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

O‘ZBEKISTON XALQARO ISLOM AKADEMIYASI

Raxmanov Qurbon Sodikovich
Tuxtanazarov Dilmurod Solijonovich

“DASTURLASH I”
FANIDAN O‘QUV QO‘LLANMA

O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta’lim vazirligi huzuridagi Muvofiqlashtiruvchi kengashning 2022-yil 28-noyabrdagi 388-sonli buyrug’iga asosan 60610400 – “Axborot xavfsizligini boshqarish” bakalavriyat ta’lim yo‘nalishi talabalari uchun o‘quv qo‘llanma sifatida tavsiya etilgan

Toshkent – 2022

UO‘K: 681(075) D-16

Q.S. Raxmanov, D.S. Tuxtanazarov

Dasturlash I // O‘quv qo’llanma. Q.S. Raxmanov, D.S. Tuxtanazarov. T.: “Zamon poligraf” nashriyoti, 2023 -y. – 167 bet.

Ushbu o‘quv qo‘llanmada C++ tilining alifbosi, identifikator, kalit so‘zlar, satrli o‘zgaruvchilar, ma’lumotlar tipi, arifmetik ifoda va amallar, inkrement, dekrement, kutubxonalar 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 massivlar, massivni funksiya parametri sifatida qo‘llanilishi yo‘llari, saralash algoritimlari, sinflar va konstruktorlardan foydalanish, obyektlar va ularni parametr sifatida qo‘llanilish usullari, satrlar va belgili o‘zgaruvchilarga ishlov berish, matnli va binar fayllar bilan ishlash usullari, inkapsulyatsiyani tashkil qilish yo‘llari, merosxo‘rlik va undan foydalanish usullari, polimorfizm va uning turlari, operatorlarni qayta yuklash, shablonlar va ularni tashkil qilish shartlari bo‘yicha nazariy va amaliy materiallar keltirilgan.

Ushbu o‘quv qo‘llanma 60610400– “Axborot xavfsizligini boshqarish” yo‘nalishi talabalarining ishchi o‘quv rejasida keltirilgan “Dasturlash I” fan asosida ishlab chiqilgan.

Taqrizchilar:

Mo‘minov B.B. - Muhammad Al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti “Axborot texnologiyalarining dasturiy ta’minoti” kafedrasi mudiri, t.f.d.

Jumayev T.S. - O‘zbekiston xalqaro islom akademiyasi “Zamonaviy axborot-kommunikatsiya texnologiyalari” kafedrasi katta o‘qituvchisi, PhD

O‘zbekiston xalqaro islom akademiyasi Kengashining 2022-yil 29-iyundagi 11-sonli majlisida muhokama qilingan va nashrga tavsiya etilgan.

KIRISH

Ma’lumki, hozirgi vaqtda C++ dasturlash tilini o‘rganish bo‘yicha adabiyotlar ko‘p. C++ dasturlash tilini o‘rganish bo‘yicha yurtimizda ham o‘zbek tilida ko‘plab kitoblar yozilgan. Shuning uchun, yana C++ tilida o‘quv qo‘llanma yozish qiyinchiliklar tug’diradi. Shuni e’tiborga olgan holda “Dasturlash I” fanidan o‘quv qo‘llanma sodda tarzda ishlab chiqilgan. O‘quv qo‘llanma O‘zbekiston xalqaro islom akademiyasining 60610400 - “Axborot xavfsizligini boshqarish” yo‘nalishi talabalari uchun mo‘ljallangan bo‘lib, C++ tilini o‘rganuvchilar uchun mo‘ljallangan.

O‘quv qo‘llanma yozishda nazariy ma’lumotlarni mumkin qadar kamroq berishga harakat qilingan. Asosiy e’tibor nazariyadan kelib chiqqan holda amaliy misollarga asosiy e’tibor berilgan.

Shuni alohida nazarda tutish kerakki, hozirgi vaqtda davlat tomonidan “Bir million dasturchi” loyihasi amalga oshirilmoqda. Albatta ushbu qo‘llanma C++ dasturlash tilini o‘rganuvchilar uchun mo‘ljallangan bo‘lib, “Bir million dasturchi” loyihasi uchun dasturchi yoshlarni tayyorlashda ozgina bo‘lsa ham hissasini qo‘shadi degan umiddamiz.

1. DASTURLASHGA KIRISH, DASTURLASHNING ASOSIY TUSHUNCHALARI

REJA:

1. Tilning bazaviy tushunchalari.
2. Arifmetik ifoda va amallar.
3. Inkrement va dekrement.
4. Preprotsessor direktivalari va vositalari.

Kalit so‘zlar: til alifbosi, identifikator va leksemlar, kalit so‘zlar, konstanta satrlar, ma’lumotlar toifasi, arifmetik ifoda va amallar, inkrement, kutubxona, dekrement, preprotsessor.

1.1. Tilning bazaviy tushunchalari

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.

C++ da ma’lumotlar toifasi va o‘lchami

Toifasi	O‘lchami	Interval
char	1 bayt	-127 dan 127 yoki 0 dan 255 gacha
unsigned char	1 bayt	0 dan 255 gacha
signed char	1 bayt	-127 dan 127 gacha
int	4 bayt	-2147483648 dan 2147483647 gacha
unsigned int	4 bayt	0 dan 4294967295 gacha
signed int	4 bayt	-2147483648 dan 2147483647 gacha
short int	2 bayt	-32768 dan 32767 gacha
unsigned short int	2 bayt	0 dan 65,535 gacha
signed short	2 bayt	-32768 dan 32767 gacha

int		
long int	8 bayt	-2 147 483 648 dan 2 147 483 647 gacha
signed long int	8 bayt	long int bilan bir xil
unsigned long int	8 bayt	0 dan 4 294 967 295 gacha
long long int	8 bayt	-(2^63) dan (2^63)-1 gacha
unsigned long long int	8 bayt	0 dan 18 446 744 073 709 551 615 gacha
float	4 bayt	10^{-38} dan 10^{+38} gacha
double	8 bayt	10^{-304} dan 10^{+304} gacha
long double	12 bayt	$3.4*10^{-4932}$ dan $1.1*10^{-4932}$ gacha

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 ko'rinishini shakllantirish uchun;

• #include <string.h> - satr tipidagi o'zgaruvchilar 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.

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 qo'yilmaydi)

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.

1.2. Inkrement va dekrement operatsiyasi

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

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).

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.

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)

1.3. 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.

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.

1.4. Preprotsessor direktivalari va vositalari

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 ko‘rib 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 bo‘lsin:

Double mix=EULER

D=alfa*EULER

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

Double mix = 2.718282

D = alfa * 2.718282

Nazorat savollari

1. Dastur tuzishda konstantalar qanday belgilanadi?
2. Kompilyatsiya interpretatsiya tushunchalari orasida qanday farq bor?
3. C++ tilida standart funksiyalar qanday yoziladi?
4. Preprotsessor deriktivasi nima?
5. Dastur tuzishda sarlavha fayllarinign ahamiyati nimada?
6. Dasturning asosiy funksiyasi. Funktsiya tuzilishi, uning sarlavhasi nima?
7. C ++ tilining ma’lumotlar tiplari nima?
8. C ++ tilidagi dasturning tuzilishi.
9. Standart kutubxonalar va ularning aloqasi nimalardan iborat?
10. #define – kalit so‘zining vazifasi nimalardan iborat?

2. DASTURLASH TILLARINING TUZILMASI

REJA:

1. Dastur tarkibi.
2. cin va cout operatorlari.
3. Simvollarni o‘qish va yozish.
4. printf() va scanf() funksiyalari.

Kalit so‘zlar: cin (console input), cout (console output), printf(), scanf(), boshqaruv simvollari, formatlash, algoritm, kompilyator, kompilyatsiya, leksema, sizeof, ternar, preprotsessor direktivalar, qiymat kiritish, qiymat chiqarish, adreslash.

2.1. Dastur tarkibi

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.

2.2. cin va cout operatorlari

C++ dasturlash tilida cout kalit so‘zi “<<” bilan birgalikda chiqarish operatori yoziladi. Operator - bu dasturlash tilida aniq bir maqsadga yo‘naltirilgan. C++ da kiritish va chiqarish operatori, takrorlash operatori, shart operatorlari va shunga o‘xshash ko‘plab operator turlari mavjud. C++ dasturlash tilida ko‘plab operatorlar so‘ng “;” belgisi bilan yakunlanadi. Ayrim operatorlar {} figuraviy qavuslar bilan ishning ma’lum bir vazifasini bajaradigan qismi yakunlaydi.

C++ dasturlash tilida chiqarish operatori uchun misol keltiramiz.

```
#include <iostream>
using namespace std;
int main() {
  cout << "Salom Talabalar! C++ dasturlash tilini o‘rganishni boshladik";
  return 0;
}
```

Natija:

```
Salom Talabalar! C++ dasturlash tilini o‘rganishni boshladik
```

Bir qatorda bir nechta jumlalarni ko‘rsatish uchun << operatoridan bir necha marta foydalanish kerak, masalan:

```
#include <iostream>
using namespace std;

int main()
{
   int a = 7;
   cout << "a = " << a;
   return 0;
```

Natija:

```
a= 7
```

cout operatoridan bir necha marta foydalansangiz ham konsolda ma’lumotlar bir qatorda chiqadi.

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Salom! ";
  cout << "Mening ismim Akbar. ";
  cout << "Men C++ ni o`rganmoqchiman.";
  return 0;
```

Natija:

```
Salom! Mening ismim Akbar. Men C++ ni o`rganmoqchiman.
```

Malumotlarni konsolda yangi qatordan namoyish qilish uchun “endl” operatoridan foydalanamiz.

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Salom! "<<endl;
  cout << "Mening ismim Akbar. "<<endl;
  cout << "Men C++ ni o`rganmoqchiman.";
  return 0;
}
```

Natija:

```
Salom!
Mening ismim Akbar.
Men C++ ni o`rganmoqchiman.
```

cin kiritish operatori. cout ma’lumotlarni konsolga “<<” chiqarish operatori yordamida chiqarsa, cin “>>” kiritish operatori yordamida foydalanuvchidan ma’lumotlarni konsol orqali oladi. cin yordamida biz foydalanuvchi ma’lumotlarini qabul qilishimiz va qayta ishlashimiz mumkin:

```
#include <iostream>
using namespace std;
int main()
{
  int a;
    cout<< “a ni kiriting: ”;
    cin>>a;
    cout << "a ni kiritdingiz "<<a;
  return 0;
}
```

Natija:

a ni kiriting: 5
a ni kiritdingiz 5

2.3. Simvollarni o‘qish va yozish

Simvollarni konsoldan o‘qish uchun gets() operatori ishlatiladi. Cin orqali ma’lumot kiritilganda probel belgisini qabul qilmaydi, shuning uchun simvollarni kiritishda gets() operatoridan foydalanamiz.

Belgilarni cin orqali kiritamiz:

```
#include <iostream>
using namespace std;
int main()
{
   char s[10];
   cout<<"Belgilarni kiriting: " << endl;
   cin>>s;
   cout << “Natija: ” << s;
return 0;
```

Natija:

```
Belgilarni kiriting:
Salom men Akbar
Natija: Salom
```

Belgilarni gets() orqali kiritamiz:

```
#include <iostream>
using namespace std;
int main()
{
   char s[10];
   cout<<"Belgilarni kiriting: " << endl;
   gets(s);
   cout << “Natija: ” << s;
   return 0;}
```

Natija:

```
Belgilarni kiriting:
Salom men Akbar
Natija: Salom men Akbar
```

2.4. printf() va scanf() funksiyalari

printf() (chiqarish uchun) va scanf() (kiritish uchun) funksiyalari ko‘pincha chiquvchi va kiruvchi qiymatlarni formatlab chiqarish va kiritish uchun ishlatiladi.

printf() - ma’lumotlarni belgilar tasviriga aylantiradi va natijada olingan belgilar tasvirlarini ekranda ko‘rsatadi. Shu bilan birga, dasturchi ma’lumotlarni formatlash, ya’ni ularning ekranda chiqarish imkoniyatiga ega.

printf() – umumiy tuzilishi:

- printf("Formatlash satri", obyekt1, obyekt2, ..., obyektn);

Formatlash satri quyidagicha elementlardan tashkil topadi:

- boshqaruv simvollari;
- chiqarilagan matn;
- har xil turdagi o‘zgaruvchilar qiymatlarini ko‘rsatish uchun mo‘ljallangan formatlar.

Boshqarish simvollari ekranda ko‘rsatilmaydi, lekin ko‘rsatilgan belgilarning tartibini boshqaradi. Boshqaruv simvolining o‘ziga xos xususiyati uning oldida "\" slesh chiziqning bo‘ladi.

Asosiy boshqaruv simvollari:

- ‘\n’ — yangi satrga o‘tish;
- ‘\t’ — gorizontal tabulyatsiya;
- ‘\v’ — vertikal tabulyatsiya;
- ‘\b’ — simvolni qaytarish;
- ‘\r’ — satr boshiga qaytish;
- ‘\a’ — tovush chiqarish.

Ekranda ma’lumot ko‘rsatiladigan shaklni ko‘rsatish uchun formatlar kerak. Formatning ajralib turadigan xususiyati uning oldida "%" foiz belgisining mavjudligidir:

- %d - o‘nlik sanoq tizimidagi int tipidagi butun son;
- %u - unsigned int tipidagi butun son;
- %x - o‘n oltilik sanoq tizimidagi int tipidagi butun son;
- %o - sakkizlik sanoq tizimidagi int tipidagi butun son;
- %hd - o‘nlik sanoq tizimidagi short tipidagi butun son;
- %hu - unsigned short tipidagi butun son;
- %hx - o‘n oltilik sanoq tizimidagi short tipidagi butun son;
- %ld - o‘nlik sanoq tizimidagi long int tipidagi butun son;
- %lu - unsigned long int tipidagi butun son;
- %lx - o‘n oltilik sanoq tizimidagi long int tipidagi butun son;

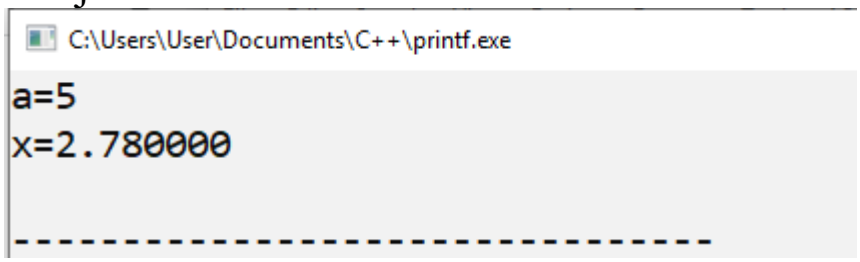
- %f – haqiqiy son tipidagi format (float tipidagi suzuvchi nuqta raqamlar);
- %lf — ikki tomonlama aniqlikdagi haqiqiy son tipidagi format (double tipidagi suzuvchi nuqta raqamlar);
- %e - eksponensial shakldagi haqiqiy son tipidagi format (eksponensial shakldagi float tipidagi suzuvchi nuqta raqamlari);
- %c - simvolli format;
- %s satrli format.

