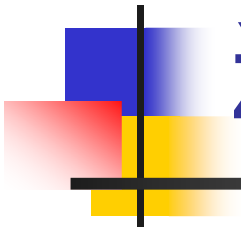
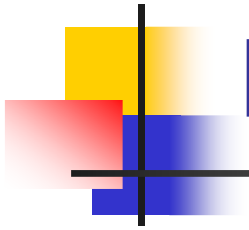


Programski jezik C – doseg, životni vek i povezanost

Programiranje 1,

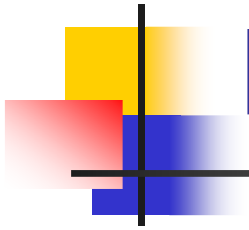
9.2.2-9.2.4



Doseg, životni vek i povezanost

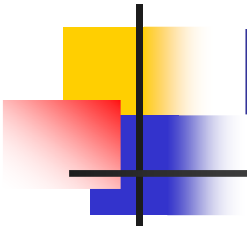
- Doseg
- Životni vek
- Povezanost

- Zavise od
  - mesta deklarisanja / definisanja objekta
  - načina deklarisanja / definisanja
    - kvalifikatora auto, register, static, extern



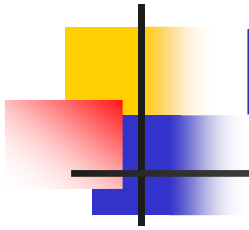
Doseg identifikatora (engl. *scope*)

- Doseg identifikatora: da li se neka imena mogu koristiti u čitavim jedinicama prevođenja ili samo u njihovim manjim delovima (najčešće funkcijama ili blokovima)
- Smanjuje se međusobna zavisnost između raznih delova programa



Doseg identifikatora

- Doseg **nivoa datoteke** (engl. *file level scope*): ime važi od tačke uvođenja do kraja datoteke;
- Doseg **nivoa bloka** (engl. *block level scope*): ime važi od tačke uvođenja do kraja bloka u kojem je uvedeno;
- Doseg **nivoa funkcije** (engl. *function level scope*): 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*): ime važi u okviru prototipa (deklaracije) funkcije;

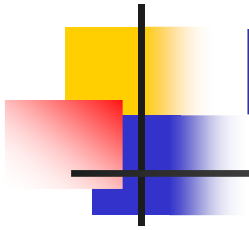


Doseg identifikatora

- Najznačajni nivoi dosega su **doseg nivoa datoteke** i **doseg nivoa bloka**
- Identifikatori koji imaju doseg nivoa datoteke najčešće se nazivaju ***spoljašnjim*** ili ***globalnim***
- Identifikatori koji imaju ostale nivoe dosega (najčešće doseg nivoa bloka) nazivaju se ***unutrašnjim*** ili ***lokalnim***

