9. Funkcije

1

9.1. FUNKCIJA main()
9.2. DEFINICIJA FUNKCIJE
9.3. STAVEK return
9.4. KLIC FUNKCIJE IN PRENOS PARAMETROV
9.5. PREKRIVANJE FUNKCIJ

Kaj je funkcija?

2

- **funkcija** (podprogram) je samostojna enota ukazov, ki rešijo nek problem
- preprosi problemi rešljivi z eno funkcijo
- zahtevnejši problemi
  - razdelimo na manjše (lažje rešljive) probleme
  - opišemo jih s funkcijami
- funkcije
  - omogočajo delitev naloge na samostojne enote
  - stavke v funkciji zapišemo enkrat, izvedemo večkrat s klicem funkcije
  - lahko ponovno uporabimo znotraj različnih programov

9.1. Funkcija `main()`

3

- kliče jo operacijski sistem
  - ne kliče je uporabnik
  - v njej se prične program izvajati
  - ima dve možni obliki

```
// brez parametrov
int main() {

  ...

  return 0;
}
```

```
// s parametri
int main(int argc, char **argv) {

  ...

  return 0;
}
```

Še o funkciji `main()`

4

- stavek return znotraj funkcije main() smemo izpustiti
  - brez tega stavka prevajalnik vrne privzeto vrednost 0

OPOMBA:

- V jeziku C je dovoljeno tudi, da je funkcija main() tipa void. Standard C++ to eksplicitno prepoveduje!

9.2. Definicija funkcije

5

- **definicija funkcije** je del programa, ki določa:
  - ime funkcije
  - tip rezultata funkcije
  - parametre funkcije
  - kaj funkcija dela
- sintaksa definicije funkcije je naslednja

```
glava funkcije {
    stavki
}
```

telo funkcije

Tip funkcije

6

- glavi funkcije pravimo tudi prototip funkcije
- sintaksa glave funkcije je naslednja:

```
tip ime(deklaracije_parametrov)
```

POMEN:

- `tip` določa tip vrednosti, ki je rezultat funkcije. Pravimo, da funkcija vrača to vrednost. Če funkcija ne vrača nobene vrednosti, je tipa `void`.

Ime funkcije

- sintaksa glave funkcije je naslednja:

```
tip ime(deklaracije_parametrov)
```

POMEN:

- `ime` določa identifikator, ki predstavlja funkcijo. S tem identifikatorjem izvedemo stavke v funkciji – pravimo, da kličemo funkcijo.

Parametri funkcije

- **parametri**
  - predstavljajo podatke brez katerih ne moremo rešiti naloge
  - lahko jih je nič ali več
  - če parametrov ni, lahko pustimo seznam prazen ali pišemo void
- **deklaracija parametra**
  - je deklaracija (lokalne) spremenljivke
    - lahko jih uporabimo znotraj funkcije
  - več parametrov ločimo z vejicami
- **formalni parametri**
  - parametri so spremenljivke
  - vsebujejo vrednost, ki jo podamo ob klicu funkcije

Telo funkcije

9

- **telo funkcije**
  - je zaporedje stavkov, ki reši dano nalogo
    - stavki, ki pridejo do rezultata
  - lahko vsebuje poljubne stavke
  - uporabljamo lahko tudi formalne parametre
    - deklarirani so v glavi funkcije

Kje definiramo funkcije?

10

- funkcije definiramo "kjerkoli" v programu
  - ne smemo jih definirati znotraj drugih funkcij
  - definirati jih moramo, preden jih uporabimo
    - če je funkcija napovedana s prototipom funkcije, jo lahko definiramo tudi za tem, ko jo uporabimo. Deklaracija funkcije mora biti podana pred uporabo.

9.3. Stavek `return`

11

- stavek `return`
  - nemudoma prekine izvajanje funkcije
  - nadzor izvajanja se vrne na mesto klica funkcije
  - če stavku `return` sledi še izraz, se na mesto klica funkcije posreduje še vrednost izraza
- sintaksa dopušča dve možni obliki:

```
return;          ali          return izraz;
```

9.3. Stavek `return` (2)

12

- funkcije tipa void:
  - uporabljati smemo le prvo obliko stavka return (brez izraza)
  - ne rabimo uporabiti stavka `return`
    - izvajanje funkcije se v tem primeru zaključi, ko pridemo na konec funkcije
- če je funkcija tipa različnega od void:
  - mora vsebovati vsaj en stavek `return`
    - z njim povemo, kaj je rezultat funkcije
  - uporabiti moramo drugo obliko (z izrazom)
  - izraz mora biti združljivega tipa s tipom funkcije

Primer 01 (definicija funkcije)

13

```
/*
    Funkcije vrne večje od dveh celih
    števil.
*/
int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}
```

Če iščemo maksimum dveh celih števil, moramo ti dve števili poznati. Predstavimo ju s formalnima parametroma.