Har bir chiqish formati % belgisi bilan boshlanadi. Format satridan keyin ko‘rsatiladigan o‘zgaruvchilar nomlari vergul bilan ajratilgan holda ko‘rsatiladi.

Format satridagi % belgilar soni chiqarilishi kerak bo‘lgan o‘zgaruvchilar soniga mos kelishi kerak. Har bir formatning turi o‘sha joyga chiqariladigan o‘zgaruvchining turiga mos kelishi kerak.

```
int main()
{
  int a = 5;
  float x = 2.78;
  printf("a=%d\n", a);
  printf("x=%f\n", x);
  return 0;
}
```

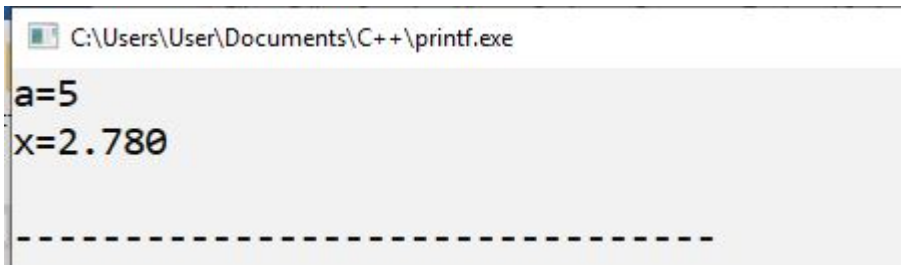
Natija:



Bunda float tipida xususiy holda (.) tadan keyin 6 ta raqamni chiqaradi. Agarda biz (.) dan keyingi raqamlar sonini boshqarishimiz mumkin.

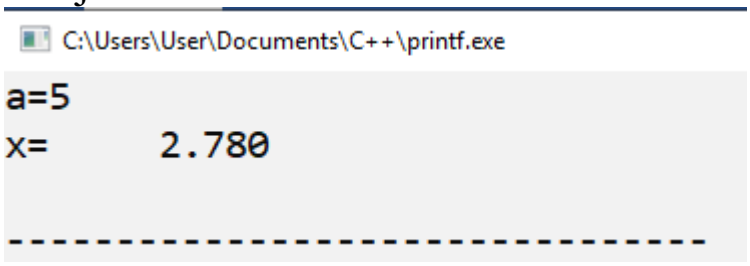
```
int main()
{
  int a = 5;
  float x = 2.78;
  printf("a=%d\n", a);
  printf("x=%.3f\n", x);
  return 0;}
```

Natija:




```
int main()
{
  int a = 5;
  float x = 2.78;
  printf("a=%d\n", a);
  printf("x=%10.3f\n", x);
  return 0;
}
```

Natija:



Bunda chiquvchi o‘zgaruvchi uchun 10 ta katak ajratilgan, kasr qism uchun esa 5 ta katak ajratilgan.

scanf("Formatlash satri", obyekt1, obyekt2, ..., obyektn);

scanf() ning formatlanishi printf() bilan deyarli bir xil. O‘zgaruvchilarni formatlash uchun ‘&’ ampersand belgisidan foydalaniladi. manzil = &obyekt1

```
int main()
{
  int a = 5;
  float y;
  printf("y ni kiriting: ");
  scanf("%f", &y);
  printf("y ning qiymati quyidagicha=%f", y);
  return 0;
}
```

Natija:

Nazorat savollari

1. Algoritm nima?
2. C++ dasturlash tilida tiplar turi va ularning xotiradagi hajmi qanday aniqlanadi?
3. C++ tilida qanday funksiyalar bor?
4. Standart matematik funksiyalarni sanab o‘ting.
5. Formatli kiritish operatorini ayting.
6. Formatli chiqarish operatorini ayting.
7. Uzunligi 64 bitdan kichik bo‘lmagan ma’lumotning haqiqiy tipi qaysi so‘z orqali ifodalanadi?
8. Ham asosiy dasturda ham boshqa funksiyalarda ishlashi mumkin bo‘lgan o‘zgaruvchilar qaysilar?
9. Dasturlashda xatoliklar bilan qanday ishlanadi?

3. TARMOQLANISH VA UZILISHLARNI TASHKIL ETISH OPERATORLARI

REJA

1. To‘liqsiz tarmoqlanish if operatori.
2. To‘liq tarmoqlanish. if – else operatori.
3. Tanlash operatori. switch operatori.
4. Ternar operator.
5. Shartsiz o‘tish operatori.
6. Break operatori.
7. Continue operatori.

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

3.1. 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.

3.2. To‘liq tarmoqlanish. if – else operatori

Misol. Ikkita butun sonni kiriting va ulardan kichigini ekranga chiqaring. 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 ko‘rsatmasini bajarish orqali dastur o‘z ishini tugatadi.

3.3. 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.

3.4. Ternar operator

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.

3.5. Shartsiz o‘tish operatorlari

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.

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.

3.6. Break operatori

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.

3.7. 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.

Nazorat savollari

1. To‘liqsiz tarmoqlanish if operatori va uning vazifasi.
2. To‘liq tarmoqlanish if operatori va uning vazifasi.
3. C++ dasturlash tilida takrorlash operatorlari nima uchun kerak?
4. Switch operatori va uning vazifasiga nimalar kiradi?
5. Ternar operatori nima uchun kerak?
6. Shartsiz o‘tish operatorlari hamda uning imkoniyatlari.
7. Break operatori va uning vazifasi nimalardan iborat?
8. Continue operatori nima uchun kerak?
9. Nima uchun goto operatori ko‘p ishlatilmaydi?

4. TAKRORLANISH OPERATORLARI

REJA:

1. for operatori.
2. while operatori.
3. do while operatori

Kalit so‘zlar: takrorlash, takrorlash tanasi, while, do while, for, iteratsiya, takrorlanish, parametrli, shartli takrorlash, shartli takrorlash.

4.1. for operatori

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 ;}
```

4.2. while 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 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.

4.3. do while 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.

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;}
```

Nazorat savollari

1. *for* operatorining umumiy ko‘rinishi. Ichki sikllar qanday tashqil etiladi? *for* operatori qanday qo‘llaniladi?
2. For operatorining takrorlanishini tugashi uning qaysi qismiga bo‘liq?
3. *while* operatorining umumiy ko‘rinishi.
4. *do while* operatori *while* operatoridan qanday farq qiladi?
5. While operatori qanday ishlaydi?
6. *for* bilan *while* va *do-while* operatorlarining qanday farqli tomonlari bor?
7. Takrorlanishlar algoritmlarda qanday ko‘rinishda bo‘ladi?
8. *while* operatorida qachon cheksiz takrorlash ro‘y beradi?
9. Takrorlanishlar ichma-ich bo‘lishi mumkinmi?
10. *do-while* operatori qanday ishlatiladi?

5. FUNKSIYALAR

REJA:

1. Funksiya.
2. Rekursiv funksiyalar.
3. Funksiyalarni qayta yuklash.

Kalit so‘zlar: funksiya, prototip, qiymat qaytarish, foydalanuvchi kutubxonasi, local va global o‘zgaruvchi, funksiya tanasi, qayta yuklash.

5.1. Funksiya

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) {	double f (n, x) int n; float x;

funksiya tanasi; }	{ 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 degan axborot chiqadi.

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

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.

#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.

5.2. Rekursiv funksiyalar

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;
- hech qachon tugallanmaydi va xatolik yuz beradi.

Bunday holatlarda rekursiv funksiyalar uchun rekursiyani to‘xtatish shartini berish zarur, chunki rekursiyada xotira yetishmasligi xavfi bor.

1-misol. F = n! ni hisoblash 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); }
```

2-misol. Fibonachi sonlarini hosil qilish dasturini tuzing. Fibonachi sonlari quyidagicha topiladi:

$f_0 = 1; f_1 = 1; f_2 = f_1 + f_0 ; \ldots\ldots$

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

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)); }
```

5.3. 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.

Nazorat savollari

1. Funksiya va uning prototipi nima?
2. Funksiya parametrlariga ta’rif bering?
3. Qiymat berish deganda nimani tushunasiz?
4. Funksiya bilan protsedurani farqi nimada?
5. Qiymat qaytarmaydigan funksiyalarni yana qanday nomlash mumkin?
6. Rekursiv funksiyaga ta’rif bering.
7. Funksiyalarni qayta yuklash nima uchun zarur?
8. C++ dasturlash tilida kutubxona fayli qanday yaratiladi?

6. MASSIVLAR
REJA

1. Massiv.
2. Massiv elementlariga qiymat kiritish va chiqarish usullari.
3. Statik massivlar.
4. Massiv elementlarini qidirish usullari.
5. Massiv elementlarini saralash usullari

Kalit so‘zlar: massiv, massiv indeksi, massiv elementi, ko‘rchsatkich, adres olish &, bo‘shatish, ko‘rsatkich, virtual destruktor, xotira, xotira chiqishi.

6.1. Massiv

Massiv - bu bitta nom ostida birlashtirilgan bir xil tipdagi elementlar guruhi sifatida taqdim etilgan ma’lumotlar strukturasi. Massivlar bir xil tipdagi katta hajmdagi ma’lumotlarni qayta ishlash uchun ishlatiladi.

Alohida massiv ma’lumotlar katakchasi massiv elementi deyiladi. Massiv elementlari har qanday turdagi ma’lumotlar bo‘lishi mumkin. Massivlar bir yoki bir nechta o‘lchovga ega bo‘lishi mumkin. O‘lchovlar soniga qarab massivlar bir o‘lchovli massivlarga, ikki o‘lchovli massivlarga, uch o‘lchovli massivlarga va hokazo n o‘lchovli massivgacha bo‘linadi. Ko‘pincha dasturlashda bir o‘lchovli va ikki o‘lchovli massivlardan foydalaniladi, shuning uchun biz faqat shu massivlarni ko‘rib chiqamiz.

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>]

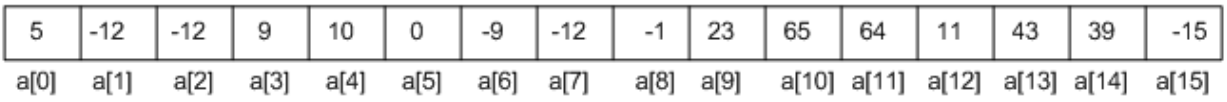
Bu ko‘rinishga *xususiy o‘zgaruvchi* deyiladi, chunki uning qiymati massivning alohida elementidir.

Masalan, 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.

Bir o‘lchovli massiv - bir o‘lchovli massivning elementlari sonini ifodalovchi bitta parametrga ega massiv. Aslida, bir o‘lchovli massiv faqat bitta qator va n ta ustunga ega bo‘lishi mumkin. Bir o‘lchovli massivdagi ustunlar massivning elementlari hisoblanadi. 1-rasmda butun sonli bir o‘lchovli a massivning tuzilishi ko‘rsatilgan. Ushbu massivning o‘lchami 16 ga teng.



1-rasm. C++ da bir o‘lchamli massiv

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}.

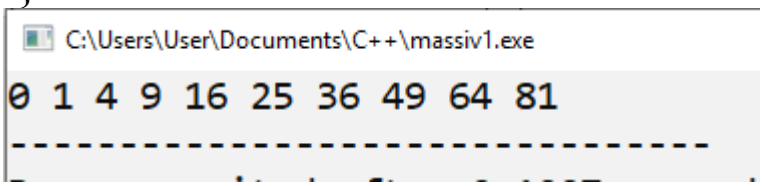
Bu yerda <uzunlik> - o‘zgarmas ifoda. Misollar:

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

6.2. Massiv elementlariga qiymat kiritish va chiqarish usullari

Massiv elementlariga avtomatik qiymat berish va uni chiqarish quyidagicha amalga oshiriladi.

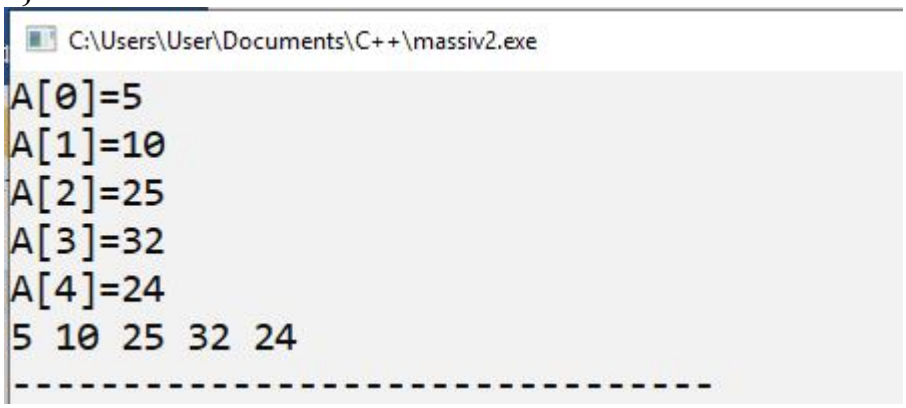
```
int main()
{
  const int N = 10;
  int A[N];
  int i;
  for ( i = 0; i < N; i++ )      A[i] = i*i;
for ( i = 0; i < N; i++ )      cout <<A[i]<<" ";
}
```



Demak, bu dasturda A massivning 10 ta elementiga 0 dan 9 gacha bo‘lgan sonlarning kvadratlari o‘zlashtirilayapdi.

Massiv elementlarini klaviatura orqali ham kiritish va kiritilgan massiv elementlarini chiqarish mumkin. Bunda A massiv elementlari ketma-ket kiritilib boriladi va eng oxirgi element kiririlgandan so‘ng kiritilgan elementlar chiqariladi.

```
int main()
{
  const int N = 5;   int i,  A[N];
  for ( i = 0; i < N; i++ ) {
   cout << "A[" << i << "]=";
   cin >> A[i]; }
  for ( i = 0; i < N; i++ )
    cout <<A[i]<<" ";
}
```

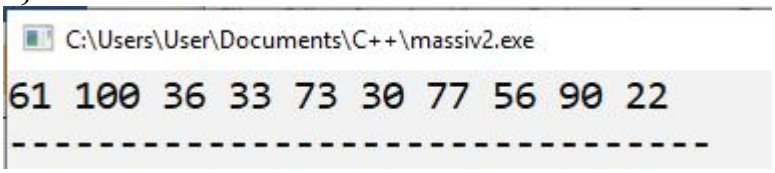


Bu dasturdagi " "; massiv elementlarini orasida bitta probel tashlab chiqarish uchun ishlatilgan.

Bundan tashqari massiv elementlarini rand() finksiyasi orqali biror bir oraliqdagi sonlar bilan ham to‘ldirishimiz mumkin.

```
#include <cstdlib>
using namespace std;
int random ( int a, int b )
{  return a + rand()% (b - a + 1);}
//1 dan b-a gacha oraliq
int main()
{
  const int N = 10;   int i,  A[N];
for ( i = 0; i < N; i++ )
  {
  A[i] = random ( 20, 100 );
  cout << A[i] << " ";
```

```
  }
}
```



6.3. Statik massivlar

Masssivlar bilan ishlashda uning elementlari soni oldindan ma’lum bo‘lsa va u dastur bajarilishi davomida o‘zgarmasa bunday massivlar static massivlar deyiladi.

```
int a[5]={3,1,8,4,2}; //statik massiv
```

Demak, a massivda elementlari soni aniq, ya’ni 5 ta. Bu elementlar soni dastur bajarilishi davomida umuman o‘zgarmaydi.

