Programski jezici i strukture podataka

8

STRUKTURE PODATAKA

Dinamička alokacija memorije

- **Prilikom definisanja skalarnih promenljivih ili nizova, memorija se zauzima statički**
  - sve potiebe za memorijom moraju biti poznate unapred, pre prevođenja izvornog programa
  - alocira se najčešce više prostora nego što je potrebno
  - prostor se zauzima u statičkoj zoni memorije
  - bilo kakva izmena kapaciteta zahteva ponovno prevođenje programa
- **Rešenje - dinamička alokacija memorije**
  - radi se u vreme izvršavanja programa
  - broj i veličinu podataka ne moramo znati unapred
  - memorija se alocira u dinamičkoj zoni memorije (eng. heap)
  - podacima u dinamičkoj zoni memorije se pristupa preko pokazivača
  - ograničenje samo ukupan raspoloživ memorijski prostor u sistemu
- **Odgovornost programera za ispravnu alokaciju i dealokaciju dinamičke memorije!**

Manipulacija dinamičkom memorijom

- Zauzimanje dinamičke memorije

  void* **malloc**(vel)

  - **vel** predstavlja traženu veličinu memorije u bajtovima

  void* **calloc**(n,vel)

  - n predstavlja broj elemenata niza
  - **vel** predstavlja veličinu jednog elementa
  - inicijalizuje zauzeti prostor nulama
- Obe funkcije vraćaju generički (**void**) pokazivač!
  - ovakav pokazivač sadrži samo adresu, a nije poznat tip podatka na koji ukazuje
  - ne može se dereferencirati, niti koristiti u adresnoj aritmeticij, osim poređenja
  - ukoliko želimo da pristupimo memoriji preko njeqa, moramo izvršiti eksplicitnu konverziju (cast) u neki tip

- Linearne strukture podataka
  - Nizovi
  - Spregnute liste
  - Stekovi
  - Redovi
  - Dekovi
- Nelinearne strukture podataka
  - Stabla
  - Grafovi

Linearna struktura podataka

- Zajednički imenitelj za sve linearne strukture je postojanje serije podataka jednodimenzijalnog poretka.
- Za broj elemenata linearne strukture se kaže **dužina** i uobičajeno se obeležava sa n.
- Kada je n=0, linearna struktura je prazna.
- Kada je n>0, linearna struktura ima elemente.
- Prvi element **nema prethodnika**, poslednji element **nema sledbenika**, svi ostali elementi imaju i prethodnika i sledbenika.

Operacije nad linearnom strukturom podataka

- Pristup elementima strukture (0,...,n-1).
- Pretraživanje strukture i vraćanje pozicije identifikovanog elementa.
- Pristupanje vrednosti pojedinog elementa na pisanje ili čitanje.
- Umetanje novog elementa na proizvoljnu poziciju.
- Umetanje novog elementa pre prve pozicije.
- Umetanje novog elementa iza poslednje pozicije.
- Brisanje elementa.
- Brisanje cele strukture.
- Pronalaženje prethodnika ili sledbenika posmatranog elementa.
- Određivanje dužine strukture.
- Spajanje dve ili više struktura u jednu.
- Razdvajanje strukture na dve ili više.

Fizička realizacija koncepta linearne strukture podataka

- Sekvencijalna fizička realizacija.
- Spregnuta fizička realizacija.

- Strukture u obe fizičke realizacije mogu da podrže prethodno navedene operacije, ali sa različitom performansom.
  - Primer pristupa elementu...
  - Primer ubacivanja ili brisanja elementa...

Odabir fizičke realizacije zavisi od namene

- Za specifičnu implementaciju linearne strukture podataka, pre svega, treba detektovati namenu i podskup operacija koje treba realizovati.
- Pogotovo treba obratiti pažnju na izbor fizičke realizacije kod struktura sa specifičnim pravilima pristupa (FIFO, LIFO,...).

NIZ

Niz je linearna struktura podataka koja se sastoji od konačno mnogo elemenata istog skalarnog ili strukturnog tipa.

Svaki element niza je jednako dostupan, i svakom se elementu može pristupati u proizvoljnom redosledu.

Nizovi mogu biti statički ili dinamički.

Vrste nizova

- **Jednodimenzionalni niz** ili **vektor**, za identifikaciju pojedine pozicije u vektoru koristi se indeks, vrednost indeksa u C programskom jeziku uzima vrednosti iz skupa 0,...,n-1 pri čemu je **n** dužina vektora.
- **Višedimenzionalni niz** (dvodimenzionalni se naziva **matrica**), predstavlja generalizaciju vektora.

