Programiranje 2

Pokazivači i dinamička alokacija memorije

Milena Vujošević Janičić
Jelena Graovac

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

Programiranje 2
Beograd, 13. februar, 2020.

Pregled

Pregled

Pokazivači

- Pokazivači (engl. pointer) predstavljaju tip podataka u C-u čije su vrednosti memorijske adrese.
- Broj bajtova koje pokazivač zauzima zavisi od sistema (2,4,8 bajtova).
- Pokazivači implicitno čuvaju informaciju o tipu onoga na šta ukazuju (osim pokazivača na tip void):

```
int *p1;
float* p2;
char* p3, p4;
```

Pokazivači

- Unarni operator &, operator referenciranja ili adresni operator vraća adresu svog operanda.

```
int a=10, *p;
p = &a;
*p = 5; /*operator dereferenciranja*/
```

- Simbolička konstanta NULL je definisana u zaglavlju `<stdio.h>`
- Podrazumeva se da pokazivač koji ima vrednost 0 ne može da pokazuje ni na šta smisleno.

Pokazivači

- Opšti pokazivač `void *` — pokazivač koji može da ukazuje na promenljive različitih tipova.
- Nije moguće vršiti dereferenciranje pokazivača tipa void* jer nije moguće odrediti tip takvog izraza kao ni broj bajtova u memoriji koji predstavljaju njegovu vrednost.
- Pre dereferenciranja, neophodno je konvertovati vrednost ovog pokazivačkog tipa u neki konkretan pokazivački tip.

```
int a = 10;
void *p;
p = &a;
*(int*)p = 5;
```

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?
- Ako je promenljiva tipa `double *` na konkretnom sistemu zauzima 4 bajta, koliko bajtova na istom sistemu zauzima promenljiva tipa `unsigned char *`?

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?
- Ako je promenljiva tipa `double *` na konkretnom sistemu zauzima 4 bajta, koliko bajtova na istom sistemu zauzima promenljiva tipa `unsigned char *`?
- Ako je promenljiva `p` pokazivačkog tipa, da li je dozvoljeno koristiti izraz `&p`?

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?
- Ako je promenljiva tipa `double *` na konkretnom sistemu zauzima 4 bajta, koliko bajtova na istom sistemu zauzima promenljiva tipa `unsigned char *`?
- Ako je promenljiva `p` pokazivačkog tipa, da li je dozvoljeno koristiti izraz `&p`? Da li se umesto vrednosti `p` može pisati i `*(&p)`?

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?
- Ako je promenljiva tipa `double *` na konkretnom sistemu zauzima 4 bajta, koliko bajtova na istom sistemu zauzima promenljiva tipa `unsigned char *`?
- Ako je promenljiva `p` pokazivačkog tipa, da li je dozvoljeno koristiti izraz `&p`? Da li se umesto vrednosti `p` može pisati i `*(&p)`? Da li se umesto vrednosti `p` može pisati i `&(*p)`?

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?
- Ako je promenljiva tipa `double *` na konkretnom sistemu zauzima 4 bajta, koliko bajtova na istom sistemu zauzima promenljiva tipa `unsigned char *`?
- Ako je promenljiva `p` pokazivačkog tipa, da li je dozvoljeno koristiti izraz `&p`? Da li se umesto vrednosti `p` može pisati i `*(&p)`? Da li se umesto vrednosti `p` može pisati i `&(*p)`?
- Ako je promenljiva tipa `int*`, da li joj se može dodeliti celobrojna vrednost?

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?
- Ako je promenljiva tipa `double *` na konkretnom sistemu zauzima 4 bajta, koliko bajtova na istom sistemu zauzima promenljiva tipa `unsigned char *`?
- Ako je promenljiva `p` pokazivačkog tipa, da li je dozvoljeno koristiti izraz `&p`? Da li se umesto vrednosti `p` može pisati i `*(&p)`? Da li se umesto vrednosti `p` može pisati i `&(*p)`?
- Ako je promenjliva tipa `int*`, da li joj se može dodeliti celobrojna vrednost? Ako je promenjliva tipa `double*`, da li joj se može dodeliti celobrojna vrednost?

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?
- Ako je promenljiva tipa `double *` na konkretnom sistemu zauzima 4 bajta, koliko bajtova na istom sistemu zauzima promenljiva tipa `unsigned char *`?
- Ako je promenljiva `p` pokazivačkog tipa, da li je dozvoljeno koristiti izraz `&p`? Da li se umesto vrednosti `p` može pisati i `*(&p)`? Da li se umesto vrednosti `p` može pisati i `&(*p)`?
- Ako je promenljiva tipa `int*`, da li joj se može dodeliti celobrojna vrednost? Ako je promenljiva tipa `double*`, da li joj se može dodeliti celobrojna vrednost?
- Kog tipa je promenljiva `a`, a kog tipa promenljiva `b` nakon deklaracije `short* a, b;`?

Pokazivači

- Koliko bajtova zauzima promenljiva tipa `char`? A koliko bajtova zauzima promenljiva tipa `char*`?
- Ako je promenljiva tipa `double *` na konkretnom sistemu zauzima 4 bajta, koliko bajtova na istom sistemu zauzima promenljiva tipa `unsigned char *`?
- Ako je promenljiva `p` pokazivačkog tipa, da li je dozvoljeno koristiti izraz `&p`? Da li se umesto vrednosti `p` može pisati i `*(&p)`? Da li se umesto vrednosti `p` može pisati i `&(*p)`?
- Ako je promenljiva tipa `int*`, da li joj se može dodeliti celobrojna vrednost? Ako je promenljiva tipa `double*`, da li joj se može dodeliti celobrojna vrednost?
- Kog tipa je promenljiva `a`, a kog tipa promenljiva `b` nakon deklaracije `short* a, b;`? Koju vrednost ima promenljiva `b` nakon izvršavanja naredbi `b = 2; a = &b; *a = 3; b++;`?