Doseg identifikatora – 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 */
}
```



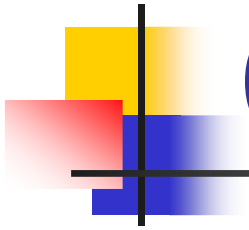
Doseg identifikatora - primer

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

Konflikt identifikatora - različit doseg

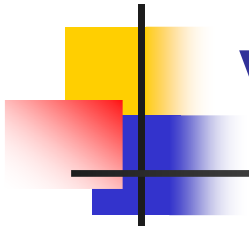
```
void f() {
   int a = 3, i;
   for (i = 0; i < 4; i++) {
     int a = 5;
     printf("%d ", a);
   }
}
```

Životni vek promenljivih (lifetime)



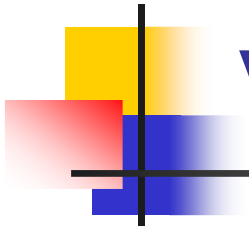
- Trajanje objekta (promenljive) – memorija i korišćenje
- U vezi ali nezavisan od dosega
- Trajanje samo pojedinačnog izvršavanja neke funkcije / trajanje čitavog izvršavanja programa
- Ušteda memorije / komunikacija između modula

Životni vek promenljivih - vrste



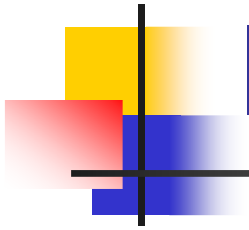
- ***statički*** (engl. *static*) životni vek – objekat je dostupan tokom celog izvršavanja programa;
- ***automatski*** (engl. *automatic*) životni vek - promenljive koje se automatski stvaraju i uklanjaju prilikom pozivanja funkcija;
- ***dinamički*** (engl. *dynamic*) životni vek - promenljive koje se alociraju i dealociraju na eksplicitan zahtev programera

Životni vek promenljivih - vrste



- Životni vek se određuje na osnovu
  - pozicije u programu 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

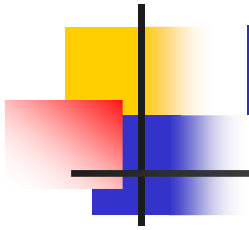


- Lokalne promenljive podrazumevano su automatskog životnog veka (osim ako je na njih primenjen kvalifikator static ili extern)
- Može i eksplicitno: auto (obično se ne radi)
- Početna vrednost lokalnih automatskih promenljivih nije određena

Životni vek promenljivih – lokalne promenljive - primer

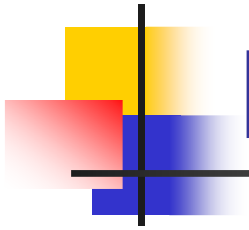
```
#include <stdio.h>
void f() {
            int a = 0;
            printf("f: %d ", a);
            a = a + 1;
}
void g() {
             int a = 0;
              printf("g: %d ", a);
              a = a + 1;
}
int main() {
  f(); f(); g(); g();    return 0;}
```

Životni vek promenljivih – lokalne promenljive

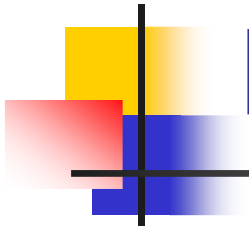


- Automatske promenljive „postoje" samo tokom izvršavanja funkcije u kojoj su deklarisane
- Formalni parametri funkcija kao i lokalne automatske promenljive
- Na lokalne automatske promenljive i parametre funkcija moguće je primeniti i kvalifikator *register*

Statički životni vek – globalne promenljive

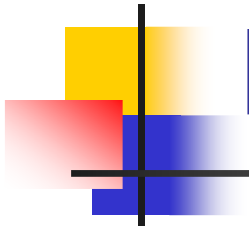


- Globalne promenljive deklarisane su van svih funkcija i njihov doseg je nivoa datoteke (od pozicije uvođenja do kraja datoteke)
- Životni vek globalnih promenljivih je statički
- Prostor rezervisan tokom celog izvršavanja programa
- Prostor se rezerviše na početku izvršavanja programa i oslobađa po završetku izvršavanja programa
- Prostor se obezbeđuje u tzv. *segmentu podataka*
- Podrazumevano se inicijalizuju na vrednost 0



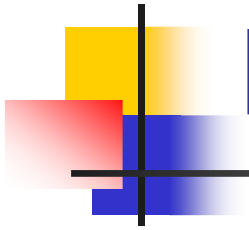
Lokalne statičke promenljive

- Nekad je poželjno da se informacija čuva između različitih poziva funkcije (primer – broj poziva funkcije)
- 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
- Drugo (poželjnije) rešenje je definisanje promenljivih
  - Lokalnog dosega
  - Statičkog životnog veka (da bi čuvale vrednost tokom čitavog izvršavanja programa)



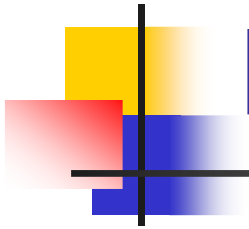
Lokalne statičke promenljive

- U deklaraciji lokalne promenljive može da se primeni kvalifikator static
- Statički životni vek - kreira se na početku izvršavanja programa i oslobađa prilikom završetka rada programa
- Čuva se takođe 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
- Statička promenljiva se implicitno iicijalizuje na 0
- Statičke promenljive se eksplicitno inicijalizuju samo jednom, konstantnim izrazom, na početku izvršavanja programa
- Doseg ovih promenljivih i dalje je nivoa bloka tj. promenljive su i dalje lokalne



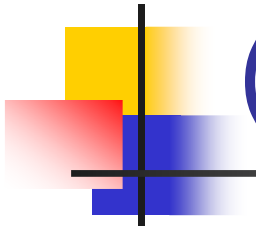
Primer 1