Operacije sa nizovima

- Operacije matričnog računa
- Operacije po elementima

Smeštanje nizova u memoriji

- Za smeštanje nizova u memoriji se koristi sekvencijalna reprezentacija.
- Logička struktura niza i fizička reprezentacija se poklapaju.
- Niz se implementira kao kontinualni skup susednih memorijskih lokacija.

Dinamički niz

- U statičkoj memoriji uvek zauzimamo maksimalnu potrebnu količinu memorije.
- Statička memorija se koristi za smeštanje malih nizova.
- Za kreiranje nizova tokom izvršavanja koristimo heap.

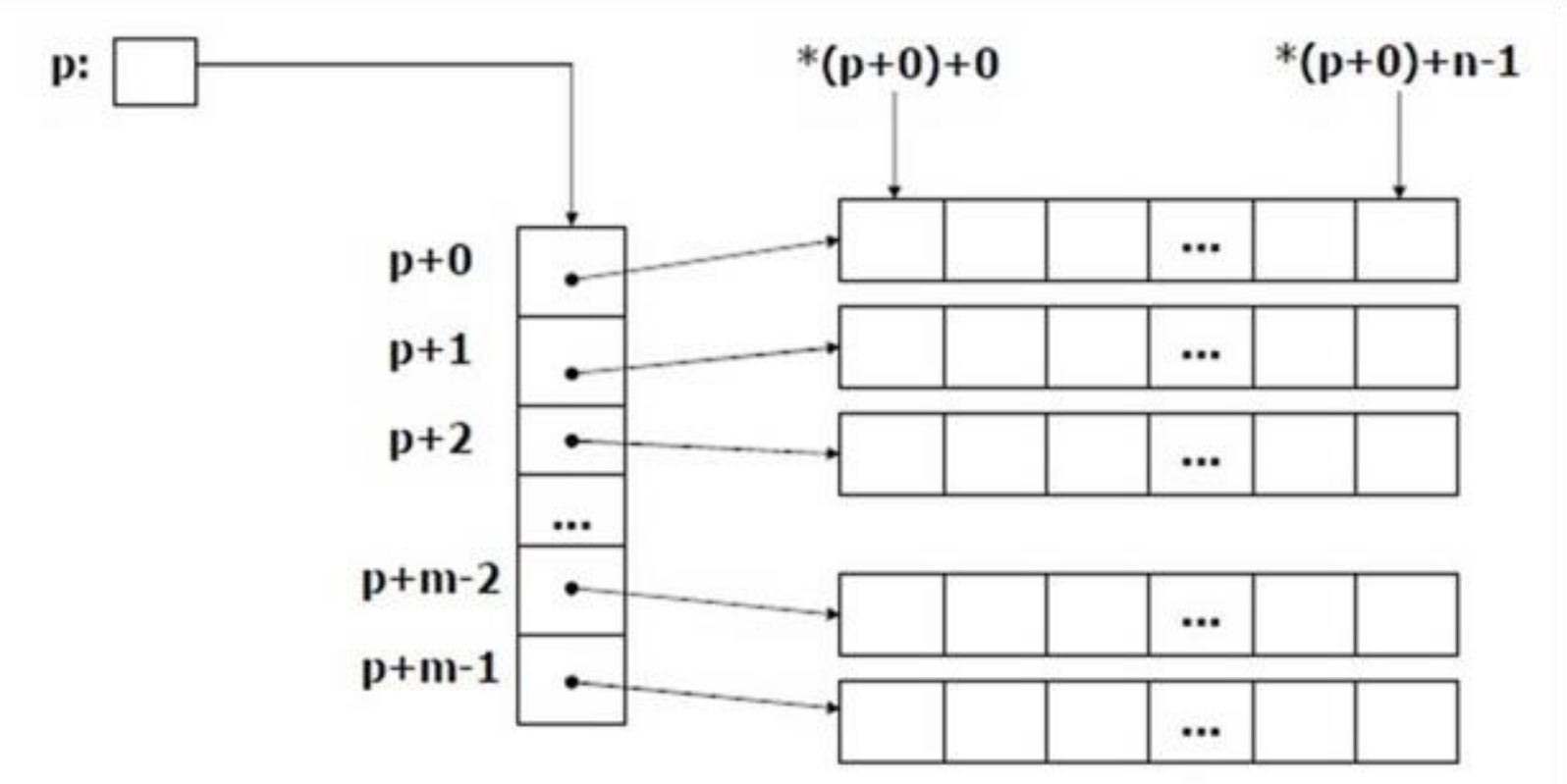
(z38.c)

Dinamička matrica

- Šta ispisuje sledeći program (**z38a.c**):
  - sadržaj matrice vrstu po vrstu, a potom sporednu dijagonalu, sleva-udesno
  - sadržaj matrice vrstu po vrstu, a potom glavnu dijagonalu, sdesna-ulevo
  - sadržaj matrice vrstu po vrstu, a potom sporednu dijagonalu, sdesna-ulevo

