

Programiranje 1

Pokazivači

Studijski program matematika

Pregled

1 Pokazivači

Pregled

1 Pokazivači

- Pokazivači i adrese
- Pokazivači i argumenti funkcija
- Pokazivačka aritmetika
- Pokazivači i nizovi

Pokazivači

- Pokazivači imaju veoma značajnu ulogu i teško je napisati iole kompleksniji program bez upotrebe pokazivača.
- Programer, u radu sa pokazivačima, ima veliku slobodu i širok spektar mogućnosti.
- Dodatne mogućnosti u radu sa pokazivačima daje dinamička alokacija memorije.
- To otvara i veliki prostor za efikasno programiranje ali i za greške.

Pokazivači

- Fon Nojmanova arhitektura: procesor+memorija
- Memorija računara organizovana je u niz uzastopnih bajtova.
- Memoriji se pristupa preko adresa
- Adrese su brojevi koji se najčešće zapisuju u heksadekadnom sistemu
- Uzastopni bajtovi memorije mogu se tretirati kao jedinstven podatak.
- Na primer, dva (ili četiri, u zavisnosti od sistema) uzastopna bajta mogu se tretirati kao jedinstven podatak celobrojnog tipa.

Pokazivači

- *Pokazivači* (engl. pointer) predstavljaju tip podataka u C-u čije su vrednosti memorijske adrese.
- Na 16-bitnim sistemima adrese zauzimaju dva bajta, na 32-bitnim sistemima četiri, na 64-bitnim osam.
- Iako su, suštinski, pokazivačke vrednosti (adrese) celi brojevi, pokazivački tipovi se razlikuju od celobrojnih.

Pokazivači

- Jezik C razlikuje više pokazivačkih tipova.
- Za svaki tip podataka (i osnovni i korisnički) postoji odgovarajući pokazivački tip.
- Pokazivači implicitno čuvaju informaciju o tipu onoga na šta ukazuju (osmi pokazivača na tip void, o čemu kasnije)
- Tip pokazivača koji ukazuje na podatak tipa `int` zapisuje se `int *`.
- Slično važi i za druge tipove, npr `float *`, `double *`, `char *`...

Primer

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


Primer

```
int *p1; /*pokazivac na int*/  
int* p2; /*pokazivac na int*/  
int* p3, p4; /*p3 je pokazivac na int, p4 je int*/
```

Šta sadrže pokazivači?

- Pokazivačka promenljiva može da sadrži adrese promenljivih ili elemenata niza.
- Postoje i pokazivači na funkcije.
- Unarni operator `&`, *operator referenciranja* ili *adresni operator* vraća adresu svog operanda.
- On može biti primenjen samo na promenljive i elemente niza, a ne i na izraze ili konstante.

Primer

```
int a=10;  
int *p;  
p = &a;
```

Kako izgleda memorija?

Operator dereferenciranja

- Unarni operator * (koji zovemo „operator dereferenciranja“) se primenjuje na pokazivačku promenljivu i vraća sadržaj lokacije na koju ta promenljiva pokazuje, vodeći računa o tipu.
- Dereferencirani pokazivač može biti l-vrednost
- Dereferencirani pokazivač nekog tipa, može se pojaviti u bilo kom kontekstu u kojem se može pojaviti podatak tog tipa.

Primer

```
int a=10, *p;  
p = &a;  
*p = 5;  
*p = *p+a+3;
```

Kako izgleda memorija i koja je vrednost promenljive a?

Odnos sa celobrojnim tipovima

- Kao što je već rečeno, pokazivački i celobrojni tipovi su različiti.
- Naredni kod je neispravan

```
int *pa, a;  
pa = a;  
a = pa;  
pa = 1234;
```

- Moguće je koristiti eksplicitne konverzije. Na primer:

```
a = (int)pa; ili  
pa = (int*)a;
```

Ove mogućnosti treba veoma oprezno koristiti!

