ra tip sifatida, umumiy **Stack** sinfida quyidagicha ta’minlash mumkin:

```
template<typename T = int>
class Stack{
...
};
```

Endi sinf shablonida jimlik qoidasi tipidan foydalanishimiz mumkin, lekin funksiya shablonida emas.

Eslatma. Biz shuningdek, *template* qo‘shimchasi tarkibida tip parametri bilan *tipsiz parametr* ni ham qo‘llashimiz mumkin. Masalan, **Stack** sinfda parametr sifatida, massiv o‘lchamini quyidagicha e’lon qilish mumkin:

```
template<typename T, int capacity>
class Stack{
...
private:
T elements[capacity];
int size;};
```

Shunday qilib, biz stek hosil qilganimizda, massiv o‘lchamini belgilashimiz mumkin. Masalan,

Stack<string, **500**> stack;

500 tagacha satrlarni saqlay oluvchi stekni e’lon qiladi.

Eslatma. Sinf shablonida statik a’zolarni aniqlash mumkin. Har bir shablon belgilanishi o‘zining statik ma’lumotlar maydonidagi nusxasiga ega.

Nazariy bilimlarni tekshirish uchun savollar

1. Shablon nima va uning qanday turlari mavjud?
2. Funksiya shabloni qanday yaratiladi?
3. Funksiya shablonida turli xil tiplardan foydalanish mumkinmi? Agar mumkin bo‘lsa bunday tipdagi funksiya shablonlari qanday ko‘rinishda yaratiladi?
4. Funksiya shablonini override qilish formasi qanday? Misol keltiring.
5. Funksiya shablonini override qilishning yangi formasi qanday ko‘rinishda?
6. Funksiya shabloni qanday overload qilinadi? Sinf shabloni qanday yaratiladi?
7. Sinf shablonida ikki va undan ortiq tiplardan foydalanish qanday amalga oshiriladi. Misol keltiring?
8. Shablonning kamchiliklari? Shablonning avzalliklari? Funksiya shablonida parametrlar qanday tipda bo‘ladi?
9. Sinf shablonida parametrlar bo‘ladimi? Bo‘lsa qanday?
10. Shablon sinfda shablon fuksiyalar ishlaydimi?Shablon sinfda shablon sinf yaratish mumkinmi? Shablon sinfni qayta yuklash mumkinmi?

Foydalanilgan adabiyotlar

1. Nazirov Sh.A., Qobulov R.V., Bobojanov M.R., Raxmanov Q.S. S va C++ tili. “Voris- nashriyot” MCHJ, Toshkent 2013, 488 b.
2. Horstmann, Cay S. C++ for everyone/Cay S. Horstmann. Printed in the United States of America - 2nd ed. 2010. – P. 562.
3. Horton I.-Beginning Visual C++ 2012/ I.Horton. Published simultaneously in Canada. –2012. –P. 988.
4. Алгоритмы. Справочник с примерами на C, C++, Java и Python. Джордж Хайнемаи, Гэри Поллис, Стэнли Селков. Москва «Альфа-Книга», 2017. –С254.
5. ASP.NET MVC Framework с примерами на C# для профессионалов. Стивен Сандерсон. Москва 2010. –С. 562.
6. Самоучитель Microsoft Visual Studio C++ и MFC. Сидорина Т.Л. Санкт-Петербург. «БХВ-Петербург» 2009. –С. 600.
7. Си Шарп: Создание приложения для Windows. В.В. Лабор. Минск, Харвест, 2003.
8. Полный справочник по C++. Герберт Шилдт. Москва, 2006.
9. Современные технологии разработки программ, взаимодействующих с базами данных. С.А. Минеев, Ю.Е. Чуманкин. Нежний Новгород, 2018.
10. C/C++. Программирование на языке высокого уровня. Т.А.Павловская. Санкт-Петербург, 2003.
11. Самоучитель C++. Герберт Шилдт. Санкт-Петербург. 2003.
12. C/C++ и MS Visual C++ 2010 для начинающих. Санкт-Петербург. 2011.
13. Программирование под Windows в среде Visual C++ 2005. М.В. Свиркин, А.С. Чуркин. Санкт-Петербург. 2008.
14. Простая графическая задача на Microsoft Visual C++ с использованием библиотеки MFC. Е.П. Дербакова. Москва 2015.
15. Программирование в Visual C++ с использованием библиотеки MFC. В.В. Васильчиков. Ярославль 2006.
16. MS Visual C++ 2010 в среде .NET. В.В. Зиборов. Санкт-Петербург, 2012
17. Herbert Schildt, Java the complete reference ninth edition, oracle press, 2014.
18. Алексеев А.П. Информатика. 2001. М., СОЛОН-Р, 2001, 364 с.
19. Арипов М.М. Internet ва электрон почта асослари. –Т.: Университет, 2000. -132 б.

