Liste

Uvod

Pri rešavanju mnogih problema potrebne su nam dinamičke liste, čiju veličinu ne moramo znati u trenutku kompajliranja, već je možemo definisati i u toku rada programa. **Povezana lista** je struktura podataka koja se koristi za modeliranje ovakvih dinamičkih lista.

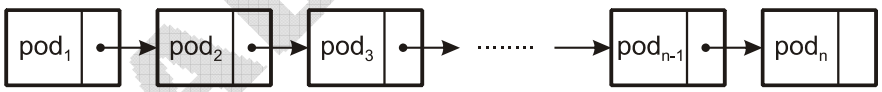
Koncept

Nizovi su u memoriji predstavljeni korišćenjem sekvencijalnog mapiranja, tako da su elementi niza podjednako udaljeni jedan od drugog. Međutim, ovakav pristup ima sledeće nedostatke:

- Umetanje novog ili brisanje postojećeg elementa iz niza je vremenski veoma skupa operacija, jer zahteva pomeranje određenog broja postojećih elemenata.
- U slučaju korišćenja statičkih nizova neophodno je poznavanje maksimalnog broja elemenata još u trenutku pisanja programa. Pored ovog nedostatka, dodatni problem je i to što bez obzira na stvaran broj elemenata koji će biti smešten u niz, program uvek alocira maksimalnu veličinu niza, što dovodi do bespotrebnog trošenja memorije.
- Korišćenje dinamičkih nizova rešava problem poznavanja maksimalnog broja elemenata u trenutku pisanja programa, ali uvodi i neke nove probleme. Svaka promena veličine niza zahteva dodatnu alokaciju i dealokaciju memorije, kao i kopiranje sadržaja elemenata u novi memorijski prostor, što drastično pogoršava performanse programa.

Da bi se prevazišli navedeni problemi uvodi se koncept povezanih elemenata. U ovakvom pristupu nije neophodno da elementi budu međusobno na podjednakom rastojanju. Umesto toga elemente možemo smestiti bilo gde u memoriji, a zatim ih povezati tako da svaki element (osim prvog) bude povezan sa prethodnim elementom u listi. To se može postići tako što se u svaki element upiše i adresa njegovog sledbenika, što zahteva da svaki element bude u mogućnosti da pored svojih podataka čuva i adresu sledećeg elementa. Zbog toga svaki element mora biti struktura koja sadrži dva dela: jedan koji čuva neophodne podatke (nazovimo ga **podatak** ili **data field**) i drugi koji čuva adresu sledbenika (**veza** ili **link**).

Dakle, **povezana lista** je lista elemenata koji su proizvoljno raspoređeni u memoriji i koji su međusobno povezani **linkom**, tj. smeštanjem adrese svakog elementa (osim prvog) u njegovog prethodnika.



Slika ### Povezana lista

Definisanje

Za realizaciju koncepta povezanih lista u programskom jeziku C koriste se slogovi (strukture) i pokazivači (pointeri). Element liste koja će sadržati celobrojne podatke se definiše na sledeći način:

```
struct element
{
   int podatak;
   struct element *sledeci;
};
```

Prethodne linije koda definišu strukturni tip *element* koji će predstavljati element liste. Svaki element liste sadrži celobrojni podatak i pokazivač na sledeći element liste. Podatak unutar liste može biti i bilo kog drugog prostog ili složenog tipa.

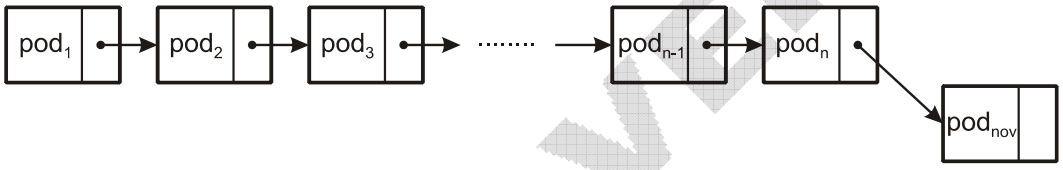
Dodavanje elementa u listu

Da bismo uopšte mogli da baratamo listom, potrebno je da nam njen prvi element (ukoliko postoji) uvek bude poznat. Kada znamo prvi element, korišćenjem pokazivača na njegovog sledbenika lako dolazimo do sledećeg elementa u listi. Pošto lista može biti i prazna, umesto da pamtimo prvi element liste, koristimo pokazivač na prvi element, koji nazivamo **glava liste**. Dakle, glava liste je pokazivač na prvi element liste, a u slučaju da je lista prazna vrednost ovog pokazivača je NULL. Veoma je bitno u svakom trenutku imati informaciju o glavi liste, jer je samo tako moguće pristupiti listi. Iz tog razloga prilikom pisanja programa treba voditi računa da uvek postoji bar jedan pokazivač na početak liste, koji se neće koristiti u druge svrhe, čime bi se došlo u opasnost da se izgubi informacija o početku liste.

Prilikom dodavanja novog elementa u listu moramo razlikovati dva slučaja:

1. **Lista je bila prazna** pre dodavanja novog elementa
2. **Lista nije bila prazna** pre dodavanja novog elementa

Pre dodavanja elementa u praznu listu, glava liste je pokazivač koji ima vrednost NULL (ne pokazuje ni na jedan element). Nakon dodavanja novog elementa, glava liste postaje pokazivač na taj element. U drugom slučaju lista nije prazna i glava pokazuje na prvi element. Da bi se dodao element na kraj liste, potrebno je proći kroz sve elemente liste kako bi se pronašao poslednji element, a zatim iza njega dodao novi element. Dodavanjem novog elementa na kraj liste očigledno ne dolazi do promene glave liste. Imajući u vidu navedene postupke dodavanja novog elementa, jasno je da se algoritmi u ova dva slučaja drastično razlikuju, pa ih iz tog razloga i u programu treba razdvojiti.