Odnos sa celobrojnim tipovima

- Jedina celobrojna vrednost koja se može koristiti i kao pokazivačka vrednost je vrednost 0 — moguće je dodeliti nulu pokazivačkoj promenljivoj i porediti pokazivač sa nulom.
- Uvek se podrazumeva da pokazivač koji ima vrednost 0 ne može da pokazuje ni na šta smisleno (tj. adresa 0 ne smatra se mogućom).

NULL

- Pokazivač koji ima vrednost 0 nije moguće dereferencirati, tj. pokušaj dereferenciranja dovodi do greške tokom izvršavanja programa (najčešće „segmentation fault“).
- Umesto konstante 0 obično se koristi simbolička konstanta `NULL`, definisana u zaglavlju `<stdio.h>`, kao jasniji pokazatelj da je u pitanju specijalna pokazivačka vrednost.

- Na primer:

```
int* p = NULL;
```


Primer

```
int* a, b = 2;  
a = &b;  
*a = 3;  
b++;
```

Koja je vrednost promenljive b?

Primer

```
int* a, b = 2;  
a = &b;  
*a = 3;  
*a = b+3;
```

Koja je vrednost promenljive b?

Konverzije

- Pokazivači se mogu eksplicitno (pa čak i implicitno) konvertovati iz jednog u drugi pokazivački tip.
- Na primer:
`int a; char* p = (char*)&a;`
ili čak `int a; char* p = &a;`
- Prilikom konverzije pokazivača vrši se samo prosta dodela adrese, bez ikakve konverzije podataka na koje pokazivač ukazuje, što često može dovesti do neželjenih efekata.
- Ovakve konverzije treba pažljivo koristiti!

Šta ispisuje naredni kod?

```
int* p ;  
float f = 5;  
p = &f;  
printf("%d\n", *p);
```

Šta ispisuje naredni kod?

```
int* p ;  
float f = 5;  
p = &f;  
printf("%d\n", *p);
```

Prilikom prevodenja biće dato upozorenje warning: assignment from incompatible pointer type. Program će ispisati 1084227584 — broj koji odgovara zapisu u pokretnom zarezu broja 5

Primer

```
int* p ;  
char f = 5;  
p = &f;  
printf("%d\n", *p);
```

Primer

```
int* p ;  
char f = 5;  
p = &f;  
printf("%d\n", *p);
```

Prilikom prevodenja biće dato upozorenje warning: assignment from incompatible pointer type. Program će ispisati vrednosti koje zavise od trenutnog stanja memorije

Pokazivač na void

- U nekim slučajevima, poželjno je imati mogućnost „opšteg“ pokazivača, tj. pokazivača koji može da ukazuje na promenljive različitih tipova.
- Za to se koristi tip `void*`.

```
void* p;  
int a = 3, b = 5, c;  
float f = 3.0, g;  
p = &a;  
c = *(int*)p + b;  
p = &f;  
g = *(float*)p + 1;
```


const

- I na pokazivače se može primeniti ključna reč const.

```
/*promenljiva i*/  
int i;
```

```
/*vrednost od a je 5 i ne moze se menjati*/  
const int a = 5;
```

```
/*pokazivac na konstantan int, npr p=&a; */  
const int* p;
```

```
/*vrednost p1 ne moze se menjati*/  
int* const p1 = &i;
```

```
/*p2 se ne moze menjati, pokazuje na const int*/  
const int* const p2 = &a;
```

Plitko i duboko kopiranje

- Vrednost pokazivača može se dodeliti pokazivaču istog tipa.
- Kako izgleda memorija?
- Taj način kopiranja nekada se naziva *plitko kopiranje*.
- *Duboko kopiranje* podrazumeva da se napravi kopija sadržaja koji se nalazi na adresi na koju pokazivač pokazuje, a da se pokazivač `q` usmeri ka kopiji tog sadržaja.
- O plitkom i dubokom kopiranju više u drugom semestru

Primer

```
int a = 5;  
int *p, *q;  
p = &a;  
q = p;  
*p = 2;  
*q = *p + 5;
```

Koja je vrednost promenljive a?

