Uvod u programiranje

- predavanja -

prosinac 2025.

---

17. Organizacija programa

Organizacija složenih programa

Prototip funkcije

Primjer

prog.c

```
#include <stdio.h>

double suma(double a,
            double b) {
   return a + b;
}

int main(void) {
   int a = 2, b = 3, s;
   s = suma(a, b);
   printf("suma = %d", s);

   return 0;
}
```

- Prevodilac analizira i prevodi izvorni kod od početka prema kraju datoteke
- Ako su funkcije `suma` i `main` poredane kao u ovom primjeru, u trenutku kada se prevode naredbe s pozivom funkcije i pridruživanja rezultata, prevodiocu su poznati tipovi parametara i tip funkcije `suma`, na temelju čega može na ispravan način obaviti potrebne konverzije tipova.

Prevođenje i povezivanje u jednom koraku

```
gcc -std=c11 -Wall -pedantic-errors -o prog.exe prog.c
```

```
suma = 5
```

- Napomena: prikazana funkcija je prikladna za ilustraciju, ali u stvarnosti nikad ne bismo pisali funkciju poput funkcije `suma`. Besmisleno je i štetno.

Primjer

prog.c

```
#include <stdio.h>

int main(void) {
   int a = 2, b = 3, s;
   s = suma(a, b);
   printf("suma = %d", s);

   return 0;
}

double suma(double a,
            double b) {
   return a + b;
}
```

- Ako su funkcije `suma` i `main` poredane kao u ovom primjeru, u trenutku kada se prevode poziv funkcije i pridruživanje rezultata, prevodiocu neće biti poznati tipovi parametara i tip funkcije `suma`, pa neće moći primijeniti potrebne konverzije podataka.
- Program se ili neće uspjeti prevesti ili neće raditi ispravno.

```
gcc -std=c11 -Wall -pedantic-errors -o prog.exe prog.c
prog.c:5:8: error: implicit declaration of function 'suma' [-Wimplicit-function-declaration]
prog1.c: At top level:
prog1.c:11:8: error: conflicting types for 'suma'
 double suma(double a,
        ^~~~
...
```

Prototip (deklaracija) funkcije

```
#include <stdio.h>

double suma(double a, double b);

int main(void) {
   int a = 2, b = 3, s;
   s = suma(a, b);
   printf("suma = %d", s);

   return 0;
}

double suma(double a,
            double b) {
   return a + b;
}
```

Prototip (deklaracija) funkcije prevodiocu pravovremeno osigurava informaciju o tipovima parametara i funkcije

- Definicija funkcije: opisuje naziv i tip funkcije, tipove i nazive parametara, **tijelo funkcije**
- Deklaracija funkcije: opisuje naziv i tip funkcije, tipove i nazive parametara, **ali ne i tijelo funkcije**

Prototip (deklaracija) funkcije

- Deklaracija jedne funkcije se u jednom programu može pojaviti više puta (pod uvjetom da je uvijek ista)
- Definicija jedne funkcije smije se pojaviti samo jednom

```
#include <stdio.h>
double suma(double a, double b);
int main(void) {
   int a = 2, b = 3, s;
   s = suma(a, b);
   printf("suma = %d", s);
   return 0;
}
      dopušteno, iako nepotrebno
double suma(double a, double b);

double suma(double a,
            double b) {
   return a + b;
}
```

```
#include <stdio.h>
double suma(double a,
            double b) {
   return a + b;
}
int main(void) {
   int a = 2, b = 3, s;
   s = suma(a, b);
   printf("suma = %d", s);
   return 0;
}
      neprihvatljivo: redefinicija funkcije
double suma(double a,
            double b) {
   return a + b;
}
```

Primjeri prototipova funkcija

- Prototipovi funkcija iz dosadašnjih primjera

```
int fakt(int n);
int binKoef(int m, int n);
unsigned long long fakt(unsigned int n);
double eksp(float x, int n);
int prebroji(void);
void ispisXY(float x, float y);
void preskoci(void);
char malo_u_veliko(char c);
void zamijeni(int *x, int *y);
void najveciPoRetcima(int *mat, int m, int n, int *rez);
```

  - kako napisati prototip funkcije: vrlo slično definiciji funkcije, ali umjesto tijela funkcije napisati terminator **;**

Programi koji sadrže više datoteka izvornog koda

zbroji.c

```
int zbroji2(int a, int b) {
   return a + b;
}

int zbroji3(int a, int b, int c) {
   return a + b + c;
}
```

mnozi.c

```
int mnozi2(int a, int b) {
   return a * b;
}

int mnozi3(int a, int b, int c) {
   return a * b * c;
}
```

glavni.c

```
#include <stdio.h>
int main(void) {
   printf("%d %d", zbroji2(15, 30), mnozi3(2, 4, 7));
   return 0;
}
```

- prevodilac *uvijek* zasebno prevodi svaku datoteku s izvornim kodom (.c)
  - za vrijeme prevođenja izvornog koda `glavni.c`, nedostaje mu informacija o tipovima funkcija i parametara funkcija `zbroji2` i `mnozi3`. Očito, nedostaju mu deklaracije tih funkcija
    - kopirati deklaracije tih funkcija u svaki program koji ih koristi?