Slika ### Dodavanje elementa u listu

Prolazak kroz listu

U radu sa listama je često potrebno kretati se kroz listu, od elementa do elementa. Najčešće je u pitanju obrada podataka u listi, pretraga liste kako bi se našao odgovarajući element ili traženje kraja liste. U svim ovim slučajevima algoritam je sličan. Uvodi se pomoćni pokazivač koji je inicijalno jednak glavi liste, odnosno pokazuje na prvi element liste ukoliko ona nije prazna. Nakon provere da li pokazivač pokazuje na neki element (ima vrednost različitu od NULL) vrši se obrada podatka u tom element (štampa, upoređivanje, račun...). Po završetku obrade podatka, pomoćni pokazivač dobija vrednost pokazivača na sledeći element, a čitav postupak se ponavlja sve dok pomoćni pokazivač ima nenultu vrednost, tj. dok pokazuje na neki element. Kada pomoćni pokazivač dobije vrednost NULL, to znači da smo došli do kraja liste.

Sledeći isečak koda pokazuje prolazak kroz listu čiji je početak definisan pokazivačem *glava* i štampanje podataka zapisanih u svim njenim elementima:

```
struct element *pom;

pom=glava;
while(pom!=NULL)
{
   printf("%d\t",pom->podatak);
   pom=pom->sledeci;
};
```

Primer

Listing programa za kreiranje i štampanje povezane liste:

```
# include <stdio.h>
# include <stdlib.h>

struct element
{
   int podatak;
   struct element *sledeci;
};

/* funkcija za dodavanje novog elementa na kraj liste */
struct element *dodaj(struct element *p, int n)
{
   struct element *pom;

   /* ako je postojeca lista prazna onda se dodaje novi element kao pocetni */
   if(p==NULL)
   {
      /* kreiranje novog elementa */
      p=(struct element *)malloc(sizeof(struct element));

      if(p==NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      /* novom elementu se dodeljuje prosledjeni podatak */
      p->podatak = n;
      p->sledeci = NULL;
   }
   else
   {
      pom = p;

      /* prolazi se kroz postojecu listu da bi se dobio pokazivac na poslednji element */
      while (pom->sledeci != NULL)      pom = pom->sledeci;

      /* kreiranje novog elementa */
      pom->sledeci = (struct element *)malloc(sizeof(struct element));
      if(pom->sledeci == NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      /* novom elementu se dodeljuje prosledjeni podatak, a njegova adresa se upisuje kao link
      prethodnog elementa */
      pom = pom->sledeci;
      pom->podatak = n;
      pom->sledeci = NULL;
   }
   return (p);
}

/* funkcija za stampanje liste */
void stampaj_listu( struct element *p )
{
   struct element *pom;
   pom = p;
   if(p!= NULL)
   {
      do
      {
         printf("%d\t",pom->podatak);
         pom=pom->sledeci;
      } while (pom!= NULL);

      printf("\n");
   }
```

```
   else      printf("Lista je prazna.");
   printf("\n");
}

void main()
{
   int n,i;
   int x;
   struct element *glava = NULL ;

   printf("Unesi broj elemenata u listi:\n");
   scanf("%d",&n);

   for(i=0;i<n;i++)
   {
      printf( "Unesi %d. element:\n", i+1);
      scanf("%d",&x);
      glava = dodaj( glava, x );
   }

   printf("Kreirana lista je:\n");
   stampaj_listu( glava );
}
```

Brisanje elementa iz liste

Da bismo obrisali određeni element iz liste potrebno je da znamo njegovu poziciju u listi. Jedan od načina je da navedemo njegov redni broj, uz pretpostavku da su elementi numerisani redom od 1 do n. Kada znamo redni broj elementa, krećemo se kroz listu kako bismo pronašli pokazivač na traženi element. Pored toga, potrebno je da znamo i pokazivač na prethodni element u listi, kako bismo njegovom linku dodelili pokazivač na element koji sledi elementu koji brišemo. Nakon toga možemo da obrišemo traženi element, odnosno da oslobodimo memoriju koju on zauzima.

Slika ### Brisanje elementa iz liste

Primer

Listing programa za kreiranje liste, brisanje zadatog elementa i štampanje povezane liste pre i posle brisanja:

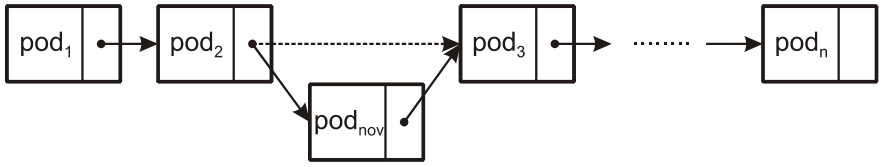
```
# include <stdio.h>
# include <stdlib.h>

struct element
{
   int podatak;
   struct element *sledeci;
};

/* funkcija za dodavanje novog elementa na kraj liste */
struct element *dodaj(struct element *p, int n)
{
   struct element *pom;

   /* ako je postojeca lista prazna onda se dodaje novi element kao pocetni */
   if(p==NULL)
   {
      /* kreiranje novog elementa */
      p=(struct element *)malloc(sizeof(struct element));

      if(p==NULL)
      {
```

```
            printf("Greska.\n");
            exit(0);
        }

        /* novom elementu se dodeljuje prosledjeni podatak */
        p->podatak = n;
        p->sledeci = NULL;
    }
    else
    {
        pom = p;

        /* prolazi se kroz postojecu listu da bi se dobio pokazivac na poslednji element */
        while (pom->sledeci != NULL)      pom = pom->sledeci;

        /* kreiranje novog elementa */
        pom->sledeci = (struct element *)malloc(sizeof(struct element));
        if(pom->sledeci == NULL)
        {
            printf("Greska.\n");
            exit(0);
        }

        /* novom elementu se dodeljuje prosledjeni podatak, a njegova adresa se upisuje kao link
        prethodnog elementa */
        pom = pom->sledeci;
        pom->podatak = n;
        pom->sledeci = NULL;
    }
    return (p);
}

/* funkcija za stampanje liste */
void stampaj_listu( struct element *p )
{
    struct element *pom;
    pom = p;
    if(p!= NULL)
    {
        do
        {
            printf("%d\t",pom->podatak);
            pom=pom->sledeci;
        } while (pom!= NULL);

        printf("\n");
    }
    else      printf("Lista je prazna.\n");
}

/* funkcija za odredjivanje duzine liste */
int duzina( struct element *p )
{
    int broj = 0 ;
    while ( p != NULL )
    {
        broj++;
        p = p->sledeci;
    }
    return ( broj ) ;
}

/* funkcija za brisanje zadatog elementa iz liste */
struct element *obrisi( struct element *p, int redni_broj )
{
    struct element *prethodni, *trenutni ;
    int i;

    if (p == NULL )
    {
        printf("Lista je prazna. \n");
    }
    else
    {
```

```
      if ( redni_broj > duzina(p) )
      {
         printf("Greska.\n");
      }
      else
      {
         prethodni = NULL;
         trenutni = p;
         i = 1;
         while ( i < redni_broj )
         {
            prethodni = trenutni;
            trenutni = trenutni->sledeci;
            i = i+1;
         }
         if ( prethodni == NULL )
         {
            p = trenutni->sledeci;
            free( trenutni );
         }
         else
         {
            prethodni->sledeci = trenutni->sledeci;
            free( trenutni );
         }
      }
   }

   return(p);
}

void main()
{
   int n,i;
   int x;
   struct element *glava = NULL;

   printf("Unesi broj cvorova u listi:\n");
   scanf("%d",&n);

   for(i=0;i<n;i++)
   {
      printf("Unesi %d. element:\n", i+1);
      scanf("%d",&x);
      glava = dodaj( glava, x );
   }

   printf("Lista pre brisanja elementa:\n");
   stampaj_listu( glava );

   printf("Unesi redni broj elementa:\n");
   scanf ("%d",&n);
   glava = obrisi(glava , n );

   printf("Lista posle brisanja elementa:\n");
   stampaj_listu( glava );
}
```