Pokazivači

- U jeziku C argumenti funkcija se uvek prenose po vrednosti.
- Ukoliko želimo izmenu vrednosti argumenata funkcije, onda argument mora da se prenese kao pokazivač.

```
void f(int *n) {
  *n = *n + 3;
}
int main() {
  ...
  f(&n);
}
```

Pokazivači

- U jeziku C argumenti funkcija se uvek prenose po vrednosti.
- Ukoliko želimo izmenu vrednosti argumenata funkcije, onda argument mora da se prenese kao pokazivač.

```
void f(int *n) {
  *n = *n + 3;
}
int main() {
  ...
  f(&n);
}
```

- Ukoliko je potrebno da funkcija vrati više od jedne vrednosti, kako se to može postići?

Pokazivači

- U slučaju nizova, prenosi se adresa početka niza, ne kopiraju se svi elementi niza.
- Prenos niza kao argumenta funkcije
  `int f(int niz[])`
  ekvivalentno je sa
  `int f(int * niz)`

Pokazivači

- Moguće je definisati i pokazivače na strukture.
  `struct razlomak *pa = &a;`
- U slučaju da se članovima strukture pristupa preko pokazivača, umesto kombinacije operatora * i ., moguće je koristiti operator `->`:
  `(*pa).imenilac <==> pa->imenilac`
- Operator `->` je operator najvišeg prioriteta.
- Prenos struktura u funkcije se najčešće vrši preko adrese, tj pokazivača.

Veza pokazivača i nizova

- Postoji čvrsta veza između pokazivača i nizova.
- Niz od deset elemenata tipa `int`

  ```
  int a[10];
  ```

  a uvek ukazuje na isti prostor koji je rezervisan za elemente niza tako da se ne može koristiti kao l-vrednost
- Tip promenljive a je `int [10]` i može se po potrebi konvertovati u pokazivač `int*`
- Vrednosti a odgovara pokazivač na prvi element niza (adresa elementa a[0]), vrednosti `a+1` odgovara pokazivač na drugi element niza (adresa elementa a[1]).
- Umesto `&a[i]` može se pisati `a+i`, a umesto `a[i]` može se pisati `*(a+i)`.

Veza pokazivača i nizova

- Ako je `p` pokazivač nekog tipa (npr. `int* p`) na njega može biti primenjen nizovski indeksni pristup (na primer, `p[3]`).
- Vrednost takvog izraza određuje se tako da se poklapa sa odgovarajućim elementom niza koji bi počinjao na adresi `p` (bez obzira što `p` nije niz nego pokazivač).
- Dakle, bez obzira da li je `x` pokazivač ili niz, `x[n]` isto je što i `*(x+n)`, tj. `x+n` isto je što i `&x[n]`.
- Ovo je i čest izvor grešaka, najčešće prilikom alokacije memorije

```
int a[10];
int *b;
a[3] = 5; /* ok */
b[3] = 8; /* greska!!! */
```

Veza pokazivača i nizova

- Ako je u okviru funkcije deklarisan niz sa `char a[10];` na koji deo memorije ukazuje a+9?

Veza pokazivača i nizova

- Ako je u okviru funkcije deklarisan niz sa `char a[10];` na koji deo memorije ukazuje a+9?
- Ako je niz deklarisan sa int a[10];, šta je vrednost izraza a?

Veza pokazivača i nizova

- Ako je u okviru funkcije deklarisan niz sa `char a[10];` na koji deo memorije ukazuje a+9?
- Ako je niz deklarisan sa int a[10];, šta je vrednost izraza a? Šta je vrednost izraza `a + 3`?

Veza pokazivača i nizova

- Ako je u okviru funkcije deklarisan niz sa `char a[10];` na koji deo memorije ukazuje a+9?
- Ako je niz deklarisan sa int a[10];, šta je vrednost izraza a? Šta je vrednost izraza `a + 3`? Šta je vrednost izraza `*(a+3)`?

Veza pokazivača i nizova

- Ako je u okviru funkcije deklarisan niz sa `char a[10];` na koji deo memorije ukazuje a+9?
- Ako je niz deklarisan sa int a[10];, šta je vrednost izraza a? Šta je vrednost izraza `a + 3`? Šta je vrednost izraza `*(a+3)`?
- Da li se komanda `ip = &a[0]` može zameniti komandom (a) `ip = a[0];` (b) `ip = a;` (c) `ip = *a;` (d) `ip = *a[0]`?

Veza pokazivača i nizova

- Ako je u okviru funkcije deklarisan niz sa `char a[10];` na koji deo memorije ukazuje a+9?
- Ako je niz deklarisan sa int a[10];, šta je vrednost izraza a? Šta je vrednost izraza `a + 3`? Šta je vrednost izraza `*(a+3)`?
- Da li se komanda `ip = &a[0]` može zameniti komandom `(a) ip = a[0]; (b) ip = a; (c) ip = *a; (d) ip = *a[0]`?
- Da li je vrednost a[i] ista što i (a) `a[0]+i;` (b) `a+i;` (c) `*(a+i);` (d) `&(a+i)`?

Veza pokazivača i nizova

- Ako je u okviru funkcije deklarisan niz sa `char a[10];` na koji deo memorije ukazuje a+9?
- Ako je niz deklarisan sa int a[10];, šta je vrednost izraza a? Šta je vrednost izraza `a + 3`? Šta je vrednost izraza `*(a+3)`?
- Da li se komanda `ip = &a[0]` može zameniti komandom (a) `ip = a[0];` (b) `ip = a;` (c) `ip = *a;` (d) `ip = *a[0]`?
- Da li je vrednost a[i] ista što i (a) `a[0]+i;` (b) `a+i;` (c) `*(a+i);` (d) `&(a+i)`?
- Ukoliko je niz deklarisan sa: `float* x[10]`, kako se može dobiti adresa drugog elementa niza?