Organizacija složenih programa

Datoteka zaglavlja

Datoteke zaglavlja (*header files*)

zbroji.h

```
int zbroji2(int a, int b);

int zbroji3(int a, int b, int c);
```

zbroji.c

```
#include "zbroji.h"

int zbroji2(int a, int b) {
   return a + b;
}

int zbroji3(int a, int b, int c) {
   return a + b + c;
}
```

mnozi.h

```
int mnozi2(int a, int b);

int mnozi3(int a, int b, int c);
```

mnozi.c

```
#include "mnozi.h"

int mnozi2(int a, int b) {
   return a * b;
}

int mnozi3(int a, int b, int c) {
   return a * b * c;
}
```

1. Deklaracije funkcija (sadržaj datoteke zaglavlja) *uključiti* u datoteku s pripadnim definicijama funkcija
   - prednosti: funkcije ne moraju biti poredane, prevodilac može tijekom prevođenja provjeriti usklađenost deklaracija i definicija funkcija
   - naziv datoteke zaglavlja koja ne pripada standardnoj biblioteci navodi se pod dvostrukim navodnicima

Datoteke zaglavlja (*header files*)

2. Deklaracije funkcija (sadržaj datoteke zaglavlja) *uključiti* u datoteku s izvornim kodom koji koristi funkcije deklarirane u datoteci zaglavlja
   - prednosti: prevodilac dobiva informaciju potrebnu za ispravnu konverziju argumenata i rezultata funkcije

glavni.c

```
#include <stdio.h>
#include "zbroji.h"
#include "mnozi.h"
int main(void) {
   printf("%d", zbroji2(15, mnozi3(2, 4, 7)));
   return 0;
}
```

   - naziv datoteke zaglavlja koja pripada standardnoj biblioteci navodi se unutar znakova <>. Nazivi ostalih datoteka zaglavlja navode se pod dvostrukim navodnicima.

Organizacija složenih programa

Modul

Modul

- Datoteka s izvornim kodom i pripadna datoteka zaglavlja (zajedno) predstavljaju *modul*
  - modul sadrži definicije i deklaracije funkcija, varijabli, tipova, itd. koji čine logičku cjelinu: npr. modul za matematičke operacije, modul za rad s nizovima znakova, modul za upravljanje standardnim ulazom/izlazom, itd.
    - datoteka zaglavlja (.h) uobičajeno sadrži *deklaracije* funkcija, varijabli, struktura, makro definicije
    - datoteka s izvornim kodom (.c) uobičajeno sadrži definicije funkcija i varijabli

zbroji.h

```
int zbroji2(int a, int b);

int zbroji3(int a, int b, int c);
```

zbroji.c

```
#include "zbroji.h"

int zbroji2(int a, int b) {
   return a + b;
}

int zbroji3(int a, int b, int c) {
   return a + b + c;
}
```

Prevođenje i povezivanje jednim pozivom

- Jednim pozivom prevodioca mogu se prevesti i povezati *svi* moduli koji čine jedan program

```
gcc -std=c11 -Wall -pedantic-errors -o prog.exe glavni.c mnozi.c zbroji.c
```

  - prevodilac će zasebno prevesti glavni.c, mnozi.c, zbroji.c i stvoriti:
    - datoteke s pretprocesiranim kodom
    - datoteke sa simboličkim kodom
    - datoteke s objektnim kodom
  - povezivač će povezati objektni kod s objektnim kodom iz standardne biblioteke, nastat će datoteka s izvršnim kodom prog.exe, a datoteke s pretprocesiranim, simboličkim i objektnim kodom će se obrisati

Odvojeni pozivi prevođenja i povezivanja

compile, but do not link

```
gcc -std=c11 -Wall -pedantic-errors -c glavni.c
```

```
gcc -std=c11 -Wall -pedantic-errors -c mnozi.c
```

```
gcc -std=c11 -Wall -pedantic-errors -c zbroji.c
```

- ili

```
gcc -std=c11 -Wall -pedantic-errors -c glavni.c mnozi.c zbroji.c
```

- zatim povezivanje

```
gcc -o prog.exe glavni.o mnozi.o zbroji.o
```

  - module iz standardne biblioteke najčešće nije potrebno navoditi u naredbi za povezivanje

- Problem: što ako se promijeni izvorni kod samo jednog ili samo nekoliko modula nekog vrlo velikog programa?

Prevoditi samo ono što je potrebno

- Zašto stalno ponavljanje prevođenja *svih* modula nije uvijek prihvatljivo?
  - nepotrebno trošenje vremena i resursa računala
    - Linux Kernel (2017.): ~ 25 000 000 linija koda u ~ 60 000 datoteka

- Kod većih programa prevođenje treba organizirati tako da se prevedu samo moduli
  - čiji se izvorni kod promijenio nakon posljednjeg prevođenja ili
  - čiji izvorni kod ovisi o izvornom kodu modula koji se promijenio
    - npr. ako se promijenio tip funkcije u jednom modulu, treba prevesti dotični modul i sve module koji koriste tu funkciju

  - potporu za organizaciju prevođenja na opisani način (ali ne samo to) pružaju različiti programski alati za automatsku izgradnju

Programski alati za automatsku izgradnju

- veći programi razvijaju se pomoću specijaliziranih softvera
  - integrirana razvojna okruženja (Eclipse, Visual Studio, NetBeans, GNU Emacs, VSCode, ...)
    - grafičko okruženje
    - upravljanje verzijama
    - pronalaženje pogrešaka (*debugging*)
    - automatsko prevođenje (samo onog što je potrebno), povezivanje, pokretanje testova
  - alati za automatsku izgradnju (make, Apache Ant, Apache Maven, Gradle, ...)

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **nema**
  - **17. vježbe uz predavanja**