Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Функции

Структурно програмирање

ФИНКИ 2024

Функции во C++

- Програмите претставуваат комбинација од кориснички и библиотечни функции
  - стандардните библиотеки содржат голем број различни функции
  - функциите го прават програмирањето поедноставно
  - секоја функција извршува точно дефинирана работа, што е корисна за останатите делови на програмата или за други програми
    - остатокот од програмата и не мора „да знае” како се извршува задачата

Повиканата функција ја извршува својата задача и потоа повторно извршувањето на програмата продолжува од наредбата по функцискиот повик

Предности при користење на функции

- поедноставно одржување на програмата
- можност за користење претходно развиени функции
  - се користат постоечки функции како градбени блокови за новите програми
- се одбегнува повторување на еден ист код на различни места во програмата
- апстракција – се кријат внатрешните детали
  - користењето на функциите е слично со шефот што задава задача на вработените
    - секој работник ја добива информацијата, ја извршува задачата и ги враќа резултатите
    - криење на информации: шефот не мора да ги знае деталите како е извршена работата, него го интересираат резултатите

Функциски повик

- Користењето на функциите во програмите се нарекува функциски повик
  - при повикување на функција се обезбедуваат име и аргументи за функцијата (податоци)
- Формат за повикување на функциите

Име (листа на аргументи)

```
cout<<sqrt(900);
```

Во наредбата се **повикува** функцијата sqrt, која ја пресметува и враќа вредноста на квадратен корен од **аргументот**

- Ако има повеќе аргументи – тие се одделуваат со запирки

```
cout<<pow(2,3);
```

Аргументи на функција

- Аргументите на функциите можат да бидат :
  - константи
  - променливи
  - цели изрази

- Ако **c1=13, d=3.0, f=4.0,**

  тогаш наредбата

```
cout<<(sqrt(c1+d*f));
```

  ќе го пресмета и испечати квадратниот корен на

  13.0+3.0*4.0=25, т.е **5**

Често користени математички функции – cmath библиотека

`sqrt(x)`	квадратен корен од x
`exp(x)`	експоненцијална функција e^x
`log(x)`	природен логаритам од x (со основа e)
`log10(x)`	логаритам од x со основа 10
`fabs(x)`	апсолутна вредност од x
`ceil(x)`	го заокружува x на најмалиот цел број не помал од x
`floor(x)`	го заокружува x на најголемиот цел број не поголем од x
`pow(x, y)`	x на степен y
`fmod(x, y)`	остаток од x/y како реален број
`sin(x)`	синус од x (во радијани)
`cos(x)`	косинус од x (во радијани)
`tan(x)`	тангенс од x (во радијани)

Пример

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{

   cout<<fabs(-500)<<" ";
   cout<<log10(100)<<" "<<sqrt(900)<<" "<<pow(2,3)<<endl;
   return 0;
}
```

Излез

500 2 30 8

Градење на функции

- Програмерот може да дефинира и готови функции
  - Кориснички дефинирани функции

- Три аспекти се важни кога се работи за функциите:
  - како се декларираат и дефинираат
  - како се повикуваат
  - како се комуницира помеѓу функциите

Дефинирање на функции

- Формат:

```
tipVratenaVrednost imeFunkcija(tip Pr1, tip Pr2, ...)
{
        /* telo na funkcijata */
}
```

- imeFunkcija – име на програмски елемент

Дефинирање на функции

- tipVratenaVrednost – тип на вредност што функцијата ја враќа како резултат
  - може да биде:
    - стандарден податочен тип (int, float, double итн.)
    - сложен тип
    - кориснички дефиниран тип (набројувачки, ...)
    - void - функцијата не враќа вредност

Дефинирање на функции

- tip Pr1, tip Pr2, ... - листа на параметри (формални параметри, формални аргументи) одделени меѓусебе со запирки
  - во оваа листа се декларираат променливи
  - функцијата може и да не содржи формални параметри (листата е празна)
  - променливите декларирани во листата на параметри важат и може да се употребат само во телото на функцијата
  - теориски, бројот на формални параметри не е ограничен, но во практиката е ограничен

Препорака: одбегнувајте го користењето на голем број формални параметри!

Дефинирање на функции

- Тело на функцијата
  - ги содржи истите елементи како и main() функцијата
    - декларација на променливи
      - Сите променливи дефинирани во рамките на дефинициите на функциите се локални променливи
      - Тие се познати (видливи) само во функцијата во која се дефинирани
    - наредби

Пример: Функцијата `Ploshtina` е од типот `int`

```
int Ploshtina (int l, int w)
{
     return l * w;
}


int main()
{
     int a=7, b=5;
     cout<<Ploshtina(a,b);
     return 0;
}
```

Излез: 35

Пример: Функцијата `kvadrati` е од типот `void`

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

void kvadrati()
{
  int i;
  for(i=1; i<10; i++)   cout<<i*i<<" ";
}

int main()
{
  kvadrati();
  return 0;
}
```

