

13-Ma'ruza. Funksiyalarda argument sifatida lokal, global o'zgaruvchilardan va havolalardan foydalanish.

Ma'ruza rejasi:

13.1 Lokal o'zgaruvchilar

13.2 Global o'zgaruvchilar

13.3 Havolalar tushunchasi

Kalit so'zlar:, *ro'yxat, manzil, nolinchi ko'rchsatkich, tugun, adres olish &, bo'shatish, ko'rsatkich, virtual destruktor, xotira, xotira chiqishi, destruktor, toifani o'zlashtirish, resurslar chiqishi, a'zo destruktori.*

Ko'rinish sohasi. Lokal va global o'zgaruvchilar

O'zgaruvchilar funksiya tanasida yoki undan tashqarida e'lon qilinishi mumkin. Funksiya ichida e'lon qilingan o'zgaruvchilarga *lokal o'zgaruvchilar* deyiladi. Bunday o'zgaruvchilar xotiradagi prog-ramma stekida joylashadi va faqat o'zi e'lon qilingan funksiya tanasida amal qiladi. Boshqaruv asosiy funksiyaga qaytishi bilan lokal o'zgaruvchilar uchun ajratilgan xotira bo'shatiladi (o'chiriladi).

Har bir o'zgaruvchi o'zining amal qilish sohasi va yashash vaqti xususiyatlari bilan xarakterlanadi.

O'zgaruvchi *amal qilish sohasi* deganda o'zgaruvchini ishlatish mumkin bo'lgan programma sohasi (qismi) tushuniladi. Bu tushuncha bilan o'zgaruvchining *ko'rinish sohasi* uzviy bog'langan. O'zgaruvchi amal qilish sohasidan chiqqanda ko'rinmay qoladi. Ikkinchi tomondan, o'zgaruvchi amal qilish sohasida bo'lishi, lekin ko'rinmas-ligi mumkin. Bunda ko'rinish sohasiga ruxsat berish amali «::» yordamida ko'rinmas o'zgaruvchiga murojat qilish mumkin bo'ladi.

O'zgaruvchining *yashash vaqti* deb, u mavjud bo'lgan programma bo'lagining bajarilishiga ketgan vaqt intervaliga aytiladi.

Lokal o'zgaruvchilar o'zlari e'lon qilingan funksiya yoki blok chegarasida ko'rinish sohasiga ega. Blokdagi ichki bloklarda xuddi shu nomdagi o'zgaruvchi e'lon qilingan bo'lsa, ichki bloklarda bu lokal o'zgaruvchi ham amal qilmay qoladi. Lokal o'zgaruvchi yashash vaqti - blok yoki funksiyani bajarish vaqti bilan aniqlanadi. Bu hol shuni anglatadiki, turli funksiyalarda bir-biriga umuman bog'liq bo'lma-gan bir xil nomdagi lokal o'zgaruvchilarni ishlatish mumkin.

Quyidagi programmada main() va sum() funksiyalarida bir xil nomdagi o'zgaruvchilarni ishlatish ko'rsatilgan. Programmada ikkita sonning yig'indisi hisoblanadi va chop etiladi:

```

#include <iostream.h>
// funksiya prototipi
int sum(int a;int b);
int main()
{
    // lokal o'zgaruvchilar
    int x=r;
    int y=4;
    cout<<sum(x, y);
    return 0;
}
int sum(int a,int b)
{
    // lokal o'zgaruvchi
    int x=a+b;
    return x;
}

```

Global o'zgaruvchilar programma matnida funksiya aniqlanishi-dan tashqarida e'lon qilinadi va e'lon qilingan joyidan boshlab programma oxirigacha amal qiladi.

```

#include <iostream.h>
int f1(); int f2();
int main()
{
    cout<<f1()<<" "<<f2()<<endl;
    return 0;
}
int f1()
{
    return x;// kompilyasiya xatosi ro'y
    beradi
}
int x=10; // global o'zgaruvchi e'loni
int f2(){ return x*x;}

```

YUqorida keltirilgan programmada kompilyasiya xatosi ro'y beradi, chunki f1() funksiya uchun x o'zgaruvchisi noma'lum hisob-lanadi.