Dinamička matrica

- **Matrica se predstavlja kao dinamički niz pokazivača na dinamičke nizove elemenata**



Dinamička matrica

- **Moguće definisanje pokazivača na neki drugi pokazivač, do proizvoljne "dubine"**
  - pokazivač na pokazivač na celobrojni podatak:

```
int **p;
```

- **Prvi korak u stvaranju dinamičke matrice je alokacija memorije za niz pokazivača**

```
p = calloc(m,sizeof(int*));
```

  - p je statički alociran dvostruki pokazivač u koji će biti smeštena adresa niza pokazivača
  - **m** je broj vrsta (redova) matrice
  - svaki pokazivač u alociranom nizu pokazivača će ukazivati na jedan niz elemenata koji će predstavljati vrste (redove) matrice
  - p mora biti dvostruki pokazivač, jer da bi se pristupilo jednom elementu matrice (p [i][j]), mora se prvo odrediti adresa i-te vrste i pristupiti i-tom pokazivaču u nizu pokazivača, a zatim odrediti adresa j-tog elementa u nizu elemenata i pristupiti tom elementu

```
p[i][j] isto što i: *(*(p+i)+j)
```

Dinamička matrica

- **Drugi korak u stvaranju dinamičke matrice je alokacija vrsta matrice**

```
for (i=0; i<m; i++)
*(p+i)= calloc(n, sizeof(int));
```

  - u svakoj iteraciji petlje se alocira prostor za jednu od m vrsta matrice
  - alocirani nizovi koji predstavljaju vrste matrice ne moraju biti fizički susedni u memoriji, već mogu biti razbacani bilo gde
- **Uništavanje (dealociranje) matrice se radi u obrnutom redosledu od redosleda kreiranja**
  - prvo se dealociraju nizovi koji predstavljaju vrste, a zatim niz pokazivača na vrste

```
for (i=0; i<m; i++)
    free (p[i]);
free (p);
```

Dinamička matrica

```
printf("N="); scanf("%d", &n) ;
/*alocira memoriju za n pokazivaca na vrste*/
a = calloc(n, sizeof(int*));
for (i=0; i<n; i++)
{
/*alocira memoriju za n elemenata vrste koji su tipa int*/
      *(a+i)= calloc(n, sizeof(int));
      for (j=0; j<n; j++)
            *(*(a+i)+j) =
rand()/((double)RAND_MAX+1)*10;
}
```

- Biće stvorena kvadratna dinamička matrica dimenzija n x n
  - dimenzija n se unosi sa glavnog ulaza
- Matrica će biti popunjena pseudoslučajnim brojevima u rasponu od 0 do 9

Dinamička matrica

- **Deo koda:**

```
for (i=0; i<n; printf("\n"), i++)
     for (j=0; j<n; j++)
          printf("%d ", *(*(a+i)+j));
```

- **ispisuje sadržaj matrice, vrstu po vrstu**
  - nakon svake iteracije spoljne petlje će u post uslovu biti odštampan znak za novi red
- **Šta radi sledeći deo koda?**

```
for (i=n-1; i>=0; i--)
printf("%d ", *(*(a+i)+n-1-i));
```

- **Uzmimo primer kvadrate matrice za n = 4**

Dinamička matrica

- Izraz: *(*(a+i)+n-1-j));

Za i = 3

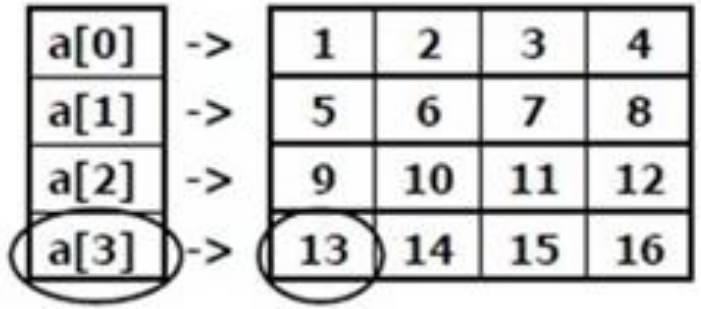
*(a + 3) je isto kao: a[3]

