Programiranje 2

Milena Vujošević Janičić
Jelena Graovac

www.matf.bg.ac.rs/~milena
www.matf.bg.ac.rs/~jgraovac

Beograd, 13. februar, 2020.

Pregled

Pregled

Organizacija izvornog koda

- Problemi koje ste do sada rešavali obuhvatali su male i jednostavne programe koji su bili precizno definisani
- Za organizovanje složenih programa neophodno je
  - napraviti funkcionalnu dekompoziciju problema, tj. podelu složenih zadataka na jednostavnije poslove i izdvajanje u zasebne funkcije
  - definisanje odgovarajućih tipova podataka i organizovanje podataka definisanjem odgovarajućih promenljivih.
- Primer: matematički softver

Doseg, životni vek i povezanost

- Postoj tri važna pojma: doseg, životni vek i povezanost
- Odnos između dosega, životnog veka i povezanosti je veoma suptilan i sva ova tri aspekta objekata određuju se na osnovu mesta i načina deklarisanja tj. definisanja objekata, ali i primenom kvalifikatora `auto`, `register`, `static` i `extern`

Doseg identifikatora

- Podelu koda po datotekama omogućavaju različiti mehanizmi
- *Doseg identifikatora* (engl. scope) određuje da li se neka imena mogu koristiti u čitavim jedinicama prevođenja ili samo u njihovim manjim delovima (najčešće funkcijama ili još užim blokovima).
- Postojanje promenljivih čija je upotreba ograničena na samo određene uske delove izvornog kôda olakšava razumevanje programa i smanjuje mogućnost grešaka i smanjuje međusobnu zavisnost između raznih delova programa.

Doseg identifikatora

- *doseg nivoa datoteke* (engl. file level scope) koji podrazumeva da ime važi od tačke uvođenja do kraja datoteke;
- *doseg nivoa bloka* (engl. block level scope) koji podrazumeva da ime važi od tačke uvođenja do kraja bloka u kojem je uvedeno;
- *doseg nivoa funkcije* (engl. function level scope) koji podrazumeva da ime važi u celoj funkciji u kojoj je uvedeno; ovaj doseg imaju jedino labele koje se koriste uz `goto` naredbu;
- *doseg nivoa prototipa funkcije* (engl. function prototype scope) koji podrazumeva da ime važi u okviru prototipa (deklaracije) funkcije.

Doseg identifikatora

- Najznačajni nivoi dosega su doseg nivoa datoteke i doseg nivoa bloka.
- Identifikatori koji imaju doseg nivoa datoteke najčešće se nazivaju *spoljašnji* ili *globalni*
- Identifikatori koji imaju ostale nivoe dosega (najčešće doseg nivoa bloka) nazivaju *unutrašnji* ili *lokalni*

Primer

```
int a; /* a je globalna promenljiva - doseg nivoa datoteke */

void f(int c) { /* f je globalna funkcija - doseg nivoa datoteke */
          /* c je lokalna promenljiva - doseg nivoa bloka (tela funkcije f) */
  int d;  /* d je lokalna promenljiva - doseg nivoa bloka (tela funkcije f) */

  void g() { printf("zdravo"); }
  /* g je lokalna funkcija - doseg nivoa bloka (tela funkcije f) */

  for (d = 0; d < 3; d++) {
     int e; /* e je lokalna promenljiva - doseg nivoa bloka (tela petlje) */
     ...
  }
  kraj:   /* labela kraj - doseg nivoa funkcije */
}

/* h je globalna funkcija - doseg nivoa datoteke */
void h(int b); /* b - doseg nivoa prototipa funkcije */
```

Primer

```
void f() {
   int a = 3, i;
   for (i = 0; i < 4; i++) {
     int a = 5;
     printf("%d ", a);
   }
}
```

Životni vek promenljivih

- U određenoj vezi sa dosegom, ali ipak nezavisno od njega je pitanje trajanja objekata (pre svega promenljivih).
- *Životni vek (engl. storage duration, lifetime)* promenljive je deo vremena izvršavanja programa u kojem se garantuje da je za tu promenljivu rezervisan deo memorije i da se ta promenljiva može koristiti.
- Postojanje promenljivih koje traju samo pojedinačno izvršavanje neke funkcije znatno štedi memoriju, dok postojanje promenljivih koje traju čitavo izvršavanje programa omogućava da se preko njih vrši komunikacija između različitih funkcija modula.

Životni vek promenljivih

- U jeziku C postoje sledeće vrste životnog veka:
  - *statički (engl. static)* životni vek koji znači da je objekat dostupan tokom celog izvršavanja programa;
  - *automatski (engl. automatic)* životni vek koji najčešće imaju promenljive koje se automatski stvaraju i uklanjaju prilikom pozivanja funkcija;
  - *dinamički (engl. dynamic)* životni vek koji imaju promenljive koje se alociraju i dealociraju na eksplicitan zahtev programera.
- Životni vek nekog objekta se određuje na osnovu pozicije u kôdu na kojoj je objekat uveden i na osnovu eksplicitnog korišćenja nekog od kvalifikatora `auto` (automatski životni vek) ili `static` (statički životni vek).