Programma matnida global o'zgaruvchilarni ular e'lonidan keyin yozilgan ixtiyoriy funksiya ishlatish mumkin. SHu sababli, global o'zgaruvchilar programma matnining boshida yoziladi. Funksiya ichidan global o'zgaruvchiga murojat qilish uchun funksiya uning nomi bilan mos tushadigan lokal o'zgaruvchilar bo'lmasligi kerak. Agar global o'zgaruvchi e'lonida unga boshlang'ich qiymat berilmagan bo'lsa, ularning qiymati 0 hisoblanadi. Global o'zgaruvchining amal qilish sohasi uning ko'rinish sohasi bilan ustma-ust tushadi.

SHuni qayd etish kerakki, tajribali programma tuzuvchilar imkon qadar global o'zgaruvchilarni ishlatmaslikka harakat qilishadi, chunki bunday o'zgaruvchilar qiymatini programmaning ixtiyoriy joyidan o'zgartirish xavfi mavjudligi sababli programma ishlashida mazmunan xatolar yuzaga kelishi mumkin. Bu fikrimizni tasdiqlovchi programmani ko'raylik.

```

#include <iostream.h>
// global o'zgaruvchi e'loni
int test=100;
void Chop_qilish(void );
int main()
{
    //lokal o'zgaruvchi e'loni
    int test=10;
    //global o'zgaruvchi chop qilish
    funksiya chaqirish
    Chop_qilish();
}

```

sout<<"Lokal	{
o'zgaruvchi: "<<test<<"\n';	cout<<"Global
return 0;	o'zgaruvchi: "<<test<<"\n';
}	}

void Chop_qilish(void)

Programma boshida test global o'zgaruvchisi 100 qiymati bilan e'lon qilinadi. Keyinchalik, main() funksiyasida test nomi bilan lokal o'zgaruvchisi 10 qiymati bilan e'lon qilinadi. Programmada, Chop_qilish() funksiyasiga murojaat qilinganida, asosiy funksiya tanasidan vaqtincha chiqiladi va natijada main() funksiyasida e'lon qilingan barcha lokal o'zgaruvchilarga murojaat qilish mumkin bo'lmay qoladi. SHu sababli Chop_qilish() funksiyasida global test o'zgaruvchisining qiymatini chop etiladi. Asosiy programmaga qaytilgandan keyin, main() funksiyasidagi lokal test o'zgaruvchisi global test o'zgaruvchisini «berkitadi» va lokal test o'zgaruvchini qiymati chop etiladi. Programma ishlashi natijasida ekranga quyidagi natijalar chop etiladi:

Global o'zgaruvchi: 100

Lokal o'zgaruvchi: 10

:: amali

YUqorida qayd qilingandek, lokal o'zgaruvchi e'loni xuddi shu nomdagi global o'zgaruvchini «berkitadi» va bu joydan global o'zgaruvchiga murojat qilish imkoni bo'lmay qoladi. S++ tilida bunday holatlarda ham global o'zgaruvchiga murojat qilish imko-niyati saqlanib qolingan. Buning uchun «ko'rinish sohasiga ruxsat berish» amalidan foydalanish mumkin va o'zgaruvchi oldiga ikkita nuqta - «::» qo'yish zarur bo'ladi. Misol tariqasida quyidagi programani keltiramiz:

#include <iostream.h >	//lokal o'zgaruvchini chop etish
//global o'zgaruvchi e'loni	cout<<uzg<<"\n';
int uzg=5;	//global o'zgaruvchini chop etish
int main()	cout<<::uzg <<"\n';
{	return 0;
//lokal o'zgaruvchi e'loni	}
int uzg=70;	

Programma ishlashi natijasida ekranga oldin 70 va keyin 5 sonlari chop etiladi.

Xotira sinflari

O'zgaruvchilarning ko'rinish sohasi va amal qilish vaqtini aniqlovchi o'zgaruvchi modifikatorlari mavjud (5.1-jadval).

5.1-jadval. O'zgaruvchi modifikatorlari

Modifikator	Qo'llanishi	Amal qilish sohasi	Yashash davri
auto	lokal	blok	vaqtincha
register	lokal	blok	vaqtincha
extern	global	blok	vaqtincha
static	lokal	blok	doimiy
	global	fayl	doimiy
volatile	global	fayl	doimiy

Avtomat o'zgaruvchilar. auto modifikatori lokal o'zgaruvchilar e'lonida ishlatiladi. Odatda lokal o'zgaruvchilar e'lonida bu modifikator kelishuv bo'yicha qo'llaniladi va shu sababli amalda uni yozishmaydi:

```
#include <iostream.h>

int main()
{
    auto int X=2; // int X=2; bilan ekvivalent
    cout<<X;
    return 0;
}
```

auto modifikatori blok ichida e'lon qilingan lokal o'zgaruvchi-larga qo'llaniladi. Bu o'zgaruvchilar blokdan chiqishi bilan avtomatik ravishda yo'q bo'lib ketadi.