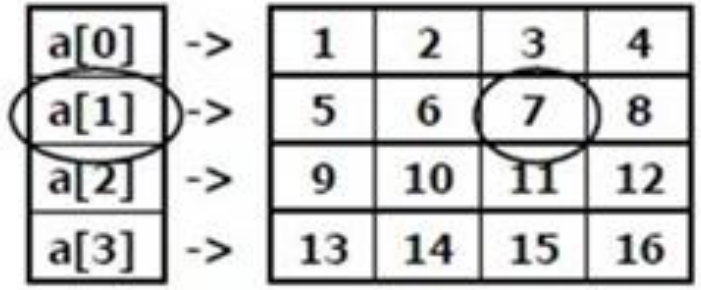
((a + 3) + 4 - 1 - 3) je isto kao: a[3][0]

Za i = 1

*(a + 1) je isto kao: a[1]

((a + 1) + 4 - 1 -1) je isto kao: a[1][2]





- Prednost nizova (kao linearnih struktura podataka) je podudaranje fizičke i logičke strukture, kao i linearna adresna funkcija.
- Nedostatak nizova možemo videti kroz komplikovane operacije umetanja i brisanja elementa na proizvoljnom mestu, a kod sekvencijalne realizacije i kroz statički alociran memorijski prostor.
- Navedeni nedostaci se mogu prevazići spregnutom realizacijom.

Spregnute strukture podataka

- Kod sekvencijalne realizacije linearne strukture podataka uzastopni elementi su susedni u memoriji.
- Kod spregnute fizičke realizacije linearne strukture podataka, elementi mogu biti bilo gde u memoriji.
- Logički linearni poredak se održava tako što elementi liste sadrže eksplicitnu adresu narednog elementa.

Spregnuta lista

- Često je pogodnije da se elementi urede na proizvoljan način pa da se povežu, nego da se fizički poređaju u sekvencu.
- Struktura koja sadrži pokazivač na objekat istog tipa kao što je ona sama naziva se **samo-upućujuća struktura** podataka.
- Samo-upućujuća struktura podataka predstavlja osnovu za spregnute liste, zovemo je ČVOR
- Upotrebom samo-upućujuće strukture realizuju se različite strukture oblika stabla kao i za spregnute realizacije reda i steka.

Spregnuta lista

- Informacioni sadržaj može biti bilo kog skalarnog ili strukturnog tipa.
- Listi se pristupa preko jednog spoljašnjeg pokazivača koji pokazuje na prvi čvor, zovemo ga **GLAVA** liste.

Spregnuta lista

- Lista je dinamička struktura koja je linearna po svakoj relaciji uspostavljenoj među njenim podacima. Zavisno od broja logičkih veza između čvorova, podataka i uspostavljenih relacija, razlikuju se sledeći tipovi lista:
  - Jednostruko spregnuta lista
  - Dvostruko spregnuta lista
  - Multilista
  - Kružna lista

Jednostruko spregnuta lista (linearna lista)

- Jednostruko spregnuta lista organizuje podatke po jednoj relaciji gde pokazivački deo čvora liste ima informaciju samo o sledećem čvoru po toj relaciji.
- Kao primer mogu poslužiti podaci o studentima (prezime, ime i broj indeksa) uređeni po rastućoj vrednosti broja indeksa. Ima mnogo primena.

Definicija

- Jednostruko spregnuta lista se definiše kao uređeni par:

$$P=(S(P),r(P))$$

sa sledećim osobinama:

- struktura je linearna
- dozvoljen je pristup svakom elementu
- moguće je ukloniti bilo koji element
- dozvoljeno je dodati element na bilo kojoj poziciji

Jednostruko spregnuta lista

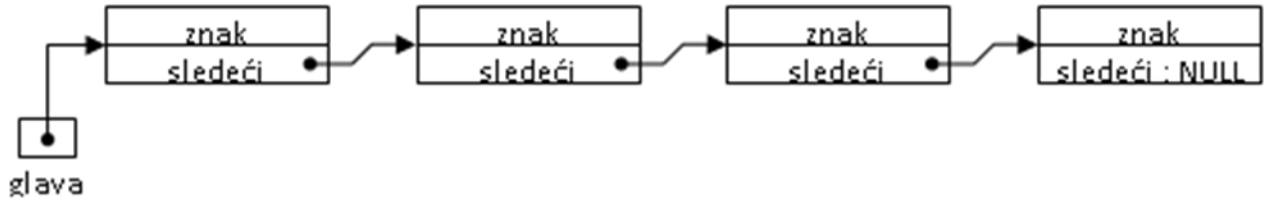
- Jednostruko spregnuta lista je veoma fleksibilna struktura.
- Pristup svakom elementu je dozvoljen (prema poziciji, prema informacionom sadržaju, navigacija).
- Element se može dodati i ukloniti sa bilo kog mesta.
- Moguće je sortiranje liste.

Uređena lista

- Ako se po informacionom sadržaju može definisati **funkcija poređenja** i ako su elementi poređani u rastućem ili opadajućem redosledu, za listu se kaže da je **uređena** (u suprotnom kažemo da je **neuređena**).