Umetanje novog elementa iza određenog elementa u listi

Da bismo umetnuli novi element na određeno mesto u listi, neophodno je da znamo redni broj elementa u listi iza koga će novi element biti umetnut. Pretpostavljamo da su svi elementi liste numerisani rednim brojevima od 1 do *n*. Kada znamo broj elementa, krećemo se kroz listu kako bismo dobili pokazivač na element sa zadatim rednim brojem. Zatim kreiramo novi element, a njegovom linku dodeljujemo pokazivač na element koji sledi iza elementa sa navedenim rednim brojem. Nakon toga, elementu sa zadatim rednim brojem postavljamo link tako da pokazuje novi element.



Slika ### Umetanje novog elementa iza određenog elementa u listi

Primer

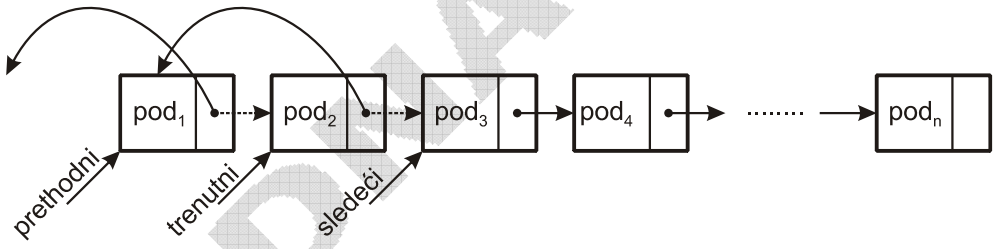
```
# include <stdio.h>
# include <stdlib.h>

struct element
{
   int podatak;
   struct element *sledeci;
};

int duzina( struct element * );

/* funkcija za dodavanje novog elementa na kraj liste */
struct element *dodaj(struct element *p, int n)
{
   struct element *pom;

   /* ako je postojeca lista prazna onda se dodaje novi element kao pocetni */
   if(p==NULL)
   {
      /* kreiranje novog elementa */
      p=(struct element *)malloc(sizeof(struct element));

      if(p==NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      /* novom elementu se dodeljuje prosledjeni podatak */
      p->podatak = n;
      p->sledeci = NULL;
   }
   else
   {
      pom = p;

      /* prolazi se kroz postojecu listu da bi se dobio pokazivac na poslednji element */
      while (pom->sledeci != NULL)      pom = pom->sledeci;

      /* kreiranje novog elementa */
      pom->sledeci = (struct element *)malloc(sizeof(struct element));
      if(pom->sledeci == NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      /* novom elementu se dodeljuje prosledjeni podatak, a njegova adresa se upisuje kao link
      prethodnog elementa */
      pom = pom->sledeci;
      pom->podatak = n;
      pom->sledeci = NULL;
   }
   return (p);
}

/* funkcija za umetanje novog elementa iza odredjenog elementa u listi */
struct element * umetni( struct element *p, int redni_broj, int vrednost )
{
   struct element *novi, *pom;
   int i;
```

```
   if ( redni_broj <= 0 || redni_broj > duzina(p) )
   {
      printf("Greska! Zadati element ne postoji.\n");
      exit(0);
   }

   if ( redni_broj == 0)
   {
      novi = ( struct element * )malloc( sizeof( struct element ) );
      if ( novi == NULL )
      {
         printf( "Greska pri alociranju memorije. \n");
         exit(0);
      }
      novi->podatak = vrednost;
      novi->sledeci = p;
      p = novi ;
   }
   else
   {
      pom = p ;
      i = 1;
      while ( i < redni_broj )
      {
         i = i+1;
         pom = pom->sledeci;
      }

      novi = ( struct element * )malloc( sizeof(struct element) );
      if ( novi == NULL )
      {
         printf ("Greska pri alociranju memorije. \n ");
         exit(0);
      }
      novi->podatak = vrednost;
      novi->sledeci = pom->sledeci;
      pom->sledeci = novi;
   }

   return (p);
}

/* funkcija za stampanje liste */
void stampaj_listu( struct element *p )
{
   struct element *pom;
   pom = p;
   if(p!= NULL)
   {
      do
      {
         printf("%d\t",pom->podatak);
         pom=pom->sledeci;
      } while (pom!= NULL);

      printf("\n");
   }
   else     printf("Lista je prazna.\n");
}

/* funkcija za odredjivanje duzine liste */
int duzina( struct element *p )
{
   int broj = 0 ;
   while ( p != NULL )
   {
      broj++;
      p = p->sledeci;
   }
   return ( broj ) ;
}

void main ()
```

```
{
   int n,x;
   int i;
   struct element *glava = NULL;

   printf("Unesi broj cvorova u listi:\n");
   scanf("%d",&n);

   for(i=0;i<n;i++)
   {
      printf("Unesi %d. element:\n", i+1);
      scanf("%d",&x);
      glava = dodaj( glava, x );
   }

   printf("Lista pre umetanja elementa:\n");
   stampaj_listu( glava );

   printf("Unesi redni broj elementa iza koga zelis da novi element bude umetnut:\n");
   scanf ("%d",&n);

   printf("Unesi vrednost novog elementa:\n");
   scanf("%d",&x);

   glava = umetni( glava, n, x );

   printf("Lista posle umetanja elementa:\n");
   stampaj_listu( glava );
}
```

Okretanje redosleda u listi

Da bismo izvršili okretanje redosleda u listi neophodno je da za svaki element znamo pokazivače na njegovog prethodnika i sledbenika. Kada su nam ovi pokazivači poznati, onda možemo link trenutnog elementa da postavimo tako da pokazuje na prethodnika. Nakon toga trenutni element proglašavamo za prethodnika, a njegovog sledbenika za trenutni.



