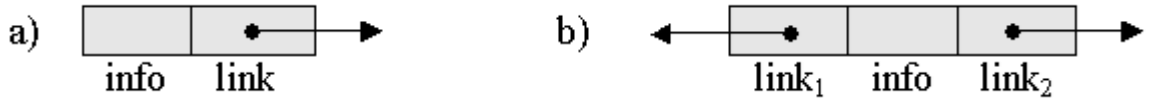
Lančane liste

Definicija

Lančane liste (*linked lists*) su linearne strukture kod kojih svaki element ukazuje na svoje susede u okviru te strukture.

Čvorovi

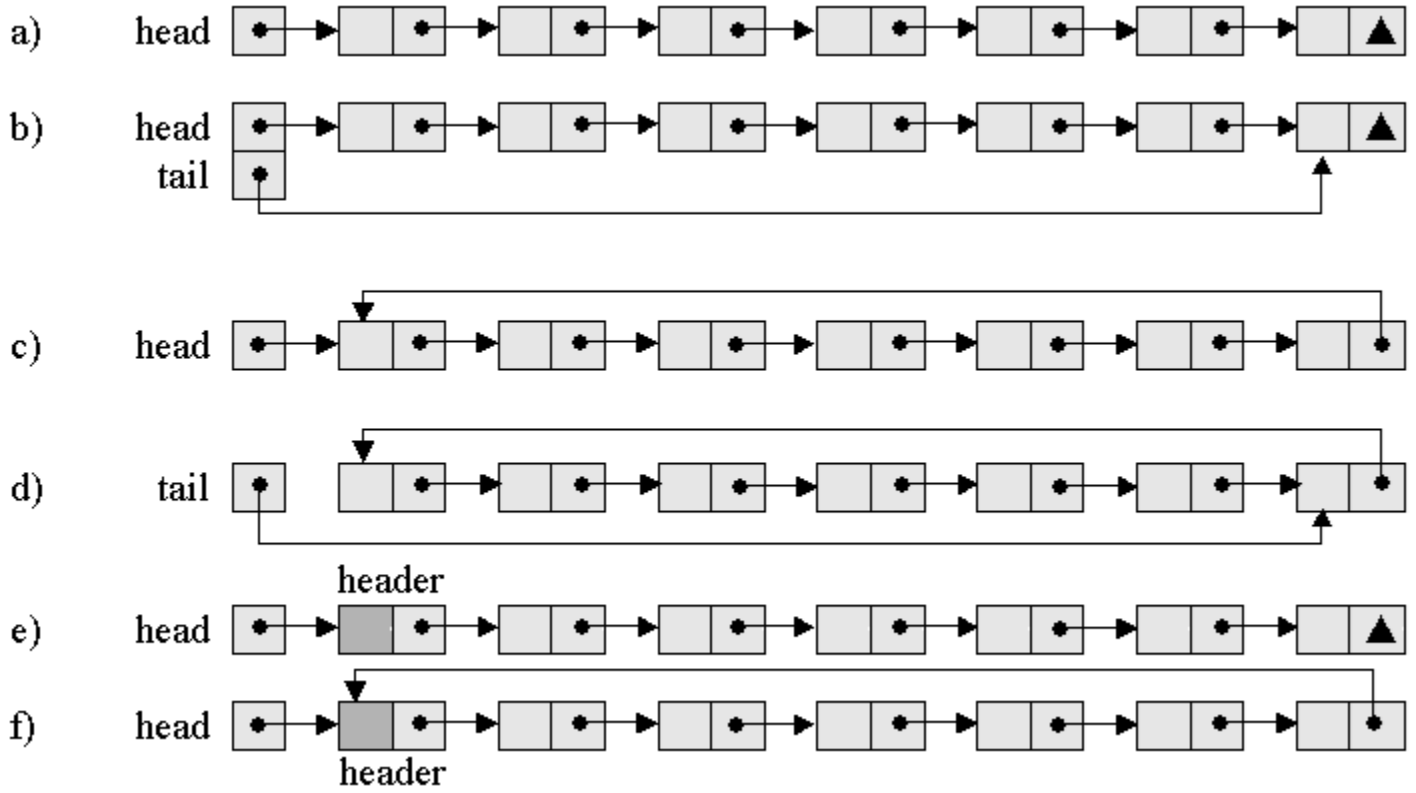


Struktura čvora lančane liste:
a) čvor jednostruko ulančane liste, b) čvor dvostruko ulančane liste