Излез: 1 4 9 16 25 36 49 64 81

Пример

Потсетување

n! = 1*2*3*...*n

3!=1*2*3
5!=1*2*3*4*5

```
/* Funkcija za presmetuvanje faktoriel */

void calc_factoriel( int n ) {
   int i, fact_num = 1;
   for( i = 1; i <= n; ++i ) fact_num *= i;
   cout<<"Faktoriel od "<<n<<" iznesuva "<<fact_num<<endl;
}
```

Пример

```
/* Programa za presmetuvanje faktoriel */
#include <iostream>
using namespace std;

void calc_factoriel( int n ) {
    int i, fact_num = 1;
    for( i = 1; i <= n; ++i ) fact_num *= i;
    cout<<"Faktoriel od "<<n<<" iznesuva "<<fact_num<<endl;
}

int main()
{
     int broj;
     cout<<"Vnesi broj za koj da presmetam faktoriel:";
     cin>>broj;
     calc_factoriel(broj);
     return(0);
}
```

```
Vnesi broj za koj da presmetam faktoriel:3
Faktoriel od 3 iznesuva 6
```

Напуштање на функцијата

- Ако во заглавјето на функцијата е дефинирано дека истата не враќа вредност, тогаш од функцијата се излегува кога ќе се стигне до изразот

  `return;`

  или до

  `}`

- Пример:

```
void print_answer(int answer) {
            if (answer < 0) {
                  cout<<"Answer corrupt\n";
                  return;
            }
            cout<<"The answer is " << answer;
      }
```

Напуштање на функцијата

- Ако во заглавјето на функцијата е дефинирано дека истата враќа вредност, тогаш од функцијата се излегува кога ќе се стигне до изразот

return *izraz;*

Типот на оваа вредност мора да соодветствува на типот на вредноста што е декларирано дека ќе ја врати функцијата

```
float add_numbers( float n1, float n2 ) {
      return n1+n2;
      /* tocno, zbirot e od tipot float */
}
```

Вредност што ја враќа функцијата

```
int add_numbers( float n1, float n2 ) {
      return 6.0;    /* ne e vo red */
}
```

```
float add_numbers( float n1, float n2 ) {
      return 6.0; /* tocno,
                     vraka realna vrednost */
}
```

Вредност што ја враќа функцијата

- Можно е една функција да има повеќе наредби за враќање на вредност:

```
int validate_input( char command ) {
    switch( command ) {
        case '+' : case '-' : return 1;
        case '*' : case '/' : return 2;
        default  : return 0;
    }
}
```

Дефинирање и користење на функции

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

float triangle(float width, float height) {
   float area;
   area = width * height / 2.0;
   return (area);
}
int main() {
   float size;
   cout<<"Triangle #1 "<< triangle(1.3, 8.3);
   cout<<"Triangle #2 "<< triangle(4.8, 9.8);
   cout<<"Triangle #3 "<< triangle(1.2, 2.0);
   return (0);
}
```

Изразот `triangle(1.3, 8.3)` е повик кон функцијата. На променливата **`width`** ќе и биде придружена вредност 1.3, а на **`height`** 8.3. Функцијата враќа вредност 5.4.

Пример за погрешно користење на функции

```
int square (float x) {
    float y;
    y = x * x;
    return (y);
}
int main ( ) {
    float a, b;
    cout<<" Vnesi broj ";
    cin>>a;
    b = square (a);
    cout<<"Kvadratot na "<<a<<" e "<< b;
    return 0;
}
```

функцијата враќа целобројна вредност

Vnesi broj 2.5
Kvadratot na 2.5 e 6

Што ќе се случи сега?

```
int main ( ) {
    float a, b;
    cout<<" Vnesi broj ";
    cin>>a;
    b = square (a);
    cout<<"Kvadratot na "<<a<<" e "<< b;
    return 0;
}

float square (float x) {
    float y;
    y = x * x;
    return (y);
}
```

Compilation failed due to following error(s).

```
main.cpp: In function ‘int main()’:
main.cpp:17:8: error: ‘square’ was not declared in this scope
   17 |     b = square (a);
      |         ^~~~~~
```

Декларирање функција - прототип

- Ако функција се користи пред да биде дефинирана, потребно е да се декларира на идентичен начин како и секоја променлива

- Прототипот на функција му кажува на компајлерот:
  - кој ќе биде типот што го враќа функцијата
  - кое е името на функцијата
  - која е листата на параметри на функцијата

Декларирање функција - прототип

- Формат:

tip imeFunkcija(listaParametri);

- *imeFunkcija* е името на функцијата
- *listaParametri* – податоци што се проследуваат во функцијата
- *tip* – податочен тип на вредноста што ја враќа функцијата

Прототипот завршува со ; и со тоа се означува дека станува збор за декларација на функција, а не нејзина дефиниција.

Имињата на параметрите на функцијата не мораат да бидат наведени во прототипот, туку се задава само нивниот тип.

Примери за функциски прототипови

`long Plostina(long dolzina,long sirina);`	Враќа long, има два параметра
`void Pecati(int brojPoraka);`	Враќа void, има еден параметар
`int GetChoice();`	Враќа int, нема параметри
`int BadFunction();`	Враќа int, нема параметри

Функциски прототипови - примери

```
float triangle (float width, float height);
```
- се проследуваат две реални вредности, враќа реална вредност

