Pokazivači

U dosadašnjem radu smo imali prilike da koristimo promenljive koje smo deklarisali na početku nekog bloka. Prilikom deklaracije promenljiva dobija jedinstveni naziv i odgovarajući prostor u memoriji u kome će biti smeštena njena vrednost. Vrednost neke promenljive se može pročitati ili zapisati korišćenjem naziva te promenljive. Ovako deklarisane promenljive su vidljive samo u okviru bloka u kome su deklarisane, a njihov životni vek počinje istovremeno sa početkom izvršavanja tog bloka. Nakon završetka izvršavanja određenog bloka, sve promenljive deklarisane u njemu se automatski brišu iz memorije.

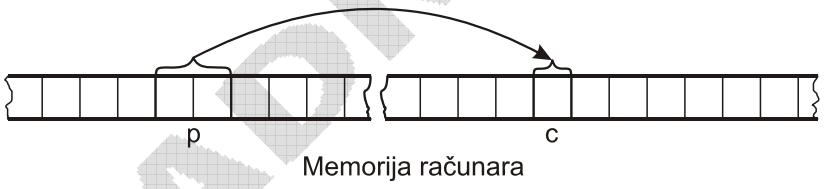
Promenljive deklarisane na ovaj način često nazivamo **statičkim** promenljivama, a ovu vrstu obezbeđivanja memorije (alokacija memorije) nazivamo statička alokacija. Nedostatak statičkog alociranja je nemogućnost da se za vreme izvršavanja programa promeni adresa ili veličina memorijskog prostora koji je dodeljen promenljivoj. Na primer, prilikom deklarisanja nizova neophodno je da u trenutku pravljenja programa znate maksimalan broj elemenata, što često nije slučaj. U realnim problemima najčešće je potrebno alocirati memorijski prostor određene veličine na osnovu informacije koja nije poznata prilikom pravljenja programa, već je ona rezultat izvršavanja programa. Takav način alociranja memorije se naziva **dinamičko** alociranje. Dinamičko alociranje moguće je ostvariti korišćenjem posebne vrste promenljivih koje nazivamo **pokazivačima** ili **pointerima**.

Pokazivači i memorija

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**. U zavisnosti od kompajlera jedan bajt memorije je dovoljan za čuvanje jednog **char** podatka, dva bajta mogu čuvati **integer**, a četiri uzastopna bajta mogu da formiraju **real**. Slično, za smeštanje pokazivača se najčešće koriste grupe od dve ili četiri uzastopne ćelija (bajta).

Pokazivač (pointer) je promenljiva koja sadrži adresu neke druge promenljive. 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.

S obzirom da pokazivač predstavlja adresu određene promenljive u memoriji, često kažemo da pokazivač pokazuje na neku promenljivu. Na sledećoj slici je shematski prikazana situacija u memoriji kada pokazivač pokazuje na promenljivu tipa char.



Slika ### Pokazivač pokazuje na promenljivu tipa char

Pokazivači u Pascal-u

U programskom jeziku Pascal pokazivače deklarišemo na sličan način kao i sve druge promenljive, sa tom razlikom što ispred tipa promenljive dodajemo i znak ^. Navedeni tip sada označava tip podatka na koji pokazivač pokazuje. Kada jednom deklarišemo pokazivač na određeni tip podatka, ovaj tip više nije moguće menjati i pokazivač može pokazivati samo na promenljive tog tipa.

Sintaksa

```
type <pokazivacki_tip>=^<tip_podatka_na_koji_pokazuje>;
```

odnosno

```
var  <pokazivacka_promenljiva>:^<tip_podatka_na_koji_pokazuje>;
```

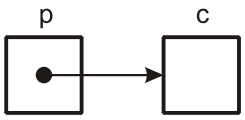
Primer

```
type  pokazivac=^integer;
      knjiga= record
                 naslov:string[30];
                 autor:string[20];
                 broj_strana:integer;
              end;
var   p:pokazivac;
      q:^integer;
      pok:^char;
      pk:^knjiga;
```

Da bismo dobili adresu neke promenljive u memoriji, koristimo operator @. Tako linija

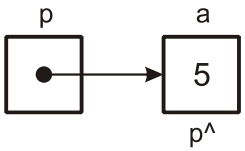
```
p:=@c;
```

pokazivaču *p* dodeljuje adresu promenljive *c* i tada kažemo da *p* „pokazuje na" *c*, kao što je prikazano na Slici ###.



