

Dinamične podatkovne strukture

- Dinamične rezervacije pomnilnika
- Enojno povezani kazalčni seznamii
 - Urejeni
 - Neurejeni
- Dvostransko povezani kazalčni seznamii
- Krožni kazalčni seznamii
- Binarno iskalno drevo
- AVL drevo (Adel'soon Velskii and Landis tree)

Dinamične rezervacije pomnilnika

- Vse deklarirane spremenljivke imenujemo statične spremenljivke, za katere program ob inicializaciji rezervira prostor v pomnilniku, ki ga preden se program konča tudi sprosti.
- Za dinamične rezervacije velja, da lahko med izvajanjem programa prostor za podatek rezerviramo ali sprostimo del pomnilnika (ukaza **new** in **delete**) za velikost neke določene podatkovne strukture, za katero potrebujemo kazalčne spremenljivke.
- Del pomnilnika, kjer je možna dinamična rezervacija imenujemo **kopica “heap“**, na kateri lahko realiziramo dinamične podatkovne strukture kot so seznamy in drevesa.

Ukazi za delo s kopico (Heapom)

New

- Sintaksa uporabe : **tip *Ksprem = new tip**
- **tip** je poljubni enostavni ali sestavljeni tip spremenljivke
- **new** je operator, ki dodeli spomin velikosti *sizeof(tip)* in vrne kazalec tipa *tip* z naslovom dodeljenega spomina
- **Ksprem** ime kazalca tipa tip, ki kaže na nov prostor za podatek. Samo preko kazalca lahko dodeljeni spomin tudi sprostimo !!

Delete

- Prostor za podatek ostane rezerviran, dokler ga ne pobrišemo. Operator **delete** sprosti spomin dodeljen z operatorjem new.
- Sintaksa uporabe : **delete Ksprem**

Primeri uporabe new in delete

```
int *kst = new int;           // dodeli en int na kopici
char *kznak = new char;      // dodeli en char na kopici
float *krealno = new float;  // dodeli na kopici en float
```

.....

```
delete kst;                // sprostimo prostor za kst
delete kznak;               // sprostimo prostor za kznak
delete krealno;             // sprostimo prostor za krealno
```

.....

```
Struct Txt
{ char niz [100];
  int indeks ; };
Txt *Mytext = new Txt;
```

.....

```
Delete Mytext;
```

Enostransko povezan kazalčni seznam (single linked list)

- Deklaracija podatkovnega tipa elementa seznama :

```
struct element
```

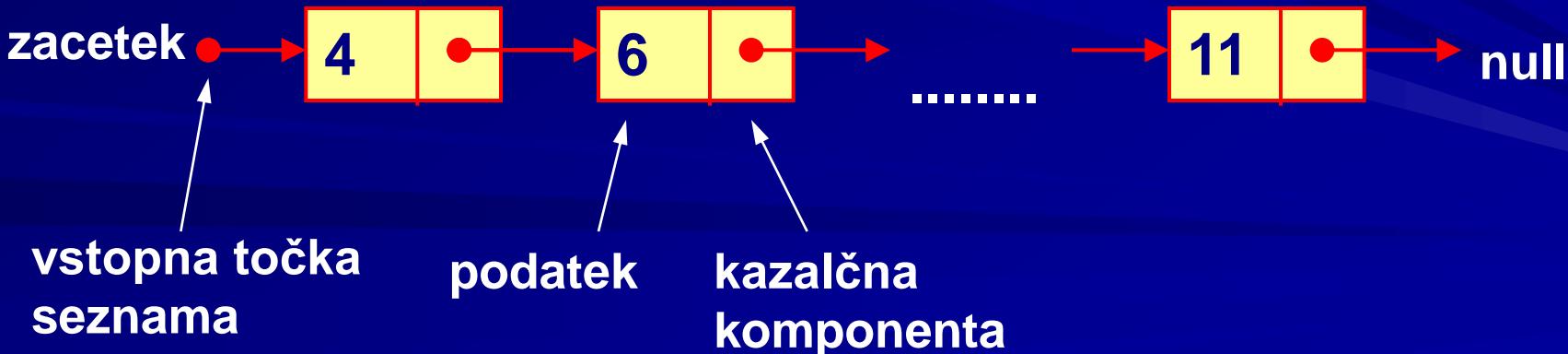
```
{ tip1 podatek1;
```

```
    tip2 podatek2;
```

```
.....
```

```
    tipN podatekN;
```

```
struct element *nasl; } *kazalec;
```