```
int prim(void);
```
- не се проследуваат вредности, враќа целобројна вредност

```
void prim1(void);
```
- не се проследуваат вредности, не враќа вредност

```
void prim(int, float);
```
- се проследуваат една целобројна и една реална вредност, не враќа вредност

```
float triangle(float, float);
```
- се проследуваат две реални вредности, враќа реална вредност

```
int maximum( int, int, int );
```
- се проследуваат три целобројни вредности, враќа целобројна вредност

Функциски прототипови - пример

```
int main ( ) {
    float square(float); //prototip
    float a, b;
    cout<<" Vnesi broj ";
    cin>>a;
    b = square (a);
    cout<<"Kvadratot na "<<a<<" e "<< b;
    return 0;
}

float square (float x) {
    float y;
    y = x * x;
    return (y);
}
```

```
Vnesi broj 2.5
kvadratot na 2.5 e 6.25
```

Функциски прототипови - пример

```
void print_message( void );

int main() {
 print_message();
 return 0;
}
void print_message( void ) {
 cout<<"Ova e funkcija narecena
print_message"<<endl;
}
```

```
Ova e funkcija narecena print_message
```

Да ја разгледаме функцијата maximum

```
int maximum( int x, int y, int z )
{
   int max = x;

   if ( y > max )
      max = y;

   if ( z > max )
      max = z;

   return max;
}
```

x, y и z се параметрите

Претпостави дека x е најголемиот

ако y е поголем од max , додели го y на max

ако z е поголем од max , додели го z на max

max е најголемата вредност

Функциски прототип - пример

```
/* Primer: Opredeluvanje na najgolemiot od tri broja */
int maximum( int, int, int ); /* funkciski prototip */

int main(){
    int a, b, c, m;
    cout<<"Vnesi tri celi broja ";
    cin>>a>>b>>c;
    m = maximum( a, b, c );
    cout<<"Najgolemiot e "<<m;
    return 0;
}
/* definicija na funkcijata */
int maximum( int x, int y, int z ){
    int max = x;
    if ( y > max ) max = y;
    if ( z > max ) max = z;
    return max;
}
```

```
Vnesi tri celi broevi: 22 85 17
Najgolemiot e: 85
```

Функциски прототип - пример

```
/* Primer: Opredeluvanje na najgolemiot od tri broja */
int maximum( int, int, int ); /* funkciski prototip */

int main(){
   int a, b, c, m;
 cout<<"Vnesi tri celi broja ";
   cin>>a>>b>>c;
   m = maximum( a, b, c );
   cout<<"Najgolemiot e "<<m;
   return 0;
}
/* definicija na funkcijata */
int maximum( int x, int y, int z ){
   int max = x;
   if ( y > max ) max = y;
   if ( z > max ) max = z;
   return max;
}
```

```
Vnesi tri celi broevi: 22 85 17
Najgolemiot e: 85
```

Вреднување на функциски повик

- Се креира мемориска локација за функцискиот повик
- Актуелните параметри на функцискиот повик формираат парови со формалните параметри на функцијата, од лево на десно
  - Нивниот број мора да е еднаков
- Се креира мемориска локација за секој формален параметар на функцијата, кон кој е придружен соодветниот актуелен параметар.
  - Типовите на двата вида параметри мора да се исти

Вреднување на функциски повик(2) – во функцијата

- Се креира мемориска локација за секоја локална променлива на функцијата
- Се изведуваат наредбите во телото на функцијата
- На крај, сите мемориски локации креирани за функцискиот резултат, формалните параметри и локалните променливи се ослободуваат
- **Вредноста на функцискиот резултат се пренесува како вредност на функцискиот повик**

Пренос на вредност

Формалните аргументи ги прифаќаат вредностите кои се задаваат при повикувањето на функцијата (напишани со задебелени букви во примерот)

Кај прототипот

```
int maximum( int x, int y, int z );
```

Кај дефинирањето

```
int maximum( int x, int y, int z )
{
   int max = x;
   if ( y > max )       max = y;
   if ( z > max )       max = z;
   return max;
}
```

Пренос на вредност

- Се наведуваат при ПОВИКУВАЊЕТО на функцијата (задебелени темно сини букви во примерот)

```
m = maximum( a, b, c );
```

- При повикување на функцијата...

  ... вредностите на вистинските аргументи се пренесуваат во формалните аргументи.