Fizička realizacija

- Fizička realizacija je zavisna od namene i u svakom slučaju je spregnuta.

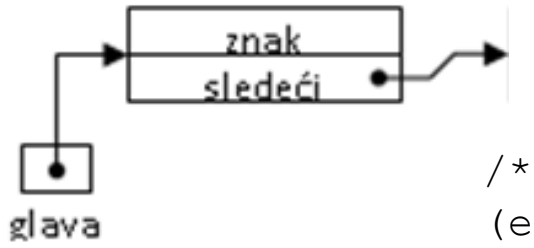


Jednostruko spregnute (povezane) liste

- Jednostruko spregnuta lista je skup čvorova povezanih pokazivačima u jednom smeru.
- Svaki čvor je strukturna promenljiva koja sadrži član koji pokazuje unapred.
- Spregnuta lista može da sadrži promenljivi broj čvorova, što je jedna od osnovnih prednosti spregnutih struktura.

Element liste

- Element liste sastoji se od dva polja:
  - podatak (u ovom slučaju proizvoljan znak)
  - pokazivač na sledeći element liste




```
/* Struktura podataka koja predstavlja slog
(element) liste. */
typedef struct cvor {
                        char znak;
                        struct cvor *sledeci;
                    }Tcvor;
```

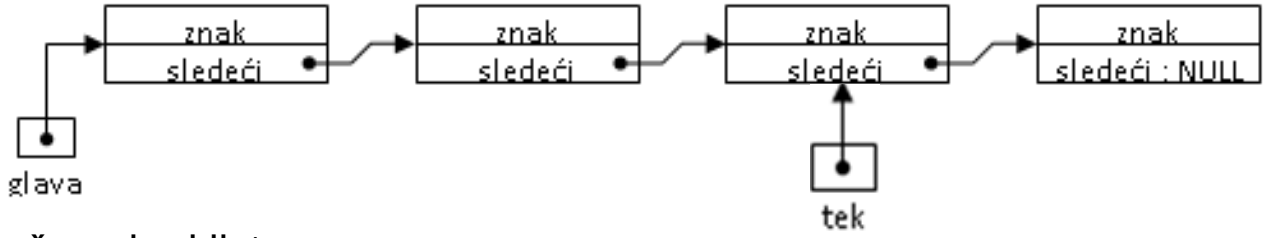
Operacije nad listom

Osnovne operacije koje se mogu izvesti nad listama uključuju sledeće:

- Umetanje čvora u listu
- Brisanje tekućeg čvora iz liste
- Pristup čvoru radi čitanja i upisa

Da bismo mogli pristupati elementima liste potreban nam je pokazivač na prvi element liste, koji se naziva **glava** liste.

Grafička predstava jednostruko spregnute liste u ovom zadatku je:



- NULL označava kraj liste.
- Uopšteno, pokazivačka promenljiva sadrži adresu elementa (sloga liste) na koji pokazuje.

Operacije sa listom su:

1) inicijalizacija liste,
2) unos novog elementa na pocetak liste,
3) listanje liste (prikaz svih elemenata iz liste),
4) brisanje elementa iz liste i
5) brisanje liste (oslobađanje memorije).

1) Inicijalizacija liste predstavlja postavljanje glave liste na NULL.

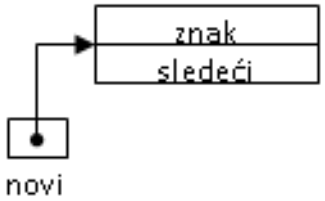
Kada glava liste ima vrednost NULL znači da je lista prazna.

```
*glava=NULL;
```

2) Unos novog elementa u listu može se obaviti dodavanjem elementa na početak liste ili dodavanjem elementa na kraj liste. Dodajemo na kraj.
Sa:

```
novi=(Tcvor *)malloc(sizeof(Tcvor));
```

formira se slog koji je tipa 'Tcvor' i pokazivač 'novi' pokazuje na novoformirani slog.