Neurejeno povezan enosmerni seznam - implementacija v C++

- Deklaracija privatne lastnosti **zacetek**, ki predstavlja kazalec na element seznama
- Najava vseh metod, ki jih potrebujemo za delo s seznamom: **konstruktor, dodajanje (push), brisanje (pop), metoda za izpis celotnega seznama**

```
class Seznam
{ struct element
  { int podatek;
    struct element *nasl; } *zacetek;

public:
  Seznam();
  void dodajElement(int st);
  bool brisiElement();
  void izpisiVse(); };
```

Neurejeno povezan enosmerni seznam

- **Definicija konstruktorja (Seznam)**
- Kazalec zacetek moramo obvezno postaviti na NULL, sicer ne moremo vstavljati novih elementov v seznam



```
Seznam::Seznam()  
{ zacetek=NULL; }
```



Neurejeno povezan enosmerni seznam

- **Metoda za dodajanje elementa v seznam (push)**
- Nov element dodamo na začetek seznama. Koraki so naslednji :
 1. Dinamična rezervacija za novi element seznama.
 2. Kazalčni komponenti novega elementa priredimo vrednost kazalca zacetek.
 3. Kazalcu zacetek priredimo vrednost kazalca tmp.

```
void Seznam::push (int st)
{ struct element *tmp=new struct element;
tmp->podatek=st;
tmp->nasl=zacetek;
zacetek=tmp; }
```

Neurejeno povezan enosmerni seznam

- **Metoda za brisanje elementa seznama (pop)**
- Metoda pop vedno briše začetni element seznama, a le če seznam ni prazen

```
bool Seznam::pop()
{ struct element *tmp;
  bool status=1;
  if (zacetek!=NULL)
  { tmp=zacetek;
    zacetek=tmp->nasl;
    delete tmp;}
    else status=0;
  return status;
}
```

Neurejeno povezan enosmerni seznam

- **Metoda za vseh elementov seznama (pop)**
- Metoda vsebuje zanko, ki se s pomočjo kazalca tmp sprehodi po vseh elementi seznama, dokler kazalec ni enak vrednosti NULL
- Na podoben način lahko realiziramo tudi ostale metode: metoda za izračun vsote, povprečja, ipd.

```
void Seznam::izpisiVse()
{ struct element *tmp;
  for (tmp=zacetek; tmp!=NULL; tmp=tmp->nasl)
    cout << tmp->podatek << "," ;
}
```

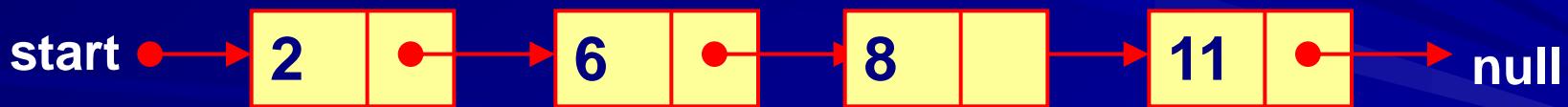
Neurejeno povezan enosmerni seznam

- Glavni program, ki izvede s pomočjo objekta kazalčni kliče vse metode razreda Seznam

```
int main(int argc, char *argv[])
{ QCoreApplication a(argc, argv);
Seznam kazalcni; int n;
cin>>n;
while (n!=0)
{ kazalcni.push(n); cin>>n; }
kazalcni.izpisiVse();
cout<<"Brisanje!"<<endl;
if (kazalcni.pop()) kazalcni.izpisiVse();
else
cout<<"Brisanje elementa neuspesno!";
return 0; }
```