Slika ### Pokazivač *p* pokazuje na promenljivu *c*

Ukoliko imamo pokazivač na neku promenljivu, toj promenljivoj možemo pristupiti korišćenjem operatora ^, koji se još naziva i operator *dereferenciranja*. Pretpostavimo da imamo pokazivač *p* koji pokazuje na celobrojnu promenljivu *a*, kao što je prikazano na Slici ###.



Slika ### Pokazivač *p* pokazuje na celobrojnu promenljivu *a*

U tom slučaju linija

```
writeln(p^);
```

štampa sadržaj memorijske lokacije na koju pokazuje *p*, što je u ovom slučaju broj 5, a linija

```
p^:=3;
```

ovoj memorijskoj lokaciji dodeljuje vrednost 3, čime je posredno i promenljiva *a* dobila vrednost 3.

Pokazivači na slogove se ponašaju na potpuno isti način kao i pokazivači na proste tipove podataka. Ukoliko je, na primer, *pk* pokazivač na slog *knjiga* iz prethodnog primera, onda bi se konkretnom slogu pristupalo sa *pk*^, a članici sloga sa *pk*^.*naslov*.

Na sledećem primeru možemo videti kako se deklarišu pokazivači i kako se mogu koristiti operatori @ i ^.

```
var   x,y:integer;
      z:array[1..20] of integer;
      k:knjiga;
      p:^integer; /* p je pokazivac na int */
      pk:^knjiga;
      ...
x:=5;
y:=1;
p:=@x;     /* p sada pokazuje na x */
```

```
y:=p^;     /* y sada ima vrednost 5 */
p^:=0;     /* x sada ima vrednost 0 */
p:=@z[5];    /* p sada pokazuje na z[5] */
pk:=@k;      /* pk sada pokazuje na knjigu k */
pk^.naslov:='Na Drini cuprija';  /* naslov knjige na koju pokazuje pk je Na... */
```

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

```
var  p:^integer;
```

što bi moglo da se čita kao „p pokazuje na integer“, ili „p^ je integer“. 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

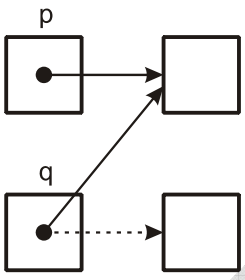
```
y:=p^+5;
```

prvo uzima vrednost na koju pokazuje *p*, uvećava je za 5 i dodeljuje promenljivoj *y*.

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*, kao što je prikazano na Slici ###.

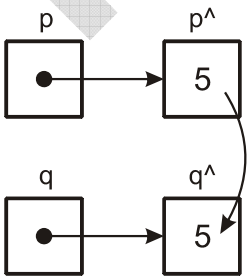


Slika ### Dodeljivanje vrednosti pokazivača *p* pokazivaču *q*

Za razliku od toga, linija

```
q^:=p^;
```

podatku na koji pokazuje pokazivač *q* dodeljuje vrednost na koju pokazuje pokazivač *p*, kao na Slici ###.



Slika ### Dodeljivanje vrednosti na koju pokazuje *p* podatku na koji pokazuje *q*.

Dinamičko alociranje memorije

U prethodnoj sekciji smo se sretali sa pokazivačima na neke postojeće, statički deklarisane promenljive. Korišćenjem ovih pokazivača posredno je moguće dodeliti ili očitati vrednost promenljivih na koju pokazivači pokazuju. Međutim, najveća prednost korišćenja pokazivača je to što se njima može dodeliti odgovarajući memorijski prostor bilo kada tokom izvršenja programa, nezavisno od statički deklarisanih promenljivih.