C++ tilida massivlar elementining turiga cheklovlar qo‘yilmaydi, lekin bu turlar chekli o‘lchamdagi obyektlarning turi bo‘lishi kerak. Chunki kompilyator massivning xotiradan qancha joy (bayt) egallashini hisoblay olishi kerak. Xususan, massiv komponentasi massiv bo‘lishi mumkin, ya’ni «vektorlar-vektori» natijada matritsa deb nomlanuvchi ikki o‘lchamli massiv hosil bo‘ladi.

Agar matritsaning elementi ham vektor bo‘lsa, uch o‘lchamli massivlar – kub hosil bo‘ladi. Shu yo‘l bilan yechilayotgan masalaga bog‘liq ravishda ixtiyoriy o‘lchamdagi massivlarni yaratish mumkin.

Ikki o‘lchamli massivning sintaksisi quyidagi ko‘rinishda bo‘ladi:

```
[] []
```

Masalan, 10 x 20 o‘lchamli haqiqiy sonlar massivining e’loni:

```
float A[10] [20] ;
```

Endi adres nuqtai – nazaridan ko‘p o‘lchamli massiv elementlariga murojaat qilishni ko‘raylik. Quyidagi e’lonlar berilgan bo‘lsin:

```
int a[3][2];
float b[2][2][2];
```

6.4. Massiv elementlarini qidirish usullari

Massiv elementlri orasidan qidirish amallarini bajarish mumkin. Quyidagi misolda berilgan massivdan X elementni qidirishni qanday amalga oshirish kodi berilgan. Bunda albatta oldindan massiv elementlari va qidirilayotgan X qiymat aniq bo‘lishi lozim.

```
nX = -1;
for ( i = 0; i < N; i++ )
  if ( A[i] == X )
```

```
    {
    nX = i;
    break;
    }
if ( nX >= 0 )
  cout << "A[" << nX << "]=" << X;
else
  cout << "Topilmadi!";
```

Berilgan massiv elementlari orasida eng katta yoki eng kichik elementni ham topish mumkin. Bunda dastlab birinchi element eng katta yoki eng kichik deb tanlab olinadi va massivning qolgan elementlari bilan solishtirib chiqiladi. Agarda M dan katta element bo‘lsa, M o‘sha katta qiymatni oladi.

```
M = A[0];
for ( i = 1; i < N; i++ )
  if ( A[i]> M )
    M = A[i];
cout << M;
```

Xuddi masalaga eng katta elemtnini adresini quyidagi kod orqali chiqarish mumkin.

```
M = A[0]; nMax = 0;
for ( i = 1; i < N; i++ )
  if ( A[i] > M ) {
    M = A[i];
    nMax = i;
    }
cout << "A[" << nMax << "]=" << M;
```

nMax o‘zgaruvchi orqali A massivning eng katta qiymati va uning nechanchi elementi ekanligini chiqaradi.

6.5. Massiv elementlarini saralash usullari

Massivlar bilan ishlashda eng asosiy funksiyalardan biri uning elementlarini tartiblash yoki saralash sanaladi. Barcha massivlarni saralashda eng katta omil saralash vaqti va xoriradan qancha joy egallashi muhim sanaladi.

Saralash vaqti – algoritmni baholaydigan asasiy parametr hisoblanadi.

Xotira – algoritm ishlashi uchun ketadigan qo‘shimcha xotira hajmi. Bunda ma’lumotlar va dastur kodi uchun ketadigan xotira hajmi

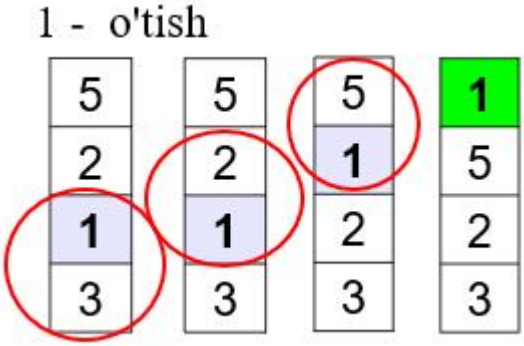
hisobga olinmaydi. Turg‘unlik – dastur(ketma-ketlik)ning boshqa qiymatlarda ham turg‘un ishlashi tushuniladi.

Saralashni turlari kop. **Saralash**– bu massiv elementlarini tartiblash (o‘sish, kamayish, oxirgi raqami, bo‘luvchilari bo‘yicha, juft, toq va boshqalar).

Saralashda har xil aolgoritmlardan foydalaniladi. Quyida ularni ko‘rib chiqamiz. Ularni ikki xilga bo‘lish mumkin.

- Oddiy va tushunarli, lekin katta massivlar uchun, samarali emas:
  - Pufakcha usuli;
  - Tanlash usuli.
- Qiyin, lekin samarali usullar:
  - «tez saralash» (Quick sort);
  - «to‘p-to‘p» saralash (Heap sort);
  - Qo‘shilib saralash;
  - Piramidali saralash.

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.



- pastdan boshlab ikkita qo‘shni elementni solishtiramiz;
- agarda ular «noto‘g‘ri» turgan bo‘lsa, ularni o‘rnini almashtiramiz;
- birinchi o‘tishda bitta element (eng kichik) o‘z joyiga o‘tadi;

```
for( j = N-2; j >= 0 ; j-- )
  if ( A[j] > A[j+1] ) {
    c = A[j];
    A[j] = A[j+1];
    A[j+1] = c;
    }
```

ikkinchi o‘tishda bitta element (eng kichik) o‘z joyiga o‘tadi;

```
for ( j = N-2; j >= 1 ; j-- )
```

```
 if ( A[j] > A[j+1] ) {
   c = A[j];
   A[j] = A[j+1];
   A[j+1] = c;
   }
```

Bu o‘tishlarni umimiy holda quyidagicha yozish mumkin:

```
for(i=0; i<N-1; i++)
{
 for(j=N-2;j>=i;j--)
     if (A[j]>A[j+1]) {
       c = A[j];
       A[j] = A[j+1];
       A[j+1] = c;
              } }
```

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. (pufakchali usul)

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.

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×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;}
```

Nazorat savollari

1. C++da massiv qanday ishlaydi?
2. Massivlarni qabday turlari mavjud? Massivlar nima uchun qo‘llaniladi? Ikki o‘lchamli massivlar.
3. Bir o‘lchovli massiv nima? Ikki o‘lchovli massiv nima? Statik bir o‘lchovli massivlar.
4. Dinamik bir o‘lchovli massivlar. Statik ko‘p o‘lchovli massivlar. Dinamik ko‘p o‘lchovli massivlar.
5. Statik massivlar indeksli o‘zgaruvchilar. Massiv e’loni sintaksisi.
6. O‘lchami ko‘rsatilgan massiv elementlarini to‘liq initsializatsiyalash.
7. O‘lchami ko‘rsatilgan massiv elementlarini to‘liqmas initsializatsiyalash.
8. Ikki o‘lchamli massivning e’loni sintaksisi. Ikki o‘lchamli massiv elementlarining xotiradagi joylashuvi.
9. Ko‘rsatkichlar va bir o‘lchovli massivlar. Ko‘rsatkichlar va ikki o‘lchovli massivlar. Ko‘rsatkichlar va uch o‘lchovli massivlar.

7. KO‘RSATKICHLAR VA DINAMIK XOTIRA BILAN ISHLASH

REJA:

1. Ko‘rsatgichlar. Obyektga ko‘rsatkich. void ko‘rsatkich.
2. Dinamik xotira bilan ishlash. Ko‘rsatkich ustida amallar.
3. Ko‘rsatgichlarga dastlabki qiymat kiritish.
4. Ko‘rsatkich ustida amallar.

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

7.1. Ko‘rsatgichlar. Obyektga ko‘rsatkich. void ko‘rsatkich

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
```

7.2. 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.

7.3. 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
```

c) 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.

7.4. 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.

7.5. 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** ko‘rsatkichga yuklanadi. Ko‘rsatkichning o‘zi uchun joy kompilyasiya vaqtida ajratiladi.

delete amali

Ikkinchi operatorda joy ajratishdan tashqari **m** adresiga boshlang‘ich qiymat - **10** sonini joylashtiradi.

Uchinchi operatorda **int** tipidagi **5 ta element** uchun joy ajratilgan va uning boshlang‘ich adresi **q** ko‘rsatkichga berilyapti.

Xotira **new** amali bilan ajratilgan bo‘lsa, u delete amali bilan bo‘shatilishi kerak. Yuqoridagi **dinamik o‘zgaruvchilar** bilan bog‘langan xotira quyidagicha bo‘shatiladi: **delete n; delete m; delete[]q;**

Agarda xotira **new[]** amali bilan ajratilgan bo‘lsa, uni bo‘shatish uchun **delete []** amalini o‘lchovi ko‘rsatilmagan holda qo‘llash kerak.

Xotira bo‘shatilganligiga qaramasdan ko‘rsatkichni o‘zini keyinchalik qayta ishlatish mumkin.

7.6. Ko‘rsatkich ustida amallar

Ko‘rsatkich ustida quyidagi amallar bajarilishi mumkin:

- obyektga vositali murojaat qilish amali;
- qiymat berish amali;
- ko‘rsatkichga o‘zgarmas qiymatni qo‘shish amali;
- ayirish amali;

– inkrement va dekrement amallari;
– solishtirish amali;
– tipga keltirish amali.

Dinamik matritsalar

DInamik xotiralar bilan ishlashda quyida misolni ko‘rib chiqamiz. 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 alohida xotira blokini ajratish;

Butun matritsa uchun bir yo‘la xotira ajratish.

Har bir satr uchun xotira bloki

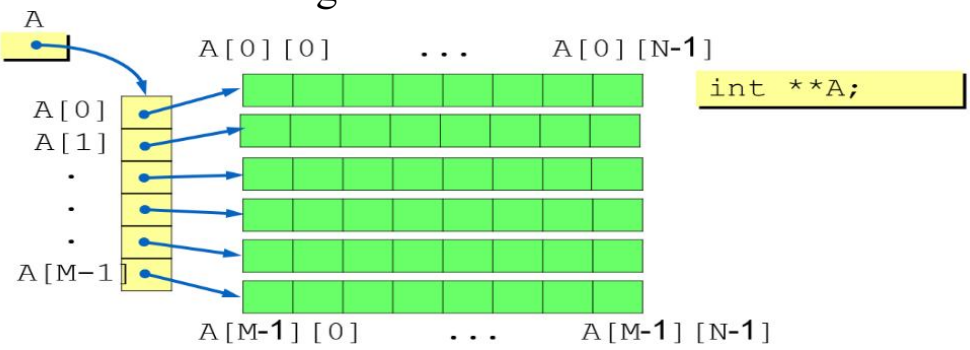
Matritsa adresi:

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
}
```

Nazorat savollari

1. Ko‘rsatkich nima?

2. Ko‘rsatkichlar qanday e’lon qilinadi va qo‘llaniladi?
3. Obyektga ko‘rsatkich nima vazifani bajaradi?
4. void ko‘rsatkich nima vazifani bajaradi?
5. Dinamik xotira bilan qanday ishlanadi?
6. Dinamik massiv va ularni funksiya parametri sifatida qo‘llanilishi.
7. Xotiraning obyektlar o‘rtasidan dinamik taqsimlanuvchi sohasidan joy ajratish uchun qanday operator ishlatiladi?

new unsigned short int deb yozish orqali biz dinamik taqsimlanuvchi xotiradan qancha joy ajratamiz?

8. OBYEKTGA YO‘NALTIRILGAN DASTURLASH ASOSLARI

REJA:

1. Obyektga yo‘naltirilgan dasturlash asoslari.
2. Inkapsulyasiya.
3. Vorislik.
4. Polimorfizm.
5. Sinflar va obyektlar.
6. Tuzilma va birlashmalar.
7. Obyektlar trassirovkasi.
8. Nusxalash konstruktori

Kalit so‘zlar: inkapsulyatsiya, sinf, OYD, konstruktor va destruktor, ro‘yxat, manzil, tugun, adres olish &, bo‘shatish, ko‘rsatkich.

8.1. Obyektga yo‘naltirilgan dasturlash asoslari

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.

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.

8.2. Inkapsulyatsiya

Agar muhandis ishlab chiqarishda diod, triod yoki rezistorni ishlatsa, u bu elementlarni yangitdan ixtiro qilmaydi, balki do‘kondan sotib oladi. Demak, muxandis ularning qanday tuzilganligiga e’tiborini qaratmaydi, bu elementlar yaxshi ishlasa yetarli. Aynan shu tashqi konstruksiyada ishlaydigan yashirinlik yoki avtonomlik xossasi inkapsulyasiya deyiladi.

Inkapsulyasiya – bu ma’lumotlar va ularni qayta ishlovchi kodni birlashtirish mexanizmidir. Inkapsulyasiya ma’lumotlar va kodni tashqi ta’sirdan saqlash imkonini beradi.

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.

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.

8.3. Vorislik

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.

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.

8.4. Polimorfizm

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.

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.