Urejeno povezan enosmerni seznam

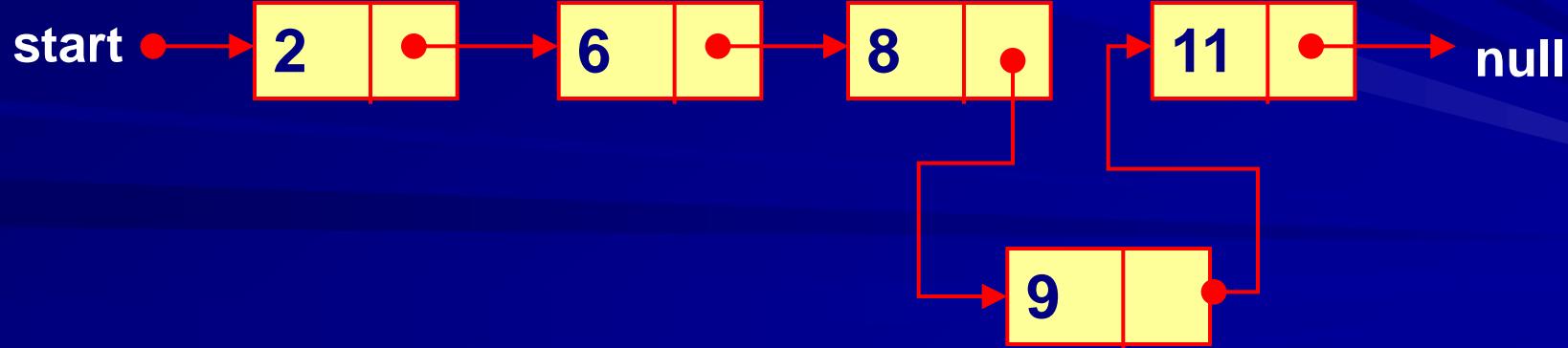
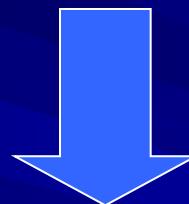
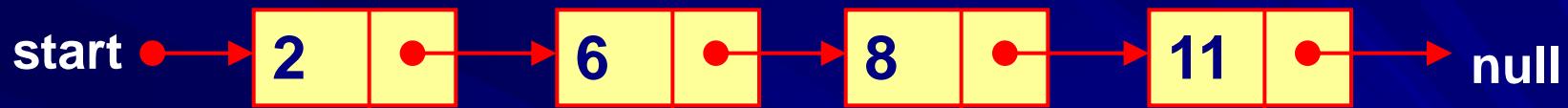
- Za urejene sezname velja, da je vrstni red elementov določen s ključem, ki ga sestavlja en ali več podatkov elementov, recimo (podatek1,podatek3).
- Pri urejenem povezanim seznamu je zato potrebno realizirati še funkcije:
 - funkcijo vstavljanja
 - funkcijo brisanja elementov seznama