Slika ### Okretanje redosleda u listi

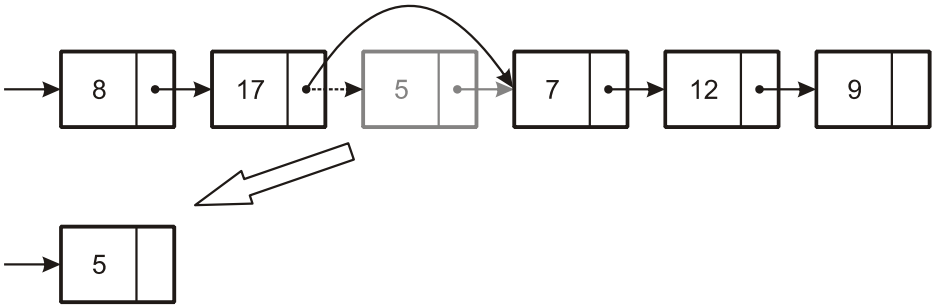
Deo koda koji bi obavljao navedenu proceduru mogao bi da izgleda ovako:

```
prethodni = NULL;
while (trenutni != NULL)
{
   sledeci = trenutni->sledeci;
   trenutni->sledeci = prethodni;
   prethodni = trenutni;
   trenutni = sledeci;
}
```

Sortiranje liste

Da bismo sortirali listu, prvo prolazimo kroz nju kako bismo pronašli element sa najmanjom vrednošću podatka. Nakon toga uklanjamo taj element iz liste i dodajemo ga na kraj druge liste, koja je inicijalno bila

prazna. Ovaj proces ponavljamo sve dok početna lista ne postane prazna. Na kraju ostaje samo da funkcija vrati pokazivač na listu u koju su prebačeni svi elementi.



Slika ### Sortiranje liste

Primer

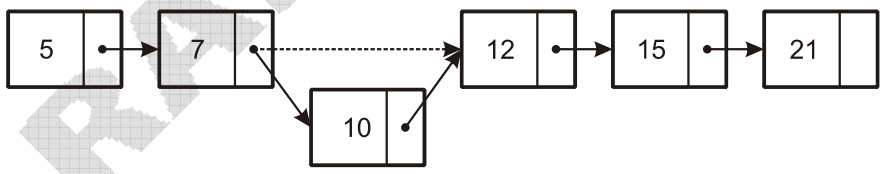
```
# include <stdio.h>
# include <stdlib.h>

struct element
{
   int podatak;
   struct element *sledeci;
};

/* funkcija za dodavanje novog elementa na kraj liste */
struct element *dodaj(struct element *p, int n)
{
   struct element *pom;

   /* ako je postojeca lista prazna onda se dodaje novi element kao pocetni */
   if(p==NULL)
   {
      /* kreiranje novog elementa */
      p=(struct element *)malloc(sizeof(struct element));

      if(p==NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      /* novom elementu se dodeljuje prosledjeni podatak */
      p->podatak = n;
      p->sledeci = NULL;
   }
   else
   {
      pom = p;

      /* prolazi se kroz postojecu listu da bi se dobio pokazivac na poslednji element */
      while (pom->sledeci != NULL)      pom = pom->sledeci;

      /* kreiranje novog elementa */
      pom->sledeci = (struct element *)malloc(sizeof(struct element));
      if(pom->sledeci == NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      /* novom elementu se dodeljuje prosledjeni podatak, a njegova adresa se upisuje kao link
      prethodnog elementa */
      pom = pom->sledeci;
      pom->podatak = n;
      pom->sledeci = NULL;
   }
   return (p);
```

```
}

/* funkcija za stampanje liste */
void stampaj_listu( struct element *p )
{
   struct element *pom;
   pom = p;
   if(p!= NULL)
   {
      do
      {
         printf("%d\t",pom->podatak);
         pom=pom->sledeci;
      } while (pom!= NULL);

      printf("\n");
   }
   else      printf("Lista je prazna.\n");
}

/* funkcija za okretanja redosleda u listi */
struct element *okreni(struct element *p)
{
   struct element *prethodni, *trenutni, *sledeci;

   prethodni = NULL;
   trenutni = p;

   while (trenutni != NULL)
   {
      sledeci = trenutni->sledeci;
      trenutni->sledeci = prethodni;
      prethodni = trenutni;
      trenutni = sledeci;
   }

   return(prethodni);
}

/* funkcija za sortiranje liste u rastucem redosledu */
struct element *sortiraj(struct element *p)
{
   struct element *pom1,*pom2,*min,*prethodni,*q;

   q = NULL;
   while(p != NULL)
   {
      prethodni = NULL;
      min = pom1 = p;

      pom2 = p->sledeci;
      while ( pom2 != NULL )
      {
         if( pom2->podatak < min->podatak )
         {
            min = pom2;
            prethodni = pom1;
         }
         pom1 = pom2;
         pom2 = pom2->sledeci;
      }

      if(prethodni == NULL)
         p = min->sledeci;
      else
         prethodni->sledeci = min->sledeci;

      min->sledeci = NULL;

      if( q == NULL)
         q = min;  /* premesta element sa najmanjom vrednoscu podatka
                      iz liste p na pocetak liste q */
      else
      {
```

```
            pom1 = q;

            /* prolazi kroz listu q da bi pronasao njen poslednji element */
            while( pom1->sledeci != NULL)  pom1 = pom1->sledeci;

            pom1->sledeci = min;   /* premesta element sa najmanjom vrednoscu podatka
                                      iz liste p na kraj liste q */
         }
      }

      return (q);
}

void main()
{
   int n,x;
   int i;
   struct element *glava = NULL;

   printf("Unesi broj cvorova u listi:\n");
   scanf("%d",&n);

   for(i=0;i<n;i++)
   {
      printf("Unesi %d. element:\n", i+1);
      scanf("%d",&x);
      glava = dodaj( glava, x );
   }

   printf("Lista pre sortiranja:\n");
   stampaj_listu( glava );

   glava = sortiraj(glava);

   printf("Sortirana lista je:\n");
   stampaj_listu( glava );

   glava = okreni(glava);

   printf("Okrenuta lista je:\n");
   stampaj_listu( glava );
}
```

Umetanje novog elementa u sortiranu listu

Da bismo umetnuli novi element u listu koja je prethodno sortirana, upoređujemo redom podatke u listi sa vrednošću podatka novog elementa. Ovaj proces se ponavlja sve dok ne dobijemo pokazivač na element koji se nalazi ispred elementa čiji je podatak veći od podatka u novom elementu.