Brez da poznamo ti dve števili, ne moremo rešiti naloge.

Če je prvo število večje, je to rezultat funkcije (povemo s stavkom return). V nasprotnem primeru je rezultat drugo število.

Maksimum dveh celih števil je celo število, zato je funkcija tipa **int**.

9.4. Klic funkcije in prenos parametrov

14

- funkcije izvedemo s **klicem funkcije**
- funkcijo kličemo z
  - imenom
  - vsemi parametri, ki jih funkcija zahteva
    - parametri se morajo ujemati v številu in tipu formalnih parametrov
- dejanski parametri
  - parametri, ki jih podamo ob klicu funkcije
- klic funkcije
  - predstavlja izraz takega tipa, kot je tip funkcije

Formalni in dejanski parametri

15

- formalni parametri
  - tisti podani v definiciji funkcije
- dejanski parametri
  - tisti podani v klicu funkcije
- povezava med njimi je odvisna od **načina prenosa parametrov**
  - način prenosa parametrov določa obnašanje formalnih parametrov funkcije

9.4.1. Prenos parametrov po vrednosti

16

- privzeti način prenosa parametrov
- formalni parametri prejmejo vrednost dejanskih parametrov
  - formalni parametri so lokalne spremenljivke
  - te lokalne spremenljivke se inicializirajo na vrednost dejanskih parametrov
- parametri, ki se prenašajo po vrednosti imenujemo tudi vhodni parametri
  - ustvarijo se nove spremenljivke z vrednostmi dejanskih parametrov

Primer 01 (celoten primer)

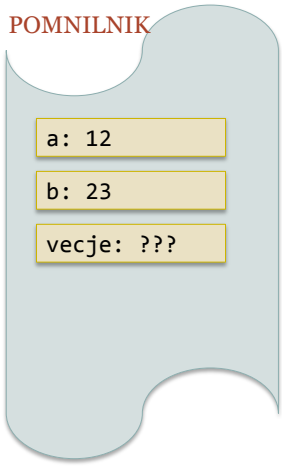
17

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

int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}

int main()
{
    int a, b;
    cout << "Vnesi dve celi stevili: ";
    cin >>a >> b;

    int vecje;
    vecje = maksimum(a,b);
    cout << "Vecje izmed prebranih stevil je " << vecje << endl;
    vecje = maksimum(2*a+b, 100);
    cout << "Vecje izmed stevil " << 2*a+b << " in 100 je " << vecje << endl;
    vecje = maksimum(a+b,2*a-b);
    cout << "Vecje izmed stevil " << a+b << " in " << 2*a-b << " je " << vecje << endl;
    return 0;
}
```

Primer 01 – kaj se dogaja

18

```
int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}

int main()
{
    int a, b;
    ...
    int vecje;
    vecje = maksimum(a,b);
    ...
```



Primer 01 – kaj se dogaja

19

```
int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}

int main()
{
    int a, b;
    ...
    int vecje;
    vecje = maksimum(a,b);
    ...
```

kličemo funkcijo `maksimum()`

Primer 01 – kaj se dogaja

20

```
int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}

int main()
{
    int a, b;
    ...
    int vecje;
    vecje = maksimum(a,b);
    ...
```

vrednosti se prekopirajo

Primer 01 – kaj se dogaja

21

```
int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}

int main()
{
    int a, b;
    ...
    int vecje;
    vecje = maksimum(a,b);
    ...
```

Primer 01 – kaj se dogaja

22

```
int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}

int main()
{
    int a, b;
    ...
    int vecje;
    vecje = maksimum(a,b);
    ...
```

Primer 01 – kaj se dogaja

23

```
int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}

int main()
{
    int a, b;
    ...
    int vecje;
    vecje = maksimum(a,b);
    ...
```

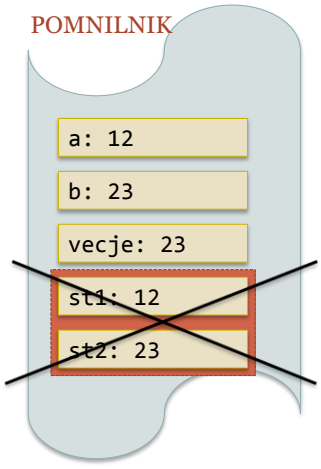
Preneha z izvajanjem funkcije.

Vrne vrednost spremenljivke st2.

Primer 01 – kaj se dogaja

24