Seznam, ki je urejen samo po celoštevilskem podatku element

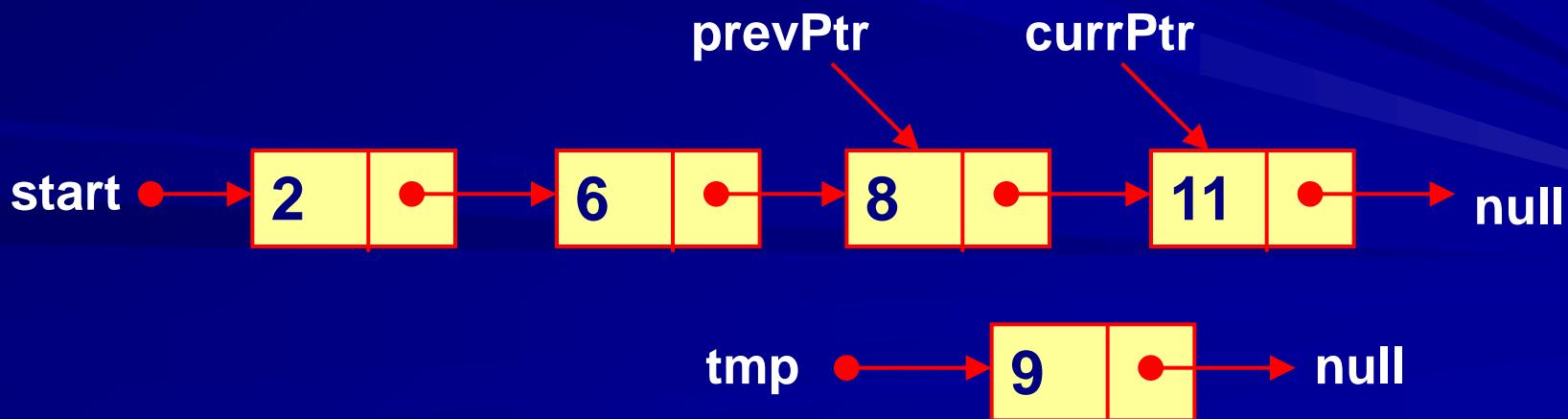
Funkcija vstavljanja

- Primer : seznamu dodamo element z vrednostjo 9



Postopek vstavljanja

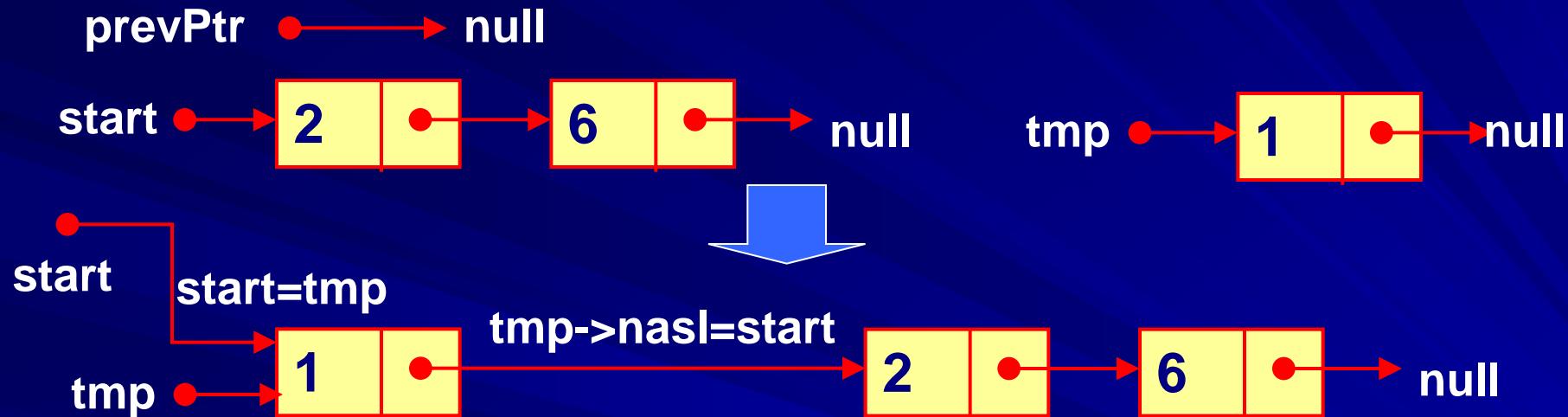
1. Dinamična rezervacija za hranjenje novega elementa
2. Določi mesto vstavljanja:
 - 2.1. postavi *currPtr* na začetek seznama in *prevPtr* na vrednost *NULL*
 - 2.2. dokler je *currPtr* različen od *NULL* in vrednost podatka, ki ga vstavljamo *< currPtr->data*, potem :
 - Spremenljivki *prevPtr* priredi vrednost spremenljivke *currPtr*
 - Premakni vrednost spremenljivke *currPtr* na naslednji element seznama