Slika ### Umetanje elementa u sortiranu listu

Primer

```
# include <stdio.h>
# include <stdlib.h>

struct element
{
   int podatak;
   struct element *sledeci;
};

struct element *dodaj(struct element *, int);
```

```
struct element *sort_dodaj(struct element*, int );
void stampaj_listu(struct element *);
struct element *sortiraj(struct element *);

/* funkcija za dodavanje novog elementa na kraj liste */
struct element *dodaj(struct element *p, int n)
{
   struct element *pom;

   /* ako je postojeca lista prazna onda se dodaje novi element kao pocetni */
   if(p==NULL)
   {
      /* kreiranje novog elementa */
      p=(struct element *)malloc(sizeof(struct element));

      if(p==NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      /* novom elementu se dodeljuje prosledjeni podatak */
      p->podatak = n;
      p->sledeci = NULL;
   }
   else
   {
      pom = p;

      /* prolazi se kroz postojecu listu da bi se dobio pokazivac na poslednji element */
      while (pom->sledeci != NULL)      pom = pom->sledeci;

      /* kreiranje novog elementa */
      pom->sledeci = (struct element *)malloc(sizeof(struct element));
      if(pom->sledeci == NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      /* novom elementu se dodeljuje prosledjeni podatak, a njegova adresa se upisuje kao link
      prethodnog elementa */
      pom = pom->sledeci;
      pom->podatak = n;
      pom->sledeci = NULL;
   }
   return (p);
}

/* funkcija za stampanje liste */
void stampaj_listu( struct element *p )
{
   struct element *pom;
   pom = p;
   if(p!= NULL)
   {
      do
      {
         printf("%d\t",pom->podatak);
         pom=pom->sledeci;
      } while (pom!= NULL);

      printf("\n");
   }
   else      printf("Lista je prazna.\n");
}

/* funkcija za sortiranje liste u rastucem redosledu */
struct element *sortiraj(struct element *p)
{
   struct element *pom1,*pom2,*min,*prethodni,*q;

   q = NULL;
   while(p != NULL)
```

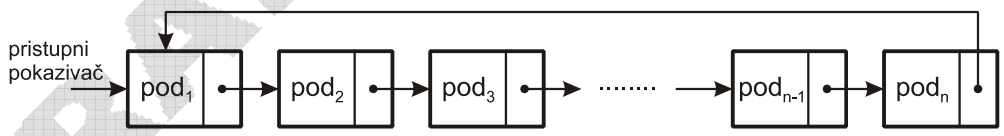
```
   {
      prethodni = NULL;
      min = pom1 = p;

      pom2 = p->sledeci;
      while ( pom2 != NULL )
      {
         if( pom2->podatak < min->podatak )
         {
            min = pom2;
            prethodni = pom1;
         }
         pom1 = pom2;
         pom2 = pom2->sledeci;
      }

      if(prethodni == NULL)
         p = min->sledeci;
      else
         prethodni->sledeci = min->sledeci;

      min->sledeci = NULL;

      if( q == NULL)
         q = min;  /* premesta element sa najmanjom vrednoscu podatka
                      iz liste p na pocetak liste q */
      else
      {
         pom1 = q;

         /* prolazi kroz listu q da bi pronasao njen poslednji element */
         while( pom1->sledeci != NULL)  pom1 = pom1->sledeci;

         pom1->sledeci = min;   /* premesta element sa najmanjom vrednoscu podatka
                                   iz liste p na kraj liste q */
      }
   }

   return (q);
}

/* funkcija za umetanje novog elementa u sortiranu listu*/
struct element *sort_dodaj(struct element *p, int n)
{
   struct element *trenutni, *prethodni, *novi;

   trenutni =p;
   prethodni = NULL;

   while( trenutni->podatak < n )
   {
      prethodni = trenutni;
      trenutni = trenutni->sledeci;
   }

   novi = (struct element *) malloc(sizeof(struct element));
   if( novi == NULL)
   {
      printf("Greska pri alociranju memorije.\n");
      exit(0);
   }

   if ( prethodni == NULL ) /* element ce biti umetnut na pocetak liste */
   {
      novi->podatak = n;
      novi->sledeci = p;
      p = novi;
   }
   else
   {
      novi->podatak = n;
      novi->sledeci = prethodni->sledeci;
      prethodni->sledeci = novi;
   }
```

```
    return(p);
}

void main()
{
    int n,x;
    int i;
    struct element *glava = NULL;

    printf("Unesi broj cvorova u listi:\n");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("Unesi %d. element:\n", i+1);
        scanf("%d",&x);
        glava = dodaj( glava, x );
    }

    printf("Lista pre sortiranja:\n");
    stampaj_listu( glava );

    glava = sortiraj(glava);

    printf("Sortirana lista je:\n");
    stampaj_listu( glava );

    printf("Unesi vrednost novog elementa:\n");
    scanf("%d",&x);

    glava = sort_dodaj( glava, x );

    printf("Lista posle umetanja elementa:\n");
    stampaj_listu( glava );
}
```

Kružne liste

Kružne ili **cirkularne** liste su liste kod kojih poslednji element pokazuje na prvim element. Iz tog razloga kružne liste se mogu posmatrati i kao liste koje nemaju ni početak ni kraj. Da bi se prošlo kroz ovakvu listu potrebno je poći od bilo kog elementa u listi i nastaviti kretanje sve dok se ponovo ne dođe do početnog elementa.