Registr o'zgaruvchilar. register modifikatori kompilyatorga, ko'rsatilgan o'zgaruvchini protsessor registrlariga joylashtirishga harakat qilishni tayinlaydi. Agar bu harakat natija bermasa o'zgaruvchi auto turidagi lokal o'zgaruvchi sifatida amal qiladi.

O'zgaruvchilarni registrlarda joylashtirish programma kodini bajarish tezligi bo'yicha optimallashtiradi, chunki protsessor xotiradagi berilganlarga nisbatan registrdagi qiymatlar bilan ancha tez ishlaydi. Lekin registrlar soni cheklanganligi uchun har doim ham o'zgaruvchilarni registrlarda joylashtirishning iloji bo'lmaydi.

```
#include < iostream.h >

int main()
{
    register int Reg;
    ...
    return 0;
}
```

register modifikatori faqat lokal o'zgaruvchilariga nisbatan qo'llaniladi, global o'zgaruvchilarga qo'llash kompilyasiya xatosiga olib keladi.

Tashqi o'zgaruvchilar. Agar programma bir nechta moduldan iborat bo'lsa, ular qandaydir o'zgaruvchi orqali o'zaro qiymat alma-shishlari mumkin (fayllar orasida). Buning uchun o'zgaruvchi birorta modulda global tarzda e'lon qilinadi va u boshqa faylda (modulda) ko'rinishi uchun u erda extern modifikatori bilan e'lon qilinishi kerak bo'ladi. extern modifikatori o'zgaruvchini boshqa faylda e'lon qilinganligini bildiradi. Tashqi o'zgaruvchilar ishlatilgan prog-rammani ko'raylik.

//Sarlavha.h faylida	extern bool Bayroq;
void Bayroq_Almashsin(void);	int main()
// modul_1.cpp faylida	{
bool Bayroq;	Bayroq_Almashsin();
void	if(Bayroq)
Bayroq_Almashsin(void){Bayroq=!Bayroq;}	cout<<"Bayroq TRUE"<<endl;
// masala.cpp faylida	else cout<<"Bayroq FALSE"<<endl;
#include <iostream.h>	return 0;
#include <Sarlavha.h>	}
#include <modul_1.cpp>	

Oldin sarlavha.h faylida Bayroq_Almashsin() funksiya sarlavhasi e'lon qilinadi, keyin modul_1.srr faylida tashqi o'zgaruvchi e'lon qilinadi va Bayroq_Almashsin() funksiyasining tanasi aniqlanadi va nihoyat, masala.cpp faylida Bayroq o'zgaruvchisi tashqi deb e'lon qilinadi.

Statik o'zgaruvchilar. Statik o'zgaruvchilar static modifikatori bilan e'lon qilinadi va o'z xususiyatiga ko'ra global o'zgaruvchi-larga o'xshaydi. Agar bu turdagi o'zgaruvchi global bo'lsa, uning amal qilish sohasi - e'lon qilingan joydan programma matnining oxirigacha bo'ladi. Agar statik o'zgaruvchi funksiya yoki blok ichida e'lon qilinadigan bo'lsa, u funksiya yoki blokka birinchi kirishda initsializatsiya qilinadi. O'zgaruvchining bu qiymati funksiya keyingi chaqirilganida yoki blokka qayta kirishda saqlanib qoladi va bu qiymatni o'zgartirish mumkin. Statik o'zgaruvchilarni tashqi deb e'lon qilib bo'lmaydi.

Agar statik o'zgaruvchi initsializatsiya qilinmagan bo'lsa, uning birinchi murojatdagi qiymati 0 hisoblanadi.

Misol tariqasida birorta funktsiyani necha marotaba chaqirilganligini aniqlash masalasini ko'raylik:

#include <iostream.h >	{
int Sanagich(void);	int natija;
int main()	for (int i=0; i<30; i++)

natija=Sanagich();	static short sanagich=0;
cout<<natija;	...
return 0;	sanagich++;
}	return sanagich;
int Sanagich(void)	}

Bu erda asosiy funksiyadan counter statik o'zgaruvchiga ega Sanagicht() funksiyasi 30 marta chaqiriladi. Funksiya birinchi marta chaqirilganda sanagich o'zgaruvchiga 0 qiymatini qabul qiladi va uning qiymati birga ortgan holda funksiya qiymati sifatida qaytariladi. Statik o'zgaruvchilar qiymatlarini funksiyani bir chaqirilishidan ikkinchisiga saqlanib qolinishi sababli, keyingi har bir chaqirishlarda sanagich qiymati bittaga ortib boradi.