```
#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;
}
```

Primer 2

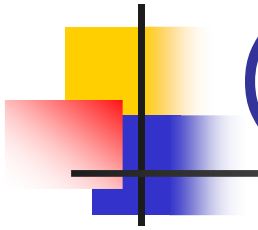
```
#include <stdio.h>

void f() {
        int a = 1;
        static int b = 2;
        printf("%d %d\n", --a, b++);
}
int main() {
        f(); f(); f();
        return 0;
}
```

Povezanost identifikatora (engl. linkage)



- Daje mogućnost korišćenja zajedničkih promenljivih i funkcija u različitim jedinicama prevođenja (modulima)
- Daje mogućnost sakrivanja nekih promenljivih tako da im se ne može pristupiti iz drugih jedinica prevođenja

Povezanost identifikatora (engl. linkage)



- Jezik C razlikuje identifikatore:
  - bez povezanosti (engl. no linkage),
  - sa spoljašnjom povezanošću (engl. external linkage) i
  - sa unutrašnjom povezanošću (engl. internal linkage)

- Objekti automatskog životnog veka –bez povezanosti
- Objekti statičkog životnog veka mogu biti svih vrsta - bez povezanosti, sa unutrašnjom iii spoljašnjom povezanošću

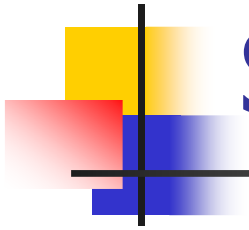
Bez povezanosti

- Bez povezanosti su najčešće
  - lokalne promenljive (bez obzira da li su automatskog ili statičkog životnog veka),
  - parametri funkcija,
  - korisnički definisani tipovi,
  - labele itd.

```
int f() {
    int i;
    ...
 }
```

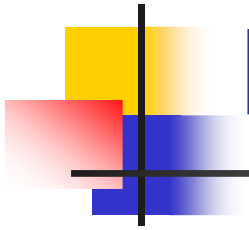
```
int g1(int i) {
        static int j;  ...
}
int g2() {
        static int i, j;
        ...
}
```

```
struct i { int a; }
```



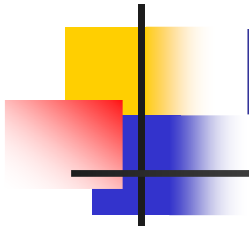
Spoljašnja povezanost

- Spoljašnja povezanost identifikatora omogućava da se isti objekat koristi u više jedinica prevođenja
- Sve deklaracije identifikatora sa spoljašnjom povezanošću u skupu jedinica prevođenja određuju jedan isti objekat
- U celom programu mora da postoji tačno jedna definicija tog objekta



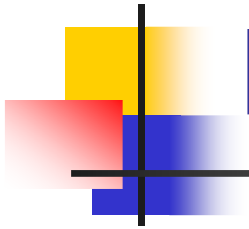
Extern

- Kvalifikator extern se koristi isključivo kod programa koji se sastoje od više jedinica prevođenja i služi da naglasi da neki identifikator ima spoljašnju povezanost
- Jedino deklaracije mogu biti okvalifikovane kvalifikatorom extern
- Nije moguće navoditi inicijalizaciju promenljivih niti telo funkcije (što ima smisla samo prilikom definisanja)
- Pošto extern deklaracije nisu definicije, njima se ne rezerviše nikakva memorija u programu već se samo naglašava da se očekuje da negde (najčešće u drugim jedinicama prevođenja) postoji definicija objekta koji se extern deklaracijom deklariše (i time uvodi u odgovarajući doseg)



Extern

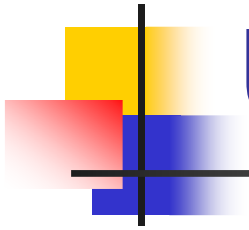
- Postoje slučajevi kada se spoljašnja povezanost podrazumeva i nije neophodno eksplicitno upotrebiti extern deklaraciju
- Sve globalne funkcije i promenljive jezika C podrazumevano imaju spoljašnju povezanost (osim ako na njih nije primenjen kvalifikator static kada je povezanost unutrašnja)



Primer

```
#include <stdio.h>
int a;
int b = 3;


void f() {
   printf("a=%d\n", a);
}
```

```
#include <stdio.h>
extern int a;
void f();

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

int main() {
   a = 1;
   /* b = 2; */
   g();
   return 0;
}
```



Unutrašnja povezanost

- Globalni objekat ima unutrašnju povezanost ako se kvalifikuje kvalifikatorom *static*
- Može da se koristi samo u toj jedinici prevođenja
- Osnovna uloga unutrašnje povezanosti obično je u tome da neki deo koda učini zatvorenim, u smislu da je nemoguće menjanje nekih njegovih globalnih promenljivih ili pozivanje nekih njegovih funkcija iz drugih datoteka

Unutrašnja povezanost

```
#include <stdio.h>
static int a = 3;
int b = 5;

static int f() {
  printf("f: a=%d, b=%d\n",
         a, b);
}

int g() {
  f();
}
```

```
#include <stdio.h>

int g(); /* int f(); */
int a, b;

int main() {
     printf("main: a=%d, b=%d\n",
     a, b);
     g(); /* f() */
     return 0;
}
```