S obzirom da ovakva lista nema početak ni kraj, korišćenje naziva "glava liste" za pokazivač koji pokazuje na listu nije prikladan. U ovom slučaju pokazivač koji pokazuje na celu listu može se nazvati **pristupni pokazivač**.



Slika ### Šematski prikaz kružne liste

Kružne liste se u praksi mogu primeniti na strukture podataka koje su po prirodi kružne. Jedan od tipičnih primera korišćenja kružne liste je deljenje procesorskog vremena različitim korisnicima od strane operativnog sistema. Kako bi se obezbedila ravnopravnost svih korisnika, oni su raspoređeni u krug i redom im se dodeljuje deo slobodnog vremena procesora. Operativni sitem uzima jednog korisnika, dodeljuje mu mali deo procesorskog vremena i prelazi na sledećeg korisnika. Korisnici koji nemaju više zahteva za procesorskim vremenom isključuju se iz liste. Ovaj proces se ponavlja sve do trenutka dok više ne postoje korisnici koji zahtevaju procesorsko vreme.

Primer

Sledeći program kreira i štampa sadržaj kružne liste:

```
# include <stdio.h>
# include <stdlib.h>

struct element
{
   int podatak;
   struct element *sledeci;
};

struct element *dodaj(struct element *p, int n)
{
   struct element *pom;

   /* ukoliko je lista prazna, novi element se dodaje kao pocetni */
   if(p==NULL)
   {
      p=(struct element *)malloc(sizeof(struct element)); /* kreiranje novog elementa */
      if(p==NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      p->podatak = n;
      p->sledeci = p; /* element pokazuje na samog sebe posto je lista kruzna */
   }
   else
   {
      pom = p;
      /* prolazak kroz celu listu kako bi se dobio pokazivac na poslednji element */
      while (pom->sledeci != p) pom = pom->sledeci;

      pom->sledeci = (struct element *)malloc(sizeof(struct element)); /* kreiranje novog elementa */
      if(pom->sledeci == NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      pom = pom->sledeci;
      pom->podatak = n;
      pom->sledeci = p;  /* poslednji element pokazuje na pocetak liste */
   }
   return (p);
}

void stampaj_listu( struct element *p )
{
   struct element *pom;

   pom = p;
   printf("Podaci u listi su:\n");
   if(p!= NULL)
   {
      do
      {
         printf(%d\t",pom->podatak);
         pom=pom->sledeci;
      } while (pom!= p);

      printf("\n");
   }
   else
      printf("Lista je prazna.\n");
}

void main()
{
   int n,i;
   int x;
```

```
   struct element *pristupni = NULL;

   printf("Unesite broj elementa liste:\n");
   scanf("%d",&n);

   for(i=0; i<n; i++)
   {
      printf("Unesite vrednost:\n");
      scanf("%d",&x);

      pristupni = dodaj( pristupni, x );
   }

   printf("Kreirana lista je:\n");
   stampaj_listu( pristupni );
}
```

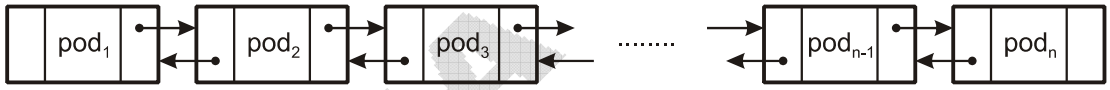
Program dodaje novi element na kraj postojeće liste, a zatim povezuje ovaj element sa prvim elementom u listi. Na ovaj način osigurano postojanje veze između poslednjeg elementa i početka liste.

Dvostruko povezane liste

Radeći sa jednostruko povezanim listama mogli smo uočiti sledeće probleme:

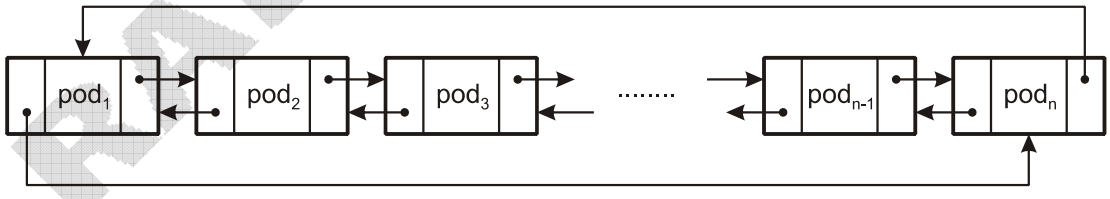
1. Jednostruko povezane liste omogućavaju kretanje kroz listu samo u jednom pravcu.
2. Brisanje elementa iz liste zahteva stalno čuvanje prethodnog elementa tokom kretanja kroz listu, kako bi se znao prethodnik elementa koji je potrebno obrisati.
3. Ukoliko dođe do gubljenja veze u nekom elementu, ostali elementi postaju nedostupni.

Ovi problemi kod jednostruko povezanih lista mogu biti prevaziđeni uvođenjem pokazivača na prethodnika u svakom elementu. Kada svaki element liste sadrži pokazivač na svog sledbenika i prethodnika, takvu listu nazivamo **dvostruko povezana lista**.



Slika ### Šematski prikaz dvostruko povezane liste

Kao i jednostruko povezana lista i dvostruko povezana lista može biti u obliku lanca ili kružna. Ukoliko je lista u obliku lanca, pokazivači na prethodnika prvog elementa i sledbenika poslednjeg elementa moraju biti NULL pokazivači.