8.5. Sinflar va obyektlar

Dasturda ishlatiladigan har bir o‘zgaruvchi o‘z tipiga ega va u quyidagilarni aniqlaydi:

– xotiradagi o‘lchovini;

– unda saqlanayotgan ma’lumotlarni;

– 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 ;   }
```

8.6. Tuzilma va birlashmalar

Misol uchun, biz talabalarni ifodalash uchun ismingiz, tug'ilgan kuningiz, bo‘yingiz, vazningiz yoki boshqa ma’lumotlarni kiritishni xohlaysiz:

string myName;
int myBirthDay;
int myBirthMonth;
int myBirthYear;
int myHeight;
int myWeight;

Ammo endi sizda 6 ta alohida mustaqil o‘zgaruvchilar mavjud. Agar siz o‘zingiz haqidagi ma’lumotni funksiyaga o‘tkazmoqchi bo‘lsangiz, har bir o‘zgaruvchini alohida o‘tkazishingiz kerak bo‘ladi.

Bundan tashqari, agar siz boshqa birov haqida ma’lumot saqlamoqchi bo‘lsangiz, har bir kishi uchun qo‘shimcha 6 ta o‘zgaruvchini e'lon qilishingiz kerak bo‘ladi!

Bularni barchasini C++ da amalga oshirish mumkin. C++ tili dasturchilarga o‘zlarining foydalanuvchi tomonidan belgilangan ma’lumotlar turlarini - bir nechta alohida o‘zgaruvchilarni birlashtiradigan turlarni yaratishga imkon beradi. Foydalanuvchi tomonidan aniqlangan eng oddiy ma’lumotlar turlaridan biri bu strukturadir. Struktura har xil turdagi o‘zgaruvchilarni bir butunga guruhlash imkonini beradi.

Strukturalar dasturchi tomonidan aniqlanganligi sababli, avvalo kompilyatorga uning umuman qanday ko‘rinishini aytishimiz kerak. Buning uchun **struct** kalit so‘zi ishlatiladi:

```
struct Ishchi
{
   short id;
   int yoshi;
   double maoshi;
};
```

Ishchi strukrurasida uchta o‘zgaruvchi bor:

id tipi short;

yoshi tipi int;

maoshi tipi double.

Strukturaning bir qismi bo‘lgan bu o‘zgaruvchilar struktura a’zolari (yoki "struktura maydonlari") deb ataladi. Ishchi - bu strukturani oddiy e’lon qilinishi. Biz kompilyatorga uning a’zo o‘zgaruvchilari borligini ko‘rsatgan bo‘lsak ham, hozirda uning uchun xotira ajratilmagan. Tuzilma nomlari o‘zgaruvchilar nomlaridan farqlash uchun bosh harf bilan yoziladi.

Ishchi strukturasini ishlatish uchun Ishchi tipidagi o‘zgaruvchini e’lon qilish kerak.

8.7. 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.

8.8. 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.

Nazorat savollari

1. Obyektga yo‘naltirilgan dasturlash tamoyillari.
2. Sinflar va obyektlar
3. Vorislik tushunchasi
4. Polimorfizm tushunchasi
5. Inkapsulyasiya nima?
6. Polimorfizm nima?
7. Vorislikning qo‘llanishi.
8. Sinfdagi o‘zgaruvchilar – sinf a’zolarining qo‘llanishi.
9. Obyektlar trassirovkasi nima?

Nusxalash konstruktori nima?

9. KONSTRUKTORLAR VA DESTRUKTORLAR

REJA

1. Konstruktorlar.
2. Destruktorlar.
3. Friend funksiyalar va sinflar.
4. Ko‘rsatkichlar va sinf metodlari
5. Obyektlar massivi.
6. this ko‘rsatkichi

Kalit so‘zlar: friend, do‘stona (friend) sinflar, sinfning statik ma’lumotlari, destruktorlar, konstruktorlar.

9.1. Konstruktor

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);

9.2. Destruktor

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.

9.3. Friend funksiyalar va sinflar

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;}
```

9.4. Ko‘rsatkichlar va sinf metodlari

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;
}
```

:: 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;
}
```

9.5. 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
```

9.6. this ko‘rsatkichi

this joriy obyektga ko‘rsatkich.

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

Nasl olingan tipga ko‘rsatkich. Bir tipdagi ko‘rsatkich boshqa bir obyekt tipiga ko‘rsata olmaydi. Faqatgina bir holat istisno. Bu ona sinf voris sinflarga ko‘rsatkich bo‘la oladi.

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

Nasl olingan tipga ko‘rsatkich.

```
base *bp;  // ona bp ko‘rsatkich
derived d;  // voris d obyekt
bp = &d;   // base ko‘rsatkich derived ga ko‘rsatadi
//derived obyektiga base ko‘rsatkich orqali 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
ko‘rsatkichni derived ko‘rsatkich tipiga o‘zgartirish kerak
    ((derived*)bp)->set_b(10);
    cout<<((derived*)bp)->get_b();
```

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

E’lon qilinishi:

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

Nazorat savollari

1. Sinflar va obyektlar tushunchasi.
2. Sinf xususiyatlari va metodlari.
3. Konstruktorni destruktordan nima ajratib turadi?
4. Obyektlar massivlarini tushuntiring.
5. Sinf statik xususiyat va metodlari.
6. Obyektlarga ko‘rsatkich qanday yaratiladi?
7. Oddiy sinf bilan nasl olingan sinfga ko‘rsatkich qanday farqlanadi?
8. Do‘st funksiyalar qanday e’lon qilinadi? Do‘st sinflar qanday e’lon qilinadi?
9. Obyektlar massivini inisalizatsiya qilish.

10. SATRLAR VA KENGAYTIRILGAN BELGILAR

REJA:

1. Satrlarga ishlov berish.
2. Standart funksiyalari.
3. Solishtirish.

Kalit so‘zlar: char tipidagi massiv, string, strlen(), sizeof(), strcpy(), strcat(), strstr(), strchr(), assign(), append(), resize(), insert(), delete(), add().

10.1. Satrlarga ishlov berish

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 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
```

Massivlarda bo‘lgani kabi satrlarda ham indeks 0 dan boshlanadi.

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)

10.2. Standart funksiyalari

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 ko‘rsatilgan pos1 o‘rnidan boshlab joylashtiradi. Bunda amal chaqiruvchi satrning pos1 o‘rindan keyin joylashgan belgilar o‘nga 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 o‘chirish uchun quyidagi funksiyani ishlatish mumkin:

erase(unsigned int pos=0,unsigned int n=npos);

Bu funksiya, uni chaqiruvchi satrning pos o‘rnidan boshlab n ta belgini o‘chiradi. Agarda pos ko‘rsatilmasa, satr boshidan boshlab o‘chiriladi. Agar n ko‘rsatilmasa, satrni oxirigacha bo‘lgan belgilar o‘chiriladi:

```
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 to‘liq tozalaydi.
Masalan:
s1.clear(); //satr bo‘sh hisoblanadi (s1=””)
```

Bir satr qismining o‘rniga boshqa satr qismini qo‘yish 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.

```
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
}
```

10.3. Solishtirish

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; }
```

Dastur matni:

```
#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
```

Nazorat savollari

1. Satr simvolli massivdan qanday farq qiladi?
2. Bir o‘lchovli massivlarni initsializatsiya qilish usullarini ko‘rsating.
3. Satrlar bilan ishlovchi standart funksiyalarni sanab o‘ting.
4. Ko‘p o‘lchovli massiv ta’rifi xususiyatlarini ko‘rsating.
5. Ko‘p o‘lchovli massivlar initsializatsiyasi xususiyatlari.
6. Satrlarni initsializatsiya qilish usullarini ko‘rsating.
7. So‘zlar massivi qanday kiritiladi?
8. Satrlar qanday solishtiriladi?
9. compare funksiyasi nima vazifani bajaradi?

11. FAYLLAR VA FAYLLAR BILAN ISHLASH

REJA:

1. Fayllar bilan ishlash. Binar fayllar.
2. Faylga ma’lumot yozish va o‘qish.
3. Fayl ko‘rsatkichi bilan ishlovchi funksiyalar.
4. Fayllar bilan ishlash. Matnli fayllar.
5. Istream va ostream sinfi funksiyalari.

Kalit so‘zlar: Binar fayllar, istream, ostream, fstream, fopen, fwrite, fclose, stream, fseek, fread, open(), is_open().

11.1. Fayllar bilan ishlash. Binar fayllar

Ma’lumotlarni saqlab qo‘yish uchun, tashqi xotiraning nomlangan qismiga fayl deyiladi. Bunday fayllar fizik fayllar deyiladi.

Mantiqiy fayllar. Fizik fayllar bilan ishlash uchun, programmalashtirish tillarida maxsus strukturalashgan, toifalangan fayllar kiritilgan. Bunday fayllar mantiqiy fayllar deyiladi. Mantiqiy fayllar, hech qanday fizik xotirani band qilmasdan ma’lumotlarning mantiqiy modelini o‘zida saqlaydi.

C++ da fizik va mantiqiy fayllar bir - biri bilan fopen funksiyasi orqali bog‘lanadi. Fayl bir nechta elementdan tashkil topgan bo‘lganligi uchun, faqat fayl ko‘rsatkichi ko‘rsatayotgan elementga murojaat qilish mumkin.

Fayldan o‘qish yoki yozish mumkin bo‘lgan o‘rinni ko‘rsatuvhi elementga fayl ko‘rsatkichi deyiladi. Fayldan ma’lumot o‘qiganda yoki yozganda fayl ko‘rsatkichi avtomat ravishda o‘qilgan yoki yozilgan bayt miqdoricha siljiydi. Fayl ko‘rsatkichini magnitafon galovkasiga o‘xshatish mumkin.

Binar fayl - har xil obyektlarni ifodalovchi baytlar ketma - ketligidir. Obyektlar faylda qanday ketma - ketlikda joylashganini programmaning o‘zi aniqlashi lozim.

Fayllar bilan ishlovchi funksiyalardan foydalanish uchun <stdio.h> sarlavha faylini programmaga qo‘shish kerak bo‘ladi. Fayldan ma’lumotlarni o‘qish yoki yozish uchun ochish fopen funksiyasi orqali amalga oshiriladi.

FILE * fopen (const char * filename, const char * mode);

filename - o‘zgaruvchisi char toifasidagi satr bo‘lib, faylning to‘liq nomini ko‘rsatishi lozim. (filename ="D:\C++\laboratoriya\example.txt"). Agar faylning faqat nomi

ko‘rsatilgan bo‘lsa, fayl joriy katalogdan qidiriladi (filename = " example.txt ").

mode - o‘zgaruvchisi ham char toifasidagi satr bo‘lib, faylni qaysi xolatda ochish lozimligini bildiradi.

mode qiymati	Faylning ochilish xolati
"w"	Faylni yozish uchun ochish. falename o‘zgaruvchisida ko‘rsatilgan fayl hosil qilinadi va unga ma’lumot yozish mumkin bo‘ladi. Agar fayl oldindan bor bo‘lsa (ya’ni oldin hosil qilingan bo‘lsa), faylning ma’lumotlari o‘chiriladi va yangi bo‘sh fayl faqat yozish uchun ochiq holda bo‘ladi.
"r"	Fayl o‘qish uchun ochiladi. Agar fayl oldindan mavjud bo‘lmasa, xatolik sodir bo‘ladi. Ya’ni ochilishi lozim bo‘lgan fayl oldindan hosil qilingan bo‘lishi shart.
"a"	Faylga yangi ma’lumotlar qo‘shish - kiritish uchun ochiladi. Yangi kiritilgan ma’lumotlar fayl oxiriga qo‘shiladi. Agar fayl oldindan mavjud bo‘lmasa, yangi fayl hosil qilinadi.
"w+"	Yozish va o‘qish uchun faylni ochish. Agar fayl oldindan bor bo‘lsa (ya’ni oldin hosil qilingan bo‘lsa), faylning ma’lumotlari o‘chiriladi va yangi bo‘sh fayl yozish va o‘qish uchun ochiqholda bo‘ladi.
"r+"	Oldindan mavjud bo‘lgan faylni o‘qish va yozish uchun ochish.
"a+"	Fayl ma’lumotlarni o‘qish va yangi ma’lumot qo‘shish uchun ochiladi. fseek, rewind

Faylni ochishda xatolik sodir bo‘lsa, fopen funksiyasi NULL qiymat qaytaradi. Ochilgan faylni yopish uchun fclose funksiyasi ishlatiladi.

int fclose (FILE * stream);

Faylni yopishda xato sodir bo‘lmasa, fclose funksiyasi nol qiymat qaytaradi. Xato sodir bo‘lsa, EOF - fayl oxiri qaytariladi.

11.2. Faylga ma’lumot yozish va o‘qish

size_t fread (void * ptr, size_t size, size_t n, FILE * stream);

fread funksiyasi, fayldan ptr ko‘rsatkichi adresiga size xajmdagi ma’lumotdan n tani o‘qishni amalga oshiradi. Agar o‘qish muvoffaqiyatli amalga oshsa *fread* funksiyasi o‘qilgan bloklar soni n ni qaytaradi. Aksholda nol qaytariladi

size_t fwrite (const void * ptr, size_t size, size_t n, FILE * stream);

fwrite funksiyasi, faylga ptr ko‘rsatkichi adresidan boshlab *size* xajmdagi ma’lumotdan n tani yozishni amalga oshiradi.

11.1-misol. fread va fwrite funksiyalarining qo‘llanilishi

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main()
{
Int n = 10; double d = 12.7;
char s[15] = "Talaba"; FILE *f; // binar faylni yozish uchun ochamiz
f = fopen("file.dan", "wb");
fwrite(&n, sizeof(int), 1, f); // n sonini faylga yozish
fwrite(&d, sizeof(double), 1, f); // d sonini faylga yozish
// satrni faylga yozish
fwrite(s, sizeof(char), strlen(s) + 1, f);
fclose(f); // faylni yopish
n = 0; d = 0; // binar faylni o‘qish uchun ochamiz
f = fopen("file.dan", "rb");
fread(&n, sizeof(int), 1, f); // n sonini fayldan o‘qish
fread(&d, sizeof(double), 1, f); // d sonini fayldan o‘qish
// satrni fayldan o‘qish
fread(s, sizeof(char), strlen(s) + 1, f);
fclose(f); // faylni yopish
cout << n << endl; cout << d << endl; cout << s << endl; return 0;
}
```

yuqoridagi misolda satrni yozish va o‘qish uchun quyidagicha kod ishlatildi:

fwrite(s, sizeof(char), strlen(s) + 1, f);

fread (s, sizeof(char), strlen(s) + 1, f);

Buning kamchiligi s satridagi har bir belgi alohida - alohida faylga yozildi va o‘qildi. Bu masalani quyidagicha hal qilish mumkin edi:

fwrite(s, sizeof(s), 1, f);

fread (s, sizeof(s), 1, f);

Lekin bu usulning ham kamchiligi bor. Ya’ni s satri belgilari soni massiv o‘lchamidan kam bo‘lgan holda, keraksiz ma’lumotlarni saqlash va o‘qish sodir bo‘ladi.

11.3. Fayl ko‘rsatkichi bilan ishlovchi funksiyalar

Fayldan ma’lumot o‘qiganda yoki yozganda fayl ko‘rsatkichi avtomat ravishda o‘qilgan yoki yozilgan bayt miqdoricha siljiydi. Fayl ko‘rsatkichining kelgan joyini aniqlash uchun *ftell* funksiyasi ishlariladi.

long int ftell (FILE * stream);

Fayl ko‘rsatkichini siljitish uchun *fseek* funksiyasi ishlatiladi.

int fseek (FILE * stream, long int offset, int whence);

Bu funksiya *offset* da ko‘rsatilgan bayt miqdoricha siljishni amalga oshiradi. *whence* o‘zgaruvchisi quyidagi qiymatlarni qabul qilishi mumkin:

O‘zgarmas	whence	Izoh
SEEK_SET	0	Fayl boshiga nisbatan siljitish
SEEK_CUR	1	Fayl ko‘rsatkichining joriy xolatiga nisbatan siljitish
SEEK_END	2	Fayl oxiriga nisbatan siljitish

Agar *whence* = 1 bo‘lsa (*SEEK_CUR*), offset musbat (o‘ngga siljish) yoki manfiy (chapga siljish) bo‘lishi mumkin.