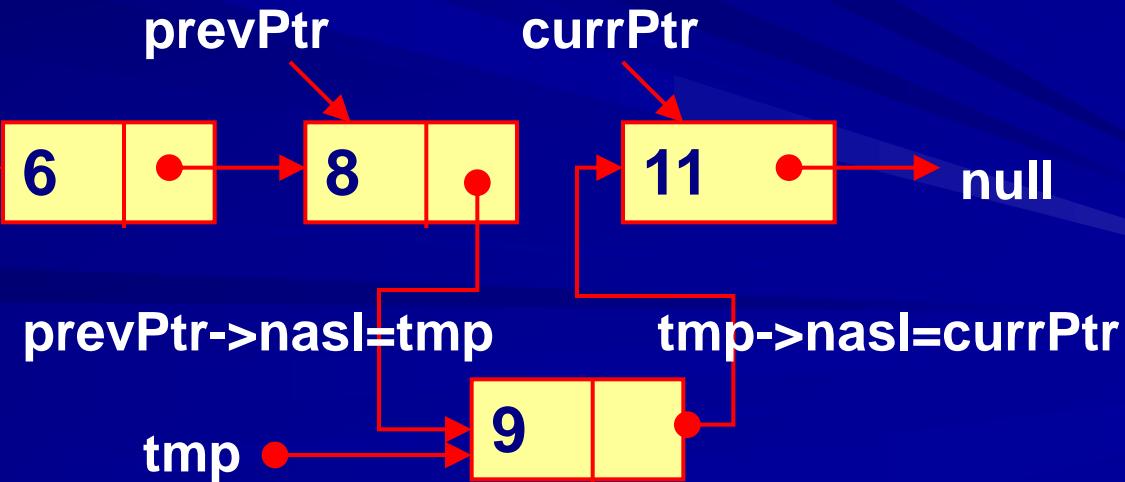
Postopek vstavljanja - nadaljevanje

3. Vstavljanje novega elementa v seznam

3.1. na prvo mesto (*prevPtr* ima vrednost *NULL*)