```
int maksimum(int st1, int st2) {
  if (st1 > st2)
    return st1;
  else
    return st2;
}

int main()
{
    int a, b;
    ...
    int vecje;
    vecje = maksimum(a,b);
    ...
```



9.4.2. Prenos parametrov po referenci

25

- pred ime formalnega parametera zapišemo operator &
- formalni parametri predstavljajo isto lokacijo v pomnilniku, kjer so dejanski parametri
  - za dejanske parametre ne smemo uporabljati konstant in izrazov
  - dejanski parameter mora biti lokacija, kamor lahko shranimo vrednost - spremenljivka
  - predstavljamo si lahko, da je formalni parameter začasno novo ime obstoječe spremenljivke
    - začasno ... znotraj funkcije

Primer 02 (z napako)

26

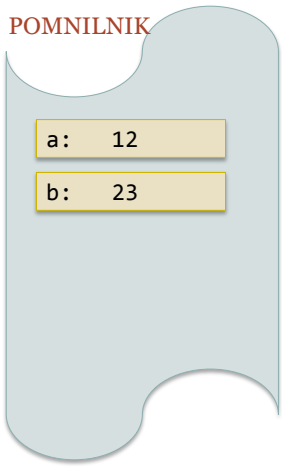
```
#include <iostream>
using namespace std;

/*
  Napiši funkcijo, ki zamenja vrednosti dveh
  celoštevilskih spremenljivk.
*/
void zamenjaj(int st1, int st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main() {
  cout << "Vnesi dve celi stevili: ";
  int a,b;
  cin >> a >> b;

  cout << "Pred zamenjavo: " << endl;
  cout << "a=" << a << "  b=" << b << endl;

  zamenjaj(a,b);

  cout << "Po zamenjavi: " << endl;
  cout << "a=" << a << "  b=" << b << endl;
  return 0;
}
```

Razmisli, zakaj imata po zamenjavi spremenljivki `a` in `b` še vedno enako vrednost!

Primer 02 – zakaj ne deluje

27

```
void zamenjaj(int st1, int st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```

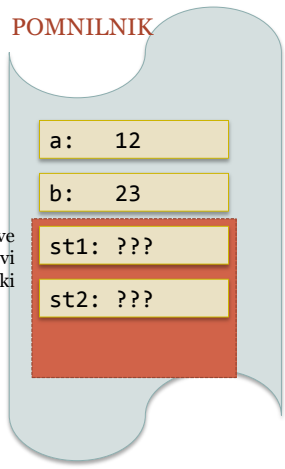


Primer 02 – zakaj ne deluje

28

```
void zamenjaj(int st1, int st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```

kličemo funkcijo zamenjaj()

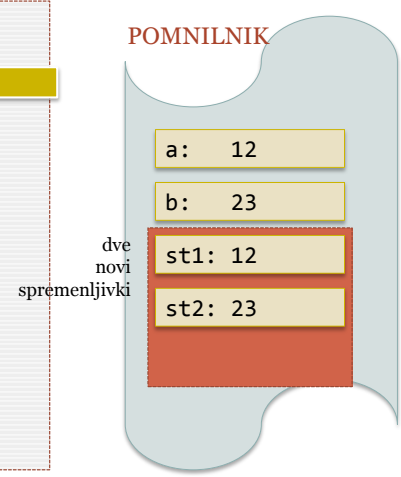


Primer 02 – zakaj ne deluje

29

```
void zamenjaj(int st1, int st2){
   int t = st1;
   st1 = st2;
   st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```

