Pokazivači i nizovi

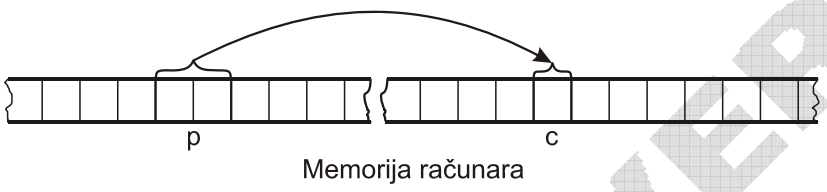
Pokazivač (pointer) je promenljiva koja sadrži adresu neke druge promenljive. Pokazivači i nizovi su u programskom jeziku C veoma bliski, pa se iz tog razloga u ovom poglavlju opisuju zajedno.

Prilikom programiranja pokazivači mogu biti veoma moćan alat, koji omogućava pisanje kompaktnijeg i efikasnijeg koda. Međutim, njihovo nepravilno korišćenje može ozbiljno narušiti čitljivost i funkcionalnost programa.

Pokazivači i adrese

Memoriju računara možemo posmatrati kao niz numerisanih (adresiranih) memorijskih ćelija, kojima se može pristupati pojedinačno ili u povezanim grupama. Najmanja memorijska jedinica (ćelija) kojoj se može pritupiti je **bajt**. Jedan bajt memorije je dovoljan za čuvanje jednog **char** podatka, dva bajta mogu čuvati **short** celobrojni tip podatka, a četiri uzastopna bajta mogu da formiraju **long**. Slično, za smeštanje pokazivača se najčešće koriste grupe od dve ili četiri uzastopne ćelija (bajta).

Na sledećoj slici je shematski prikazana situacija u memoriji kada pokazivač pokazuje na promenljivu tipa char.



Slika ### Pokazivač pokazuje na promenljivu tipa char

Da bismo dobili adresu neke promenljive u memoriji, koristimo operator **&**. Tako izraz

```
p=&c;
```

pokazivaču *p* dodeljuje adresu promenljive *c* i tada kažemo da *p* „pokazuje na“ *c*.

Ukoliko imamo pokazivač na neku promenljivu, toj promenljivoj možemo pristupiti korišćenjem operatora *****, koji se još naziva i operator *dereferenciranja*. Na sledećem primeru možemo videti kako se deklariše pointer i kako se koriste operatori & i *.

```
int x = 5, y = 1, z[20];
int *p;   /* p je pokazivac na int */

p = &x;   /* p sada pokazuje na x */
y = *p;   /* y sada ima vrednost 5 */
*p = 0;   /* x sada ima vrednost 0 */
p = &z[0]    /* p sada pokazuje na z[0] */
```

U prethodnom primeru smo videli da se pokazivač *p* deklariše pomoću

```
int *p;
```

što bi moglo da se čita kao „p pokazuje na int“, ili „*p je int“. Iz ovoga zaključujemo da se pokazivač deklariše kao pokazivač na tačno određeni tip podatka i ne može se koristiti za pokazivanje na neki drugi tip.

Ukoliko pokazivač pokazuje na celobrojnu promenljivu *x*, **p* se može pojaviti u bilo kom kontekstu gde bi se moglo pojaviti i *x*. Na primer:

```
*p = *p + 12;
```

uvećava **p* za 12, a samim tim uvećava i *x* za 12.

Unarni operatori & i * imaju veći prioritet od aritmetičkih operatora, tako da

```
y = *p + 5;
```

prvo uzima vrednost na koju pokazuje *p*, uvećava je za 5 i dodeljuje promenljivoj *y*, dok svaka od linija

```
*p += 1;
++*p;
(*p)++;
```

uvećava **p* za jedan.

S obzirom da su pokazivači promenljive kao i sve druge, moguće je njihovo korišćenje i bez dereferenciranja. Jedan od primera je i dodeljivanje vrednosti jednog pokazivača drugom:

```
q = p;
```

čime se vrši kopiranje sadržaja pokazivača *p* u pokazivač *q*, tako da sada pokazivač *q* pokazuje na istu memorijsku lokaciju, na koju pokazuje i pokazivač *p*.