Slika ### Šematski prikaz dvostruko povezane kružne liste

Realizacija dvostruko povezane liste može se obaviti pomoću sledeće strukture:

```
struct delement
{
   int podatak;
   struct delement *levi,*desni;
};
```

Primer

Sledeći program kreira i štampa dvostruko povezanu listu:

```
# include <stdio.h>
# include <stdlib.h>

struct delement
{
   int podatak;
   struct delement *levi, *desni;
};

struct delement *dodaj(struct delement *p, struct delement **q, int n)
{
   struct delement *pom;

   /* ukoliko je lista prazna, novi element se dodaje kao pocetni element liste */
   if(p==NULL)
   {
      p=(struct delement *)malloc(sizeof(struct delement)); /* kreiranje novog elementa */
      if(p==NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      p->podatak = n;
      p->levi = p->desni = NULL;  /* novi element nema ni levog ni desnog suseda */
      *q = p;   /* kraj liste je i pocetak liste */
   }
   else
   {
      pom = (struct delement *)malloc(sizeof(struct delement)); /* kreiranje novog elementa */
      if(pom == NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      pom->podatak = n;
      pom->levi = (*q);  /* poslednji element liste postaje levi novom elementu */
      pom->desni = NULL; /* novi element nema desnog suseda */
      (*q)->desni = pom; /* novi element postaje desni poslednjem elementu liste */
      (*q) = pom;        /* novi element postaje poslednji element liste */
   }

   return (p);
}

void stampaj_desno( struct delement *p )
{
   printf("Podaci u listi stampani u desno su:\n");
   while (p!= NULL)
   {
      printf("%d\t",p->podatak);
      p = p->desni;
   }
}

void stampaj_levo( struct delement *p )
{
   printf("Podaci u listi stampani u levo su:\n");
   while (p!= NULL)
   {
      printf("%d\t",p->podatak);
      p = p->levi;
   }
}

void main()
{
   int n,i;
   int x;
```

```
   struct delement *pocetak = NULL ;
   struct delement *kraj = NULL;

   printf("Unesite broj elementa liste:\n");
   scanf("%d",&n);

   for(i=0; i<n; i++)
   {
      printf( "Unesi vrednost:\n");
      scanf("%d",&x);

      pocetak = dodaj( pocetak, &kraj, x );
   }

   printf("Kreirana lista je:\n");
   stampaj_desno( pocetak );
   stampaj_levo( kraj );
}
```

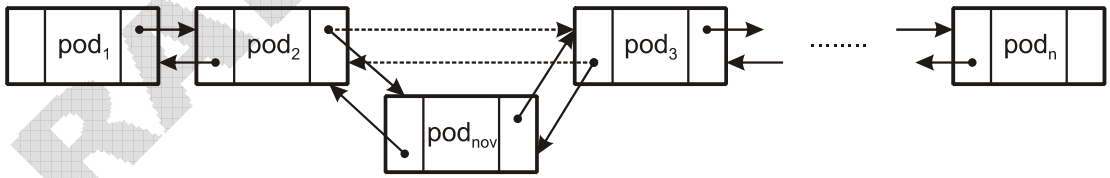
Za kreiranje dvostruko povezane liste ovaj program koristi funkciju *dodaj* koja ima tri parametra: pokazivač na početak liste, pokazivač na kraj liste i vrednost podatka koji treba dodati u listu. Koristeći zadatu vrednost, funkcija kreira novi element, dodaje ga na kraj liste i vraća pokazivač na početak liste.

Na početku je lista prazna, tako da je pokazivač na početak liste NULL. Nakon prvog poziva funkcije *dodaj* novi element postaje i početni element liste. Ukoliko lista nije bila prazna, funkcija smešta pokazivač na novi element u pomoćni pokazivač. Zatim se leva veza novog elementa postavlja da pokazuje na poslednji element postojeće liste, a desna veza postaje NULL. Na kraju se desna veza poslednjeg elementa postojeće liste postavlja da pokazuje na novi element, a zatim pokazivač na kraj dosadašnje liste postavlja da pokazuje na novi element.

Glavni program više puta poziva funkciju *dodaj* u cilju kreiranja dvostruko povezane liste sa zadatim brojem elemenata.

Umetanje elementa iza određenog elementa u dvostruko povezanoj listi

Da bismo umetnuli novi element na određeno mesto u dvostruko povezanoj listi, neophodno je da znamo redni broj elementa u listi iza koga će novi element biti umetnut. Pretpostavljamo da su svi elementi liste numerisani rednim brojevima od 1 do *n*. Kada znamo broj elementa, krećemo se kroz listu kako bismo dobili pokazivač na element sa zadatim rednim brojem. Zatim kreiramo novi element, a njegovom levom i desnom linku dodeljujemo pokazivače na element sa navedenim rednim brojem i njegovog desnog suseda. Nakon toga, levom i desnom elementu novog elementa postavljamo desnu i levu vezu tako da pokazuju na novi element.



Slika ### Umetanje elementa iza određenog elementa u dvostruko povezanoj listi

Primer

Sledeći program vrši umetanje novog elementa iza elementa u dvostruko povezanoj listi sa navedenim rednim brojem.