```
int maximum( int x, int y, int z );
```

Пренос на вредност...

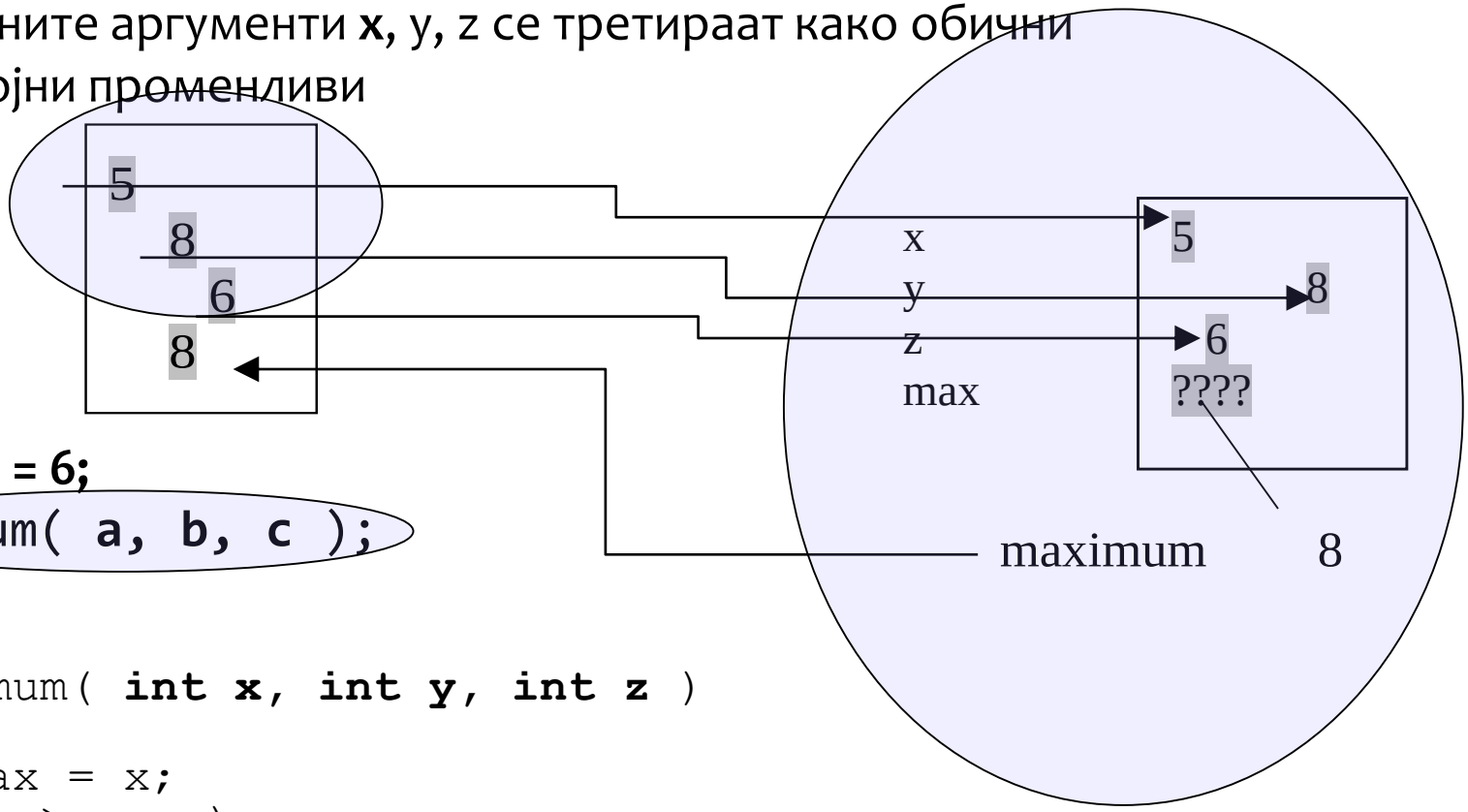
- при секој повик на функцијата се проверува бројот и типот на аргументите за време на преведувањето
- ако **бројот** на вистинските аргументи е различен од бројот на формалните, се јавува порака за грешка.
- ако **типот** на формалните аргументи е различен од типот на вистинските аргументи соодветно,
  - прво компајлерот се обидува да го „преведе" едниот тип во друг (на пример double во int). Притоа, може да се изврши повикот, но резултатот да биде погрешен.
  - Ако не успее „преведувањето" на типот, тогаш компајлерот јавува порака за грешка.

Пренос на вредност

- За секој *формален аргумент при пренесување на вредност*
  - се креира нова променлива,
  - се иницијализира променливата со вредноста на вистинскиот аргумент,
  - **измените во оваа променлива не влијаат на вистинскиот аргумент.**

Пренос на вредност...

- Формалните аргументи **x**, y, z се третираат како обични целобројни променливи

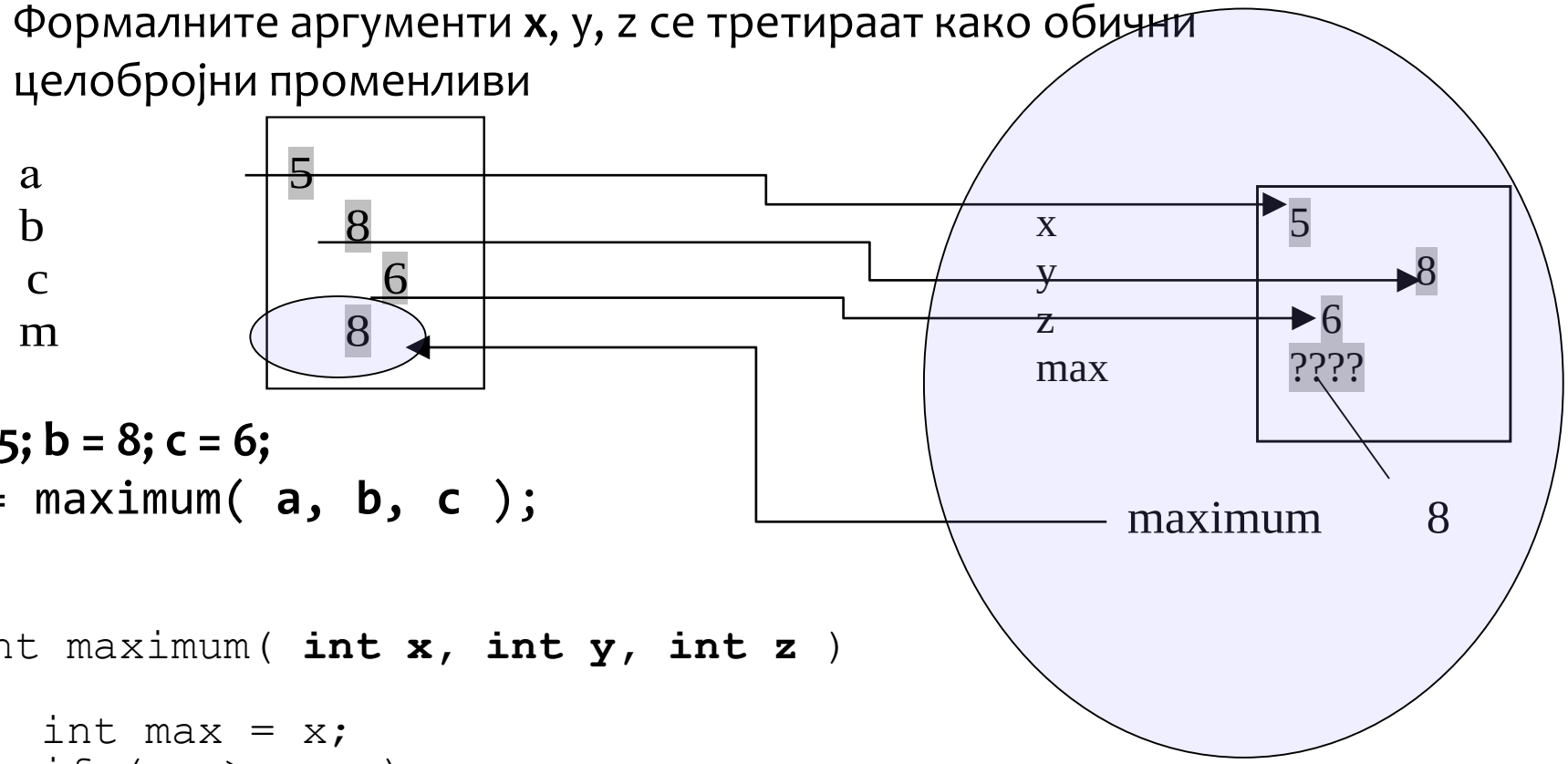