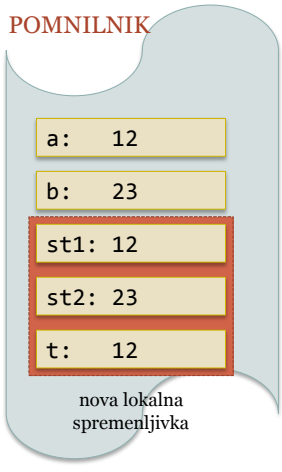
vrednosti se kopirajo



Primer 02 – zakaj ne deluje

30

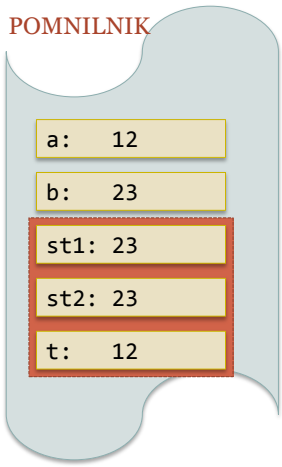
```
void zamenjaj(int &st1, int &st2){
   int t = st1;
   st1 = st2;
   st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```



Primer 02 – zakaj ne deluje

31

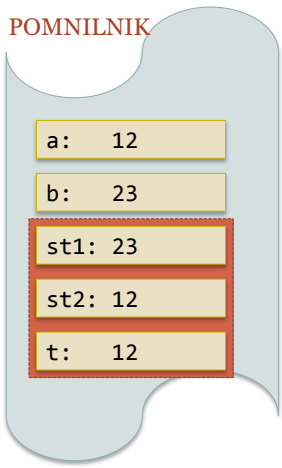
```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```



Primer 02 – zakaj ne deluje

32

```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```



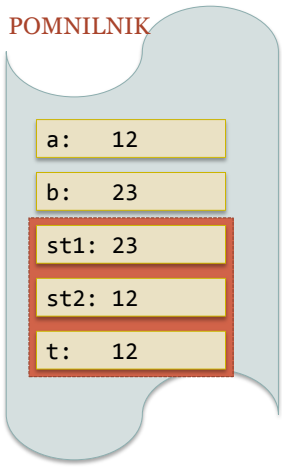
Primer 02 – zakaj ne deluje

33

```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```

Preneha z izvajanjem funkcije.

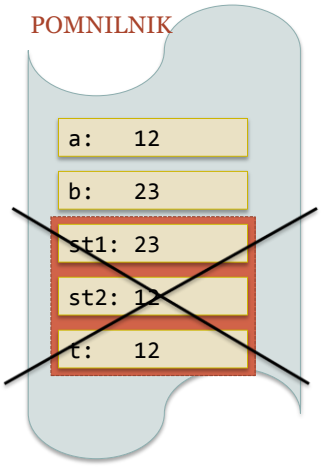
Pobriši lokalne spremenljivke.



Primer 02 – zakaj ne deluje

34

```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```



Primer 02 (pravilna rešitev)

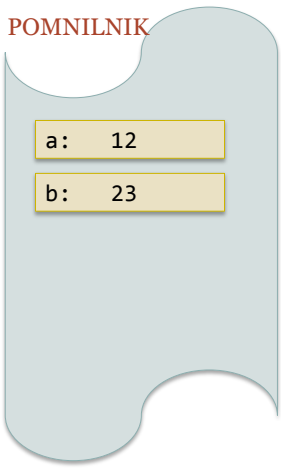
35

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

/*
  Napiši funkcijo, ki zamenja vrednosti dveh
  celoštevilskih spremenljivk.
*/
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main() {
  cout << "Vnesi dve celi stevili: ";
  int a,b;
  cin >> a >> b;

  cout << "Pred zamenjavo: " << endl;
  cout << "a=" << a << "  b=" << b << endl;

  zamenjaj(a,b);

  cout << "Po zamenjavi: " << endl;
  cout << "a=" << a << "  b=" << b << endl;
  return 0;
}
```

Primer 02 – kaj se dogaja

36

```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```

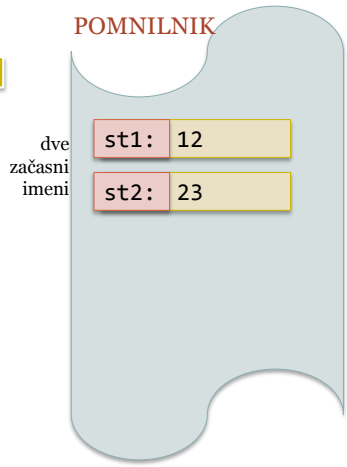


Primer 02 – kaj se dogaja

37

```
void zamenjaj(int &st1, int &st2){
   int t = st1;
   st1 = st2;
   st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```

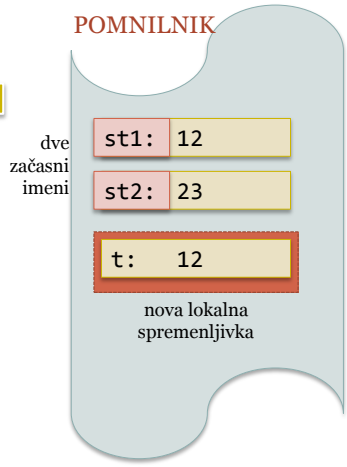
kličemo funkcijo zamenjaj()