Fayl ko‘rsatkichini faylning boshiga o‘rnatish uchun rewind funksiyasi ishlatiladi. void rewind (*FILE * stream*);

Bu amalni fayl ko‘rsatkichini siljitish orqali ham amalga oshirish mumkin.

fseek (f, 0, SEEK_SET);

Agar faylda faqat butun sonlar yozilgan bo‘lsa, uning k - elementiga murojaat quyidagicha bo‘ladi.

fseek (f, sizeof(int) * (k - 1), SEEK_SET); fread (&n, sizeof(int), 1, f);

Fayl oxirini aniqlash uchun feof funksiyasi ishlatiladi. int feof (*FILE * stream*);

feof funksiyasi fayl ko‘rsatkichi fayl oxirida bo‘lsa, noldan farqli qiymat qaytaradi. Boshqa hollarda nol qaytaradi.

11.2- misol. n natural soni berilgan. Elementlari n ta butun sondan iborat bo‘lgan faylni hosil qiluvchi va ekranga chiqaruvchi programma tuzilsin.

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main() {
   int n, k;
   FILE *f;
   f = fopen("binar", "wb+");
// binar faylni yozish va o‘qish uchun ochish
   if (f == NULL) {
      cout << "Faylni hosil qilishda xato bo‘ldi"; return 1;
   }
   cout << "n="; cin >> n;
   for (int i = 0; i < n; i++) {
      cin >> k;
      fwrite(&k, sizeof(k), 1, f);
   }
// fayl ko‘rsatkichini fayl boshiga qo‘yish
    rewind(f);
    while (fread(&k, sizeof(k), 1, f))   {
//fayl boshidan fayl ko‘rsatkichi turgan o‘ringacha bo‘lgan baytlar
       int bayt = ftell(f);
       cout << k <<" ftell(f)=" << bayt << endl;
   }
fclose(f);
return 0;
}
```

11.3- misol. “Guruh.txt” nomli faylda guruh talabalarining familiya va ismlar ro‘yhati keltirilgan. Ushbu fayldan foydalanib qizlar va yigitlarni familaya va ismini aloxida fayllarga yozish dasturini tuzaylik:

```
#include <iostream>
```

```
#include <fstream>
#include <string>
using namespace std;
int main ()
{
ifstream Guruh("Guruh.txt");
ofstream erkak("erkak.txt");
ofstream ayol("ayol.txt");
if (!Guruh.is_open())
{
cout << "Guruh.txt - fayli topilmadi\n";
return 1;
}
string s, erk, ayl;
int p;
cout << "Guruh.txt fayli ma'lumotlari\n";
while (!Guruh.eof())
{
getline(Guruh, s);
p = s.find(" ");
if (s[p-1]=='a'){
   ayol << s << endl;
}
else
    erkak << s << endl;
}
Guruh.close();
erkak.close();
ayol.close();
//system (“pause”);
return 0;
}
```

11.4. Fayllar bilan ishlash. Matnli fayllar

Matnli fayllar bilan ishlash binar fayllar bilan ishlashdan bir oz farq qiladi. Matnli fayllarda ma`lumotlar satrlarda saqlanadi. Matnli fayl elementilari har xil uzunlikdagi satrlardir. Bu satrlar bir biridan satr oxiri belgisi bilan ajratiladi. Matnli fayl elementlari indekslanmagan

bo‘lganligi uchun, faylning istalgan elementiga bevosita murojaat qilib bo‘lmaydi.

C++ da matnli yoki binar fayllar bilan ishlash uchun keng imkoniyatlar berilgan. Matnli fayllar bilan ishlashda oddiy C ning funksiyalaridan ham foydalanish mumkin. Masalan, formatli o‘qish va yozish funksiyalari yoki oldingi mavzudagi funksiyalardan foydalanishimiz mumkin. Matnli fayllar bilan ishlashning bunday usuli kitoblarda keng yoritilgan. Ularni mustaqil o‘qib - o‘rganishingiz mumkin.

Bu mavzu fayllar bilan ishlovchi oqimlarni qisqacha o‘rganamiz va buni matnli fayl misolida ko‘ramiz.

Standart kiritish / chiqarish kutubxonasi sinflari quyidagicha shajaraga ega:

Fayllar bilan ishlash uchun quyidagi sifnlar obyektlari hosil qilinadi:

- ofstream - faylga ma'lumot yozish uchun;
- ifstream - fayldan ma'lumot o‘qish uchun;
- fstream - fayldan ma'lumot o‘qish uchun va yozish uchun.

Bu sinflarni dasturda ishlatish uchun <fstream> sarlavha faylini qo‘shish kerak bo‘ladi. Bundan keyin programmada aniq fayllar oqimini aniqlash mumkin. Masala:

ofstream yozish; // faylga yozish oqimini e'lon qilish

ifstream oqish; // fayldan o‘qish oqimini e'lon qilish

fstream yoz_oqi; // faylga yozish va o‘qish oqimini e'lon qilish

Keyin faylni ochish kerak bo‘ladi. Faylni ochish deganda, uning ustida nima amal qilinishi haqida amaliyot tizimiga xabar berish tushuniladi.

void open (const char * filename, ios_base::openmode mode = ios_base::out); mode parametri quyidagicha qiymatlarni qabul qilishi mumkin:

ios::in	faqat ma'lumot o‘qish uchun
ios::out	faqat ma'lumot yozish uchun
ios::ate	faylni ochishda fayl ko‘rsatkichini fayl oxiriga qo‘yish
ios::app	fayl oxiriga ma'lumotlarni yozish uchun
ios::trunc	bor bo‘lgan faylning ustidan yangi faylni yozish
ios::binary	binar holda ma'lumotlarni almashish uchun

Har bir sinf uchun mode parametrining odatiy qiymatlari mavjud:

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

Fayl ustida o‘qish yoki yozish amalini bajarib bo‘lgandan song, faylni yopish kerak bo‘ladi. Faylni yopish uchun close funksiyadi ishlatiladi. Masalan:

yozish.close(); oqish.close();

11.4-misol. Matnli faylga ma'lumot yozish

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
ofstream yozish; // faylga yozish oqimini hosil qilish
yozish.open("example.txt");
// yangi example.txt nomli fayl hosil qilinadi.
// agar example.txt fayli oldindan bo‘lsa,
// uning eski qiymatlari o‘chiriladi
// va yangi fayl hosil qilinadi
yozish << "Matnli faylga " << endl;
yozish << "ixtiyori malumotlar " << endl;
yozish << "yozildi!" << endl;
yozish.close(); // faylni yopish
return 0;
}
```

11.5-misol. Matnli fayldan o‘qish

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main ()
{
ifstream oqish; // fayldan o‘qish oqimini hosil qilish
```

```
string satr;

oqish.open("example.txt");

// faylni ochishda xatolik sodir bo‘lsa
if (!oqish.is_open())
{
cout << "Faylni ochishda xatolik sodir bo‘ldi." << endl;
exit(1); // dasturni tugatish
}

while (!oqish.eof())
{
// fayldan o‘qish
getline(oqish, satr);
// ekranga chiqarish
cout << satr << endl;
}

// faylni yopish
oqish.close();
return 0;
}
```

11.5. Istream va ostream sinfi funksiyalari

istream& seekg (streampos pos);

istream& seekg (streamoff off, ios_base::seekdir dir);

oqish oqimi ko‘rsatkichini o‘rnatish (siljitish).
pos - oqim buferining yangi pozitsiyasi.
dir parametri quyidagilardan birini qabul qilishi mumkin:

Qiymat	Izoh
ios::beg	oqimning boshlanishi
ios::cur	oqimning joriy xolari
ios::end	oqim oxiri

long tellg(); -o‘qish oqimining joriy xolatini (pozitsiyasi) aniqlash.

ostream& seekp (streampos pos);

ostream& seekp (streamoff off, ios_base::seekdir dir);

yozish oqimi o‘rnini (pozitsiyasini) o‘rnatish.
pos - oqim buferining yangi pozitsiyasi

dir parametri beg, cur, end qiymatlaridan birini qabul qilishi mumkin.

long tellp() - yozish oqimining kelgan joyini aniqlash.

11.6-misol. Fayldan nusxa olish

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
int length;
char * buffer, fayl[] = "matn.txt", yangi[]="yangi_fayl.txt";
 // fayl - nusxalanadigan fayl nomi
// yangi - yangi nusxalangan fayl nomi
// o‘qish oqimi
ifstream fromfile(fayl, ios::binary );
if (!fromfile.is_open())
{
cout << "faylni o‘qishda xatolik sodir bo‘ldi\n"; exit(1);
}
// yozish oqimi
ofstream tofile(yangi, ios::binary );
// fayl xajmini aniqlash:
fromfile.seekg (0, ios::end); // fayl oxiriga o‘tish
length = fromfile.tellg();
fromfile.seekg (0, ios::beg); // fayl boshiga o‘tish
// xotira ajratish:
buffer = new char [length];
// blokka ma'lumotlarni o‘qish:
fromfile.read (buffer, length); fromfile.close();
// nusxalanishi kerak bo‘lgan faylga yozish
tofile.write (buffer, length);
// xotirani bo‘shatish
delete[] buffer;
cout << "fayl nusxalandi\n";
return 0;
}
```

dic.txt nomli fayl berilgan. Faylning har bir satrida inglizcha va o‘zbekcha so‘zlar "-" belgisi bilan ajratilgan. Inglizcha so‘zlarni

english.txt fayliga, o‘zbekcha so‘zlarni uzbek.txt fayliga o‘tkazuvchi programma tuzilsin.

dic.txt fayli quyidagicha bo‘ladi:

hello - salom
bread - non car - mashina

11.7-misol. Fayldan o‘qish va faylga yozish

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main ()
{
ifstream dic("dic.txt"); ofstream uzbek("uzbek.txt");
ofstream english("english.txt");
if (!dic.is_open())
{
cout << "dic.txt - fayli topilmadi\n"; exit(1);
}
string s, uzb, eng; int p;
cout << "dic.txt fayli ma'lumotlari\n"; while (!dic.eof())
{
getline(dic, s); p = s.find("-");
eng.assign(s, 0, p - 1);
uzb.assign(s, p + 1, s.length() - (p + 1));
uzbek << uzb << endl; english << eng << endl;

cout << s << endl;
}

dic.close(); uzbek.close(); english.close();

return 0;
}
```

Nazorat savollari

1. Fizik fayl deb qanday fayllarga aytiladi?
2. Mantiqiy fayl deb qanday fayllarga aytiladi?
3. Fizik va matiqiy fayllar qanday bog`laniladi?
4. Fayllar nima uchun va qanday ochiladi?
5. Fayl qanday yopiladi?

6. Fayl ko‘rsatkichi deb nimaga aytiladi?
7. Fayllar necha xil xolatda bo‘lishi mumkin?
8. Fayllar bilan ishlovchi qaysi sinflarni bilasiz?
9. Matnli fayllarni toifali fayllardan qanday farqi bor?
10. Fayllarga oqimli yozish va o‘qishda, oqimni ochish qanday bo‘ladi?
11. istream sinfi funksiyalarni tushuntirib bering?
12. ostream sinfi funksiyalarni tushuntirib bering?

12. INKAPSULYATSIYA VA MEROSXO‘RLIK

REJA:

1. Obyektga yo‘naltirilgan dasturlash (OYD).
2. Sinf va obyekt yaratish.
3. Bir nechta obyektlar.
4. C++da inkapsulatsiya.
5. Merosxo‘rlik.

Kalit so‘zlar: Object Oriented Programming, class, object, public, private, protected, Sinf yaratish, obyekt yaratish, encapsulation, merosxo‘rlik, ota sin, bola sinf, vorislik.

12.1. Obyektga yo‘naltirilgan dasturlash (OYD)

OYD (OOP - Object Oriented Programming) - bu biron bir maqsadga yo‘naltirilgan dasturlash degan ma'noni anglatadi.

Protseduraviy dasturlash - bu ma'lumotlarga ishlov beradigan protseduralar yoki funktsiyalarni yozish, obyektga yo‘naltirilgan dasturlash esa ma'lumot va funktsiyalarni o‘z ichiga olgan obyektlarni yaratishga mo‘ljallangan.

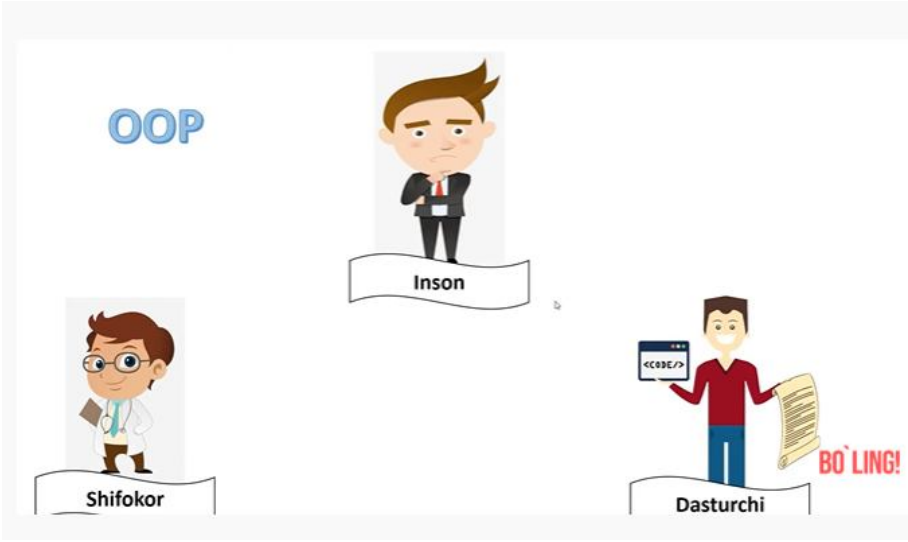
Obyektga yo‘naltirilgan dasturlash protsessual(oddi) dasturlashdan bir qator afzalliklarga ega:

- OOP tezroq va bajarilishi osonroq
- OOP dasturlarning aniq tuzilishini ta'minlaydi
- OOP C ++ kodini DRY "Don't Repeat Yourself" saqlashga yordam beradi va kodni saqlash, o‘zgartirish va disk raskadrovka qilishni osonlashtiradi.
- OOP kodni kam va ishlab chiqarish vaqtini qisqartirgan holda to‘liq qayta ishlatiladigan ilovalarni yaratishga imkon beradi.

12.2. C ++ da sinf va obyektlar

Sinflar va obyektlar obyektga yo‘naltirilgan dasturlashning ikkita asosiy jihati. Quyidagi sinf va obyektlar o‘rtasidagi farqlarga ma’lumotlar keltirilgan:

sinf	**obyektlar**
Meva	olma Banan Mango
Avtomobil	Daewoo Audi Toyota



12.1-rasm. Klass va obyektlar o‘rtasidagi bog‘liqlik.

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 funksiyaa’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.

Obyekt - biz yashayotgan olamdagi biror elementga hos bo‘lgan barcha ma’lumot va hulqlarni, ya’ni shu element ustida bajarish mumkin bo‘lgan xarakatlarni ifodalaydi hamda ma’lumotlarning tugal abstraksiyasidan iborat bo‘ladi. Bu ma’lumot va hulqlar obyektga yo‘naltirilgan dasturlash atamasida mos ravishda hususiyat va metod deb ataladi. Hususiyatni obyektning maydoni deb ham yuritiladi. Masalan, shashka obyekti rang, vertikal maydondagi o‘rni, gorizontal maydondagi o‘rni kabi maydonlarga, surish, urish, «damka» ga chiqish, shashka taxtasidan chetga olish kabi metodlarga ega bo‘ladi. Maydon va metodlar birgalikda obyektning a’zolari deyiladi. Obyektlarning strukturasi ularning o‘zaro aloqasini ifodalaydi. Obyektlar o‘zlarining barcha xarakteristika va hulqlarining o‘ziga hos tomonlarini birgalikda saqlaydi.

12.3. Sinf va obyekt yaratish

Sinf yaratish uchun class kalit so‘zdan foydalaning. "MyClass" nomli sinf yaratamiz.

12.1-misol. Klass yaratish.