```
a = 5; b = 8; c = 6;
m = maximum( a, b, c );
```

```
int maximum( int x, int y, int z )
{
   int max = x;
   if ( y > max )       max = y;
   if ( z > max )       max = z;
   return max;
}
```

Пренос на вредност...

- Формалните аргументи **x**, y, z се третираат како обични целобројни променливи




```
a = 5; b = 8; c = 6;
m = maximum( a, b, c );
```

```
int maximum( int x, int y, int z )
{
   int max = x;
   if ( y > max )        max = y;
   if ( z > max )        max = z;
   return max;
}
```

Пренос на вредности во функцијата

- Со овој метод сите промени што ќе ги претрпи вредноста на формалниот аргумент при извршувањето на функцијата немаат никаков ефект на вредноста на аргументот со кој е повикана функцијата.

- Со тоа се одбегнуваат несаканите ефекти поради промена на вредноста на формалниот аргумент

Пренос на вредности во функцијата

```
int swapv (int x, int y) {
    int t;
    t = x; x = y; y = t;
    cout<<" x = "<<x<<" y = "<< y<<endl;
    return 0;
}
int main ( ) {
    int a = 10, b =20;
    cout<<" a = "<<a<<" b = "<< b<<endl;
    swapv (a, b);
    cout<<" a = "<<a<<" b = "<< b<<endl;
}
```