Veza pokazivača i nizova

- Neka je niz a deklarisan sa int a[10], i neka je p pokazivač tipa `int*`. Da li je naredba a = p; ispravna?

Veza pokazivača i nizova

- Neka je niz a deklarisan sa int a[10], i neka je p pokazivač tipa `int*`. Da li je naredba a = p; ispravna? Da li je naredba p = a; ispravna?

Veza pokazivača i nizova

- Neka je niz a deklarisan sa int a[10], i neka je p pokazivač tipa `int*`. Da li je naredba a = p; ispravna? Da li je naredba p = a; ispravna? Koja je vrednost izraza sizeof(p) nakon p = a;, ako `int*` zauzima 4 bajta?

Veza pokazivača i nizova

- Neka je niz a deklarisan sa int a[10], i neka je p pokazivač tipa `int*`. Da li je naredba a = p; ispravna? Da li je naredba p = a; ispravna? Koja je vrednost izraza sizeof(p) nakon p = a;, ako `int*` zauzima 4 bajta?
- Niz je deklarisan sa int a[10];. Da li je a+3 ispravan izraz?

Veza pokazivača i nizova

- Neka je niz a deklarisan sa int a[10], i neka je p pokazivač tipa `int*`. Da li je naredba a = p; ispravna? Da li je naredba p = a; ispravna? Koja je vrednost izraza sizeof(p) nakon p = a;, ako `int*` zauzima 4 bajta?
- Niz je deklarisan sa int a[10];. Da li je a+3 ispravan izraz? Da li mu je vrednost ista kao i vrednost `a[3], &a[3], *a[3] ili a[10]`?

Veza pokazivača i nizova

- Neka je niz a deklarisan sa int a[10], i neka je p pokazivač tipa `int*`. Da li je naredba a = p; ispravna? Da li je naredba p = a; ispravna? Koja je vrednost izraza sizeof(p) nakon p = a;, ako `int*` zauzima 4 bajta?
- Niz je deklarisan sa int a[10];. Da li je a+3 ispravan izraz? Da li mu je vrednost ista kao i vrednost `a[3], &a[3], *a[3] ili a[10]`? Da li je `*(a+3)` ispravan izraz?

Veza pokazivača i nizova

- Neka je niz a deklarisan sa int a[10], i neka je p pokazivač tipa `int*`. Da li je naredba a = p; ispravna? Da li je naredba p = a; ispravna? Koja je vrednost izraza sizeof(p) nakon p = a;, ako `int*` zauzima 4 bajta?
- Niz je deklarisan sa int a[10];. Da li je a+3 ispravan izraz? Da li mu je vrednost ista kao i vrednost `a[3], &a[3], *a[3] ili a[10]`? Da li je `*(a+3)` ispravan izraz? Da li mu je vrednost ista kao i vrednost `a[3], a[10], *a[3] ili *a[10]`?

Veza pokazivača i nizova

- Izraz `&a` ima istu vrednost (tj. sadrži istu adresu) kao i `a`, ali drugačiji tip — tip `int (*)[10]` — to je tip pokazivača na niz od 10 elemenata koji su tipa `int`. Taj tip ima u narednom primeru promenljiva `c`:

```
int a[10]; /*niz od 10 elemenata tipa int*/
int *b[10]; /*niz od 10 pokazivaca na int*/
int (*c)[10]; /*pokazivac na niz od 10
elemenata koji su tipa int*/
c = &a; /* (*c)[0] <==> a[0] */
```

Pokazivačka aritmetika

- Pokazivačka aritmetika se razlikuje od običnog izračunavanja: izraz p+1 ne označava dodavanje vrednosti 1 na p, već dodavanje dužine jednog objekta tipa na koji ukazuje p.
- Na primer, ako p ukazuje na int, onda p+1 i p mogu da se razlikuju za dva ili četiri — za onoliko koliko bajtova na tom sistemu zauzima podatak tipa int.
- Od pokazivača je moguće oduzimati cele brojeve.
- Pokazivače nije moguće sabirati (jer to nema smisla!), ali ih možemo oduzimati (razlika je broj objekata izmedju).
- Moguće je koristiti operatore ++ i --
- Pokazivače je moguće porediti.

Prioritet operatora

Operator	Opis	Asocijativnost
()	Poziv funkcije	sleva na desno
[]	indeksni pristup elementu niza	
.	pristup članu strukture ili unije	
->	pristup članu strukture ili unije preko pokazivača	
++ --	postfiksni inkrement/dekrement	
++ --	prefiksni inkrement/dekrement	zdesna na levo
+ -	unarni plus/minus	
! ~	logička negacija/bitski komplement	
(type)	eksplicitna konverzija tipa (cast)	
*	dereferenciranje	
&	referenciranje (adresa)	
sizeof	veličina u bajtovima	

Prioritet operatora

- Unarni operatori & i * imaju viši prioritet nego binarni aritmetički operatori.
- Koje je značenje izraza *p+1? Da li je to (*p)+1 ili *(p+1)?
- Koje je značenje ++*p?
- Koje je značenje *++p?
- Koje je značenje *p++? Da li se uzima vrednost na lokaciji p i zatim inkrementira pokazivač ili se inkrementira sadržaj na lokaciji p?

Pokazivačka aritmetika

- Ako je p tipa `int*`, šta je efekat naredbe `*p += 5;` a šta efekat naredbe `p += 5;`?

