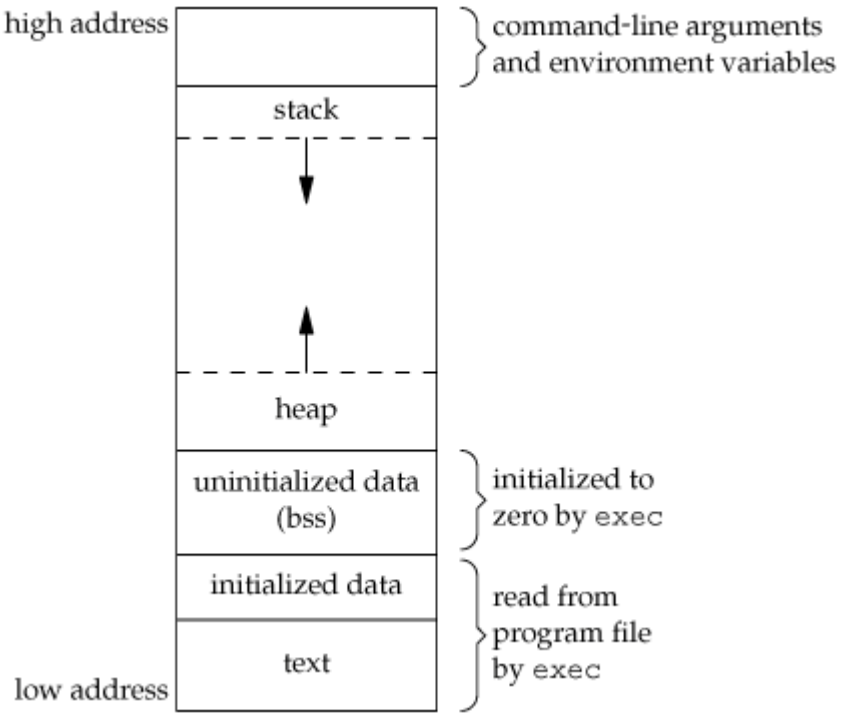
Organizacije memorije dodeljene programu



Kada operativni sistem učita izvršni program, dodeljuje mu određenu memoriju i započinje njegovo izvršavanje. Dodeljena memorija organizovana je u nekoliko delova koje zovemo *segmenti* ili *zone*:

- segment koda (eng. code segment ili text segment)
- segment podataka (eng. data segment)
- stek segment (eng. stack segment)
- hip segment (eng. heap segment)

Segment koda

U ovom segmentu se nalazi sâm izvršni kôd programa — njegov mašinski kôd koji uključuje sve korisnikove funkcije a može da uključuje i korišćene sistemske funkcije. Na nekim operativnim sistemima, ukoliko je pokrenuto više instanci istog programa, onda sve te instance dele isti prostor za izvršni kôd, tj. u memoriji postoji samo jedan primerak koda. U tom slučaju, za svaku instancu se, naravno, zasebno čuva informacija o tome do koje naredbe je stiglo izračunavanje.