Lančana lista može biti:

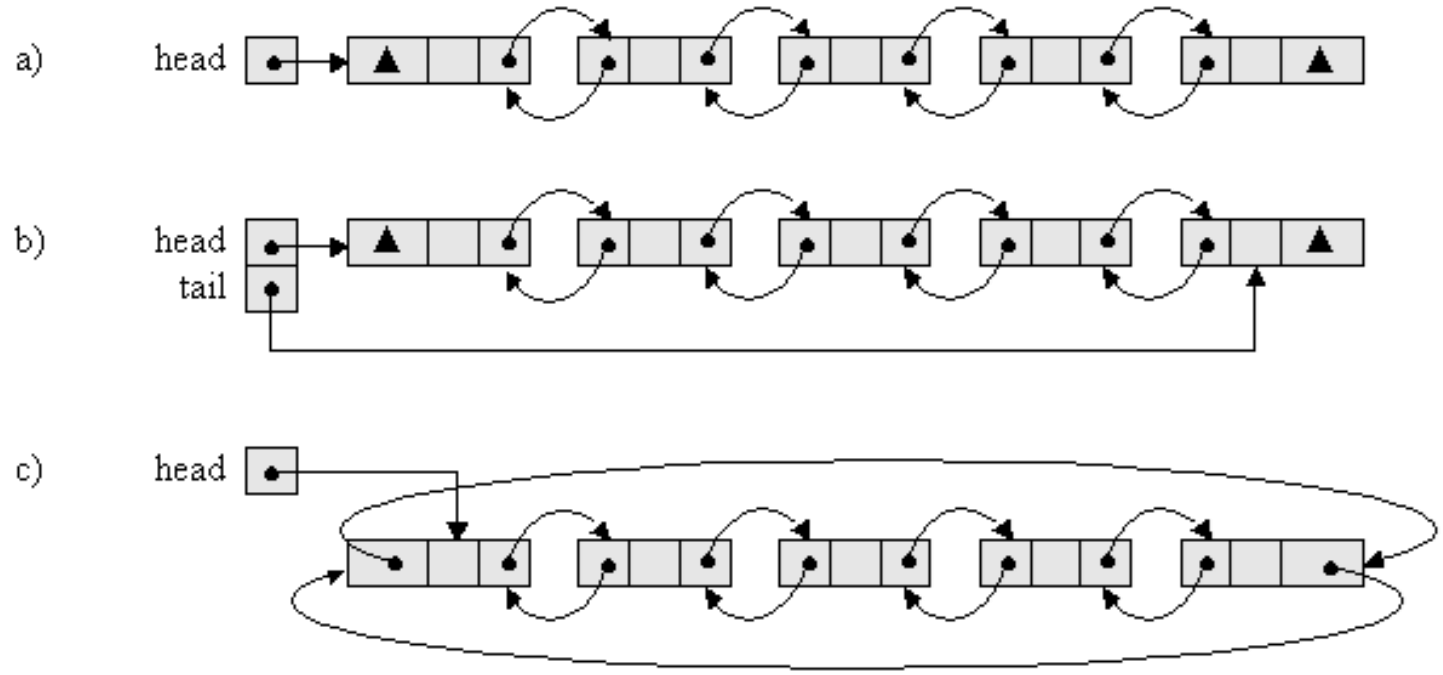
- **jednostruko ulančana** (engl. *singly linked list*) ili
- **dvostruko ulančana** (engl. *doubly linked list*)