3.2 na izbrano mesto

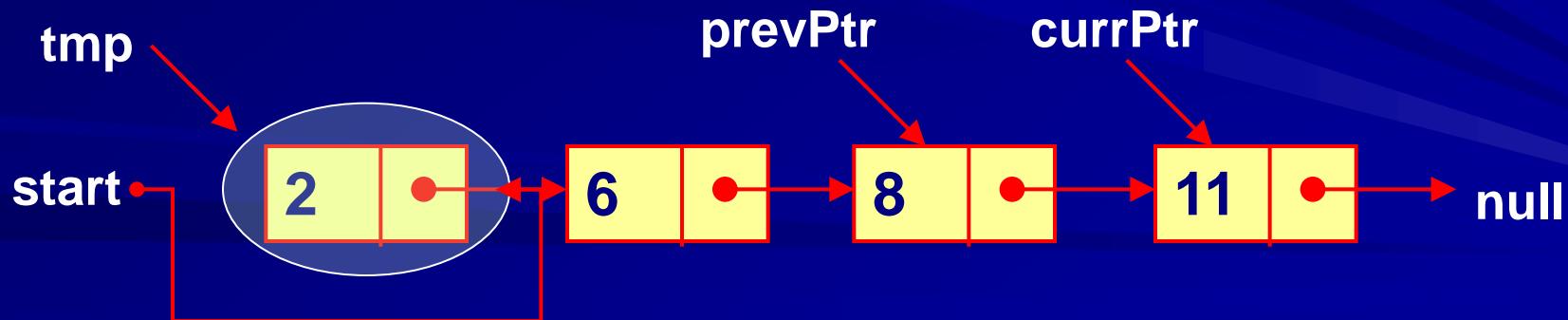


Podprogram za vstavljanje novega elementa v urejeno povezan enostranski seznam

```
void InsertN (int data)
{ struct element *prevPtr, *currPtr;
  prevPtr=NULL; currPtr=start;
  struct element *tmp=new struct element;
  tmp->data=data;
  while ( (currPtr != NULL) && (data > currPtr->data))
  { prevPtr=currPtr;
    currPtr=currPtr->nasl; }
  if (prevPtr==NULL)           /* vstavljanje na začetek seznama */
  { tmp->nasl=start;
    start=tmp; }
  else
  { tmp->nasl=currPtr;
    prevPtr->nasl=tmp; }
}
```

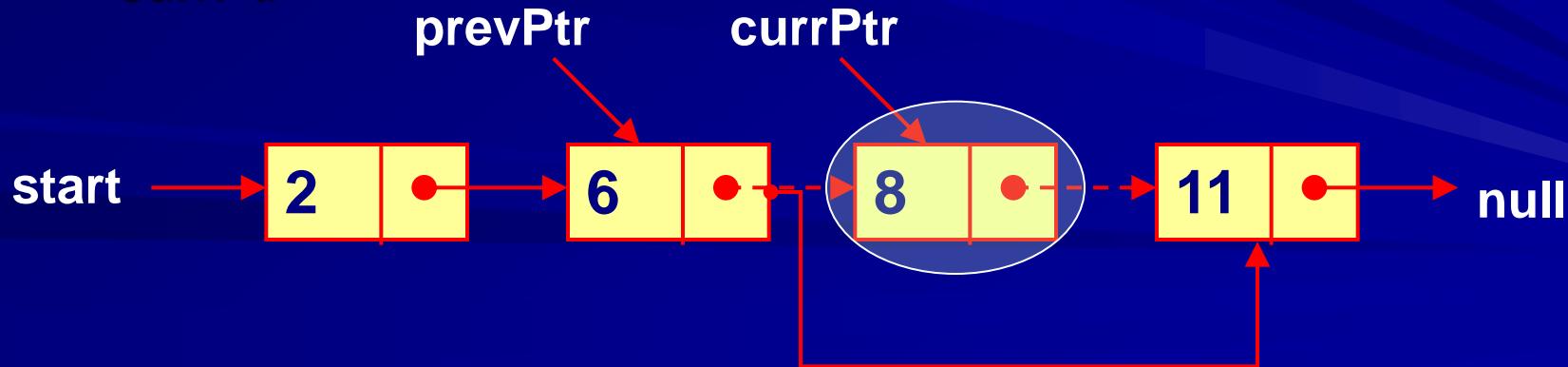
Funkcija brisanja

1. Če element s ključem, ki ga brišemo na začetku seznama:
 1. kazalčni spremenljivki *tmp* priredimo naslov prvega elementa seznama
 2. kazalec *start* postavi na naslednji element seznama
 3. sprostitev elementa seznama na katerega kaže začasni kazalec



Funkcija brisanja - nadaljevanje

2. Če element s ključem, ki ga brišemo ni na začetku seznama, poišči vozlišče za brisanje:
1. inicializiraj *prevPtr* z začetkom seznama (1.element) in *currPtr* na naslednji element
 2. ponavljaj zanko, dokler je *currPtr* različen od NULL in je vrednost ključa, ki ga brišemo različna od *currPtr->data*
 - 2.1. priredi *prevPtr* na *currPtr*
 - 2.2. premakni *currPtr* na naslednje vozlišče
 3. če vrednost kazalčne spremenljivke *currPtr* različna od NULL
 - 3.1 postavi *prevPtr->nasl* na *currPtr->nasl*
 - 3.1 sprosti element seznama na katerega kaže spremenljivka *currPtr*