```
a = 10 b =20
x = 20 y =10
a = 10 b =20
```

Вметнати (inline) функции

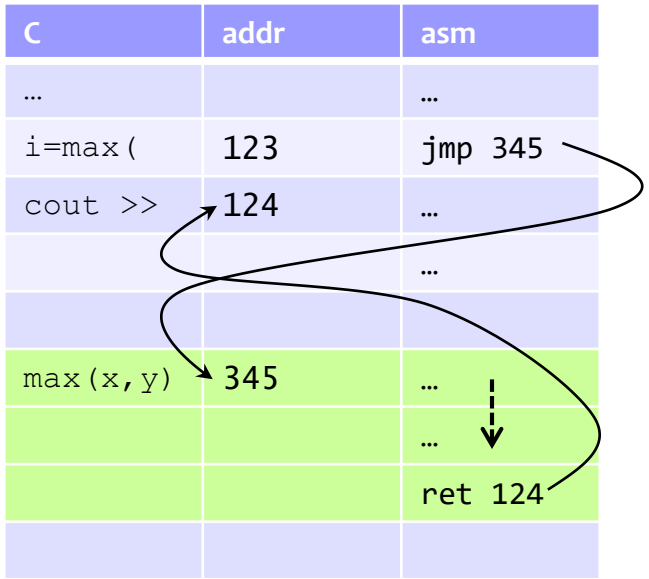
- Во C++ постојат *вградени (inline)* функции кои овозможуваат заштеда на времето потребно да се направи функциски повик. Ова особено важи ако функциите се мали.
- Квалификаторот **inline** се става пред повратниот тип на функцијата за да му се сугерира на компајлерот да генерира копија на функцискиот код на даденото место каде што треба да се направи функциски повик и со тоа да се избегне функцискиот повик.
- Лошото тука е што повеќе копии од функцискиот код се вметнуваат во програмот, а со тоа го прават многу поголем, отколку да имаме само една копија од функцијата и да ја повикуваме со функциски повик. Компајлерот има можност да не го регистрира квалификаторот **inline**, а ова вообичаено и се случува, освен за мали функции.

inline функции (1)

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

int max(int a, int b) {
    if (a > b)
        return a;
    return b;
}

int main()
{
    int i, x=23, y=45;
    i=max(x++, y++);
    cout << "x= " << x << " y= " << y
    << " max(x++,y++)= " << i << endl;
    return 0;
}
```




```
x= 24 y= 46 max(x++,y++)= 45
x= 24 y= 46
```

inline функции (2)

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

inline int max(int a, int b) {
    if (a > b)
        return a;
    return b;
}

int main()
{
    int i, x=23, y=45;
    i=max(x++, y++);
    cout << "x= " << x << " y= " << y
    << " max(x++,y++)= " << i << endl;
    return 0;
}
```

C	addr	asm
…		…
i=max(	123	jmp 345
max(x,y)	345	…
		…
		…
cout >>	124	…
		…

```
x= 24 y= 46 max(x++,y++)= 45
x= 24 y= 46
```

Генерираниот код за телото на функцијата **се вметнува** на местото на нејзиниот повик наместо да се прави скок на друга адреса

Пример – пресметување волумен на 3 коцки

```
inline double cube( const double s ) { return s * s * s; }
int main()
{
  double side;
  for ( int k = 1; k < 4; k++ ) {
            cout << "Vnesi dolzhina na stranata na kockata:  ";
            cin >> side;
            cout << "Volumenot na kocka so strana "
                        << side << " e " << cube( side ) << endl;
  }
   return 0;
}
```

Пример – пресметување волумен на 3 коцки

```
inline double cube( const double s ) { return s * s * s; }

int main()
{
  double side;
  for ( int k = 1; k < 4; k++ ) {
        cout << "Vnesi dolzhina na stranata na kockata:  ";
        cin >> side;
        cout << "Volumenot na kocka so strana "
             << side << " e " << cube( side ) << endl;
  }
   return 0;
}
```

```
Vnesi dolzhina na stranata na kockata:  4
Volumenot na kocka so strana 4 e 64
Vnesi dolzhina na stranata na kockata:  3
Volumenot na kocka so strana 3 e 27
Vnesi dolzhina na stranata na kockata:  2
Volumenot na kocka so strana 2 e 8
```

Области на важење на променливите

- Променливите се разликуваат според типот, областа на важење и начинот на креирање

- Според областа на важење на променливите, тие грубо може да се поделат на:
  - Глобални променливи
  - Локални променливи

Локални и глобални променливи

- **Локални променливи**
  - постојат само во рамките на функциите и блоковите наредби во кои се креирани
  - непознати се за сите останати функции или блокови наредби
  - се уништуваат кога ќе се напушти функцијата или блокот наредби во кои се креирани
  - повторно се креираат при секој следен повик на функцијата или блокот наредби
- **Глобални променливи**
  - се дефинираат надвор од секоја функција и нив може да ги користи секоја функција во програмата
  - променливите постојат се додека се извршува програмата

Правила за подрачјето на важење на променливите

1. Променливата не може да се користи надвор од подрачјето на важење,
2. Глобалните променливи можат да се користат во целата програма,
3. Променливите декларирани во еден блок можат да се користат само во него,
4. Променлива декларирана во една функција не може да се користи во друга,
5. Променливата може да биде скриена во некој дел од нејзиното подрачје на важење,
6. Не може две различни променливи со исто име да имаат исто подрачје на важење.

Пример - глобални променливи

```
#include <iostream>
using namespace std;
int add_numbers( void );
int value1, value2, value3;

int add_numbers( void ) {
   int result;
   result = value1 + value2 + value3;
   return result;
}
int main() {
   int result;
   value1 = 10;
   value2 = 20;
   value3 = 30;
   result = add_numbers();
   cout<<value1<<" + "<< value2<<" + "<<value3<<" = "<< result;
   return 0;
}
```

```
10 + 20 + 30 = 60
```

Дефинирање на глобални променливи

Областа на важење на глобалните променливи може да се ограничи со внимателно поставување на декларациите на променливите во датотеката.
Пример