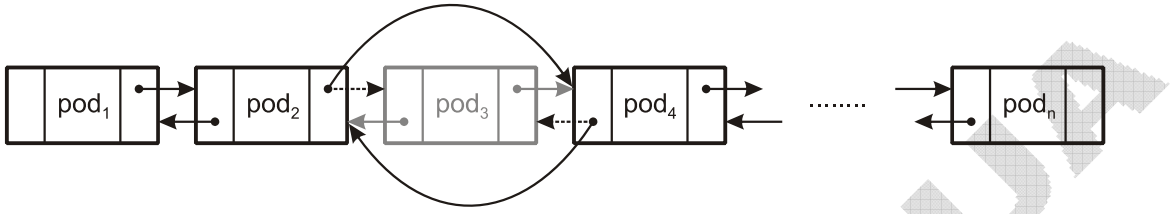
```
# include <stdio.h>
# include <stdlib.h>

struct delement
{
   int podatak;
   struct delement *levi, *desni;
```

```
};

struct delement *dodaj(struct delement *p, struct delement **q, int n)
{
   struct delement *pom;

   /* ukoliko je lista prazna, novi element se dodaje kao pocetni element liste */
   if(p==NULL)
   {
      p=(struct delement *)malloc(sizeof(struct delement)); /* kreiranje novog elementa */
      if(p==NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      p->podatak = n;
      p->levi = p->desni = NULL;  /* novi element nema ni levog ni desnog suseda */
      *q = p;   /* kraj liste je i pocetak liste */
   }
   else
   {
      pom = (struct delement *)malloc(sizeof(struct delement)); /* kreiranje novog elementa */
      if(pom == NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      pom->podatak = n;
      pom->levi = (*q);  /* poslednji element liste postaje levi novom elementu */
      pom->desni = NULL; /* novi element nema desnog suseda */
      (*q)->desni = pom; /* novi element postaje desni poslednjem elementu liste */
      (*q) = pom;        /* novi element postaje poslednji element liste */
   }

   return (p);
}

void stampaj_desno( struct delement *p )
{
   printf("Podaci u listi stampani u desno su:\n");
   while (p!= NULL)
   {
      printf("%d\t",p->podatak);
      p = p->desni;
   }
}

/* funkcija za odredjivanje broja elemenata u listi */
int duzina( struct delement *p )
{
   int broj=0;

   while (p != NULL)
   {
      broj++;
      p = p->desni;
   }

   return(broj);
}

/* funkcija za umetanje novog elementa iza odredjenog elementa u dvostruko poveyanoj listi */
struct delement * umetni( struct delement *p, int redni_broj, int vrednost )
{
   struct delement *pom, *pom1;
   int i;

   if ( redni_broj <= 0 || redni_broj > duzina(p) )
   {
      printf("Greska! Zadati element ne postoji.\n");
      exit(0);
   }
```

```
   if ( redni_broj == 0)
   {
      pom = ( struct delement * )malloc( sizeof( struct delement ));
      if ( pom == NULL )
      {
         printf( "Greska pri alociranju memorije.\n");
         exit (0);
      }

      pom->podatak = vrednost;
      pom->desni = p;
      pom->levi = NULL;
      p->levi = pom;
      p = pom;
   }
   else
   {
      pom = p ;
      i = 1;
      while ( i < redni_broj )
      {
         i = i+1;
         pom = pom->desni;
      }

      pom1 = ( struct delement * )malloc( sizeof(struct delement));
      if ( pom == NULL )
      {
         printf( "Greska pri alociranju memorije.\n");
         exit(0);
      }

      pom1->podatak = vrednost;
      pom1->desni = pom->desni;
      pom1->levi = pom;
      pom1->desni->levi = pom1;
      pom1->levi->desni = pom1;
   }

   return (p);
}

void main()
{
   int n;
   int x;
   struct delement *pocetak = NULL ;
   struct delement *kraj = NULL;

   printf("Unesite broj elemenata u listi:\n");
   scanf("%d",&n);

   for(i=0; i<n; i++)
   {
      printf( "Unesite element:\n");
      scanf("%d",&x);

      pocetak = dodaj( pocetak, &kraj, x );
   }

   printf("Lista pre umetanja je:\n");
   stampaj_desno( pocetak );

   printf("Unesite redni broj elementa iza koga zelite da umetnete novi element:\n");
   scanf("%d",&n);

   printf("Unesite vrednost novog elementa:\n");
   scanf("%d",&x);

   pocetak = umetni( pocetak, n, x);

   printf("Lista posle umetanja je:\n");
   stampaj_desno( pocetak );
}
```

Brisanje elementa iz dvostruko povezane liste

Da bismo obrisali određeni element iz dvostruko povezane liste potrebno je da znamo njegovu poziciju u listi. Jedan od načina je da navedemo njegov redni broj, uz pretpostavku da su elementi numerisani redom od 1 do n. Kada znamo redni broj elementa, krećemo se kroz listu kako bismo pronašli pokazivač na traženi element. Nakon toga možemo da obrišemo traženi element, odnosno da oslobodimo memoriju koju on zauzima. Za razliku od brisanja elementa iz jednostruko povezane liste, u ovom slučaju nije potrebno da znamo i pokazivač na prethodni element u listi, s obzirom da u dvostruko povezanoj listi svaki element ima pokazivač na svog levog i desnog suseda.



Slika ### Šematski prikaz brisanja elementa iz dvostruko povezane liste

Primer

Sledeći program vrši brisanje elementa iz dvostruko povezane liste:

```
# include <stdio.h>
# include <stdlib.h>

struct delement
{
   int podatak;
   struct delement *levi, *desni;
};

struct delement *dodaj(struct delement *p, struct delement **q, int n)
{
   struct delement *pom;

   /* ukoliko je lista prazna, novi element se dodaje kao pocetni element liste */
   if(p==NULL)
   {
      p=(struct delement *)malloc(sizeof(struct delement)); /* kreiranje novog elementa */
      if(p==NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      p->podatak = n;
      p->levi = p->desni = NULL;  /* novi element nema ni levog ni desnog suseda */
      *q = p;   /* kraj liste je i pocetak liste */
   }
   else
   {
      pom = (struct delement *)malloc(sizeof(struct delement)); /* kreiranje novog elementa */
      if(pom == NULL)
      {
         printf("Greska.\n");
         exit(0);
      }

      pom->podatak = n;
      pom->levi = (*q);   /* poslednji element liste postaje levi novom elementu */
      pom->desni = NULL;  /* novi element nema desnog suseda */
      (*q)->desni = pom;  /* novi element postaje desni poslednjem elementu liste */
      (*q) = pom;         /* novi element postaje poslednji element liste */
   }

   return (p);
}
```

```
/* funkcija za odredjivanje broja elemenata u listi */
int duzina( struct delement *p )
{
   int broj=0;

   while (p != NULL)
   {
      broj++;
      p = p->desni;
   }

   return(broj);
}

/* funkcija za brisanje elementa iz dvostruko povezane liste */
struct delement * obrisi( struct delement *p, struct delement **q, int redni_broj)
{
   struct delement *pom;
   int i;

   if (redni_broj <= 0 || redni_broj > duzina(p))
   {
      printf("Greska! Zadati element ne postoji.\n");
      exit(0);
   }

   pom = p ;
   i = 1;
   while ( i < redni_broj )
   {
      i = i+1;
      pom = pom->desni;
   }

   if(pom->levi != NULL)
      pom->levi->desni = pom->desni;
   else
      p = pom->desni;

   if(pom->desni != NULL)
      pom->desni->levi = pom->levi;
   else
      (*q) = pom->levi;

   free(pom);

   return (p);
}

void main()
{
   int n;
   int x;
   struct delement *pocetak = NULL ;
   struct delement *kraj = NULL;

   printf("Unesite broj elemenata liste:\n");
   scanf("%d",&n);

   for(i=0; i<n; i++)
   {
      printf( "Unesite vrednost elementa:\n");
      scanf("%d",&x);
      pocetak = dodaj( pocetak, &kraj, x );
   }

   printf("Lista pre brisanja je:\n");
   stampaj_desno( pocetak );
   printf("Unesite redni broj elementa koji treba obrisati:\n");
   scanf("%d",&n);

   pocetak = obrisi( pocetak, &kraj, n );
```

```
    printf("Lista posle brisanja je:\n");
    stampaj_desno( pocetak );
}
```