```
#include <string>
using namespace std;
class MyClass {       // class yaratish
  public:             // ochiqlik siyosati
    int myNum;        // Attribute (int tipiga tegishli)
    string myString;  // Attribute (string tipiga tegishli)
};
int main() {
return 0;
}
```

Sinf yaratish jarayoni:

- **class** kalit so‘z **MyClass** deb atalgan bir sinf yaratish uchun ishlatiladi;
- **public** kalit so‘z murojaatlarni ochiqligini bildiradi. Bu degani classdan tashqarida ham class attributlardan foydalanish mumkin;
- Sinf ichida butun son **myNum** va satr o‘zgaruvchi **myString** e’lon qilingan. O‘zgaruvchilar sinf ichida e'lon qilinganida, ular atributlar deb nomlanadi;
- sinf tanasini nuqta-vergul bilan tugatiladi.

C++ dasturlash tilida **MyClass** nomli class yaratilgan va shu class asosida obyekt yaratishni ko‘rib o‘tamiz. Obyektni yaratish uchun **MyClass** sinf nomini, so‘ngra obyekt nomini ko‘rsatiladi.

12.2-misol. Klass va obyekt yaratish.

```
#include <string>
using namespace std;
class MyClass {
  public:
    int myNum;
    string myString;
};
int main() {
  MyClass myObj;
  myObj.myNum = 15;
  myObj.myString = "Mening birinchi dasturim";
  cout << myObj.myNum << "\n";
```

```
  cout << myObj.myString;
  return 0;
}
```

Natija:

```
15
Memimg birinchi dasturim
```

12.4. Bir nechta obyektlar

Siz bitta sinfning bir nechta obyektlarini yaratishingiz mumkin:

12.3-misol. Klass va bir necha obyekt yaratish.

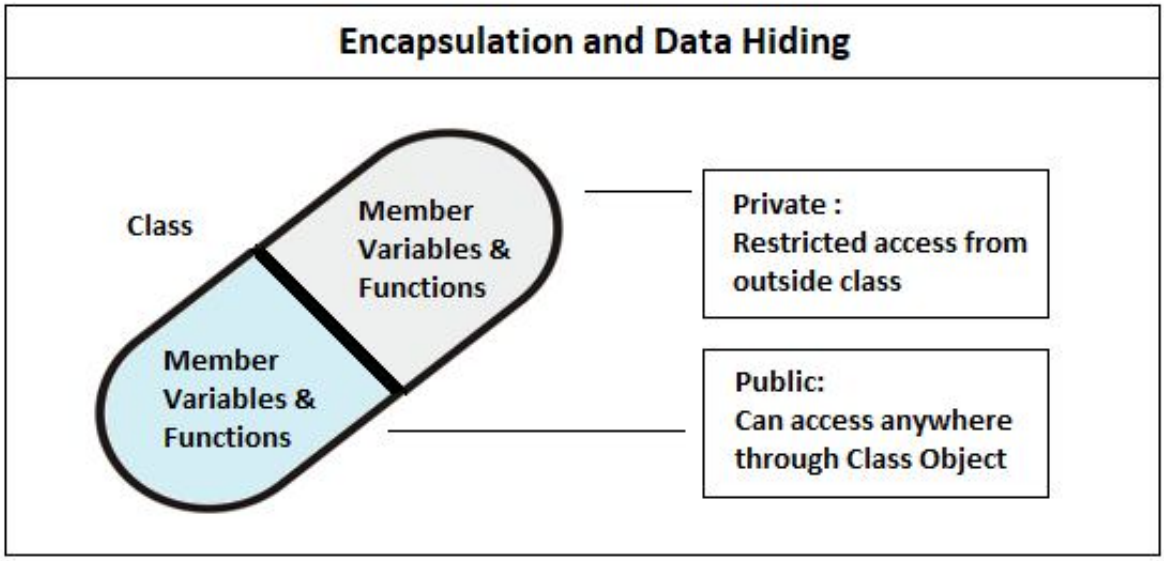
```
#include <string>
using namespace std;
class Car {
  public:
    string brand;
    string model;
    int year;
};
int main() {
  Car carObj1;
  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;
  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;
cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year
<< "\n";
cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year
<< "\n";
  return 0;
}
```

Natija:

```
BMW X5 1999
Ford Mustang 1969
```

12.5. C++da inkapsulatsiya

Obektga yo‘naltirilgan dasturlashda klasslardan foydalanamiz. Klasslar tarkibidagi o‘zgaruvchi va funksiyalar ommaviy (public) hamda hususiy(private) bo‘lishi mumkin. Klass tarkibidagi o‘zgaruvchi va funksiyalarga faqat klassni ichida murojat qilish, obyektlarga klass o‘zgaruvchilarini va funksiyalarini ko‘rinmas holatga keltirib qo‘yish jarayoni **Inkapsulyatsiya** deyiladi.



12.2-misol. Inkapsulatsiya strukturasi

12.4-misol. Mobile va home nomerlarni class orqali aniqlash dasturi. Telefon nomerlarni inkapsulatsiya orqali ifodalash

```
#include <iostream>
using namespace std;
class Contact
{
   private:
      int mobile_number;          // private variable
      int home_number;            // private variable
   public:
      void print_numbers()
      {
         mobile_number = 12345678;
         home_number = 87654321;
         cout << "Mobile number: " << mobile_number;
         cout << ", home number: " << home_number << endl;
      }
};
```

```
int main()
{
   Contact Tony;
   Tony.print_numbers();
    return 0;
}
```

12.5-misol. Kiritilayotgan sonlarni yig`indisini topish. Yig`indi o‘zgaruvchisini inkapsulya orqali ifodalash

```
#include <iostream>
using namespace std;
class Adder {
  public:
    // interface to outside world
    void addNum(int number) {
      total += number;
    }
    // interface to outside world
    int getTotal() {
      return total;
    };
  private:
    // hidden data from outside world
    int total;
};
int main() {
  Adder a;
  a.addNum(11);
  a.addNum(20);
  a.addNum(30);
  cout << "Total " << a.getTotal() <<endl;
  return 0;
}
```

Inkapsulatsiyani nima uchun kerak:

- xavfsizlik uchun juda muhim. Yashirin ma'lumotlarni saqlash va undan foydalanuvchilarni cheklash;
- **class** ni ichida xavfsizlikga ega bo‘lgan attributga murojaat qilish imkoniyati mavjudligi.

12.6. Merosxo‘rlik

Merosxo‘rlik - ma’lum obyekt asosida boshqa obyektni yaratish jarayoniga aytiladi. Bir klassning boshqa klassdan meros olishi yordamida amalga oshiriladi. Meros olingan obyekt ota obyektdagi xususiyatlarni tanlovga ko‘ra meros oladi.

Masalan, avtoulov bu ota obyekt. Bu obyekt yordamida yengil mashina, yuk mashinasi, poyga mashinasi kabi boshqa obyektlarni yaratib olishimiz mumkin. Ota klassda bo‘lgan 4 g’ildirak farzand klasslarda ham mavjud bo‘ladi. Ya’ni poyga mashinasi, avtoulovdan g’ildiraklarni meros oladi. Ota klassdan meros olayotgan bola klassimiz ota klassning shaxsiy bo‘lmagan (**private**) barcha o‘zgaruvchilari, funksiyalari va h.k larni meros qilib oladi.

Meros olish:

- derived class (bola) - boshqa sinfdan meros qolgan sinf.
- base class (ota-ona) - meros bo‘lib o‘tgan sinf.

Sinfdan meros olish uchun : belgidan foydaniladi.

12.6-misol. Car sinf (bola) atributlar va usullarni Vehicle sinfdan (ota-onadan) meros qilib oladi:

```
#include <iostream>
#include <string>
using namespace std;
// Base class
class Vehicle {
  public:
    string brand = "GM";
    void honk() {
      cout << "Salom avtomobil ishqibozlari ! \n" ;
    }
};
// Derived class
class Car: public Vehicle {
  public:
    string model = "Cobalt";
};
int main() {
  Car myCar;
  myCar.honk();
  cout << myCar.brand + " " + myCar.model;
  return 0;
```

```
}
```

12.7-misol. *Person* klassidan *teacher* va *dasturchi* klasslarini meros olish

```
#include <iostream>
using namespace std;
class Person{
public:
   string Fulname;
   int age;
   int salary;
   void inson(){
      cout<<" Ushbu shaxs haqida umumiya
               ma'lumotlar:"<<endl;    }
};
class teacher: public Person{
public:
   void full(){
   cout << " O‘qituvchi"<< endl;
   cout << " yoshi: "<<age<< endl;
   cout << " Maoshi: "<<salary<< endl;
   }
} ;
class dasturchi:public Person{
public:
   void full(){
   cout << " O‘qituvchi"<< endl;
   cout << " yoshi: "<<age<< endl;
   cout << " Maoshi: "<<salary<< endl;  }
};
   int main()
{
   teacher t1;
   t1.Fulname='Akramov Alisher';
   t1.age=35;     t1.salary=5300000;
   t1.inson();       t1.full();
   dasturchi d1;
   d1.Fulname="Alisherov Akram";
   d1.age=40;      d1.salary=13500000;
```

```
    d1.inson();      d1.full();
    return 0;
}
```

Nazorat savollari

1. Sinf nima ?
2. Obyekt nima ?
3. OYD nima uchun kerak ?
4. Inkapsulatsiya qanday amalga oshiriladi ?
5. Ota va bola klasslar tushunchasi.
6. Klasslardan meros olish jarayoni.

13. POLIMORFIZM.
REJA:

1. Polimorfizm nima?
2. Polimorfizmning turlari.
3. Virtual funksiya.
4. Abstrakt sinf va funksiyalar

Kalit so‘zlar: Polimorfizm, virtual funksiya, abstrakt sinf, compile time, run time, function overloading, operator overloading, bitta interfeys, overload, override.

13.1. Polimorfizm va uning turlari

Polimorfizm bu "ko‘p shakllar" degan ma'noni anglatadi. Biz Meros orqali bir-biri bilan bog'liq bo‘lgan ko‘plab sinflarga ega bo‘lganimizda paydo bo‘ladi. **Polimorfizm**da turli xil vazifalarni bitta harakatda turli yo‘llar bilan bajarish imkoniyatiga ega bo‘lamiz.

Obyektga abstrakt darajada qarash xususiyati. Masalan, turli xil oynalar mavjud: deraza oynasi, eshik oynasi, mashina oynasi, telefon oynasi. Bularning barchasi bir biridan ishlatilish sohasi, tuzulishi, shakli bilan farq qiladi. Lekin barchasini umumiy qilib oyna deb qarash mumkin. Polimorfizm turli xil obyektlar bilan bir xil uniformada ishlash imkoniyatini beradi.

Polimorfizm orqali bir jarayonni turli yo‘llar bilan tashkillashtirishimiz mumkin. Polimorfizm so‘zi yunoncha ikki so‘zning birikmasidan tashkil topgan «poly» — ko‘p va «morphs» — formalar. Polimorfizm ham ***ko‘p formalar*** degan ma'noni anglatadi.

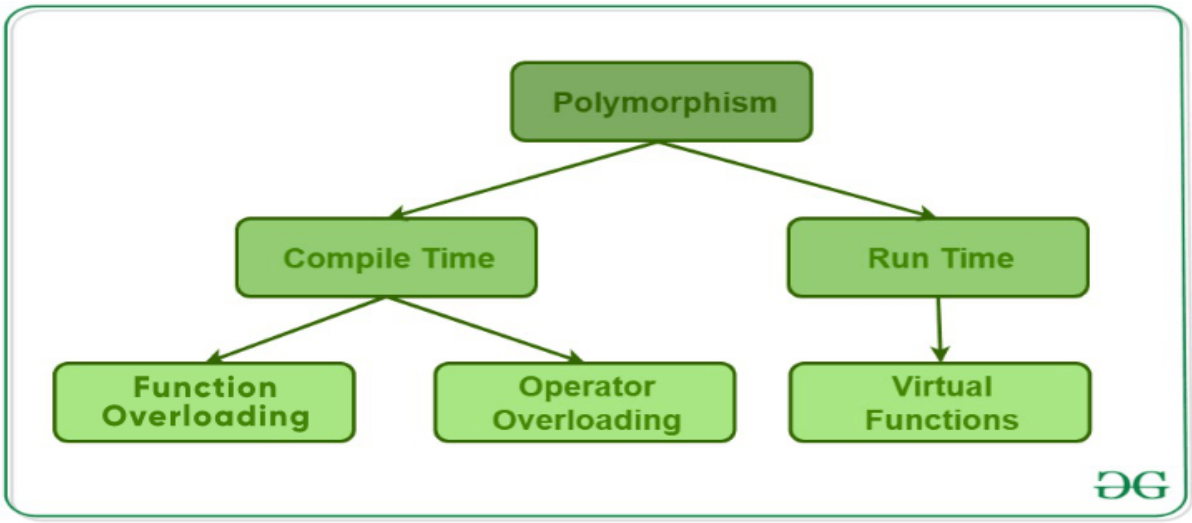
13.1-misol.