```
#include <iostream>
void no_access( void );
void all_access( void );
int n2; /*n2 e poznata od ovaa tocka*/
void no_access( void ) {
   n1 = 10; /* netocno, n1 ne e deklarirana, i
   kompajlerot pri preveduvanje na programata vo
                ovaa tocka ke javi greska*/
   n2 = 5;    /* tocno */
}
int n1;         /* n1 e deklarirana i vazi od ovaa tocka
   vo ostatokot od programata */

void all_access( void ) {
   n1 = 10;    /* tocno */
   n2 = 3;     /* tocno */
}
```

Покривање (криење) на глобални променливи

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

void prikazi();
int x = 20;
int main() {
   cout<<" vo glavnata programa
"<<x<<endl;
   prikazi();
   return 0;
}
void prikazi() {
   cout<<" vo funkcija "<< x<<endl;
}
```

```
vo glavnata programa 20
vo funkcija 20
```

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

void prikazi();
int x = 20;
int main() {
   int x = 10;
   cout<<" vo glavnata programa
"<<x<<endl;
   prikazi();
   return 0;
}
void prikazi() {
   cout<<" vo funkcija "<< x<<endl;
}
```

```
vo glavnata programa 10
vo funkcija 20
```

Локални променливи

Локални променливи може да се декларираат и за секој блок наредби и за нив важат истите правила - променливите се креираат при секое повторно влегување во блокот и важат за блокот и сите останати блокови вгнездени во истиот.

```
for(i=0; i<10; i++) {
      float x=0.0;
      . . .
}
```

x ќе важи само во рамките на **for** блокот и тоа при секое повторување на циклусот: на почетокот ќе се резервира простор за истата и ќе се иницијализира, а на крајот се уништува.

Локални променливи

```
int main() {
    int x = 1;
    {
        int x = 2;
        {
            int x = 3; cout << "x=" << x;
        }
        cout << " x=" << x;
    }
    cout << " x=" << x;
    return 0;
}
```

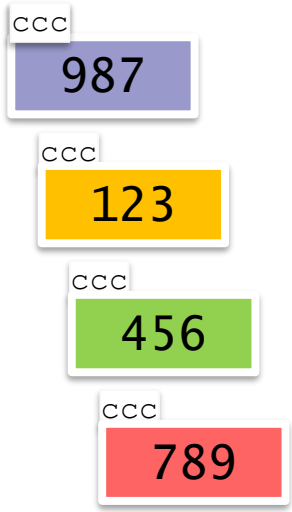
```
x=3 x=2 x=1
```

scope оператор ::

```
#include <iostream>
   using namespace std;
   int ccc=987;
   int main()
   {
      int ccc=123;
      cout << "main   ccc=" << ccc << endl;
      cout << "Global ccc=" << ::ccc << endl;
      {
         int ccc=456;
         cout << "Inner  ccc=" << ccc << endl;
         cout << "Global ccc=" << ::ccc << endl;
         {
            int ccc=789;
            cout << "Innest ccc=" << ccc << endl;
            cout << "Global ccc=" << ::ccc << endl;
            ::ccc=321;
         }
         cout << "Inner  ccc=" << ccc << endl;
         cout << "Global ccc=" << ::ccc << endl;
      }
      cout << "main   ccc=" << ccc << endl;
      cout << "Global ccc=" << ::ccc << endl;
   }
```

пристап до глобалната променлива

```
main   ccc=123
Global ccc=987
Inner  ccc=456
Global ccc=987
Innest ccc=789
Global ccc=987
```

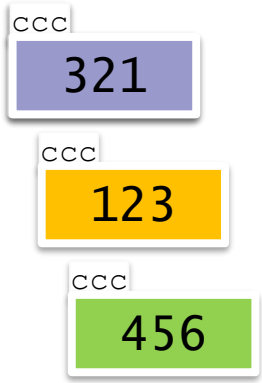


scope оператор ::

```
#include <iostream>
   using namespace std;
   int ccc=987;
   int main()
   {
      int ccc=123;
      cout << "main   ccc=" << ccc << endl;
      cout << "Global ccc=" << ::ccc << endl;
      {
         int ccc=456;
         cout << "Inner  ccc=" << ccc << endl;
         cout << "Global ccc=" << ::ccc << endl;
         {
            int ccc=789;
            cout << "Innest ccc=" << ccc << endl;
            cout << "Global ccc=" << ::ccc << endl;
            ::ccc=321;
         }
         cout << "Inner  ccc=" << ccc << endl;
         cout << "Global ccc=" << ::ccc << endl;
      }
      cout << "main   ccc=" << ccc << endl;
      cout << "Global ccc=" << ::ccc << endl;
   }
```

пристап до глобалната променлива

```
main   ccc=123
Global ccc=987
Inner  ccc=456
Global ccc=987
Innest ccc=789
Global ccc=987
Inner  ccc=456
Global ccc=321
```



Подразбирани вредности за аргументи на функција