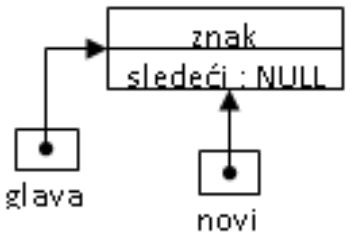


Polja sloga popunjavaju na osnovu sledećih naredbi:

```
c=getc();
novi->znak=c;
novi->sledeci=NULL;
```

- Ako je lista prazna tada se novi element ubacuje kao prvi element liste:

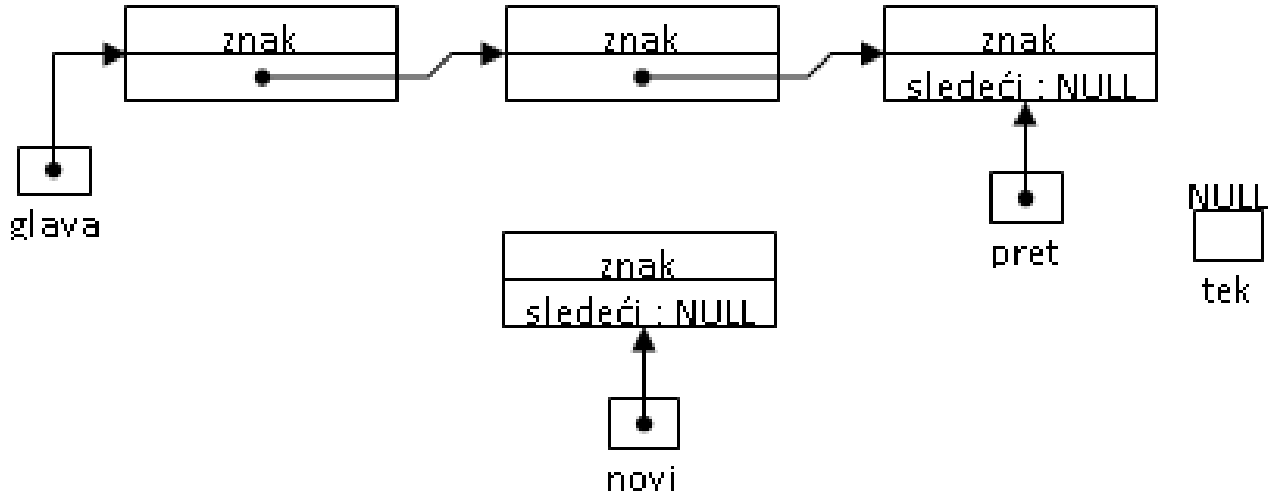
```
if(glava==NULL)
{
        glava=novi;
        return;
}
```



Ako lista već postoji preskočiće se gornja if naredba i unos će se obaviti sledećim delom koda:

```
tek=glava;
pret=glava;
while(tek!=NULL)
{
        pret=tek;
        tek=tek->sledeci;
}
pret->sledeci=novi;
```

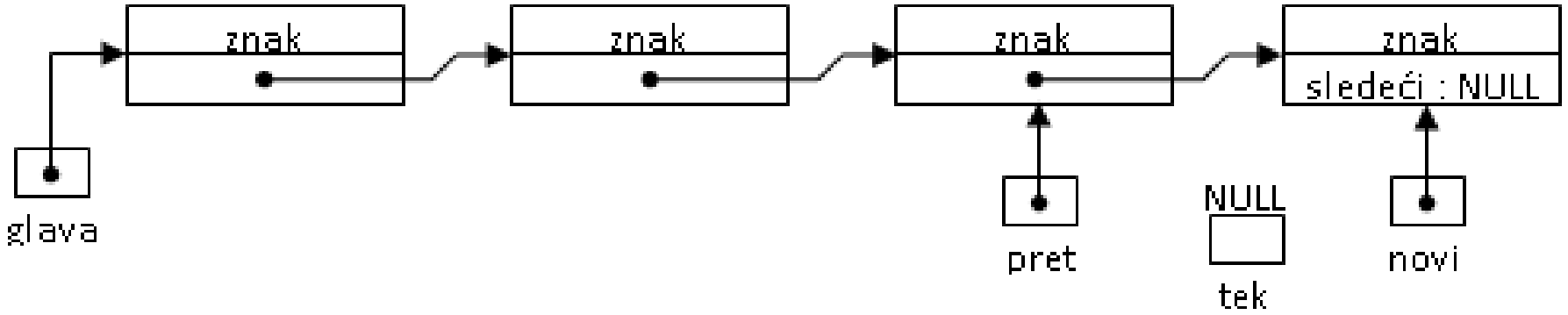
- Nakon prolaska kroz while ciklus imaćemo sledeću situaciju:
  '**tek**' ne pokazuje ni na šta, a '**pret**' koji ga je "pratio" pokazaće na poslednji slog iz liste.



Nakon:

```
pret->sledeci=novi;
```

novi slog će biti povezan kao poslednji element u listi.



3) Listanje liste predstavlja prikazivanje svih elemenata iz liste

- Ako je glava==NULL znači da je lista prazna i nemamo šta prikazati.
- Ako lista nije prazna, novim pokazivačem polazi se od prvog sloga liste (tek=glava) i dok se ne dođe do kraja liste (tek==NULL) prikazuju se slogovi liste.

```
tek=glava;
while(tek!=NULL)
{
    printf("   %c",tek->znak);     /* Sa strelicom (->) se pristupa
                                   elementu sloga (strukture). */
    tek=tek->sledeci;              /* Prelazimo na sledeći element iz liste. */
}
```

4) Brisanje elementa iz liste realizuje se tako što se mora element prvo pronaći (vrši se traženje elementa u listi).