Kako izgleda memorija?

Pokazivači i argumenti funkcija

- Argumenti funkcija se uvek prenose po vrednosti

```
#include <stdio.h>
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
int main() {
    int x = 3, y = 5;
    swap(x, y);
    printf("x = %d, y = %d\n", x, y);
}
```

Pokazivači i argumenti funkcija

- Ukoliko se želi da funkcija `swap` zaista zameni vrednosti argumentima, onda ona mora biti definisana drugačije:

```
void swap(int *pa, int *pb) {  
    int tmp = *pa;  
    *pa = *pb;  
    *pb = tmp;  
}
```

- Zbog drugačijeg tipa argumenata, funkcija `swap` više ne može biti pozvana na isti način kao ranije — `swap(x, y)`.
- Njeni argumenti nisu tipa `int`, već su pokazivačkog tipa `int*`.
- Zato se kao argumenti ne mogu koristiti vrednosti `x` i `y` već njihove adrese — `&x` i `&y`, tj poziv je `swap(&x, &y)`
- Isti taj mehanizam koristi se i u funkciji `scanf`.

Primer

```
#include <stdio.h>
void deljenje(unsigned a, unsigned b,
              unsigned* pk, unsigned* po) {
    *pk = 0; *po = a;
    while (*po >= b) {
        (*pk)++;
        *po -= b;
    }
}
int main() {
    unsigned k, o;
    deljenje(14, 3, &k, &o);
    printf("%d %d\n", k, o);
    return 0;
}
```

Pokazivači i strukture

- Moguće je definisati i pokazivače na strukture.
- ```
struct razlomak *pa = &a;
(*pa).imenilac = 2;
(*pa).brojilac = 1;
```

Zagrade su neophodne zbog prioriteta operatora
- Umesto kombinacije operatora `*` i `.`, moguće je koristiti operator `->` koji je najvišeg prioriteta.

```
struct razlomak *pa = &a;
pa->imenilac = 2;
pa->brojilac = 1;
```

# Pokazivači i strukture

- Strukture se kao argumenti u funkciju prenose po vrednosti.
- S obzirom na to da strukture često zauzimaju više prostora od elementarnih tipova podataka, čest je običaj da se umesto struktura u funkcije proslede njihove adrese, tj. pokazivači na strukture.

```
void saberi_razlomke(const struct razlomak *pa, const struct razlomak *pb,
 struct razlomak *pc)
{
 pc->brojilac = pa->brojilac*pb->imenilac +
 pa->imenilac*pb->brojilac;
 pc->imenilac = pa->imenilac*pb->imenilac;
}
```



# Pokazivačka aritmetika

- Pokazivač možemo uvećati za neki ceo broj
- Izraz  $p+1$  i slični uključuju izračunavanje koje koristi *pokazivačku aritmetiku* i koje se razlikuje od običnog izračunavanja.
- Naime, izraz  $p+1$  ne označava dodavanje vrednosti 1 na  $p$ , već dodavanje dužine jednog objekta tipa na koji ukazuje  $p$ .
- Na primer, ako je  $p$  tipa  $\text{int}^*$ , onda  $p+1$  i  $p$  mogu da se razlikuju za dva ili četiri, tj za  $\text{sizeof}(\text{int})$ .
- Na primer, ako  $p$  sadrži adresu 100, i ako je  $\text{sizeof}(\text{int})$  jednako 4, vrednost  $p+3$  će biti adresa  $100 + 3 \cdot 4 = 112$ .

# Pokazivačka aritmetika

- Od pokazivača je moguće oduzimati cele brojeve (na primer,  $p-n$ ), pri čemu je značenje ovih izraza analogno značenju u slučaju sabiranja.
- Na pokazivače je moguće primenjivati prefiksne i postfiksne operatore  $++$  i  $--$ , sa sličnom semantikom.
- Dva pokazivača je moguće oduzimati. I u tom slučaju se ne vrši prosto oduzimanje dve adrese, već se razmatra veličina tipa pokazivača, sa kojom se razlika deli.
- Dva pokazivača nije moguće sabirati.