```
#include <iostream>
using namespace std;
void show( int = 1, float = 2.3, long = 4 );
int main()
{
    show();                 // All three arguments default
    show( 5 );              // Provide 1st argument
    show( 6, 7.8 );         // Provide 1st and 2nd
    show( 9, 10.11, 12L );   // Provide all three arguments
// show(  , 3.5, 7L); ** Error: cannot omit only first
argument
}
void show( int first, float second, long third )
{
    cout << "first = " << first;
    cout << ", second = " << second;
    cout << ", third = " << third;
    cout << endl;
    return;
}
```

```
first = 1, second = 2.3, third = 4
first = 5, second = 2.3, third = 4
first = 6, second = 7.8, third = 4
first = 9, second = 10.11, third = 12
```

Пресметување волумен на кутија

```
#include <iostream>
using namespace std;
int boxVolume( int length = 1, int width = 1, int height = 1
);
int main()
{
   cout << "Predefiniraniot volumen e: " << boxVolume()
        << "\nVolumenot na kutija so dolzhina 10, shirina 1 i
visina 1 e: " << boxVolume( 10 )
        << "\nVolumenot na kutija so dolzhina 10, shirina 5 i
visina 1 e: " << boxVolume( 10, 5 )
        << "\nVolumenot na kutija so dolzhina 10, shirina 5 i
visina 2 e: " << boxVolume( 10, 5, 2 )<< endl;
      return 0;
}

int boxVolume( int length, int width, int height )
{
   return length * width * height;
}
```

Преоптоварување (обременување) на функции

- Функциите може
  - да се декларирани над исто подрачје на важење
  - да имаат исто име
  - НО, да се разликуваат во бројот и/или типот на аргументите – потпис на функцијата (signature)
- Името и потписот прават функциите да се разликуваат на единствен начин

Преоптоварување (обременување) на функции

Функциите:

- `int foo(int x)`
- `int foo(float x)`
- `int foo(char x)`

- `void goo(int x, float y)`
- `void goo(char x, float y)`

се различни функции (исто име но различен тип на аргументи)

Функциите:

- `void goo(int x, float y)`
- `void goo(int x, float y, char z)`

се различни функции (исто име но различен број на аргументи)

Помеѓу функциите:

- `int func(char c)`
- `float func(char c)`

не може да се прави разлика (исто име и исти аргументи)

Пример за преоптоварување на функции

```
#include <iostream>
using namespace std;
void show(int val)
{
    cout << "Integer: " << val << endl;
}
void show(double val)
{
    cout << "Double: " << val << endl;
}


int main()
{
    show(12);
    show(3.1415);
    return 0;
}
```

```
Integer: 12
Double: 3.1415
```

Пример за преоптоварување на функции (2)

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

int square( int x ) { return x * x; }

double square( double y ) { return y * y; }

int main()
{
   cout << "Kvadrat na celiot broj 7 e " << square( 7 )
        << "\nKvadratot na double vrednosta 7.5 e " <<
   square( 7.5 )
        << endl;

   return 0;
}
```

```
Kvadrat na celiot broj 7 e 49
Kvadratot na double vrednosta 7.5 e 56.25
```

Класи променливи според начинот на креирање

- перманентни или привремени променливи
- перманентните се најчесто глобални променливи
  - се креираат и иницијализираат на почетокот на секоја програма и постојат до завршувањето на програмата
- локалните променливи се најчесто привремени променливи
  - се креираат и иницијализираат при извршувањето на блокот во кој се декларирани
  - се уништуваат при напуштање на блокот во кој се декларирани
- Следните клучни зборови се користат да се опише кога и каде променливите ќе се креираат и уништат

```
auto
static
extern
register
```

Static и auto променливи

- Променливите декларирани како **`static`** се креираат и иницијализираат еднаш, на првиот повик на функцијата
  - При секој следен повик на функцијата не се креираат ниту се реиницијализираат статичките променливи.
  - Кога ќе заврши функцијата овие променливи се уште постојат, но не може да им се пристапи од другите функции.

- Променливите декларирани како **`auto`** се користат кога очекуваме од компајлерот автоматски да претпостави кој тип на променлива треба да се користи.
- Променливата мора да е дефинирана за да може компајлерот да го претпостави типот
- зборот auto пред име на функцијата или кога имаме auto повратна вредност значи дека типот на повратна вредност ќе биде евалуиран од вредноста која се враќа во време на извршување

Пример Static и auto променливи

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

void demo( void );
void demo( void ) {
    auto local = 0; //avtomatski se odreduva
    static int svar = 0;
    cout<<"local = "<<local<<", static = "<< svar<<endl;
    ++local; ++svar;
}
int main() {
    int i=0;
    while( i < 3 ) { demo(); i++; }
    return 0;
}
```

```
local = 0, static = 0
local = 0, static = 1
local = 0, static = 2
```

Локални променливи декларирани како static

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

int main() {
  int counter; /* brojac */
  for (counter = 0; counter < 3; ++counter) {
    int temporary = 1;   /* privremena promenliva */
    static int permanent = 1; /* permanentna promenliva */
    cout<<"Temporary "<< temporary<<" Permanent "<<
permanent<<endl;
    ++temporary;
    ++permanent;
   }
   return (0);
}
```

```
Temporary 1 Permanent 1
Temporary 1 Permanent 2
Temporary 1 Permanent 3
```