Pokazivačka aritmetika

- Ako je p tipa `int*`, šta je efekat naredbe `*p += 5;` a šta efekat naredbe `p += 5;`?
- Ako su promenljive p i q tipa `double *`, za koliko će se razlikovati vrednosti p i q nakon komande `p = q+n` (pri čemu je n celobrojna promenljiva)?

Pokazivačka aritmetika

- Ako je p tipa `int*`, šta je efekat naredbe `*p += 5;` a šta efekat naredbe `p += 5;`?
- Ako su promenljive p i q tipa `double *`, za koliko će se razlikovati vrednosti p i q nakon komande `p = q+n` (pri čemu je n celobrojna promenljiva)?
- Ako je promenljiva p tipa `double *`, za koliko će se promeniti njena vrednost (a) nakon komande `++*p;`? (b) nakon komande `++(*p);`? (c) nakon komande `*p++;`? (d) nakon komande `(*p)++;`?

Pokazivačka aritmetika

- Ako je p tipa `int*`, šta je efekat naredbe `*p += 5;` a šta efekat naredbe `p += 5;`?
- Ako su promenljive p i q tipa `double *`, za koliko će se razlikovati vrednosti p i q nakon komande `p = q+n` (pri čemu je n celobrojna promenljiva)?
- Ako je promenljiva p tipa `double *`, za koliko će se promeniti njena vrednost (a) nakon komande `++*p;`? (b) nakon komande `++(*p);`? (c) nakon komande `*p++;`? (d) nakon komande `(*p)++;`?
- Da li postoji razlika između komandi `(*p)++` i `*(p++)`?

Pokazivači i niske

- Koja je razlika između

```
char *p  = "informatika";
char a[] = "informatika";
```

- p je pokazivač za koji se memorija odvaja na steku, niska informatika se čuva u segmentu podataka
- a je niz od 12 karaktera koji su inicijalizovani niskom informatika i za niz je rezervisana memorija na steku
- p je pokazivač pa može da promeni vrednost (da pokazuje na nešto drugo)
- a je niz i ne može da promeni vrednost.
- Promena vrednosti p[0] nije regularna.
- Promena vrednosti a[0] je regularna.

Pokazivači i niske

```
int strlen(char s[])
{
 int i, n=0;
 for(i=0; s[i]!='\0'; i++)
  n++;
 return n;
}
```

```
int strlen(char s[])
{
 int i=0;
 while(s[i])
  i++;
 return i;
}
```

Povratna vrednost treba da bude tipa unsigned.
`size_t` označava neoznačen celobrojni tip, koji obično služi za reprezentovanje veličine objekata u memoriji — najčešće je to `unsigned int`.

strlen

Kako se funkcija strlen implementira korišćenjem pokazivačke aritmetike?

```
int strlen(char s[])
{
 int i, n=0;
 for(i=0; s[i]!='\0'; i++)
  n++;
 return n;
}
```

```
int strlen(char s[])
{
 int i=0;
 while(s[i])
  i++;
 return i;
}
```

```
size_t strlen(const char *s) {
 size_t n;
 for (n = 0; *s != '\0'; s++)
   n++;
 return n;
}
```

```
size_t strlen(const char *s) {
  const char* t = s;
  while(*s)
    s++;
  return s - t;
}
```

Pokazivači i niske

Kako se funkcija strlen implementira korišćenjem pokazivačke aritmetike?

```
size_t strlen(const char *s) {
 size_t n;
 for (n = 0; *s != '\0'; s++)
   n++;
 return n;
}
```

```
size_t strlen(const char *s) {
  const char* t = s;
  while(*s)
    s++;
  return s - t;
}
```

Zbog neposrednih veza između nizova i pokazivača, strlen može da se primeni i na konstantnu nisku (npr strlen("informatika") i za argument koji je ime niza (npr strlen(a) gde je niz deklarisan sa char a[10]) kao i za pokazivač (npr strlen(p) gde je p deklarisano sa char* p;)

Višedimenzioni nizovi

- Pored jednodimenzionih mogu se koristiti i višedimenzioni nizovi, koji se deklarišu na sledeći opšti način:

  ```
  tip ime_niza[dimenzija_1]...[dimenzija_N];
  ```

  Na primer, dvodimenzioni niz (matrica) sa celobrojnim elementima:

  ```
  int a[10][20];
  ```

- Dvodimenzioni nizovi (matrice) tumače se kao jednodimenzioni nizovi čiji su elementi nizovi. Npr, svaki od izraza a[0], a[1], a[2] do a[9] označava niz od 20 elemenata tipa int.
- Elementu i-te vrste i j-te kolone dvodimenzionalnog niza pristupa se sa:

  ```
  a[i][j]
  ```

Matrice

Inicijalizacija matrice (prikazati na steku):

```
char a[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

Inkrementiranje svih elemenata matrice dimenzije nxm:

```
for(i=0; i<n; i++)
  for(j=0; j<m; j++)
    a[i][j]++;
```

Matrice

- Razni zadaci sa matricama
  - Izračunati nešto što je funkcija svih elemenata matrice (suma, maksimum, minimum...)
  - Transfromisati elemenate matrice (apsolutne vrednosti, uvećanje, umanjenje...)
  - Izračunati neku osobinu svake vrste/kolone matrice (maksimum/minimum, zbir, proizvod...)
  - Koristiti samo deo matrice za izračunavanje (elemente ispod glavne dijagonale, elemente iznad sporedne dijagonale...)
  - Na osnovu dve matrice formirati treću (zbir, razlika, proizvod ...)
  - ...

Niz nizova i niz pokazivača