20. Страуструп. Б. Язык программирования С++. Специальное издание.-М.:ООО «Бином-Пресс», 2006.-1104 с.
21. Вирт Н. Алгоритмы + структуры данных = программа.-М.:Мир,1985.-405с.
22. Говорухин В., Цибулин В. Компьютер в математическом исследовании. Maple, Matlab, Latex и др. Учебный курс. Питер. 2001. – 624 с.
23. Информатика. Базовой курс. Учебник для Вузов., Санк-Петербург, 2001. под редакцией С.В.Симоновича.
24. Павловская Т.С. Щупак Ю.С. С/С++. Структурное программирование. Практикум.-СПб.: Питер,2002-240с
25. Павловская Т.С. Щупак Ю.С. С++. Объектно-ориентированное программирование. Практикум.- СПб.: Питер,2005-265с
26. Подбельский В.В. Язык СИ++.- М.; Финансы и статистика-2003 562с.
27. Романчик В.С., Люлькин А.Е. Программирование в С++ BUILDER. Учебное пособие. Мн.: БГУ, 2007. –126 с.
28. Юлдашев У.Ю., Боқиев Р.Р., Зокирова Ф.М. Информатика. – Т.: Ғ.Ғулом номидаги нашриёт-матбаа ижодий уйи, 2002. - 237 б.
29. Abduqodirov A.A., Hayitov A.G‘., Shodiyev R.R. Axborot texnologiyalari //Akademik litsey va kasb-hunar kollejlari uchun darslik. -T.:O‘qituvchi, 2003. –152 b.
30. Axmedov A.B., Tayloqov N.I. Informatika. Akademik litsey va kasb hunar kollejlari uchun darslik.-T.:O‘zbekiston, 2001. 2 – nashri. – 272 b.
31. Mo‘minov B.B. Microsoft Excel bo‘yicha amaliy mashg‘ulotlar va ularni bajarish tartibi. Uslubiy qo‘llanma. – Buxoro: Ziyo Rizograf, 2008. – 72 b.
32. Mo‘minov B.B. Microsoft Excel dasturi bo‘yicha laboratoriya ishlar to‘plami. Uslubiy qo‘llanma. – Buxoro: Ziyo Rizograf, 2008. – 92 b.
33. Mo‘minov B.B. Pedagogik dasturiy ta’minot yaratish texnologiyasi. Monografiya. –Buxoro, Buxoro nariyoti, 2010. -168 b.
34. Xaldjigitov A.A., Madraximov Sh.F., Adamboev U.E .Informatika va dasturlsh. O‘quv qo‘llanma. O‘zMU, 2005 yil, 145 bet.
35. http://aut.researchgateway.ac.nz/index.jsp Auckland University of Technology digital library

36. http://dastur.uz Kompyuter dasturlari va Kompyuterda dasturlashga oid Forum, Habar va Yangiliklar
37. http://google.com Google qidiruv tizimi
38. http://informatika.sch880.ru Основы логики
39. http://lex.uz/ O‘zbekiston Respublikasi qonun hujjatlari ma’lumotlari milliy bazasi
40. http://ru.wikipedia.org/wiki Википедия — свободная энциклопедия
41. http://uza.uz O‘zbekiton milliy axborot agentligi
42. http://winedt.com winedt rasmiy portali
43. http://www.aci.uz O‘zbekiston aloqa va axborotlashtirish agentligi
44. http://www.cplusplus.com cplusplus.com
45. http://www.edu.uz O‘zbekiston Respublikasi Oliy va O‘rta maxsus ta’lim vazirligi
46. http://www.exponenta.ru Образовательный математический сайт Exponenta.ru
47. http://www.intuit.ru Интернет-Университет Информационных Технологий
48. http://ziyonet.uz Oliy va O‘rta maxsus ta’lim vazirligi huzuridagi axborot ta’lim portali

MUNDARIJA

MO‘MINOV Bahodir Boltaeyich
e-mail: Mbbahodir@gmail.com

D A S T U R L A S H I

(D a r s l i k)

O‘zbek tilida

Toshkent – «NIHOL PRINT» OK – 2021

Muharrir:	A.Tog‘ayev
Tex. muharrir:	F.Tog‘ayeva
Musavvir:	B.Esanov
Musahhiha:	O.Muxammadiyeva
Kompyuterda sahifalovchi:	G.Tog‘ayeva

9323

№ 7439-765f-47f1-7ea1-a683-4648-1314.
Bosishga ruxsat etildi: . Bichimi 60x841 /16.
Shartli bosma tabog‘i 17,75. Nashr bosma tabog‘i 17,5.
Adadi 100. Buyurtma № 4.

«Nihol print» Ok da chop etildi.
Toshkent sh., M. Ashrafiy ko‘chasi, 99/101.