- Traženje elementa liste se može realizovati sledećim delom koda:

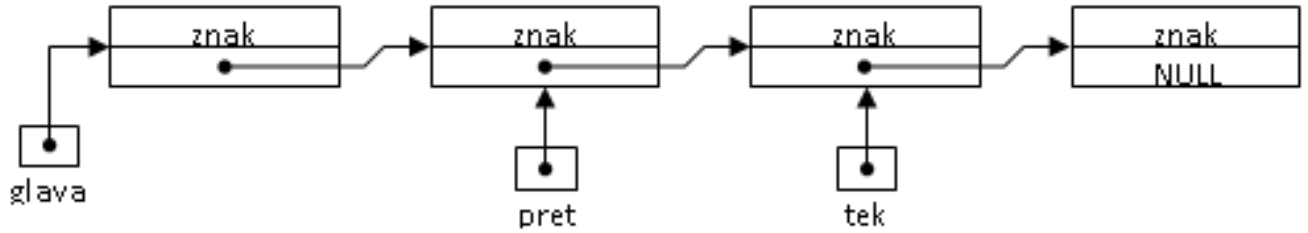
```
tek=glava;
pret=glava;
while(tek!=NULL && (tek->znak != c))
{
        pret=tek;
        tek=tek->sledeci;
}
```

Pri čemu je 'c' promenljiva čija vrednost se traži. Nakon završetka while ciklusa ako pokazivač 'tek' ima vrednost NULL znači da elementa nema u listi. U slučaju da je tek!=NULL element se nalazi u listi, a pokazivač 'pret' pokazuje na njegovog prethodnika iz liste.

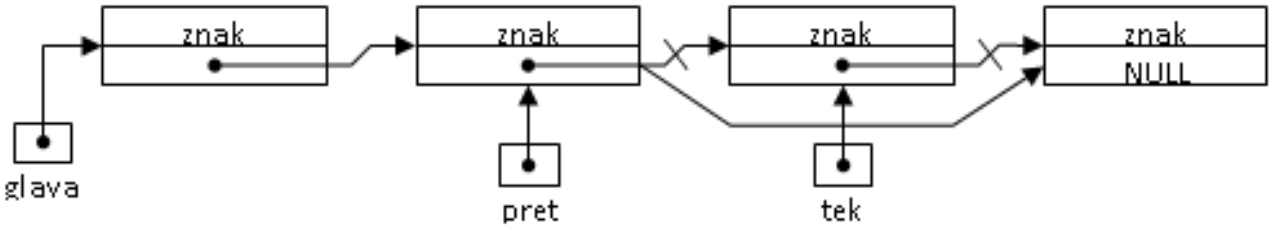
Razlikujemo dva slučaja brisanja:

a) da slog koji se briše nije prvi element liste,
b) da slog koji se briše jeste prvi element liste.

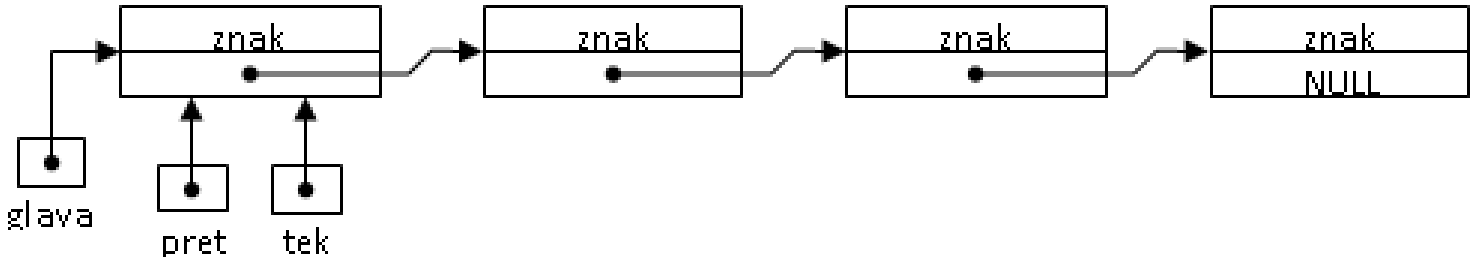
a) Slog koji se briše nije prvi element liste