```
int a[10][20]; /*Deset nizova od po 20 elemenata tipa int
                 200 lokacija za podatak tipa int
                 je rezervisano */
int *b[10];    /*Deset pokazivača na int
                 deset pokazivača na int je alocirano*/
```

- U oba slučaja može se pristupiti npr a[3][5] i b[3][5], ali u prvom slučaju imamo alociran prostor za element a[3][5] dok u drugom slučaju moramo da obezbedimo da pokazivač koji se nalazi na poziciji b[3] bude pokazivač na niz koji sadrži najmanje 6 elemenata.
- U slučaju da za svaki pokazivač niza b obezbedimo da pokazuje na niz od po 20 elemenata, onda u tom slučaju imamo tih 200 lokacija plus 10 pokazivača što zauzima više prostora nego za a.

Niz nizova i niz pokazivača

```
int a[10][20]; /*Deset nizova od po 20 elemenata tipa int
                 200 lokacija za podatak tipa int
                 je rezervisano */
int *b[10];    /*Deset pokazivača na int
                 deset pokazivača na int je alocirano*/
```

- Elementi matrice a su na uzastopnim lokacijama u memoriji dok elementi matrice b to ne moraju da budu.
- Nizovi na koje pokazuju pokazivači niza b ne moraju da budu istih veličina.

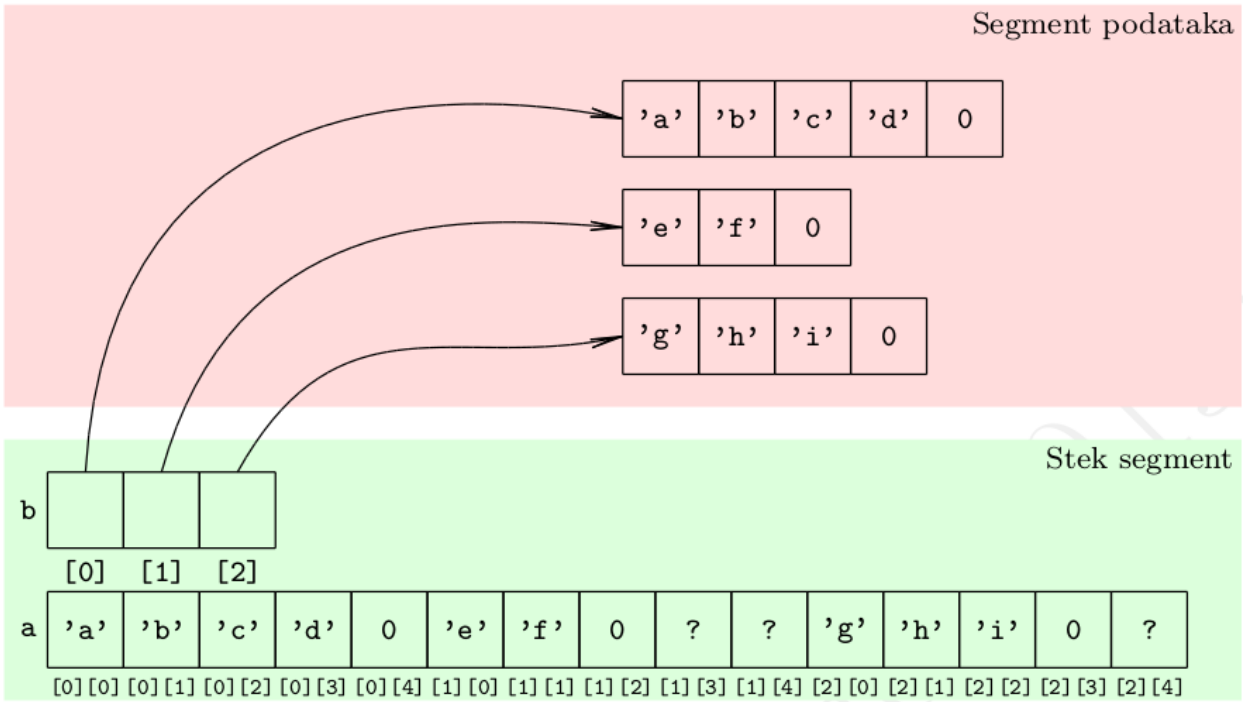
Niz nizova i niz pokazivača

Prikazati memoriju za prvi i drugi primer

```
char a[][5] = {"abcd", "ef", "ghi"};

char *b[] = {"abcd", "ef", "ghi"};
```

Niz nizova i niz pokazivača



Pokazivači na funkcije

```
void povecaj1(int a[], int n, int b[]) {
 int i;
 for (i = 0; i < n; i++)
   b[i] = a[i] + 1;
}
void pomnozi2(int a[], int n, int b[]) {
 int i;
 for (i = 0; i < n; i++)
   b[i] = a[i] * 2;
}
void parni0(int a[], int n, int b[]) {
 int i;
 for (i = 0; i < n; i++)
   b[i] = a[i] % 2 == 0 ? 0 : a[i];
}
```

Pokazivači na funkcije

```
void transformacija(int a[], int n, int b[], int (*f) (int)) {
 int i;
 for (i = 0; i < n; i++)
   b[i] = (*f)(a[i]);
}
int uvecaj1(int x) { return x + 1; }
int pomnozi2(int x) { return 2 * x; }
int parni0(int x) { return x % 2 == 0 ? 0 : x; }

/*poziv funkcije transformacija()*/
transformacija(a, N, b, &uvecaj1);
transformacija(a, N, b, &pomnozi2);
transformacija(a, N, b, &parni0);
```

Pokazivači na funkcije

- Pokazivači na funkcije se razlikuju po tipu funkcije na koje ukazuju (po tipovima argumenata i tipu povratne vrednosti).
- Deklaracija promenljive tipa pokazivača na funkciju se vrši tako što se ime promenljive kojem prethodi karakter * navede u zagradama kojima prethodi tip povratne vrednosti funkcije, a za kojima sledi lista tipova parametara funkcije.