Pokazivači kao argumenti funkcija

Imajući u vidu da C funkcijama šalje argumente preko vrednosti, ne postoji direktan način da funkcija koja je pozvana promeni promenljive u funkciji koja ju je pozvala. Na primer, funkcija za sortiranje treba da međusobno zameni vrednosti dvema promenljivama korišćenjem funkcije *zameni*, koja je definisana kao

```
void zameni(int a, int b)        /* POGRESNO */
{
   int pom;

   pom = a;
   a = b;
   b = pom;
}
```

pri čemu bi poziv funkcije bio

```
zameni(x, y);
```

Ovako definisana funkcija kopira vrednosti koje su joj poslate u svoje privremene lokalne promenljive *a* i *b*. Iako funkcija vrši promenu podataka u promenljivim *a* i *b*, ta promena ne utiče na prosleđene promenljive *x* i *y*, s obzirom da se funkciji argumenti prosledjuju preko njihovih vrednosti.

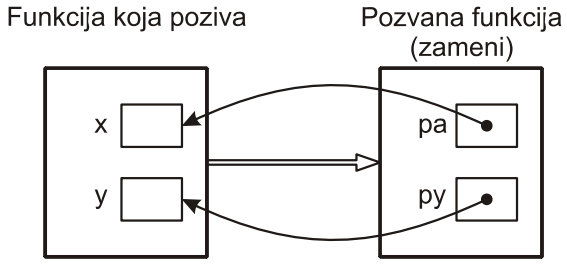
Na sreću, ovaj problem je moguće rešiti korišćenjem pokazivača. Umesto slanja vrednosti određenih promenljivih, funkciji se šalju adrese, odnosno pokazivači na te promenljive. Na ovaj način pozvana funkcija može korišćenjem prosledjenih pokazivača posredno da pristupi ovim promenljivim i da promeni njihove vrednosti. U tom slučaju funkcija *zameni* bi izgledala ovako:

```
void zameni(int *pa, int *pb)          /* ISPRAVNO */
{
   int pom;

   pom = *pa;
   *pa = *pb;
   *pb = pom;
}
```

pri čemu bi poziv funkcije bio

```
zameni(&x, &y);
```



Slika ### Shematski prikaz prosleđivanja pokazivača funkciji

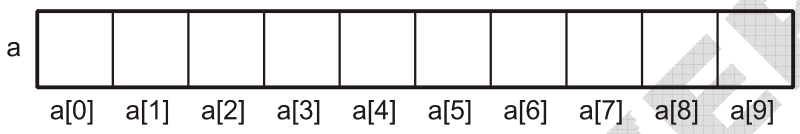
Pokazivači i nizovi

U programskom jeziku C postoji čvrsta veza između pokazivača i nizova, tako da se o njima može govoriti istovremeno. Svaka operacija koja se može obaviti korišćenjem nizova, može se obaviti i pomoću pokazivača. Varijanta koja koristi pokazivače je brža, ali teža za razumevanje.

Deklaracija

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

definiše niz *a* veličine 10, odnosno niz od 10 uzastopnih celih brojeva *a[0]*, *a[1]*, ..., *a[9]*.



Slika ### Shematski prikaz niza *a* u memoriji računara.

Notacija *a[i]* ukazuje na *i*-ti element niza.

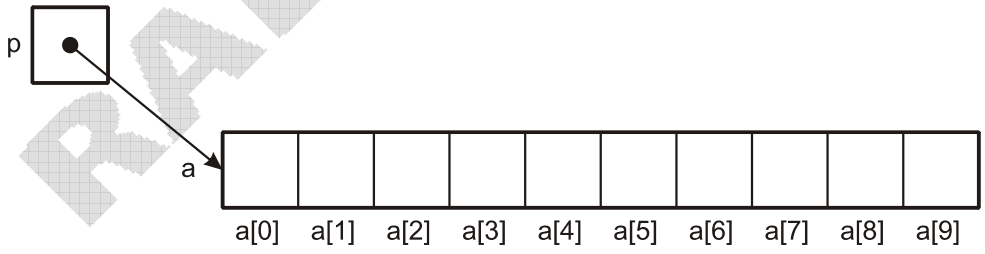
Ukoliko je *p* pokazivač na ceo broj, deklarisan kao

```
int *p;
```

onda linija

```
p = &a[0];
```

postavlja pokazivač *p* da pokazuje na nulti element niza *a* (indeksi nizova u C-u počinju od nule), ili drugačije rečeno pokazivač *p* sadrži adresu elementa *a[0]*.



Slika ### Pokazivač *p* pokazuje na nulti element niza *a*

Sada linija

```
x = *p;
```

dodeljuje promenljivoj *x* vrednost elementa *a[0]*.