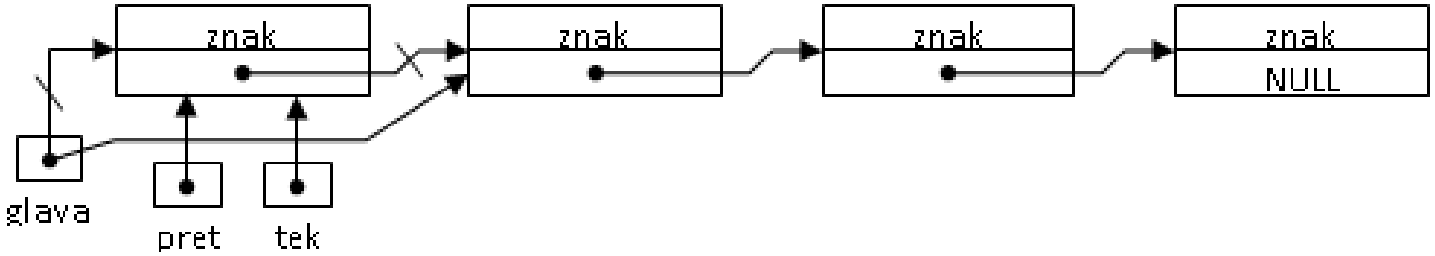

```
pret->sledeci=tek->sledeci;
tek->sledeci=NULL;
```



b) Slog koji se briše jeste prvi element liste

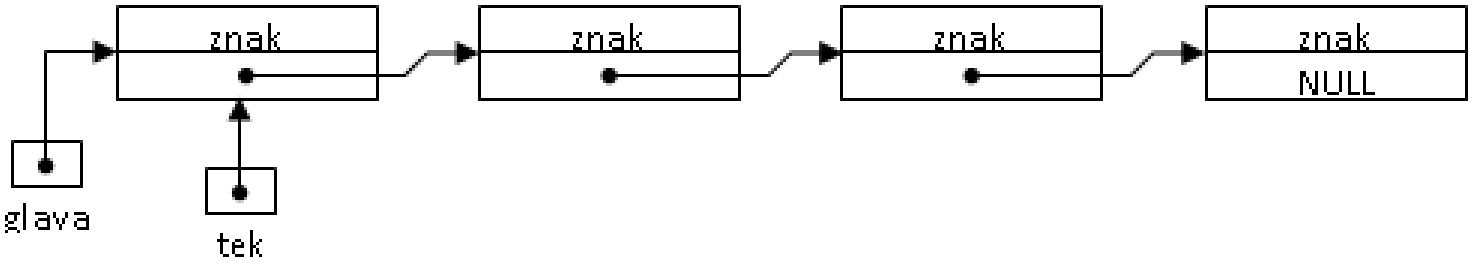


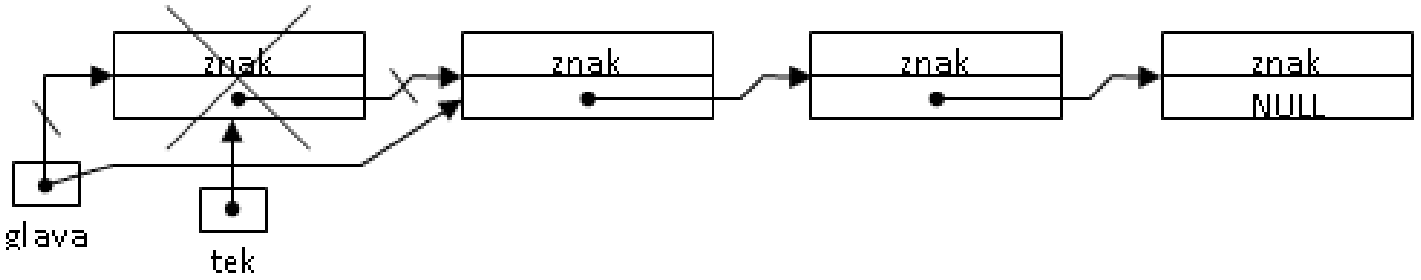

```
glava=tek->sledeci;
tek->sledeci=NULL;
```



5) Brisanje liste

```
while(glava!=NULL)
{
        tek=glava;
        glava=tek->sledeci;
        free(tek);
}
```





Kružne jednostruko spregnute liste

- Vrednost pokazivača na sledeći čvor u čvoru na kraju jednostruko spregnute liste ima vrednost NULL, pošto on ne pokazuje na sledeći čvor.
- U kružnoj listi svaki čvor pokazuje na sledeći u listi. Poslednji čvor pokazuje nazad na početak liste formirajuđi na taj način krug.
- U slučaju kružne liste jednostavnije se izvode operacije kao što su pretraživanje od nekog unutrašnjeg čvora ka bilo kom delu liste.