```
double *a(double, int);   /*Funkcija koja vraca pokazivac na
                            double i prima dva argumenta
                            double i int*/
double (*b)(double, int); /*Pokazivac na funkciju koja vraca
                            double i prima dva argumenta
                            double i int*/
```

Pokazivači na funkcije

Moguće je kreirati i nizove pokazivača na funkcije. Ovi nizovi se mogu i incijalizovati (na uobičajeni način). U primeru

```
int (*nf[3]) (int) = {&povecaj1, &pomnozi2, &parni0};
```

nf predstavlja niz od 3 pokazivača na funkcije koje vraćaju int, i primaju argument tipa int.
Funkcije čije se adrese nalaze u nizu se mogu direktno i pozvati. Na primer, naredni kôd ispisuje vrednost 4:

```
printf("%d", (*nf[0])(3));
```

Pokazivači na funkcije

- Kog je tipa x deklarisano sa:
  1. `double (*x[3])(int);`

Pokazivači na funkcije

- Kog je tipa x deklarisano sa:
  1. `double (*x[3])(int);`
  2. `int (*x) (double);`

Pokazivači na funkcije

- Kog je tipa x deklarisano sa:
  1. `double (*x[3])(int);`
  2. `int (*x) (double);`
  3. `int *x (double);`

Pokazivači na funkcije

- Kog je tipa x deklarisano sa:
  1. `double (*x[3])(int);`
  2. `int (*x) (double);`
  3. `int *x (double);`
  4. `double* (*x) (float*);`

Pokazivači na funkcije

- Kog je tipa x deklarisano sa:
  1. `double (*x[3])(int);`
  2. `int (*x) (double);`
  3. `int *x (double);`
  4. `double* (*x) (float*);`
  5. `int (*f) (float*);`

Pregled

Dinamička alokacija memorije

- U većini realnih aplikacija, u trenutku pisanja programa nije moguće precizno predvideti memorijske zahteve programa.
- Memorijski zahtevi zavise od interakcije sa korisnikom.
- Gornje ograničenje veličine niza: ne možemo da radimo sa manjim nizom, a ako koristimo mnogo veći niz onda zauzimamo bespotrebno prostor u memoriji.

Dinamička alokacija memorije

- Dinamička alokacija memorije omogućava da program u toku svog rada, u fazi izvršavanja, zahteva (od operativnog sistema) određenu količinu memorije.
- U trenutku kada mu memorija koja je dinamički alocirana više nije potrebna, program može i dužan je da je oslobodi i tako je vrati operativnom sistemu na upravljanje.
- Alociranje i oslobađanje vrši se funkcijama iz standardne biblioteke i pozivima rantajm biblioteke.

Hip i dinamički životni vek

- Memorija dodeljena programu je organizovana u segment kôda, segment podataka, stek segment i hip segment.
- Hip segment predstavlja tzv. slobodnu memoriju iz koje se crpi memorija koja se dinamički alocira.
- Objekti koji su alocirani u slobodnom memorijskom prostoru nisu imenovani, već im se pristupa isključivo preko adresa.
- Svi objekti koji su dinamički alocirani imaju dinamički životni vek.
- Ovo znači da se memorija i alocira i oslobađa isključivo na eksplicitni zahtev i to tokom rada programa.

Dinamička alokacija memorije

- Standardna biblioteka jezika C podržava dinamičko upravljanje memorijom kroz nekoliko funkcija koje su deklarisane u zaglavlju `<stdlib.h>`.
- Prostor za dinamički alociranu memoriju nalazi se u segmentu memorije koji se zove hip (engl. heap).
- malloc, calloc, realloc i free

void *malloc(size_t n);

- malloc alocira blok memorije (tj. niz uzastopnih bajtova) veličine n bajtova i vraća adresu alociranog bloka u vidu generičkog pokazivača (tipa void*).
- U slučaju da zahtev za memorijom nije moguće ispuniti (na primer, zahteva se više memorije nego što je na raspolaganju), ova funkcija vraća NULL.
- Memorija na koju funkcija malloc vrati pokazivač nije inicijalizovana i njen sadržaj je, u principu, nedefinisan (tj. zavisi od podataka koji su ranije bili čuvani u tom delu memorije).

void *malloc(size_t n);

- Kada je nakon komandi

```
char *p;
p = (char*)malloc(n);
```

komanda `strcpy(p,"test");` bezbedna?

void *calloc(size_t n, size_t size)

- calloc vraća pokazivač na blok memorije veličine n objekata navedene veličine size.
- U slučaju da zahtev nije moguće ispuniti, vraća se NULL.
- Za razliku od malloc, alocirana memorija je inicijalizovana na nulu.
- Nakon poziva funkcije malloc() ili calloc() obavezno treba proveriti povratnu vrednost kako bi se utvrdilo da li je alokacija uspela.
- Ukoliko alokacija ne uspe, pokušaj pristupa memoriji na koju ukazuje dobijeni pokazivač dovodi do dereferenciranja NULL pokazivača i greške.

Primer

```
int* p = (int*) malloc(n*sizeof(int));
if (p == NULL)
  /* prijaviti gresku */
```

- Nakon uspešne alokacije, u nastavku programa se p može koristiti kao (statički alociran) niz celih brojeva, npr:

  ```
  p[0] = 5;
  ```

free

- U trenutku kada dinamički alociran blok memorije više nije potreban, poželjno je osloboditi ga:

