Programiranje 2

Predavanje 03 - struktura programa

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

18. ožujka 2024.

Struktura C programa

C program je skup definicija:

- Objekata (varijabli)
- Funkcija

Varijable **zauzimaju memoriju** a funkcije **sadrže instrukcije**.
Funkcije komuniciraju preko **argumenata** i **povratnih vrijednosti**, **vanjskih ili globalnih varijabli** (varijable definirane izvan bilo koje funkcije).

Funkcije mogu biti u **proizvoljnom poretku** u izvornom programu, a program može biti smješten (rastavljen) u **više datoteka**.
Dijeljenje neke funkcije na više datoteka **nije dozvoljeno**.

Objekti u C programu mogu biti:

- Globalni ili vanjski (eng. external) - definirani izvan bilo koje funkcije
- Lokalni ili unutarnji (eng. internal) - definirani lokalno unutar neke funkcije

Struktura C programa

Funkcije u C-u su **uvijek globalne ili vanjske**. Nije dozvoljeno definirati funkcije unutar drugih funkcija. U Pascal-u je to moguće.

Kod **vanjskih objekata** svaka referenca na takav objekt s istim imenom se odnosi na taj konkretan objekt. Imena na vanjske objekte su **vanjski** simboli, a pripadni objekti su univerzalno dohvatljivi (čak i kada su funkcije koje ih dohvaćaju smještene u raznim datotekama). Univerzalna dohvatljivost se može ograničiti korištenjem ključne riječi `static`.

Unutarnji objekti nisu univerzalno dohvatljivi. Unutarnji objekti su argumenti funkcija i varijable definirane unutar funkcija.

Blokovska struktura jezika - lokalne varijable

Blok naredbi je **svaki niz** naredbi koji se nalazi unutar vitičastih zagrada (npr. tijelo funkcije). C dozvoljava da se u svakom bloku naredbi deklariraju varijable (lokalne varijable). Deklaracija varijabli unutar bloka mora prethoditi prvoj izvršnoj naredbi u bloku (standard C90). U standardu C99 je taj uvjet ukinut.

```
1  if (n > 0) {
2  int i; /* deklaracija varijable */
3  for (i = 0; i < n; ++i)
4  ...
5  } //C90
6
7  if (n > 0) {
8  for (int i = 0; i < n; ++i)
9  ...
10 } //C99
```

Blokovska struktura jezika - lokalne varijable

- Varijabla definirana unutar nekog bloka vidljiva je samo unutar tog bloka.
- Izvan bloka u kojem je definirana lokalna varijabla se toj varijabli ne može pristupiti (nije definirana). Na tom mjestu može biti deklarirana varijabla istog imena (nedostupna je unutar bloka jer je maskirana lokalnom varijablom).
- Varijabla deklarirana izvan bloka vidljiva je u tom bloku ako nije pokrivena lokalnom varijablom istog imena. Dostupna je najlokalnija varijabla istog imena.

Blokovska struktura jezika - lokalne varijable

```
1  int main(void) {
2  int x, y;
3  ...
4  if (x > 0){
5      double y; /* int y NIJE vidljiv u bloku */
6  /* int x JE vidljiv u bloku */
7  ...
8    }
9  }
```

Blokovska struktura jezika - lokalne varijable

```
1  int main(void) {
2  int x;
3  ...
4  if (x > 0){
5      double y;
6  ...
7    }
8  int y; //OK! double y nije vidljiv van bloka
9  }
```

Funkcije - doseg formalnog argumenta

Formalni argument funkcije vidljiv je unutar funkcije i nije dohvatljiv (niti definiran) izvan nje. Doseg (vidljivost) formalnog argumenta je isti kao i doseg lokalne varijable definirane na početku funkcije.

```
1  int x, y;
2  ...
3  void f(double x) {
4        double y; /* int x i int y NISU */
5  ... /* vidljivi unutar funkcije */
6  }
7
8  int main(void) { ... }
```

Atributi varijable

Varijabla je ime (sinonim) za sadržaj neke lokacije ili bloka lokacija u memoriji. Varijable imaju tri atributa: a) tip, b) doseg (engl. scope) i c) vijek trajanja (eng. lifetime).