```
#include <iostream>
#include <string>
using namespace std;
// Base class
class Animal {
  public:
    void animalSound() {
      cout << "Bu jonzotlar shunday ovoz chiqaradi \n" ;
    }
};
// Derived class
class Pig : public Animal {
```

```
  public:
    void animalSound() {
      cout << "Cho‘chqa: vee vee \n" ;
    }
};
// Derived class
class Dog : public Animal {
  public:
    void animalSound() {
      cout << "It: vov vov \n" ;
    }
};
int main() {
  Animal myAnimal;
  Pig myPig;
  Dog myDog;
  myAnimal.animalSound();
  myPig.animalSound();
  myDog.animalSound();
  return 0;}
```

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

- compile time;
- run time;



13.1-rasm. Polimorfizm va uning turlari

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.

Unga quyidagicha ham ta’rif berish mumkin:

- Polimorfizm – bitta nom bilan bir nechta funksiyalarni qayta yuklanishi.

- Polimorfizm - funksiyaning har xil turdagi ma'lumotlar bilan ishlash qobiliyati.

13.2. Virtual funksiya

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

Virtual funksiya - bu shunday funksiyaki, uni chaqirish va mos amallarni bajarish, uni chaqirgan obyekt turiga bog‘liq bo‘ladi.

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

13.2-misol. Virtual funksiyalar bilan ishlash

```
class base {
public:
        virtual void vfunc() {
                cout << "Bu asosiy funksiya vfunc() \n";
        }
};
class derived1 : public base {
public:
        void vfunc() {
```

```
    cout << "Bu 1-vorisda funksiya vfunc().\n";
        }
};
class derived2 : public base {
public:
        void vfunc() {
                cout << "Bu 2-vorisda funksiya vfunc().\n";
        }
};
int main()
{
        base *p, b;
        derived1 d1;
        derived2 d2;
        // base sinfga ko‘rsatkich
        p = &b;
        p->vfunc(); // base sinf vfunc() virtual funksiyasi
        // derived1 sinfga ko‘rsatkich
        p = &d1;
        p->vfunc(); // derived1 sinf vfunc() virtual funksiyasi
        // derived2 sinfga ko‘rsatkich
        p = &d2;
        p->vfunc(); // derived2 sinf vfunc() virtual funksiyasi
        return 0;}
```

Ushbu dasturga ko‘ra, **base** sinfida **vfunc( )** nomli virtual funksiya e’lon qilingan.

Bu yerda ishlatilgan **virtual** kalit so‘zi qolgan voris sinfdagi funksiyalar ishlashi uchun imkoniyat yaratadi.

derived1 va **derived2** voris sinflarda **vfunc( ) funksiya uchun virtual** kalit so‘zini ishlatish shart emas.

Demak, asos sinf ko‘rsatkich obyekti (*p) voris sinf a’zolariga ya’ni funksiyalariga murojaat qila oladi, agar ushbu funksiya asos sinfda virtual deb e’lon qilingan bo‘lsa

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

Asos sinfdan voris olinganda ushbu sinfdagi virtual funksiya ham vorislik xususiyatiga ega bo’ladi.

Bu shuni bildiradiki, asos sinf virtual funksiyasi voris sinf uchun mavjud bo'lgani bilan birga, ushbu voris sinfdan yana voris olingan holda ham ushbu virtuallik xususiyati saqlanib qoladi. Ya'ni ushbu funksiya ikkinchi voris sinf uchun ham override qilinadi.

Bu xususiyat bir nechta vorislikda ham saqlanib qoladi.

Quyidagi misolda ko'ramiz:

```
#include <iostream>
using namespace std;
class Asosiy{
 public:
  Asosiy(int _x) {x=_x;}
  virtual int Qiymat_X(){return x;} //virtual funksiya
  virtual void Chop_X(); //virtual funksiya
 private:
  int x;};
```

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.

```
#include <iostream>
using namespace std;
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); }
};
```

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.

13.3. Abstrakt sinf va funksiyalar

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

13.3-misol. Abstrakt sinflar

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

Nazorat savollari

1. Polimorfizm nima ?
2. Dasturlashda polimorfizm nima uchun kerak ?
3. Vertual funksiyalar.
4. Abstrakt sinflar.
5. Abstrakt funksiyalar.

14. OPERATORLARNI QAYTA YUKLASH

REJA:

1. C++da operatorlarni qayta yuklanish nima?
2. Kiritish va chiqarish operatorlarini qayta yuklash.
3. Qo‘shish va ayirish operatorlarini qayta yuklash.
4. Binar operatorlar qanday qayta yuklanadi?

Kalit so‘zlar: operator, binar, unar, qayta yuklash, const, kiritish operatori, chiqarish operatori, qo‘shish operatori, ayirish operatori, ostream, class.

14.1. C++da operatorlarni qayta yuklash

Standart tiplar uchun o’zining vazifasiga qarab barcha amallar qayta yuklangan. Shuning uchun ular bilan e’lon qilingan identifikatorlar ushbu operatorlarni qo’llaydi. Masalan:

```
int a, b, c;
c = a + b; // bunday yozish mumkin
```

Agar sun’iy tip bo’lsa u holda undan olingan obyekt uchun operatorlarni qayta yuklash zarur. Masalan:

```
Myclass a, b, c;
c = a + b; // bu xolatda xatolik yuz beradi
```

Buning oldini olish uchun operatorlarni qayta yuklash zarur!

Operatorlarni qayta yuklash uchun **operator** kalit so’zidan foydalaniladi. Ushbu ***operator funksiyasi*** sinf ob’yektlari orasidagi qo’shimcha amallarni yuklaydi va sinf a’zolariga mos ishlaydi.

Demak operator funksiyasi sinf a’zosi bo’lmasa albatta bu funksiyaga mos friend (do’stona) sinf bo’lishi lozim.

Operator funksiyasi quyidagi forma bo’yicha yaratiladi:

```
qaytarish_tipi sinf_nomi::operator#(argumentlar)
{
// ifoda      }
class sampleClass {
Public:
qaytarish_turi operator symbol (arguments) { }
};
```

Bu yerda «operator» — kalit so’z, «symbol» — qayta yuklanadigan operator.

C++da operatorlarni qayta yuklanishida “operator” kalit so‘zidan foydalaniladi va undan so‘ng aniqlanayotgan operator belgisi keltiriladi.

Qayta yuklanadigan funksiyalar singari operatorlarni ham qayta yuklanish turi va parametrlari mavjud.

Box operatori+(const Box&);

Ikkita Box obyektini qo‘shish uchun ishlatilishi mumkin bo‘lgan qo‘shish operatorini e'lon qiladi va yakuniy Box obyektini qaytaradi. Qayta yuklangan operatorlarning aksariyati oddiy a'zo bo‘lmagan funktsiyalar yoki sinf a'zosi funktsiyalari sifatida aniqlanishi mumkin. Agar biz yuqoridagi funktsiyani sinfning a'zosi bo‘lmagan funktsiyasi sifatida aniqlasak, har bir operand uchun ikkita argumentni quyidagicha yozishimiz kerak bo‘ladi:

Box operator+(const Box&, const Box&);

Quyida a'zo funksiyadan foydalangan holda operatorni qayta yuklash strukturasiga misol keltirilgan.

14.1-misol.

```
#include <iostream>
using namespace std;
class sana{
public:
   int kun;
   int oy;
   int yil;
};
istream &operator >> (istream &kirit, sana &x)
{
   cout<<"kun: "; kirit>>x.kun;
   cout<<"oy: ";  kirit>>x.oy;
   cout<<"yil: "; kirit>>x.yil;
}
ostream &operator << (ostream &chiqar, sana &x)
{
   chiqar << x.kun <<"."<<x.oy << "."<<x.yil<< endl;
}
int main()
{
   sana a,b;
   //a obyekt
   cout<<"a: "<<endl;     cin >> a;
   cout << a;     //b obyekt
   cout<<"b: "<<endl;     cin >> b;
```

```
    cout << b;     return 0;
}
```

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.

14.2. Kiritish va chiqarish operatorlarini qayta yuklash

Point klass uchun “<<” operatorining qayta yuklanishini amalga oshirish ko‘rib chiqamiz. Biz shu darsimizgacha c++ da mavjud tip identifikatorlarini natijaga chiqarish organib keldik. Agar yaratilgan klass obyektini kiritsh yoki chiqarish kerak bo‘lsa bu ishni to‘g`ridan-to‘g`ri amalga oshirib bo‘lmaydi. Buning uchun dastlab kiritish va chiqarish operatorlarini qayta yuklash lozim.

14.2-misol. “<<” chiqarish operatorini qayta yuklash.

```
#include <iostream>
class Point
{
private:
    double m_x, m_y, m_z;
public:
    Point(double x=0.0, double y=0.0, double z=0.0): m_x(x),
m_y(y), m_z(z)
    {
    }
    friend std::ostream& operator<< (std::ostream &out, const Point
&point);
};
std::ostream& operator<< (std::ostream &out, const Point &point)
{
out << "Point(" << point.m_x << ", " << point.m_y << ", " <<
point.m_z << ")";
    return out;
}
```

```
int main()
{
   Point point1(3.0, 4.7, 5.0);
   Point point2(9.0, 10.5, 11.0);
   std::cout << point1 << " " << point2 << '\n';
   return 0;
}
```

Bundan tashqari kiritish operatorini ham qayta yuklash mumkin. Hammasi chiqish operatori bilan deyarli bir xil, lekin esda tutish kerak bo‘lgan asosiy narsa std::cin std::istream tipidagi obyektdir.

14.3-misol. “>>” kiritish operatorini qayta yuklash.

```
#include <iostream>
class Point
{
private:
   double m_x, m_y, m_z;
public:
   Point(double x=0.0, double y=0.0, double z=0.0): m_x(x),
m_y(y), m_z(z)
   {
   }
   friend std::ostream& operator<< (std::ostream &out, const Point
&point);
   friend std::istream& operator>> (std::istream &in, Point
&point);
};
std::ostream& operator<< (std::ostream &out, const Point &point)
{
out << "Point(" << point.m_x << ", " << point.m_y << ", " <<
point.m_z << ")";
   return out;
}
std::istream& operator>> (std::istream &in, Point &point)
{
    in >> point.m_x;
   in >> point.m_y;
   in >> point.m_z;
   return in;
```

```
}
int main(){
   std::cout << "Enter a point: \n";
   Point point;
   std::cin >> point;
   std::cout << "You entered: " << point << '\n';
   return 0;
}
```

14.3. Qo‘shish va ayirish operatorlarini qayta yuklash

Bitta operandda ishlaydigan unar plyus (+) va minus (-) operatorlarini ko‘rib chiqamiz. Ular faqat bitta obyektga tegishli bo‘lganligi sababli, ular sinf usullari qayta yuklanishi kerak.

14.4-misol. Dollars sinfi uchun unary minus (-) operatorini qayta yuklanishini ko‘ramiz:

```
#include <iostream>
class Dollars
{
private:
   int m_dollars;
public:
   Dollars(int dollars) { m_dollars = dollars; }
    Dollars operator-() const;
   int getDollars() const { return m_dollars; }
};
Dollars Dollars::operator-() const
{
   return Dollars(-m_dollars);
}
int main()
{
   const Dollars dollars1(7);
   std::cout << "My debt is " << (-dollars1).getDollars() << "
dollars.\n";
    return 0;
}
```

14.4. 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 ko‘rinishidagi 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 ko‘rinishida aniqlangan. Umuman olganda argument sifatida obyektni o‘zini ham chaqirish mumkin, lekin funksiyadan chiqishda bu obyekt destruktor yordamida yo‘qotiladi. Funksiya parametri sinf obyektga murojaat ko‘rinishida bo‘lishining afzalligi shundaki, funksiya chaqirilganda unga obyekt emas, balki obyektga ko‘rsatkich uzatiladi va sinf nusxasi uchun chaqiriladigan destruktorni ishlatilmaydi. Operator funksiyalarning qaytaruvchi qiymati ayni shu sinf tipida va hol obyektlarni nisbatan murakkab ifodalarda qo‘llash imkonini beradi. Masalan, quyidagi amallar dastur uchun ruxsat etilgan til ko‘rsatmasi hisoblanadi:

C=A+B-C;

Ikkinchi tomondan, quyidagi ifoda ham o‘rinli:

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

Bu ifodada qo‘shish operatorining operator funksiyasidagi vaqtincha (OraliqOb) obyektining Nuqta_Qiymati() funksiyasi ishlatiladi.

Nazorat savollari

1. Operatorlarni qayta yuklash qanday amalga oshiriladi?
2. Kiritish operatorini qayta yuklash qanday amalga oshiriladi?.
3. Chiqarish operatorini qayta yuklash qanday amalga oshiriladi?
4. Qo‘shish va ayirish operatorlarini qayta yuklash qanday amalga oshiriladi?
5. Operatorlarni qayta yuklash nima uchun zarur?
6. Binar operatorlarni qayta yuklash qanday amalga oshiriladi?

15. SHABLONLAR BILAN ISHLASH

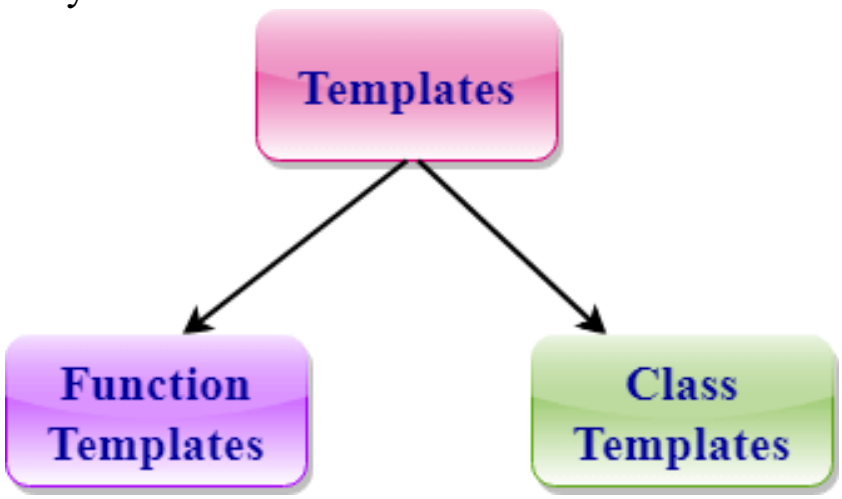
REJA:

1. Shablonlar va ularning qo‘llanilishi.
2. Funksiya shablonlari.
3. Klass shablonlari.

Kalit so‘zlar: template, class, function, typename, funksiya shablon, sinf shablon, overload, override, obyekt.

15.1. Shablonlar va ularning qo‘llanilishi

C++ tili qayta foydalaniluvchi dasturiy ta’minotni ishlab chiqish uchun shablon funksiyalar va sinflar bilan ta’minlaydi. Shablonlar funksiyalar va sinflarda turlarni muvofiqlashtirish (sozlash) qobiliyatini taqdim etadi. Bunday qobiliyat bilan, kompilyator aniq bir tur o‘rnida qabul qila oladigan umumiy turga sifatida bitta funksiya yoki bitta sinfni aniqlashimiz mumkin. Masalan, biz umumiy turdagi ikkita sondan kattasini topish uchun bitta funksiyani aniqlashimiz mumkin. Agar bu funksiyani ikkita **int** argumentlar orqali chaqirsak, umumiy tur **int** turi bilan almashadi. Agar bu funksiyani ikkita **double** argumentlar orqali chaqirsak, umumiy tur **double** turi bilan almashadi.



15.1-rasm. C++ shablon turlari

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.

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.

15.1-misol. Shablon bilan ishlash

```
#include <string>
using namespace std;
template <typename T>
T maxValue(T value1, 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
```

Funksiya shabloni **template** xizmatchi so’zi bilan aniqlangan.

15.2. Funksiya shablonlari

Funksiya shablonining aniqlanishi parametrlar ro‘yxati tomonidan berilgan **template** – kalit so‘zi bilan boshlanadi. Har bir parametr dastlab o‘zaro teng kuchli bo‘lgan **typename** yoki **class** kalit so‘zi orqali, **<typename typeParameter>** yoki **<class typeParameter>** ko‘rinishida beriladi. Masalan:

```
template <typename T>
T functionName(T parameter1, T parameter2, ...){
// code
}
```

Shablon funksiyalarda parameter tiplari va sonlarini ixtiyoriy ishlatish mumkin. Bu turdagi shablon funksiyalarni qo‘llash orqali dastur kodini bir muncha qisqartirish va ko‘p o‘xshash jarayonlarni bir funksiya orqali bajarish mukin.

15.2-misol. Ikkita turli yoki bir xil tipli o‘zgaruvchini shablon funksiya orqali qo‘shish.