Životni vek promenljivih

- Lokalne promenljive podrazumevano su *automatskog* životnog veka (sem ako je na njih primenjen kvalifikator `static` ili kvalifikator `extern`.
- Iako je moguće i eksplicitno okarakterisati životni vek korišćenjem kvalifikatora `auto`, to se obično ne radi jer se podrazumeva.
- Početna vrednost lokalnih automatskih promenljivih nije određena.

Životni vek promenljivih

- Automatske promenljive „postoje“ samo tokom izvršavanja funkcije u kojoj su deklarisane
- Formalni parametri funkcija, tj. promenljive koje prihvataju argumente funkcije imaju isti status kao i lokalne automatske promenljive.
- Na lokalne automatske promenljive i parametre funkcija moguće je primeniti i kvalifikator `register`

Statički životni vek

- Globalne promenljive deklarisane su van svih funkcija i njihov doseg je nivoa datoteke tj. mogu se koristiti od tačke uvođenja, u svim funkcijama do kraja datoteke.
- Životni vek ovih promenljivih uvek je *statički*, tj. prostor za ove promenljive rezervisan je tokom celog izvršavanja programa: prostor za njih se rezerviše na početku izvršavanja programa i oslobađa onda kada se završi izvršavanje programa.
- Prostor za ove promenljive obezbeđuje se u segmentu podataka.
- Ovakve promenljive se podrazumevano inicijalizuju na vrednost 0 (ukoliko se ne izvrši eksplicitna inicijalizacija).

Lokalne statičke promenljive

- U nekim slučajevima poželjno je čuvati informaciju između različitih poziva funkcije (npr. potrebno je brojati koliko puta je pozvana neka funkcija).
- Jedno rešenje bilo bi uvođenje globalne promenljive, statičkog životnog veka, međutim, zbog globalnog dosega tu promenljivu bilo bi moguće koristiti (i promeniti) i iz drugih funkcija, što je nepoželjno.
- Zato je poželjna mogućnost definisanja promenljivih koje bi bile statičkog životnog veka (da bi čuvale vrednost tokom čitavog izvršavanja programa), ali lokalnog dosega (da bi se mogle koristiti i menjati samo u jednoj funkciji).

Lokalne statičke promenljive

- U deklaraciji lokalne promenljive može se primeniti kvalifikator `static` i u tom slučaju ona ima statički životni vek — kreira se na početku izvršavanja programa i oslobađa prilikom završetka rada programa.
- Tako modifikovana promenljiva ne čuva se u stek okviru svoje funkcije, već u segmentu podataka.

Lokalne statičke promenljive

- Ukoliko se vrednost statičke lokalne promenljive promeni tokom izvršavanja funkcije, ta vrednost ostaje sačuvana i za sledeći poziv te funkcije.
- Ukoliko inicijalna vrednost statičke promenljive nije navedena, podrazumeva se vrednost `0`.
- Statičke promenljive se inicijalizuju samo jednom, konstantnim izrazom, na početku izvršavanja programa tj. prilikom njegovog učitavanja u memoriju.
- Doseg ovih promenljivih i dalje je nivoa bloka tj. promenljive su i dalje lokalne, što daje željene osobine.

Primer

```
#include <stdio.h>

void f() {
  int a = 0;
  printf("f: %d ", a);
  a = a + 1;
}

void g() {
  static int a = 0;
  printf("g: %d ", a);
  a = a + 1;
}

int main() {
  f(); f(); g(); g();
  return 0;
}
```

Povezanost identifikatora

- *Povezanost identifikatora* (engl. linkage) u vezi je sa deljenjem podataka između različitih jedinica prevođenja i daje mogućnost korišćenja zajedničkih promenljivih i funkcija u različitim jedinicama prevođenja (modulima), ali i mogućnost sakrivanja nekih promenljivih tako da im se ne može pristupiti iz drugih jedinica prevođenja.

Povezanost

Jezik C razlikuje identifikatore:

- *bez povezanosti* (engl. no linkage),
- identifikatore sa *spoljašnjom povezanošću* (engl. external linkage) i
- identifikatore sa *unutrašnjom povezanošću* (engl. internal linkage).

Pregled

Literatura

- Slajdovi su pripremljeni na osnovu knjiga
  Filip Marić, Predrag Janičić: Programiranje 1
  Predrag Janičić, Filip Marić: Programiranje 2
- Za pripremu ispita, slajdovi nisu dovoljni, neophodno je učiti iz knjige!