- Tip - način interpretacije sadržaja bloka (piše/čita se u bitovima), uključuje i veličinu bloka.
- Vijek trajanja - u kojem dijelu memorije programa se rezervira taj blok. Postoje tri dijela memorije: statički, programski stog (eng. run-time stack) i programska hrpa (eng. heap).
- Doseg - u kojem dijelu programa je dio memorije dohvatljiv ili vidljiv za korištenje (čitanje, mijenjanje vrijednosti).

Atributi varijable

- Varijable prema tipu mogu biti: `int`, `char`, `double` itd.
- Prema dosegu, varijable mogu biti: **lokalne** (unutarnje) i **globalne** (vanjske).
- Prema vijeku trajanja, varijable mogu biti: **automatske**, **statičke** i **dinamičke**.

Doseg i vijek trajanja određeni su mjestom deklaracije/definicije objekta (varijable) unutar ili izvan neke funkcije. Upravljanje vijekom trajanja (a ponekad i dosegom) vrši se **identifikatorima memorijske klase**. Ključne riječi `auto`, `extern`, `register` (ukinuto u verziji `C11`, ali ključna riječ zadržana za buduću uporabu) i `static` kod deklaracije objekta označavaju njegovu memorijsku klasu.

Automatske varijable

Automatska varijabla je svaka varijabla kreirana **unutar nekog bloka** (unutar svake funkcije) koja nema ključnu riječ `static` u deklaraciji. Automatske varijable kreiraju se na ulasku u blok u kome su deklarirane i uništavaju se na izlasku iz tog bloka. Memorija koju su zauzimale oslobađa se tada za druge varijable (vrijednosti se gube). Ovaj postupak se odvija na programskom (izvršnom) stogu.

```
1 ...
2 void f(double x) {
3       double y = 2.71;
4       static double z;
5 ...
6 }
```

Automatske varijable su x (formalni argument), y. Varijabla z je **statička** varijabla.

Automatske varijable - inicijalizacija

Inicijalizacija (ukoliko postoji) se vrši prilikom svakog novog ulaza u blok u kojem je varijabla definirana (dogodi se rezervacija memorije i inicijalizacija). Automatska varijabla koja nije inicijalizirana na ulasku u blok u kojem je definirana dobiva nepredvidljivu vrijednost (rezervira memoriju bez promjene sadržaja memorijske lokacije).

Inicijalizaciju automatske varijable moguće je izvršiti:

- konstantnim izrazom
- nekonstantnim izrazom

```
1 void f(double x) {//inicijalizacija konstantnim
2 //izrazom
3       double y = 2.71; }
4
5 void f(double x) {//inicijalizacija nekonstantnim
6       double y = 2*x; } //izrazom
```

Identifikatori memorijske klase

Identifikatori memorijske klase (eng. storage class) su `auto`, `extern`, `register` i `static`. U tu skupinu se može staviti i `typedef` (ali se koristi u drugačije svrhe).

Identifikatori memorijske klase služe preciziranju vijeka trajanja varijable, te dijela memorije u kojem se nalazi. `extern` omogućava deklaraciju objekata koji se nalaze u drugim datotekama. `static` se koristi i za kontrolu dosega varijabli i funkcija (svrha ovisi o načinu korištenja).