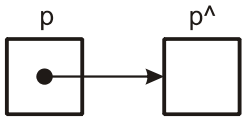
Alociranje novog memorijskog prostora na koji će pokazivati određeni pokazivač vrši se korišćenjem naredbe **new**. Nareba new obezbeđuje memorijski prostor čija veličina odgovara tipu podatka na koji pokazivač naveden unutar naredbe pokazuje. Na primer, ukoliko je pokazivač *p* deklarisan kao pokazivač na integer

```
var  p:^integer;
```

onda će linija

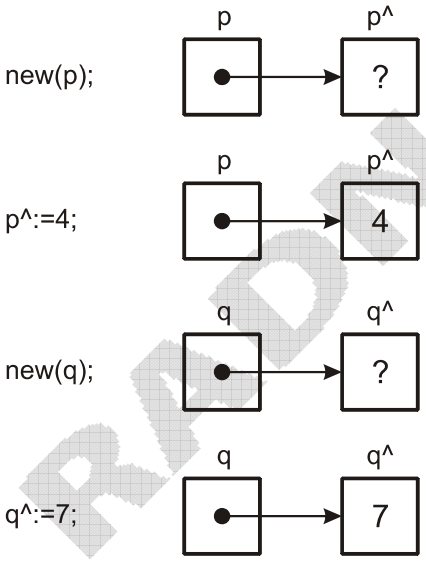
```
new(p);
```

obezbediti memorijski prostor za jedan ceo broj, a adresu te memorijske lokacije dodeliti pokazivaču *p*, kao što je prikazano na Slici ###



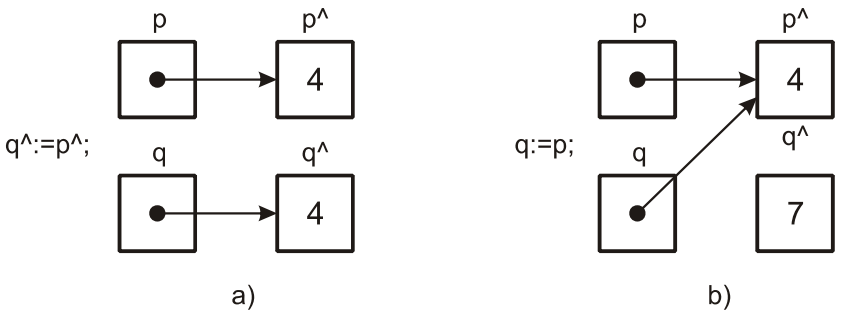
Slika ### Dodeljivanje memorijskog prostora korišćenjem naredbe *new*

Od tog trenutka pokazivač *p* pokazuje na celobrojni podatak u memoriji, kome se može pristupiti korišćenjem operatora ^. Na Slici ### dat je grafički prikaz različitih operacija nad pokazivačima.



Slika ### Šematski prikaz rezultata različitih operacija nad pokazivačima

Kao što smo ranije pomenuli vrednost na koju pokazuje jedan pokazivač se može dodeliti podatku na koji pokazuje drugi pokazivač (Slika ###a). Takođe, pokazivaču se može dodeliti vrednost drugog pokazivača, čime se on praktično usmerava da pokazuje na isti podatak kao i drugi pokazivač (Slika ###).



Slika ### Dodeljivanje pokazivača po vrednosti (a) i po adresi (referenci) (b)

Na Slici ###b smo mogli videti da nakon dodeljivanja vrednosti pokazivača *p* pokazivaču *q*, ne postoji pokazivač koji pokazuje na podatak na koji je do tada pokazivao pokazivač *q*. To znači da od tog trenutka ne postoji mogućnost pristupa toj memorijskoj lokaciji. Iako se taj memorijski prostor više ne koristi, on bespotrebno zauzima memoriju. Iz tog razloga potrebno je sve memorijske lokacije koje su zauzete korišćenjem naredbe *new* osloboditi po prestanku potrebe za njima. Oslobađanje zauzete memorije obavlja se korišćenjem naredbe **dispose**:

```
dispose(q);
```

Pozivanjem prethodne naredbe oslobađa se memorija na koju je pokazivao pokazivač *q*, a njegova vrednost postaje nedefinisana.

Napomenimo i to da pokazivač može da ne pokazuje ni na jednu memorijsku lokaciju i tada on ima vrednost **NIL**. S obzirom da pokazivač sa ovom vrednossću ne pokazuje ni na šta, korišćenje operatora ^ će izazvati grešku prilikom izvršavanja programa.