Memorijska reprezentacija jednostruko ulančane liste



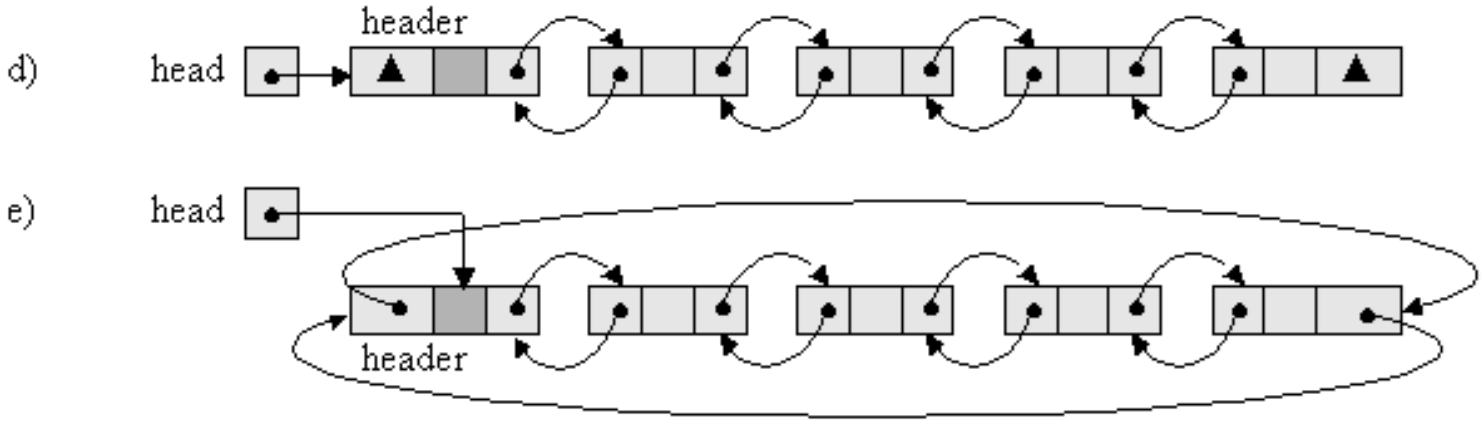
Jednostruko ulančane liste: a) sa pokazivačem na početak, b) sa pokazivačima na početak i kraj, c) ciklična lista sa pokazivačem na početak, d) ciklična lista sa pokazivačem na kraj, e) lista sa zaglavljem i f) ciklična lista sa zaglavljem

Dvostruko ulančane liste



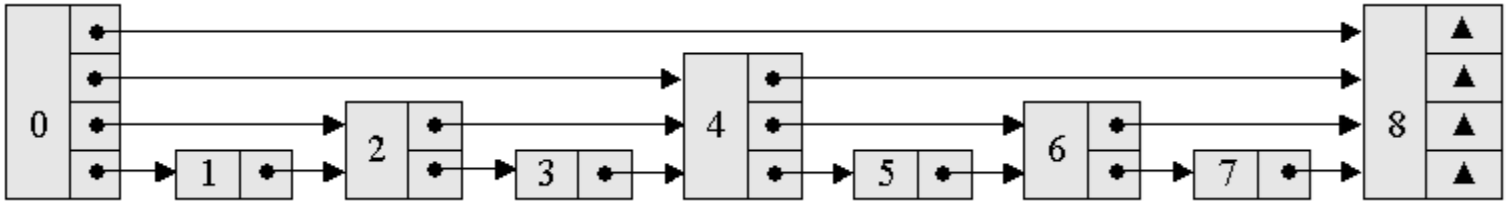
Dvostruko ulančane liste: a) samo sa pokazivačem na početak, b) sa pokazivačima na početak i kraj, c) ciklična dvostruko ulančana lista

Dvostruko ulančane liste



Dvostruko ulančane liste: d) lista sa zaglavljem i e) ciklična dvostruko ulančana lista sa zaglavljem