```
#include <iostream>
using namespace std;
template <typename T,typename T1>
T1 sum(T i,T1 j){
   return i+j;
   }
int main()
{
  cout << sum("Salom","s") << endl;
  cout << sum(7.7,8.5)<< endl;
  cout << sum(7.5,8)<< endl;
   return 0;
}
```

15.3-misol. Ixtiyoriy o‘lchamli massiv elementlari yig‘indisini hisoblashni shablon funksiya orqali tashkil qilish.

```
#include <iostream>
using namespace std;
void printmassiv(int * massiv, int i){
   int s=0;
   for (int j=0; j<i; j++)
      s+=i;
   cout <<"s= "<< s << endl;
}
int main()
{
   int a[]={2,8,9,7,4}
   int b[]={1,3,5,7,9,11,13};
   printmassiv(a,5);
   printmassiv(b,7);
```

```
    return 0;
}
```

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:

```
#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.

15.3. Klass shablonlari

Oldingi qismda funksiya uchun mo‘ljallangan tur parametrga ega funksiya shabloni aniqlandi. Biz sinf uchun mo‘ljallangan tur parametrga ega sinfni ham aniqlashimiz mumkin. Tur parametrlar sinfning tur shakllantiriladigan ixtiyoriy qismida foydalanilishi mumkin.

Funksiya shablonlariga o‘xshab, biz turli xil ma'lumotlar turlari bilan ishlash uchun bitta sinf yaratish uchun sinf shablonlaridan foydalanishimiz mumkin. Sinf shablonlari foydali bo‘ladi, chunki ular bizning kodimizni qisqartirishi va boshqarilishi mumkin.

15.3-misol. Ikkita bir xil tipli sonlarni shablon sinf orqali solishtirish.

```
using namespace std;
template <typename T>
class katta{
T son1, son2;
public:
   katta(T val1,T val2){
   son1=val1;
   son2=val2; }
    T getson(){
   return (son1>son2?son1:son2);}
};
int main(){
   katta <int> obj1(50,89);
   cout<<obj1.getson()<<endl;
   katta <float> obj2(50.8,89.7);
   cout<<obj2.getson()<<endl;
   return 0;
}
```

15.4-misol. Ikkita bir xil tipli sonlar ustida 4 ta amalni bajarishni shablon klass orqali tashkil qilish.

```
#include <iostream>
using namespace std;
```

```
template <class T>
class calculate{
   T son1,son2;
public:
   calculate(T val1, T val2):son1(val1),son2(val2){}
   void getshow(){
   cout<< "Qo‘shish= " <<qushish()<<endl;
   cout<< "Ayirish= " <<ayirish()<<endl;
   cout<< "Ko‘paytirish= " <<kupaytirish()<<endl;
   cout<< "Bo‘lish= " <<bulish()<<endl;
   }
   T qushish(){
   return son1+son2;
   }
   T ayirish(){
   return son1-son2;
   }
   T kupaytirish(){
   return son1*son2;
   }
   T bulish(){
   return son1/son2;
   }
};
int main()
{
   calculate<int> obj1(8,9);
   obj1.getshow();
   calculate<float> obj2(8.8,9.7);
   obj2.getshow();
   return 0;
}
```

Funksiya shablonlari - turli xil tipdagi o’zgaruvchilar yoki parametrik o’zgaruvchilar bilan ishlashda qo’llaniladi.

Sinf shablonlari esa – turli xil obyektlar ustida amallar bajarish uchun ishlatiladi.

Obyektlar – o’zgaruvchi va o’zgarmaslarning dinamik strukturasidan tashkil topadi.

Nazorat savollari

1. Shablon funksiyalari nima uchun zarur?
2. Shablon klasslarni tushuntirib bering.
3. Shablon funksiyalarni qayta yuklash qanday amalga oshiriladi?
4. Shablon klasslarni qayta yuklash qanday imkoniyatlar beradi?
5. Shablon funksiya va klasslarning yutuqlari nimlardan iborat.

FAN DOIRASIDA ISHLAB CHIQILGAN TESTLAR

1. O‘zgaruvchi qanday kiritiladi?
a) var str = "Hi";
b) int num = "1";
c) float x = 32,14;
d) done = true;
e) char sym = 'a';

2. Izoh qanday yoziladi?
a) # mana izoh
b) /* mana izoh
c) /* mana izoh /*
d) / mana izoh
e) // mana izoh

3. Destruktorga qancha parametr o‘tkazilishi mumkin?
a) 3 dan ortiq emas
b) 10 dan ortiq emas
c) 15 dan ortiq emas
d) Maksimum 1
e) Parametrlarni destruktorga o‘tkazib bo‘lmaydi

4. Kod nima chiqaradi?
```
char s[] = "hello", t[] = "hello";
if(s == t)
   cout << "True";
```
a) True
b) Massivlarni tenglik uchun taqqoslab bo‘lmaydi
c) Kodda xato bo‘ladi
d) Hech narsa chiqmaydi, chunki ko‘rsatkichlar solishtirilmoqda

5. Funksiyaga nechta argument berish mumkin?
a) 10 gacha
b) До 30 gacha
c) 5 gacha
d) 50 gacha
e) cheksiz

6. Ushbu kod natijasini toping.

```
char *s = "Fine";
*s = 'N';
cout << s << endl;
```

a) Nine
b) Fine
c) xatolik

7. Ushbu kod natijasini toping.

```
char s[] = "C++";
cout << s << " ";
s++;
cout << s << " ";
```

a) C++ C++
b) C++
c) C++ ++
d) xatolik

8. Ushbu kod natijasini toping.

```
int const a = 5;
a++;
cout << a;
```

a) 5
b) 6
c) 0
d) xatolik

9. x nimaga teng?

```
int x = 2 + 1;
```

a) xatolik
b) 2
c) 1
d) 0
e) 3

10. iostream standart kutubxonasini qanday ulash kerak?

a) #include “iostream.h”
b) #include <iostream.h>
c) #include iostream

d)#include 'iostream.h'
e)#include <iostream>

11. C++ dasturlash tilida ma'limotlarni chiqarish uchun qaysi xizmatchi so‘zdan foydaniladi?
a) cout
b) cout
c) int
d) float

12. C++ dasturlash tilida ma'limotlarni kiritish uchun qaysi xizmatchi so‘zdan foydaniladi?
a) cin
b) cout
c) int
d) float

13. C++ dasturlash tilida haqiqiy tiplarni ko‘rsating ?
a) string, double
b) char, float
c) double, float
d) int, Boolean

14. C++ dasturlash tilida haqiqiy tiplarni ko‘rsating ?
a) string, double
b) char, float
c) double, float
d) int, short int

15. C++ dasturlash tilida belgili tipni ko‘rsating ?
a) string
b) char
c) double
d) int

16. C++ dasturlash tilida qatorli tipni ko‘rsating ?
a) string
b) char
c) double

d) int

17. C++ da ishlatiladigan shart operatorini ko’rsating.
a) if
b) while
c) for
d) break

18. C++ da ishlatiladigan sikl operatorini ko’rsating.
a) if, for
b) while, repeat
c) for, while
d) break

19. while va for operatorlarining farqi nimada?
a) farqi yo‘q
b) while shart orqali, for oraliq orqali takrorlanadi
c) for shart orqali, while oraliq orqali takrorlanadi
d) for qulay whilega nisbatan

20. while va do while farqi nimada?
a) while dastlab shart tekshiradi, do while oldin sikl keyin shart
b) do while dastlab shart tekshiradi, while oldin sikl keyin shart
c) ikkalasi bir xil
d) do while qulay

21. Inkapsulatsiya nima ?
a) Ochiqlik siyosati
b) Klasni tarkibida va tashqarida atributlarni ishlashi
c) Klassga tegishli atributlar klassni tarkibida ishlashi va tashqarida ishlamasligi
d) Klass tarkibidagi funksiyalarni ishlamasligi

22. Merosxo‘rlik nima ?
a) Ota klassni barcha xususiyatlarini olgan farzand klas yaratish
b) Klasslarni bog‘lnishi
c) Ota klass asosida xususiyatlari umuma farq qiladigan farzand klass yaratish
d) Klasslarni ichma-ich bog‘lanishi

23. Ko‘rsatkichlar to‘g‘ri e’lon qilingan qatorni toping
a) int *a
b) int a*
c) float &a
d) int a&

24. Dastur natijasini toping
```
int main(){
int a=5, b=8;
return(a>b?a:b);
}
```
a) 5
b) 8
c) 5 8
d) 8 5

25. Dastur natijasini toping
```
int main()
{
int a=68;
cout << (char)a;
return 0;
}
```
a) A
b) D
c) C
d) G

26. C++ da identifikator nima bilan boshlanishi kerak?
a) lotin hariflari bilan yoki past chiziqcha bilan
b) kril xarflari va raqamlar bilan
c) xohlagan simvol bilan
d) raqam bilan

27. Char tipi nechta simvolni o‘zida saqlaydi?
a) 256
b) 255
c) 257

d)254

28. **float** tipiga mos qiymatni belglang.
A)-5
b)2.3
c)5
d)0

29. Quyidagi dastur uchun k=1 va m= -12 bo‘la oladimi ?
(int k, float m)
a)HA
b)YO‘Q
c)m manfiy bo‘la olmidi
d)Bu o‘zgaruvchiga aloqasi yo‘q

30. Kompilyator bu-?
a) dastur tuzish uchun, ya’ni kodlarning qonun qoida bo‘yicha terilganligini nazorat qiluvchi va dasturning natijasini chiqaruvchi amaliy dasturdir
b) Kiritilgan matinni dasturga o‘girib beradigan dastur
c)dastur tuzish uchun dastur kodlarini to‘g’riligini tekshirib beruvchi dasturdir
d)kiritilgan kodni ekranga chiqzib beruvchi dastur

31. a=1.0; b=3; x=(a+b)/a*b-a; amalining natijasi nimaga teng?
a)2
b)1
c)0.3333
d)11

32. summa=sqr(x)+3*(-a) amalida qanday o‘zgaruvchilar ishtirok etgan?
a) summa, int x, float a
b)summa float x int a
c) summa short x float a
d)summa double a short x

33. Simvolli malumotlarni qabul qiluvchi tipni belgilang.
a)char

b)byte
c)word
d)switch

34. Tanlash operatorini belgilang.
a)switch
b)choose
c)getch
d)conio

35. || bu qanday amal.
a)dizyunksiya
b)konyunksiya
c)inversiya
d)belgi

36. && bu qanday amal?
a) VA amali
b) yoki amali
c)ko‘paytirish amali
d)qo‘shish amali

37. **void** ko‘rsatkichi qanday e’lon qilinadi?
a)void *<nom>
b)void <nom>*
c)*void<nom>
d)void<*nom>

38. Sonning ildizini chiqaruvchi amal qaysi?
a)sqrt(x)
b)sqr(x)
c)sqrt[x]
d)sqr[]

39. Mantiqiy amallar nimalardan tashkil topgan bo‘ladi?
a)1 va 0 dan
b) ixtiyori harflardan
c)faqat 10 va 01 dan
d)faqat sondan

40. float va int necha bayt joy band qiladi xotiradan?
a)4 va 2 bayt
b)6 bayt
c)2 va 4 bayt
d)8 byte

41. int tipining oralig'i…
a)-32768…32767
b)-32767….32767
c)-32768…32768
d)-32766…32767

42. if(a>b && a>c && a<d) bu qanday shart?
a) murakkab
b) chiziqli
c) ko‘paytma
d) oddiy

43. ofstream nima uchun ishlatiladi?
a) malumotlarni yozish uchun
b) malumotlarni o‘qish uchun
c) o‘qish va yozish uchun
d) kutubxona

44. fstream nima uchun ishlatilinadi?
a) fayldan malumotlarni yozish va o‘qish uchun
b) faqat o‘qish uchun
c) faqat yozish uchun
d) shifrlovchi

45. ifstream nima uchun ishlatiladi?
a) o‘qish uchun
b) yozish uchun
c) o‘qish va yozish uchun
d) shart belgilovchi

FOYDALANILGAN ADABIYOTLAR RO‘YXATI

1. Mo‘minov B.B., Dasturlash I. (Darslik). – T.: «Nihol print» OK, 2021. –280 b.
2. Nazirov Sh.A., Qobulov R.V., Bobojanov M.R., Raxmanov Q.S. C va C++ tili. “Voris- nashriyot” MCHJ, Toshkent 2013, 488 b.
3. Raxmanov Q.S., Kasimova Sh.T. “C++ tilida dasturlash” o‘quv qo‘llanma // TATU. Toshkent, 2017. 520 b.
4. Страуструп. Б. Язык программирования C++. Специальное издание.-M.:ООО «Бином-Пресс», 2006.-1104 с.
5. Вирт Н. Алгоритмы + структуры данных = программа. - M.:Мир, 1985.-405с.
6. estudy.iiau.uz – O‘zbekiston xalqaro islom akademiyasi masofaviy ta’lim portali.
7. https://robocontest.uz – Onlayn masalalar yechish sayti.
8. https://codeforces.com –dasturchilar ichun jahon portali.
9. acm.iiau.uz - O‘zbekiston xalqaro islom akademiyasi dasturiy yechim to‘g’riligini tekshiruvchi tizim.
10. https://acmp.ru –Дастурчилар мактаби портали.
11. https://www.onlinegdb.com/online_c++_compiler – C++ online kompilyator.

МУНДАРИЖА:

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

O‘ZBEKISTON XALQARO ISLOM AKADEMIYASI

Raxmanov Qurbon Sodikovich
Tuxtanazarov Dilmurod Solijonovich

“DASTURLASH I”
FANIDAN O‘QUV QO‘LLANMA

O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta’lim vazirligi huzuridagi Muvofiqlashtiruvchi kengashning 2022-yil 28-noyabrdagi 388-sonli buyrug’iga asosan 60610400 – “Axborot xavfsizligini boshqarish” bakalavriyat ta’lim yo‘nalishi talabalari uchun o‘quv qo‘llanma sifatida tavsiya etilgan

Taqrizchilar:

Mo‘minov B.B. - Muhammad Al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti “Axborot texnologiyalarining dasturiy ta’minoti” kafedrasi mudiri, t.f.d.

Jumayev T.S. - O‘zbekiston xalqaro islom akademiyasi “Zamonaviy axborot-kommunikatsiya texnologiyalari” kafedrasi katta o‘qituvchisi, PhD

Босишга руҳсат этилди: 12.05.2022 Офсет қоғоз.
қоғоз бичими 64x84 1/16 Тимес гарнитураси. Шартли босма табоғи 9,8. Нашр хисоб табоғи 8,6. Адади 100

«ЗАМОН ПОЛИГРАФ» ОК босмахонасида чоп этилди.
Манзил: Ташкент шаҳри. Юнусобод тумани,
Бободеҳқон маҳалласи 45 уй.