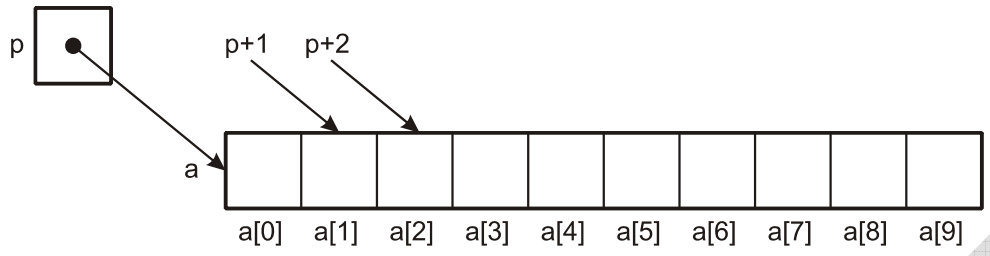
Ukoliko pokazivač *p* pokazuje na određeni element niza, po definiciji *p+1* pokazuje na sledeći element, *p+i* pokazuje na *i*-ti element iza *p*, a *p-i* pokazuje na *i*-ti element ispred *p*. Tako, ukoliko *p* pokazuje na *a[0]*, onda

`*(p+1)`

predstavlja sadržaj elementa *a[1]*, a

`*(p+i)`

sadržaj elementa *a[i]*.



Slika ### Pristupanje elementima niza pomoću pokazivača

Ovakav način pristupanja elementima niza može se koristiti bez obzira na tip elemenata niza, tako da notacija *p+1* uvek predstavlja sledeći element niza, a *p+i* predstavlja *i*-ti element iza *p*, bilo da se radi o nizu celih brojeva ili nekom drugom nizu.

Indeksiranje nizova i aritmetika pointera su veoma bliski. Po definiciji, vrednost promenljive kojom se označava niz je upravo adresa nultog elementa niza. Zbog toga nakon izvršenja linije

```
p = &a[0];
```

p i *a* imaju jednake vrednosti, tako da prethodna linija može biti napisana i kao

```
p = a;
```

Imajući ovo u vidu, ne iznenađuje činjenica da se *a[i]* može napisati kao **(a+i)*. Ukoliko primenimo operator & na oba ova oblika, dobijamo da je *&a[i]* isto što i *a+i*, tj. adresa *i*-tog elementa niza. Slično važi i za pokazivač *p*, pa je *p[i]* isto što i **(p+i)*.

Iako na prvi pogled izgleda da su pokazivači i nizovi jedno te isto i da imaju potpuno iste osobine, postoji jedan detalj koji ih razlikuje. Naime, kao što je ranije pomenuto, pokazivači su promenljive, pa su i izrazi *p=a* i *p++* dozvoljeni. Međutim, sam naziv niza nije promenljiva i na njega se ne mogu primeniti izrazi tipa *a=p* i *a++*.

Kada se funkciji prosleđuje naziv niza, ono što se zapravo šalje je adresa nultog elementa. Unutar pozvane funkcije ovaj argument je lokalna promenljiva, a parametar koji predstavlja niz je pokazivač, odnosno promenljiva koja sadrži adresu nultog elementa niza. Posmatrajmo funkciju za određivanje dužine stringa (niza karaktera):

```
/* duzina: vraca duzinu stringa s */
int duzina(char *s)
{
  int n;

  for(n = 0; *s != '\0'; s++)   n++;

  return n;
}
```

Imajući u vidu da je *s* pokazivač, njegovo povećavanje korišćenjem *s++* je dopušteno. Pošto je *s* lokalna promenljiva funkcije *duzina*, njegova promena ne utiče na string koji je prosleđen iz funkcije koja je pozvala funkciju *duzina*. Zahvaljujući tome mogući su sledeći pozivi funkcije *duzina*:

```
duzina("Zdravo, drugari!");      /* konstantan string */
duzina(niz);                     /* char niz[100]; */
duzina(pok);                     /* char *pok */
```

U definiciji funkcija dozvoljeno je deklarisanje formalnih parametara na oba načina; i kao niza i kao pokazivača. Deklaracije

```
char s[]
```

i

```
char *s
```

su ekvivalentne.

Funkciji je moguće proslediti i deo nekog niza tako što bi joj se prosledio pokazivač na podniz. Tako, ukoliko je *a* niz, pozivi

```
fun(&a[5]);
```

i

```
fun(a+5);
```

prosleđuju funkciji *fun* podniz koji počinje od petog elementa niza *a*.