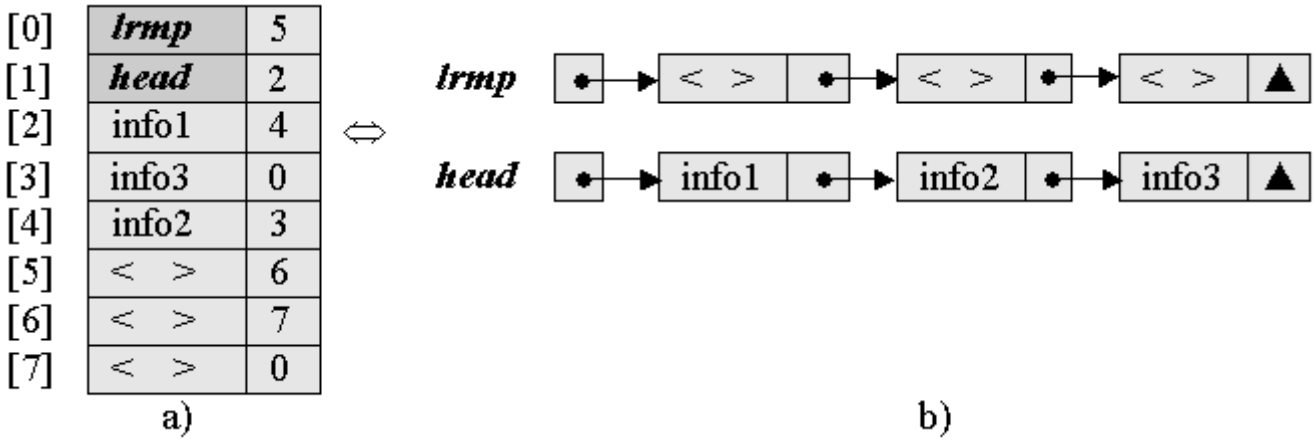
Liste sa preskokom (*skip lists*)



U listi sa preskokom sa **n** čvorova, za svako **k** ($1 \le k \le \lfloor \log_2 n \rfloor$) i **i** ($1 \le i \le \lfloor n / 2^{k-1} \rfloor - 1$), čvor na poziciji $\mathbf{i \cdot 2^{k-1}}$ ukazuje na čvor na poziciji $\mathbf{(i+1) \cdot 2^{k-1}}$. Naime, svaki drugi čvor ukazuje na čvor za dve pozicije ispred njega, svaki četvrti za četiri pozicije ispred itd.

$$\text{maxLevel} = \lfloor \log_2 n \rfloor + 1$$

Statička memorijska reprezentacija lančanih listi



Statička reprezentacija jednostruko ulančane liste:
a) lista predstavljena poljem, b) dinamički ekvivalent

Ostale definicije i pojmovi

Lančana lista se naziva **uređena lačana lista**, ako njeni elementi slede neki poredak (alfabetski, leksikografski, uređenje u rastući ili opadajući redosled i sl.).

Ako lančana lista zadržava uređenost nakon svake operacije, tada se naziva **sortirana lančana lista**.

Samoorganizujuća lista je lista koja preuređuje svoje čvorove, koristeći neku heuristiku, kako bi poboljšala efikasnost pristupa. Reorganizacija se najčešće svodi na pomeranje čvora kome je poslednji put pristupano na početak lančane liste, tako da sledeći pristup tom čvoru ne zahteva prolazak kroz listu (ova operacija se naziva **pomeranje na početak**), ili zamena mesta čvora kome je pristupano i njegovog prethodnika (ova operacija se naziva **transpozicija**).

Osnovne operacije

Osnovne operacije za rad sa lančanom listom su:

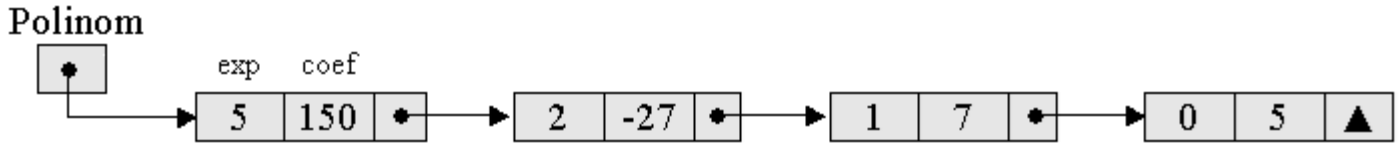
- **traverse** - obilazak lančane liste,
- **find** - pronalazi zadati čvor, ukoliko se on nalazi u listi,
- **insert** - umeće novi čvor u listu
  - na početak lančane liste (**addHead**),
  - na kraj lančane liste (**addTail**),
  - iza zadatog čvora (**insertAfter**) i
  - ispred zadatog čvora (**insertBefore**). i
- **delete** - briše zadati čvor iz liste.