Pišu se u deklaraciji varijable ili funkcije ispred identifikatora tipa varijable ili funkcije.

```
1 identif_mem_klase tip_var ime_var;
2 identif_mem_klase tip_fun ime_fun ... ;
```

Identifikatori memorijske klase

Primjeri:

```
1 auto int *pi;
2 extern double l;
3 register int z;
4 static char polje[10];
```

Identifikator `auto`

Identifikator `auto` deklarira automatsku varijablu. Međutim, sve varijable deklarirane unutar nekog bloka implicitno su automatske (ako nisu deklarirane `static`). Sve varijable deklarirane izvan svih blokova implicitno su `static`. Ključna riječ `auto` se obično ne koristi u programima iako je dopušteno.

Ekvivalentno je pisati:

```
{
    char c;
    int i,j,k;
    ...
}
```

```
{
    auto char c;
    auto int i,j,k;
    ...
}
```

Identifikator `register`

Identifikator memorijske klase `register` sugerira prevoditelju da varijablu smjesti u registar procesora. Navedenu preporuku prevoditelj ne mora izvršiti (i uglavnom je većina implementacija prevodioca ovu instrukciju ignorirala, stoga je ukinuta u verziji `C11`).

Ideja korištenja je bila skratiti vrijeme izvođenja programa koristeći brži pristup memoriji (registar procesora). Uglavnom se u registar stavljaju često korištene varijable (kontrolne varijable petlje, brojači). Identifikator `register` je primijenjiv samo na automatske varijable (unutar nekog bloka).

```
1 int f (register int m, register long n) {
2 register int i;
3 ...
4 }
```

Zabranjeno primijeniti **adresni operator** i **pokazivač** na `register` varijable.

Statičke varijable

Statička varijabla je varijabla definirana **izvan svih funkcija** ili varijabla deklarirana u nekom bloku (npr. tijelu funkcije) identifikatorom memorijske klase `static`.

Statičke varijable su aktivne tijekom cijelog izvršavanja programa. Kreiraju se na početku izvršavanja programa i uništavaju tek na završetku programa. Statičke varijable je moguće eksplicitno inicijalizirati konstantnim izrazima. Neinicijalizirane statičke varijable prevoditelj inicijalizira na nulu (svi bitovi su jednaki 0).

```
1 int f(int j)
2 {
3 static int i = j; /* greska */
4 ...
5 }
```

Gornji kod nije ispravan jer statička varijabla nije inicijalizirana konstantnim izrazom.

Statičke varijable

Statička varijabla deklarirana unutar nekog bloka inicijalizira se **samo** pri prvom ulazu u blok, međutim zadržava svoju vijednost pri izlasku iz bloka (iako više nije dohvatljiva).

```
1 void foo() {
2       int a = 10; static int sa = 10;
3       a += 5; sa += 5;
4       printf("a = %d, sa = %d\n", a, sa);
5 return;
6 }
```

Varijabla *a* je automatska i inicijalizira se prilikom svakog ulaska u funkciju. *sa* je statička varijabla i inicijalizira se prilikom prvog ulaska u funkciju. *sa* zadržava vrijednost nakon izlaska iz funkcije.

Statičke varijable

```
1  int main(void) {
2      int i;
3
4      for (i = 0; i < 3; ++i)
5          foo();
6  return 0;
7  }
8
9  Izlaz je:
10 a = 15, sa = 15
11 a = 15, sa = 20
12 a = 15, sa = 25
```

Doseg varijable

Doseg varijable je područje programa u kojem je varijabla dostupna (vidljiva). Prema dosegu, varijable se dijele na:

- lokalne (imaju doseg bloka)
- globalne (imaju doseg datoteke)

Svaka varijabla definirana unutar nekog bloka je lokalna za taj blok. Lokalne varijable (uključujući statičke) koje su definirane unutar bloka, nisu definirane izvan tog bloka. Statička lokalna varijabla postoji za vrijeme cijelog izvršavanja programa, ali se može dohvatiti samo iz bloka u kojem je deklarirana.

Globalne varijable

Globalna varijabla je varijabla definirana izvan svih funkcija. Globalna varijabla (deklarirana izvan svih blokova) vidljiva je od mjesta deklaracije do kraja datoteke ako nije prekrivena varijablom istog imena unutar nekog bloka.