## Operatori poredjenja

- Nad pokazivačima se (ako su istog tipa) mogu primenjivati i relacijski operatori (na primer,  $p1 < p2$ ,  $p1 == p2$ , ...).
- Na primer,  $p1 < p2$  je tačno akko  $p1$  ukazuje na raniji element (u smislu linearno uređene memorije) niza od pokazivača  $p2$ .

## Prioriteti operatora

- Unarni operatori `&` i `*` imaju viši prioritet nego binarni aritmetički operatori.
- Zato je značenje izraza `*p+1` zbir sadržaja lokacije na koju ukazuje `p` i vrednosti 1 (a ne sadržaj na adresi `p+1`).
- Unarni operatori `&`, `*`, i prefiksni `++` se primenjuju zdesna nalevo, pa naredba `++*p`; inkrementira sadržaj lokacije na koju ukazuje `p`.
- Postfiksni operator `++`, kao i svi unarni operatori koji se primenjuju sleva na desno, imaju viši prioritet od unarnih operatora koji se primenjuju zdesna nalevo, pa `*p++` vraća sadržaj na lokaciji `p`, dok se kao bočni efekat `p` inkrementira.

# Primer

```
int a = 3, *p;
p = &a;
*p = *p+1;
++*p;
```

Koja je vrednost promenljive a?

# Nizovi

- Postoji čvrsta veza između pokazivača i nizova
- Deklaracija `int a[10]`; deklariše niz od 10 elemenata tipa `int`.
- Izraz `sizeof(a)` imaće istu vrednost kao izraz `10*sizeof(int)`.
- Početni element niza je `a[0]`, a deseti element je `a[9]` i oni su u memoriji poređani uzastopno.
- U fazi kompilacije, imenu niza `a` pridružena je informacija o adresi početnog elementa niza, o tipu elemenata niza, kao i o broju elemenata niza.

# Pokazivači i nizovi

- Vrednost a nije pokazivačkog tipa, ali mu je vrlo bliska.
- Za razliku od pokazivača koji jesu l-vrednosti imena nizova to nisu (jer a uvek ukazuje na isti prostor koji je rezervisan za elemente niza).

```
int a[50];
int b[50];
a = b; /*neispravna dodela*/
```

# Pokazivači i nizovi

- Ime jednodimenzionog niza biće, sem ako je navedeno kao argument `sizeof` ili `&` operatora, implicitno konvertovano u pokazivač na prvi element niza.

```

a <=> &a[0] *a <=> a[0]
a+1 <=> &a[1] *(a+1) <=> a[1]
...
a+i <=> &a[i] *(a+i) <=> a[i]
...
a+n-1 <=> &a[n-1] *(a+n-1) <=> a[n-1]

```

- Nema provere granica niza, pa je dozvoljeno pisati (tj. prolazi fazu prevođenja) i `a[100]`, `*(a+100)`, `a[-100]`, `*(a-100)`, iako je veličina niza samo 10.



## Pokazivači i nizovi

- Kao što se elementima niza može pristupiti korišćenjem pokazivačke aritmetike, 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).
- Na primer, ako je `sizeof(int)` jednako 4 i ako pokazivač `p` ukazuje na adresu 100 na kojoj počinje niz celih brojeva, tada je `p[3]` ceo broj koji se nalazi smešten u memoriji počevši od adrese 112.
- Isti broj se može dobiti i izrazom `*(p+3)` u kome se koristi pokazivačka aritmetika.