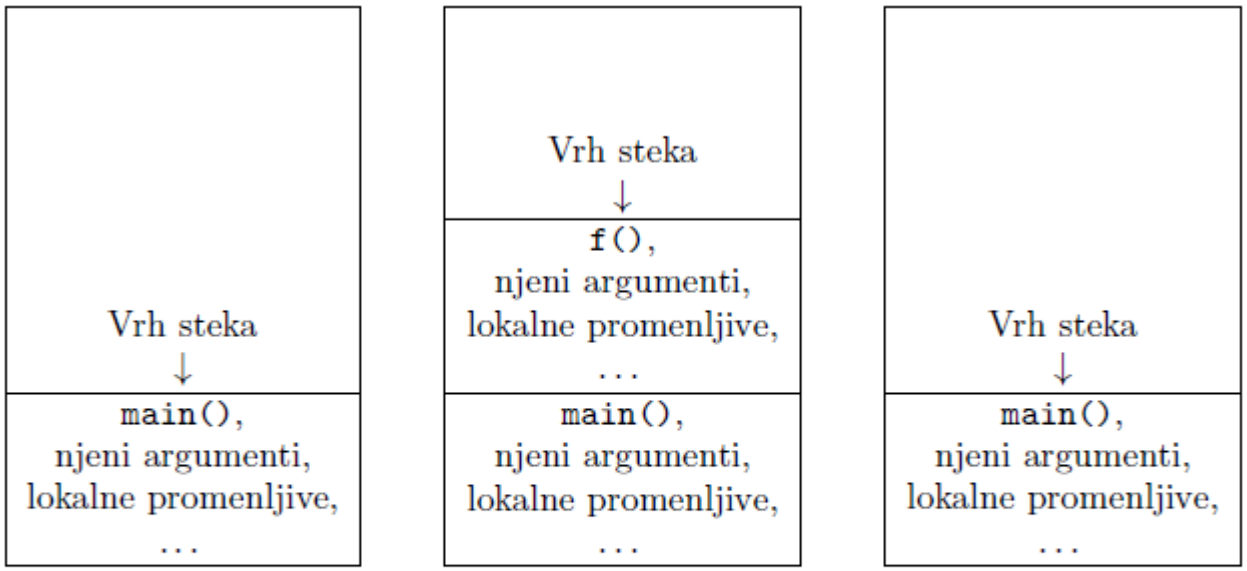
Segment podataka

U data segmentu čuvaju se određene vrste promenljivih koje su zajedničke za ceo program (tzv. statičke i spoljašnje promenljive), kao i konstantne niske. Ukoliko se istovremeno izvršava više instanci istog programa, svaka instanca ima svoj zaseban segment podataka.

Stek segment

U stek segmentu (koji se naziva i *programski stek poziva (eng. call stack)*) čuvaju se svi podaci koji karakterišu izvršavanje funkcija. Podaci koji odgovaraju jednoj funkciji (ili, preciznije, jednoj instance jedne funkcije — jer, na primer, jedna funkcija može da poziva samu sebe i da tako u jednom trenutku bude aktivno više njenih instanci) organizovani su u takozvani *stek okvir (eng. stack frame)*. Stek okvir jedne instance funkcije obično, između ostalog, sadrži:

- argumente funkcije;
- lokalne promenljive (promenljive deklarisane unutar funkcije);
- međurezultate izračunavanja;
- adresu povratka (koja ukazuje na to odakle treba nastaviti izvršavanje programa nakon povratka iz funkcije);
- adresu stek okvira funkcije pozivaoca.



Organizacija steka i ilustracija izvršavanja funkcije. Levo: tokom izvršavanja funkcije `main()`. Sredina: tokom izvršavanja funkcije `f()` neposredno pozvane iz funkcije `main()`. Desno: nakon povratka iz funkcije `f()` nazad u funkciju `main()`.

Doseg i životni vek identifikatora

Doseg identifikatora (eng. scope) određuje deo teksta programa u kome je moguće koristiti određeni identifikator i u kome taj identifikator identifikuje određeni objekat (na primer, promenljivu ili funkciju). Jezik C spada u grupu jezika sa statičkim pravilima dosega što znači da se doseg svakog identifikatora može jednoznačno utvrditi analizom izvornog koda, bez obzira na to kako teče proces izvršavanja programa. U programskom jeziku C, razlikuje se *doseg nivoa datoteke* (eng. file level scope) koji podrazumeva da ime važi od tačke uvođenja do kraja datoteke, *doseg nivoa bloka* (eng. block level scope) koji podrazumeva da ime važi od tačke uvođenja do kraja bloka u kome je uvedeno, *doseg nivoa funkcije* (eng. function level scope) koji podrazumeva da ime važi od tačke uvođenja do kraja funkcije u kojoj je uvedeno i *doseg nivoa prototipa funkcije* (eng. function prototype scope) koji podrazumeva da ime važi u okviru prototipa (deklaracije) funkcije. Prva dva nivoa dosega su najznačajnija. Identifikatori koji imaju doseg nivoa datoteke najčešće se nazivaju *spoljašnji* ili *globalni*, dok se identifikatori koji imaju ostale nivoe dosega (najčešće doseg nivoa bloka) nazivaju *unutrašnji* ili *lokalni*. Doseg nivoa funkcije imaju samo labele (koje se najčešće koriste uz `goto` naredbu), dok doseg nivoa prototipa funkcije imaju samo imena parametara u okviru prototipova funkcije. U ranijim verzijama jezika C nije bilo moguće definisati funkciju u okviru definicije druge funkcije (te nema lokalnih funkcija), dok standard C99 uvodi i takvu mogućnost.

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

/* f je globalna funkcija - doseg nivoa datoteke */
void f(int c) {
  /* 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 */
```

Jezik C dopušta tzv. konflikt identifikatora tj. moguće je da postoji više identifikatora istog imena, pri čemu su njihovi dosezi jedan u okviru drugog i identifikator u užoj oblasti dosega sakriva identifikator u široj oblasti dosega. Na primer, u narednom programu, promenljiva a inicijalizovana na vrednost 5 sakriva promenljivu a inicijalizovanu na vrednost 3.

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

```
5 5 5 5
```

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 oslobađaju prilikom završetka rada programa. Tako modifikovana promenljiva ne čuva se u stek okviru svoje funkcije, već u segmentu podataka. 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 rada programa[1]. Doseg ovih promenljivih je i dalje doseg nivoa bloka tj. promenljive su i dalje lokalne.