Uobičajeno se globalne varijable deklariraju na početku datoteke, prije svih funkcija. Svaka funkcija može koristiti takvu globalnu varijablu i promijeniti joj vrijednost. Više funkcija može komunicirati koristeći globalne varijable bez upotrebe formalni argumenata.

Primjer: brojači poziva kod rekurzivnih funkcija.

Broj particija - globalni brojač

```
1  #include <stdio.h>
2  int broj = 0; /* globalni brojac */
3
4  void particije(int suma, int prvi){
5  int i;
6
7  if (suma == 0)
8        ++broj;
9  else
10       for (i = prvi; i <= suma; ++i)
11             particije(suma - i, i);
12 return;
13 }
```

Broj particija - main

```
1  int main(void){
2
3  int n;
4
5  printf("␣Upisi␣prirodni␣broj␣n:␣");
6  scanf("%d", &n);
7  particije(n, 1);
8  printf("\n␣Broj␣part.␣p(%d)␣=␣%d\n", n, broj);
9  return 0;
10 }
```

Globalne varijable

```
1  int a; /* static */
2
3  void f(int);
4
5  int main(void) {
6  int c, d; /* auto */
7  ...
8  }
9
10 int b; /* static */
11 void f(int i) {
12 int x, y; /* auto */
13 ...
14 }
```

Varijabla *a* vidljiva je i u funkciji `main` i u funkciji *f*, dok je varijabla *b* vidljiva u funkciji *f*, ali ne i u funkciji `main`.

Program smješten u više datoteka

C program može biti smješten u više datoteka. Npr. svaka funkcija definirana u programu može biti smještena u zasebnu `.c` datoteku. Globalne varijable i funkcije definirane u jednoj datoteci mogu se koristiti i u bilo kojoj drugoj datoteci (uz korektnu deklaraciju u drugoj datoteci).

Objekt koji je definiran u nekoj drugoj datoteci se deklarira koristeći ključnu riječ `extern` (vanjski). `extern` ispred deklaracije objekta (varijable ili funkcije) informira prevoditelj da se radi o objektu koji je definiran u nekoj drugoj datoteci.

Program smješten u više datoteka

```
1  //Datoteka 1
2  #include <stdio.h>
3
4  int g(int);
5
6  void f(int i) {
7        printf("i = %d\n", g(i));
8  }
9
10 int g(int i) {
11       return 2 * i - 1;
12 }
```

Program smješten u više datoteka

```
1 //Datoteka 2
2 extern void f(int); /* extern i prototip. */
3
4 int main(void) {
5     f(3);
6 return 0;
7 }
```

Nakon prevođenja i povezivanja obje datoteke, izvršavanje daje rezultat `i = 5`.

Vanjski simboli

U programu smještenom u više datoteka sve funkcije i globalne varijable mogu se koristiti i u drugim datotekama (ako su deklarirane u tim datotekama). Zato su imena funkcija i globalnih varijabli **vanjski simboli** (javni). Povezivanje deklaracija vanjskih simbola s njihovim definicijama radi linker (povezivač).

Deklariranjem funkcije ili globalne varijable kao `static` **ograničimo doseg** funkcije ili varijable (ona se može dohvatiti samo iz datoteke u kojoj je definirana, postaje privatan simbol).