# Pokazivači i nizovi

- Dakle, bez obzira da li je  $x$  pokazivač ili niz, važi  $x[n] \Leftrightarrow *(x+n)$  i  $x+n \Leftrightarrow \&x[n]$ .
- Ovako tesna veza pokazivača i nizova daje puno prednosti
- Na primer, iako funkcije ne primaju niz već samo pokazivač na prvi element, implementacija tih funkcija može to da zanemari i sve vreme da koristi indeksni pristup kao da je u pitanju niz, a ne pokazivač
- Međutim, ovo je i čest izvor grešaka:

```
int a[10];
int *b;
a[3] = 5; b[3] = 8;
```

# Pokazivači i nizovi

- Izraz `a` ima vrednost adrese početnog elementa i tip `int [10]` koji se, po potrebi, može konvertovati u `int *`.
- 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`.

```
int a[10]; /*niz celobrojnih vrednosti*/
int *b[10]; /*niz pokazivaca*/
int (*c)[10]; /*pokazivac na niz sa 10 elemenata*/
```

```
c = &a;
(*c)[0] = 0; /*a[0] = 0;*/
```

# Pokazivači i nizovi

- Niz se ne može preneti kao argument funkcije — kao argument funkcije se može navesti ime niza i time se prenosi samo pokazivač koji ukazuje na početak niza.
- Funkcija koja prihvata takav argument za njegov tip može da ima `char a[]` ili `char *a`.
- Ovim se u funkciju kao argument prenosi (kao i uvek — po vrednosti) samo adresa početka niza, ali ne i informacija o dužini niza.

## Primer

```

#include <stdio.h>
int main() {
 int a[] = {8, 7, 6, 5, 4, 3, 2, 1}; /* Odnos izmedju prioriteta
 int *p1, *p2, *p3; * i ++ */
 /* Ispis elemenata niza */ *p1++; printf("%d ", *p1);
 printf("%d %d %d\n", a[0], a[3], a[5]); *(p1++); printf("%d ", *p1);
 (*p1)++; printf("%d\n", *p1);
 /* Inicijalizacija pokazivaca - return 0;
 ime niza se konvertuje u pokazivac */ }
 p1 = a; p2 = &a[3]; p3 = a + 5;
 printf("%d %d %d\n", *p1, *p2, *p3); 8 5 3
 8 5 3
 /* Pokazivaci se koriste u nizovskoj 7 3 4
 sintaksi */ 7 6 1 6
 printf("%d %d %d\n", p1[1], p2[2], p3[-1]); 6 5 6

 /* Pokazivacka aritmetika */
 p1++; --p2; p3 += 2; /* a++ nije dozvoljeno */
 printf("%d %d %d %lu\n", *p1, *p2, *p3, p3 - p1);

```

## Pokazivači na funkcije

- Funkcije se ne mogu direktno prosleđivati kao argumenti drugim funkcijama, vraćati kao rezultat funkcija i ne mogu se dodeljivati promenljivima.
- Ipak, ove operacije je moguće posredno izvršiti ukoliko se koriste pokazivači na funkcije.
- Više o tome u okviru kursa Programiranje 2

## Dinamička alokacija memorije

- U većini realnih aplikacija, u trenutku pisanja programa nije moguće precizno predvideti memorijske zahteve programa.
- Naime, memorijski zahtevi zavise od interakcije sa korisnikom i tek u fazi izvršavanja programa korisnik svojim akcijama implicitno određuje potrebne memorijske zahteve (na primer, koliko elemenata nekog niza će biti korišćeno).
- Rešenje ovih problema je *dinamička alokacija memorije* koja omogućava da program u toku svog rada, u fazi izvršavanja, zahteva (od operativnog sistema) određenu količinu memorije.
- Više o tome u okviru kursa Programiranje 2

# Literatura

Slajdovi su pripremljeni na osnovu materijala iz desetog poglavlja knjige:

Predrag Janičić, Filip Marić: Programiranje 1.

Nastali su dopunom slajdova prof. dr Milene Vujošević Janičić.

Za pripremu ispita nisu dovoljni slajdovi, potrebno je koristiti knjigu!