```
#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;
}
```

Kada se prevede i pokrene, prethodni program ispisuje:

```
f: 0 f: 0 g: 0 g: 1
```

Globalne statičke promenljive i funkcije

Globalne promenljive su promenljive deklarisane van svih funkcija i njihov doseg je doseg nivoa datoteke što znači da su dostupne od tačke deklaracije do kraja izvorne datoteke. Globalne promenljive mogu da budu korišćene u svim funkcijama koje su u njihovom dosegu. Životni vek ovih promenljivih je statički, tj. prostor za ove promenljive je trajanje 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).

Primeri

1. Primer koji ilustruje vidljivost promenjivih. Šta je rezultat rada?

```
#include <stdio.h>
int i=10;
void main()
{

      {  int i=3;

            {   int i=1;
                 printf("%d\n", i);
             }

        printf("%d\n",i);
      }

printf("%d\n",i);
}
```

2. Primer koji ilustruje prenos parametara po vrednosti *(preneti parametri se ne mogu menjati !!!)* ***Šta je rezultat rada?***

```
#include <stdio.h>
void f(int x)  { x++;}
main()
{ int x=3;   f(x);
printf("%d\n", x); }
```

Izlaz: 3

3. Primer koji ilustruje zivotni vek i oblast vazenja promenjivih (scope). ***Šta je rezultat rada?***

```
#include <stdio.h>
/* Globalna promenjiva */
int a = 0;

/* Uvecava se globalna promenjiva a */
void increase()
{  a++;   printf("increase::a = %d\n", a);}

/* Umanjuje se lokalna promenjiva a. Globalna promenjiva zadrzava svoju vrednost. */
void decrease()
{/* Ovo a je nezavisna promenjiva u odnosu na globalno a */
int a = 0;
a--;   printf("decrease::a = %d\n", a);}

void nonstatic_var()
{/* Nestaticke promenjive ne cuvaju vrednosti kroz pozive funkcije */
    int s=0;
     s++;   printf("nonstatic::s=%d\n",s);
}

void static_var()
{/* Staticke promenjive cuvaju vrednosti kroz pozive funkcije. Inicijalizacija se odvija samo u okviru prvog
poziva. */
  static int s=0;
```

```
    s++;printf("static::s=%d\n",s);
}

main()
{
/* Promenjive lokalne za funkciju main */
int i;
int x = 3;

printf("main::x = %d\n", x);

for (i = 0; i<3; i++)
{    /* Promenjiva u okviru bloka je nezavisna od spoljne promenjive. Ovde se koristi promenjiva x lokalna
za blok petlje koja ima vrednost 5, dok originalno x i dalje ima vrednost 3*/
     int x = 5;
    printf("for::x = %d\n", x);
}

/* U ovom bloku x ima vrednost 3 */
printf("main::x = %d\n", x);

increase();
decrease();

/* Globalna promenjiva a */
printf("main::a = %d\n", a);



/* Demonstracija nestatickih promenjivih */
for (i = 0; i<3; i++)   nonstatic_var();

/* Demonstracija statickih promenjivih */
for (i = 0; i<3; i++)   static_var();

}
```

Izlaz iz programa:

```
main::x = 3
for::x = 5
for::x = 5
for::x = 5
main::x = 3
increase::a = 1
decrease::a = -1
main::a = 1
nonstatic::s=1
nonstatic::s=1
nonstatic::s=1
static::s=1
static::s=2
static::s=3
```

Mini kontrolni

4. ***Šta je rezultat rada sledećeg programa?***

```
#include <stdio.h>
int b[10];
main()
{
   int a[5]={5},i;
   for(i=0;i<5;i++)
   printf("%d %d\n",b[i], a[i] );
   return 0;
}
```

5. ***Šta je rezultat rada sledećeg fragmenta programa?***

```
void f(int n) { n = (n++ < 3 ? ++n : n--); }
int main() { int n = 1; f(n); f(n); f(n); printf("%d\n", n); }
```

6. U kom segmentu memorije se čuvaju mašinske instrukcije prevedenog C programa? A argumenti funkcija?

7. Navesti barem tri vrste podataka koje se čuvaju u svakom stek okviru.

8. Šta je rezultat rada sledećeg programa ?

```
#include <stdio.h>
void f(int x, int *y)
{ x++;
  *y*=x+3;
  x+=3;
  printf("x=%d, y=%d\n", x, *y);
  return;
}
  int main()
{ int x=1, y=-1;
  f(x, &y);
  printf("x=%d, y=%d\n", x, y);
  return 0;
}
```

9. Šta je rezultat rada sledećeg programa?

```
#include <stdio.h>
int x=1;
int fun(int x)
```

```
{int  i=1;
 x++;
 printf("i=%d, x=%d\n", i, x);
 return (i-x);
}
void main()
{int i;
 i=5;
 printf("i=%d x=%d\n", i, x);
 i-=fun(i);
 printf("i=%d x=%d\n", i, x);
}
```