Podprogram za brisanje elementa iz seznama

```
short Delete (int data)
{ struct element *prevPtr, *currPtr, *tmp;
short status=0;
if (start!=NULL)      /* seznam ni prazen */
{ if (start->data==data)    /*brisanje prvega elementa seznama */
 { tmp=start;
  start=start->nasl;
  delete tmp;
  status=1; } else
{ prevPtr=start; currPtr=start->nasl;
 while ((currPtr != NULL) && (data != currPtr->data))
{ prevPtr=currPtr;
  currPtr=currPtr->nasl; }
if (currPtr!=NULL)
{ prevPtr->nasl=currPtr->nasl;
  delete currPtr;
  status=1; } }
} return (status);}
```

Dvostransko povezan seznam

- Za dvostransko povezane sezname velja, da vsak element poleg osnovnih podatkov vsebuje še dve kazalčni komponenti in sicer kazalec na naslednji in prejšnji element seznama.



Lastnosti dvostransko povezanega seznama:

- element seznama vsebuje dve kazalčni komponenti (prejšnji, naslednji, element)
- dve vstopni točki v seznam (glava, rep)
- pomikanje po seznamu ni več strogo linearne, kot pri enostransko povezanih seznamih

Deklaracija dvostransko povezanega seznama

```
struct element  
{ tip1 podatek1;  
    tip2 podatek2;  
    .....  
    tipN podatekN;  
struct element *nasl;  
struct element *prej; } *glava,*rep;
```

- Primer deklaracije dvostranskega seznama realnih števil:

```
struct realni  
{ float stevilo;  
struct realni *nasl, *prej; } *glava,*rep;
```

Operaciji dodajanja (push) in brisanja (pop)

■ Operacija dodajanja na 1.mesto seznama :

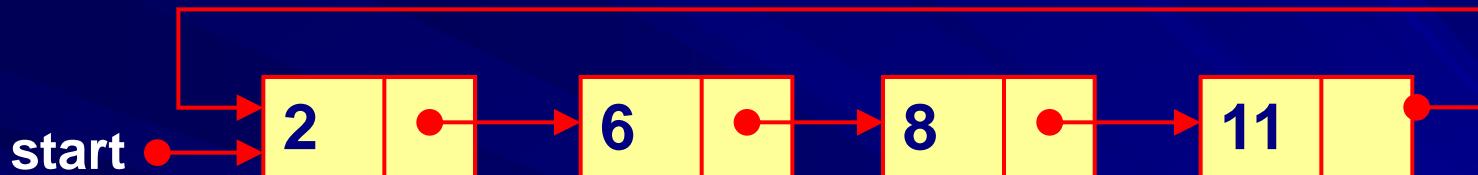
```
void Push (int Data)
{ struct element *tmp=new struct element;
  tmp->data=Data;
  Glava->prej=tmp;
  tmp->nasl=Glava;
  tmp->prej=NULL;
  Glava=tmp;}
```

■ Operacija brisanja s 1.mesta seznama :

```
void Pop ()
{ if (Glava != NULL)
  { struct element *tmp=new struct element;
    tmp=Glava;  Glava=Glava->nasl;
    Glava->prej=NULL;
    delete tmp; } }
```

Krožni seznam (circular list)

- Za krožno povezane sezname (enostranske ali dvostranske) velja, da zadnji element v seznamu vsebuje povezavo na 1. element seznama



Lastnosti krožnega seznama :

- vstopno točko seznama lahko poljubno prestavljamo
- enostavna realizacija operaciji brisanja in dodajanja (ni potrebno paziti na začetek in konec seznama)
- ni konca seznama (vsi elementi seznama so med seboj povezani)