Masala. Berilgan ishorasiz butun sonning barcha tub bo'luv-chilari aniqlansin. Masalani echish algoritmi quyidagi takrorla-nuvchi jarayondan iborat bo'ladi: berilgan son tub songa (1-qadamda 2 ga) bo'linadi. Agar qoldiq 0 bo'lsa, tub son chop qilinadi va bo'linuv-chi sifatida bo'linma olinadi, aks holda navbatdagi tub son olinadi. Takrorlash navbatdagi tub son bo'linuvchiga teng bo'lguncha davom etadi.

Programma matni:

#include<iostream.h>	else p=Navb_tub();
#include<math.h>	}
int Navb_tub();	return 0;
int main()	}
{	int Navb_tub()
unsigned int n,p;	{
cout<<"\nn qiymatini kiritng: ";	static unsigned int tub=1;
cin>>n;	for(;;)
cout<<"\n1";	{
p=Navb_tub();	tub++;
while(n>=p)	short int ha_tub=1;
{	for(int i=2;i<=tub/2;i++)
if(n%p==0)	if(tub%i==0)ha_tub=0;
{	if(ha_tub)return tub;
cout<<"*"<<p;	}
n=n/p;	return 0;
}	}

Programmada navbatdagi tub sonni hosil qilish funktsiya ko‘ri-nishida amalga oshirilgan. Navb_tub() funksiyasining har chaqirili-shida oxirgi tub son dan keyingi tub son topiladi. Oxirgi tub sonni «eslab» qolish uchun tub o‘zgaruvchisi static qilib aniqlangan.

Programma ishga tushganda klaviaturadan n o‘zgaruvchisining qiymati sifatida 60 soni kiritilsa, ekranga quyidagi ko‘paytma chop etiladi:

1*2*2*3*5

volatile sinfi o‘zgaruvchilari. Agar programmada o‘zgaruvchini birorta tashqi qurilma yoki boshqa programma bilan bog‘lash uchun ishlatish zarur bo‘ladigan bo‘lsa, u volatile modifikatori bilan e‘lon qilinadi. Kompilyator bunday modifikatorli o‘zgaruvchini registrga joylashtirishga harakat qilmaydi. Bunday o‘zgaruvchilar e‘loniga misol quyida keltirilgan:

volatile short port_1;

volatile const int Adress=0x00A2;

Misoldan ko‘rinib turibdiki, volatile modifikatorli o‘zgarmas ham e‘lon qilinishi mumkin.

onsider this sample program: **1** int x;

2 int mystery(int x)

3 {

4 int s = 0;

5 for (int i = 0; i < x; i++)

6 {

7 int x = i + 1;

8 s = s + x;

9 }

10 return x;

11 }

12 int main()

13 {

14 x = 4;

15 int s = mystery(x);

16 cout << s << endl;

17 }

28. Which line defines a global variable?

29. Which lines define local variables named x?

30. Which lines are in the scope of the definition of x in line 2?

31. Which variable is changed by the assignment in line 14?
32. This program defines two variables with the same name whose scopes don't overlap. What are they?

The diagram illustrates the state of variables during a function call. It consists of four panels:

- 1 Function call:** The variable `harrys_account` is set to 1000. The function `withdraw(harrys_account, 100);` is called. The variables `balance` and `amount` are shown as empty input fields.
- 2 Initializing function parameter variables:** The variable `harrys_account` remains 1000. The parameter `balance` is initialized to 1000, and `amount` is initialized to 100.
- 3 About to return to the caller:** The function body executes `balance = balance - amount;`, changing `balance` to 900. `harrys_account` remains 1000, and `amount` remains 100.
- 4 After function call:** The function returns, and the state is the same as in panel 1, with `harrys_account = 1000` and `balance` and `amount` as empty input fields.