  ```
  void free(void* p);
  ```

- Poziv free(p) oslobađa memoriju na koju ukazuje pokazivač p, pri čemu je neophodno da p pokazuje na blok memorije koji je alociran pozivom funkcije malloc ili calloc (i ne sme se sa free oslobađati memorija koja nije alocirana na ovaj način).
- Ne sme se koristiti nešto što je već oslobođeno niti se sme dva puta oslobađati ista memorija.
- Redosled oslobađanja memorije ne mora da odgovara redosledu alociranja.

free

- Ukoliko neki dinamički alociran blok nije oslobođen ranije, on će biti oslobođen prilikom završetka rada programa, zajedno sa svom drugom memorijom koja je dodeljena programu.
- Ipak, preporučeno je eksplicitno oslobađanje sve dinamički alocirane memorije pre kraja rada programa, a poželjno čim taj prostor nije potreban.

free

- Ukoliko neki dinamički alociran blok nije oslobođen ranije, on će biti oslobođen prilikom završetka rada programa, zajedno sa svom drugom memorijom koja je dodeljena programu.
- Ipak, preporučeno je eksplicitno oslobađanje sve dinamički alocirane memorije pre kraja rada programa, a poželjno čim taj prostor nije potreban.
- Šta je, nakon poziva free(p), vrednost pokazivača p, a šta vrednost *p?

void *realloc(void *memblock, size_t size);

- U nekim slučajevima potrebno je promeniti veličinu već alociranog bloka memorije što se postiže korišćenjem funkcije realloc.
- Parametar memblock je pokazivač na prethodno alocirani blok memorije, a parametar size je nova veličina u bajtovima.
- Funkcija realloc vraća pokazivač na realociran blok memorije, a NULL u slučaju da zahtev ne može biti ispunjen.
- Zahtev za smanjivanje veličine alociranog bloka memorije uvek uspeva.

void *realloc(void *memblock, size_t size);

- U slučaju da se zahteva povećanje veličine alociranog bloka memorije:
  - ukoliko iza postojećeg bloka postoji dovoljno slobodnog prostora, taj prostor se jednostavno koristi za proširivanje.
  - ukoliko iza postojećeg bloka ne postoji dovoljno slobodnog prostora, onda se u memoriji traži drugo mesto dovoljno da prihvati prošireni blok i, ako se nađe, sadržaj postojećeg bloka se kopira na to novo mesto i zatim se stari blok memorije oslobađa.
- Ova operacija može biti vremenski zahtevna.

Greške u radu sa dinamičkom memorijom

- Curenje memorije
- Pristup oslobođenoj memoriji
- Oslobađanje istog bloka više puta
- Oslobađanje neispravnog pokazivača
- Prekoračenja i potkoračenja bafera

Curenje memorije

- Curenje memorije — situacija kada se u tekućem stanju programa izgubi informacija o lokaciji dinamički alociranog, a neoslobođenog bloka memorije.

```
char* p;
p = (char*) malloc(1000);
....
p = (char*) malloc(5000);
```

- U slučaju curenja memorije program više nema mogućnost da oslobodi taj blok memorije i on biva „zauvek“ izgubljen (rezervisan za korišćenje od strane programa koji više nema načina da mu pristupi).

Greške u radu sa dinamičkom memorijom

- Ukoliko se curenje memorije dešava u nekoj petlji, može da se gubi malo po malo memorije, ali tokom dugtrajnog izvršavanja programa, pa ukupna količina izgubljene memorije može da bude veoma velika.
- Može da se iscrpi sva memorija i da operativni sistem naglo prekine program, ili da se ekstremno uspori izvršavanje programa usled prebacivanja memorija-disk.
- Curenje memorije je greška koju je teško uočiti i pronaći njene uzroke, a veoma je opasna.

Pristup oslobođenoj memoriji

- Pristup oslobođenoj memoriji — nakon poziva free(p), memorija na koju pokazuje pokazivač p se oslobađa i ona više ne bi trebalo da se koristi.
- Međutim, poziv free(p) ne menja sadržaj pokazivača p. Ukoliko programer nastavi da koristi ovu memoriju, može da se desi da se ovaj problem nekada ne ispoljava, a da u drugim situacijama dolazi do pucanja programa. Ovake greške se teško otkrivaju.
- Preporuka: nakon poziva free(p), odmah p postaviti p na NULL — tako se osigurava da će svaki pokušaj pristupa oslobođenoj memoriji biti odmah prepoznat tokom izvršavanja programa.

Oslobađanje istog bloka više puta

- Nakon poziva free(p), svaki naredni poziv free(p) za istu vrednost pokazivača p prouzrokuje nedefinisano ponašanje programa i trebalo bi ga izbegavati.
- Takozvana višestruka oslobađanja mogu da dovedu do pada programa a poznato je da mogu da budu i izvor bezbednosnih problema.

Oslobađanje neispravnog pokazivača

- Funkciji free(p) dopušteno je proslediti isključivo adresu vraćenu od strane funkcije malloc, calloc ili realloc.
- Nije dozvoljeno čak ni prosleđivanje pokazivača na lokaciju koja pripada alociranom bloku (a nije njegov početak). Na primer,