Primer 02 – kaj se dogaja

38

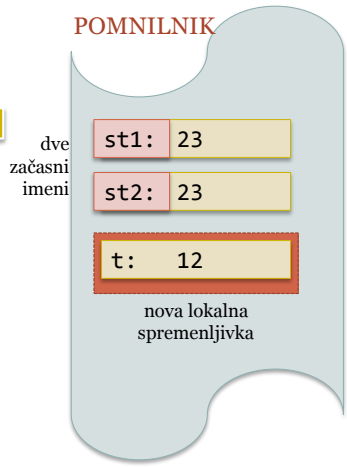
```
void zamenjaj(int &st1, int &st2){
   int t = st1;
   st1 = st2;
   st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```



Primer 02 – kaj se dogaja

39

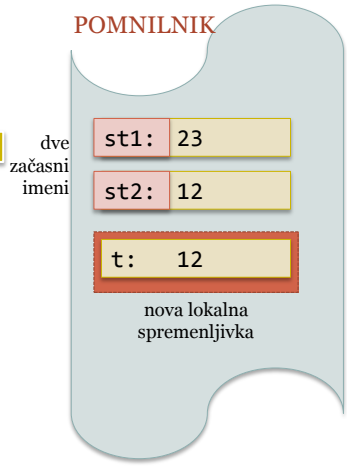
```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```



Primer 02 – kaj se dogaja

40

```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...

    zamenjaj(a,b);
    ...
```



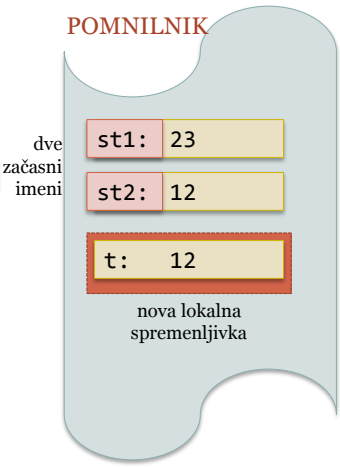
Primer 02 – kaj se dogaja

41

```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...
    zamenjaj(a,b);
    ...
```

Preneha z izvajanjem funkcije.

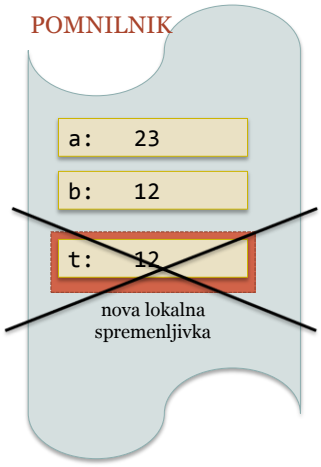
Pobriši začasna imena in lokalne spremenljivke.



Primer 02 – kaj se dogaja

42

```
void zamenjaj(int &st1, int &st2){
  int t = st1;
  st1 = st2;
  st2 = t;
}

int main()
{
    int a, b;
    ...
    zamenjaj(a,b);
    ...
```



9.5. Prekrivanje funkcij

43

- že vemo:
  - znotraj istega bloka ne smemo deklarirati dveh spremenljivk z istim identifikatorjem
- kaj pa funkcije?
  - C++ dovoljuje, da imamo več funkcij z istim imenom
  - **morajo se razlikovati v**
    - številu parametrov
      in / ali
    - tipu parametrov

9.5. Prekrivanje funkcij (2)

44

- prekrivanje funkcij
  - imamo dve ali več funkcij z istim imenom in **različnim seznamom parametrov**
- tip funkcije ni merodajen
  - ne smemo imeti funkcije z istim imenom in istim seznamom parametrov, tudi, če je se tipa funkcije razlikujeta
- katero funkcijo kličemo, se razbere iz seznama parametrov

Primer 03


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

/* Funkcija izračuna kvadrat števila. */
int kvadrat(int x) {
    return x*x;
}

double kvadrat(double x) {
    return x*x;
}

int main()
{
    int a;
    double b;
    cout << "Vnesi eno celo stevilo: ";
    cin >> a;
    cout << "Vnesi eno realno stevilo: ";
    cin >> b;

    cout << "Kvadrat stevila " << a << " je " << kvadrat(a) << endl;
    cout << "Kvadrat stevila " << b << " je " << kvadrat(b) << endl;
    return 0;
}
```

Naslednje funkcije ne moremo imeti v tem programu, zakaj?

```
double kvadrat(int x)
{
    return x*x;
}
```