ch05/account.cpp

```
1 #include <iostream>
2
3 using namespace std;
4 5 /**
5
6 Withdraws the amount from the
given balance, or withdraws
7 a penalty if the balance is
insufficient.
```

```
8 @param balance the balance from
which to make the withdrawal 9 @param
amount the amount to withdraw
10 */
11 void withdraw(double& balance,
double amount)
12 {
13 const double PENALTY = 10;
14 if (balance >= amount)
15 {
16 balance = balance - amount;
```



```

17 }
18 else
19 {
20     balance = balance - PENALTY;
21 }
22 }
23
24 int main()
25 {
26     double harrys_account = 1000;
27     double sallys_account = 500;
28     withdraw(harrys_account, 100);
29     // Now harrys_account is 900
30     withdraw(harrys_account, 1000);
// Insufficient funds
31     // Now harrys_account is 890
32     withdraw(sallys_account, 150);
33     cout << "Harry's account: " <<
harrys_account << endl; 34 cout << "Sally's
account: " << sallys_account << endl; 35
36     return 0;
37 }

```

program run

```

Harry's    account:    890    Sally's
account: 350

```

ch05/intname.cpp

```

1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 /**
7  Turns a digit into its English name.

```

```

8  @param digit an integer between 1
and 9
9  @return the name of digit ("one" ...
"nine")
10 */
11 string digit_name(int digit)
12 {
13     if (digit == 1) return "one"; 14 if
(digit == 2) return "two"; 15 if (digit == 3)
return "three";
16 if (digit == 4) return "four"; 17 if
(digit == 5) return "five"; 18 if (digit == 6)
return "six";
19 if (digit == 7) return "seven"; 20
if (digit == 8) return "eight"; 21 if (digit ==
9) return "nine";
22 return "";
23 }
24
25 /**
26  Turns a number between 10 and
19 into its English name.
27  @param number an integer
between 10 and 19
28  @return the name of the given
number ("ten" ... "nineteen")
29 */
30 string teen_name(int number)
31 {
32     if (number == 10) return "ten";
33     if (number == 11) return "eleven";
34     if (number == 12) return "twelve";
35     if (number == 13) return
"thirteen";

```

```

36 if (number == 14) return
"fourteen";
37 if (number == 15) return "fifteen";
38 if (number == 16) return "sixteen";
39 if (number == 17) return
"seventeen";
40 if (number == 18) return
"eighteen";
41 if (number == 19) return
"nineteen";
42 return "";
43 }
44
45 /**
46 Gives the name of the tens part of
a number between 20 and 99.
47 @param number an integer
between 20 and 99
48 @return the name of the tens part
of the number ("twenty" ... "ninety") 49 */
50 string tens_name(int number)
51 {
52 if (number >= 90) return "ninety";
53 if (number >= 80) return "eighty";
54 if (number >= 70) return
"seventy";
55 if (number >= 60) return "sixty";
56 if (number >= 50) return "fifty";
57 if (number >= 40) return "forty";
58 if (number >= 30) return "thirty";
59 if (number >= 20) return "twenty";
60 return "";
61 }
62
63 /**

```

```

64 Turns a number into its English
name.
65 @param number a positive integer
< 1,000
66 @return the name of the number
(e.g. "two hundred seventy four")
67 */
68 string int_name(int number)
69 {
70 int part = number; // The part that
still needs to be converted
71 string name; // The return value
72
73 if (part >= 100)
74 {
75 name = digit_name(part / 100) + "
hundred"; 76 part = part % 100;
77 }
78
79 if (part >= 20)
80 {
81 name = name + " " + tens_name(part);
82 part = part % 10;
83 }
84 else if (part >= 10)
85 {
86 name = name + " " + teen_name(part);
87 part = 0;
88 }
89
90 if (part > 0)
91 {
92 name = name + " " +
digit_name(part);
93 }

```

```

94
95 return name;
96 }
97
98 int main()
99 {
100 cout << "Please enter a positive
integer: "; 101 int input;

```

```

102 cin >> input;
103 cout << int_name(input) << endl;
104 return 0;
105 }
program run
Please enter a positive integer: 729
seven hundred twenty nine

```

Nazorat savollari

10. C++da funksiya qanday ishlaydi?
11. funksiya kutubxona kerakmi?
12. For operatori funksiyada qanday ishlatiladi?
13. Matematik funksiyalar qanday ishlaydi?
14. Funksiya parametrlar nima?
15. Funksiya qanday chaqiriladi?
16. Funksiya parametrlari orqali nima uzatiladi?
17. If operatorining nechta turi bor?
18. O'zgaruvchilar nima uchun qo'llaniladi?