Vanjski simboli

```c
#include <stdio.h>
static int g(int); /* static ogranicava doseg. */

void f(int i) {
    printf("i = %d\n", g(i));
}

int g(int i) {
     return 2 * i - 1;
}
```

Vanjski simboli

```c
#include <stdio.h>

extern void f(int); /* extern i prototip. */
extern int g(int); /* Nije vanjski simbol! */

int main(void) {
f(3); /* Ispravno. */
printf("g(2) = %d\n", g(2)); /* Neispravno. */
return 0;
}
```

Vanjski simboli

Korištenje globalnih varijabli i funkcija u jednoj datoteci i korištenje u drugoj.

```
1  #include <stdio.h> //datoteka 1
2  int z = 3; /* Definicija varijable z. */
3  void f(int i) /* Definicija funkcije f. */
4  {
5      printf("i = %d\n", i); }
6
7  //druga datoteka
8  extern void f(int); /* Deklaracija funkcije f.*/
9  extern int z; /* Deklaracija varijable z.*/
10
11 int main(void) {
12     f(z);
13 return 0; }
```

Definicija i deklaracija globalnih varijabli

Kod globalnih varijabli razlikujemo **definiciju** varijable i **deklaraciju** varijable (slično i kod funkcija).

U definiciji varijable deklarira se njezino ime i tip te se rezervira memorijska lokacija za varijablu. Kod deklaracije se samo deklarira ime i tip bez rezervacije memorije. Za deklarirane varijable se podrazumijeva da je varijabla definirana negdje drugdje i da joj je tamo pridružena memorijska lokacija.

Definicija varijable je uvijek i njezina deklaracija. Globalne varijable mogu imati više deklaracija, ali samo jednu definiciju.

Pravila kod definicije i deklaracije

- Globalne (vanjske) varijable dobivaju prostor u **statičkom** dijelu memorije programa kao i **statičke** varijable.
- Pravila inicijalizacije **su ista**. Globalna varijabla može biti inicijalizirana konstantnim izrazom (kod definicije). Globalne varijable koje nisu eksplicitno inicijalizirane, inicijaliziraju se automatski nulom.
- Kod deklaracije globalne varijable mora se koristiti ključna riječ `extern`, a inicijalizacija nije moguća.
- Kod definicije globalnog polja, mora biti definirana njegova dimenzija (zbog rezervacije memorije). Kod deklaracije se ne mora navoditi dimenzija.

Sužavanje dosega globalnih varijabli `static`

Oznaka memorijske klase `static` može se primijeniti i na globalne varijable, s istim djelovanjem kao i za funkcije. `static` sužava doseg (područje vidljivosti) varijable na datoteku u kojoj je definirana. Ime takve varijable više nije dohvatljivo kao vanjski simbol (postaje privatna).

Oznaka memorijske klase `static` ispred globalne i lokalne varijable ima različito značenje:

```
1 static int z = 3; /*z nevidljiv izvan datoteke.*/
2
3 void f(int i) {
4       static double x;
5       /* x je staticka varijabla. */
6 ... }
```

Datoteke zaglavlja

Kad se program sastoji od više datoteka, grupe deklaracija vanjskih simbola (varijabli i funkcija) smještaju se u posebnu datoteku zaglavlja (`*.h`), koja se uključuje s `#include "*.h"` u svaku `*.c` datoteku u kojoj su te deklaracije potrebne. Na taj se način osigurava konzistentnost svih deklaracija.

Primjer deklaracije vanjskih simbola u `.h` datoteci:

```
1 extern void f(int); /* extern NE treba pisati! */
2 extern int g(int);
3 extern int z;
```

Datoteke zaglavlja

```c
//datoteka 1
#include <stdio.h>
#include "dekl.h"

int z = 3; /* Definicija varijable z. */
void f(int i) /* Definicija funkcije f. */
{
    printf("i = %d\n", g(i));
}

int g(int i) /* Definicija funkcije g. */
{
    return 2 * i - 1;
}
```

Datoteke zaglavlja

```c
//datoteka 2
#include <stdio.h>
#include "dekl.h"
int main(void)
{
    f(z);
    printf("g(2) = %d\n", g(2));
return 0;
}
```

U datotekama zaglavlja `*.h` implicitno se sve deklaracije tretiraju kao `extern`. Podrazumijeva se da su svi objekti definirani negdje drugdje. Programe s više datoteka možemo u `CodeBlocks`-u prevoditi koristeći **projekte**.