```
free(p+10); /* Oslobodi sve osim prvih 10 elemenata bloka */
```

neće osloboditi „sve osim prvih 10 elemenata bloka“ i sasvim je moguće da će dovesti do neprijatnih posledica, pa čak i do pada programa.

Prekoračenja i potkoračenja bafera

- Nakon dinamičke alokacije, pristup memoriji je dozvoljen samo u okviru granica bloka koji je dobijen.
- Kao i u slučaju statički alociranih nizova, pristup elementima van granice može da prouzrokuje ozbiljne probleme u radu programa.
- Upis van granica bloka najčešće je opasniji od čitanja.
- U slučaju dinamički alociranih blokova memorije, obično se nakon samog bloka smeštaju dodatne informacije potrebne alokatoru memorije kako bi uspešno vodio evidenciju koji delovi memorije su zauzeti, a koji slobodni i ukoliko se to poremeti dolazi do pada sistema za dinamičko upravljanje memorijom.

Greške u radu sa dinamičkom memorijom

- Da li će doći do curenja memorije ako nakon komande p = (int*)malloc(sizeof(int)*5) slede komanda/komande:

  1) q = (int*)malloc(sizeof(int)*5);

Greške u radu sa dinamičkom memorijom

- Da li će doći do curenja memorije ako nakon komande p = (int*)malloc(sizeof(int)*5) slede komanda/komande:

  1) q = (int*)malloc(sizeof(int)*5);

  2) p = (int*)malloc(sizeof(int)*5);

Greške u radu sa dinamičkom memorijom

- Da li će doći do curenja memorije ako nakon komande p = (int*)malloc(sizeof(int)*5) slede komanda/komande:

  1) q = (int*)malloc(sizeof(int)*5);

  2) p = (int*)malloc(sizeof(int)*5);

  3) free(p); free(p);

Greške u radu sa dinamičkom memorijom

- Da li će doći do curenja memorije ako nakon komande p = (int*)malloc(sizeof(int)*5) slede komanda/komande:

  1) q = (int*)malloc(sizeof(int)*5);

  2) p = (int*)malloc(sizeof(int)*5);

  3) free(p); free(p);

  4) free(p+1);

Fragmentisanje memorije

- Čest je slučaj da ispravne aplikacije u kojima ne postoji curenje memorije (a koje često vrše dinamičku alokaciju i dealokacije memorije) tokom dugog rada pokazuju degradaciju u performansama i na kraju prekidaju svoj rad na nepredviđeni način.
- Uzrok ovome je najčešće fragmentisanje memorije.

Fragmentisanje memorije

- Ukoliko 0 označava slobodni bajt, a 1 zauzet, a memorija trenutno ima sadržaj 100101011000011101010110, postoji ukupno 12 slobodnih bajtova. Međutim, pokušaj alokacije 5 bajtova ne može da uspe, jer u memoriji ne postoji prostor dovoljan za smeštanje 5 povezanih bajtova. S druge strane, memorija koja ima sadržaj 111111111111111100000000 ima samo 8 slobodnih bajtova, ali jeste u stanju da izvrši alokaciju 5 traženih bajtova.

Pregled

Višedimenzioni nizovi

- Statički niz:

```
int niz[10];
```

- Dinamički niz:

```
int* p;
...
p = (int*) malloc(n*sizeof(int));
if(p==NULL) ...
```

- Statički alocirana matrica

```
int a[10][20];
```

- Kog je tipa dinamički alocirana matrica?
- Kako se dinamički alocira matrica?

Višedimenzioni nizovi

- Niz pokazivača (nacrtati memoriju!)

```
int* a[10];
```

- Ovako definisana matrica može da ima najviše deset vrsta, dok broj kolona možemo da podesimo po potrebi.
- Za svaki pokazivač iz tog niza, tj za svaku vrstu matrice, dinamički alociramo prostor za elemenate matrice.

```
n = n<10 ? n : 10;
/*za svaku vrstu alociramo prostor*/
for(i=0; i<n; i++) {
  a[i] = (int*)malloc(m*sizeof(int));
  if(a[i]==NULL) ...
}
```

Višedimenzioni nizovi

- Fleksibilnije rešenje, bez ograničavanja broja vrsta: a je pokazivač koji čuva adresu drugog pokazivača.

  ```
  int** a;
  ```

- a je pokazivač za koji ćemo dinamički da alociramo prostor, tj a će da čuva adresu početka niza čiji su elementi pokazivači na vrste matrice.
- Za svaki pokazivač iz tog niza, tj za svaku vrstu matrice, dinamički alociramo prostor za elemenate matrice.

Višedimenzioni nizovi

- Nacrtati memoriju!
- Dakle, najpre alociramo prostor za niz pokazivača:

```
int **a = NULL;
...
a = (int**)malloc(n*sizeof(int*)); /*n vrsta*/
if(a==NULL) ...
```

Zatim za svaki pokazivač, alociramo prostor za niz na koji pokazuje:

```
for(i=0; i<n; i++) {
  /*za svaku vrstu alociramo prostor*/
  a[i] = (int*)malloc(m*sizeof(int));
  if(a[i]==NULL) ...
}
```

Višedimenzioni nizovi

- Upotreba dinamički alocirane matrice se ne razlikuje od upotrebe statički alocirane matrice, tj pristup elementima matrice je isti `a[i][j]`
- Za dinamički alociranu matricu neophodno je obezbediti dealokaciju memorije.
- U prvom slučaju (tj za `int* a[10]`) to je

```
/*za svaku vrstu dealociramo prostor*/
for(i=0; i<n; i++) {
  free(a[i]);
}
```

Višedimenzioni nizovi

- U drugom slučaju (`int** a`), prethodnoj dealokaciji potrebno je dodati i dealokaciju niza pokazivača:

```
/*za svaku vrstu dealociramo prostor*/
for(i=0; i<n; i++) {
  free(a[i]);
}
/*dealociramo prostor koji je zauzimao niz pokazivaca*/
free(a);
```

- Važno je voditi računa o redosledu dealokacije.

Pregled

Literatura

- Slajdovi su pripremljeni na osnovu desetog poglavlja knjige
  Filip Marić, Predrag Janičić: Programiranje 1
  i šestog poglavlja knjige
  Predrag Janičić, Filip Marić: Programiranje 2
- Za pripremu ispita, slajdovi nisu dovoljni, neophodno je koristiti knjigu!