Osim ovih operacija, često se implementiraju i sledeće:

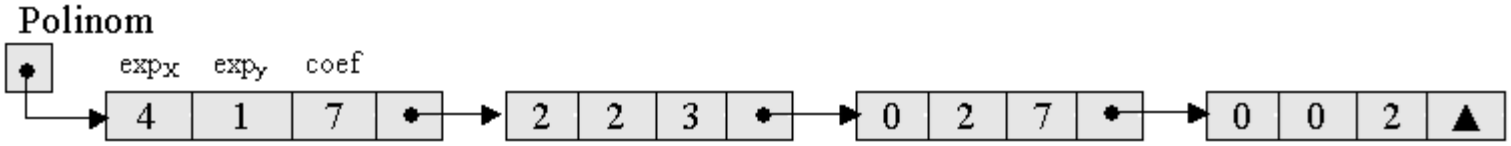
- **sort** - uređuje elemente lančane liste po zadatom kriterijumu,
- **copy** - kopira elemente jedne liste u drugu,
- **concatenate** - nadovezuje jednu listu na kraj druge i
- **split** - cepa jednu lančanu listu na dve zasebne liste.

Lančana reprezentacija polinoma

$$P_n(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_i \cdot x^i + \ldots + a_1 \cdot x + a_0$$

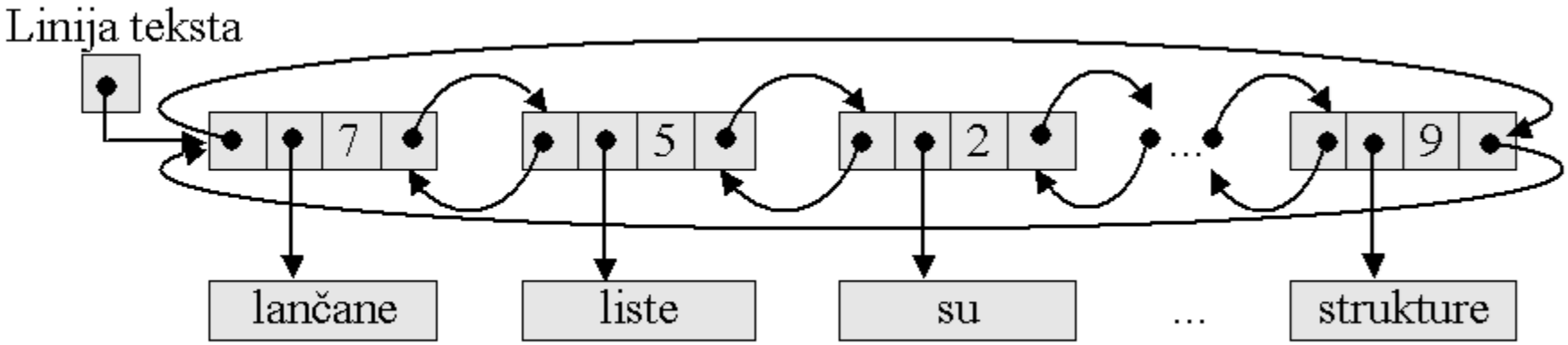


Primer memorijske reprezentacije polinoma $150 \cdot x^5 - 27 \cdot x^2 + 7 \cdot x + 5$ korišćenjem jednostruko ulančane liste

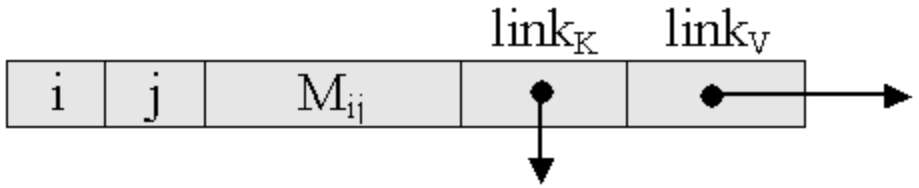


Primer memorijske reprezentacije polinoma $7 \cdot x^4 \cdot y + 3 \cdot x^2 \cdot y^2 + 7 \cdot y^2 + 2$ korišćenjem jednostruko ulančane liste.

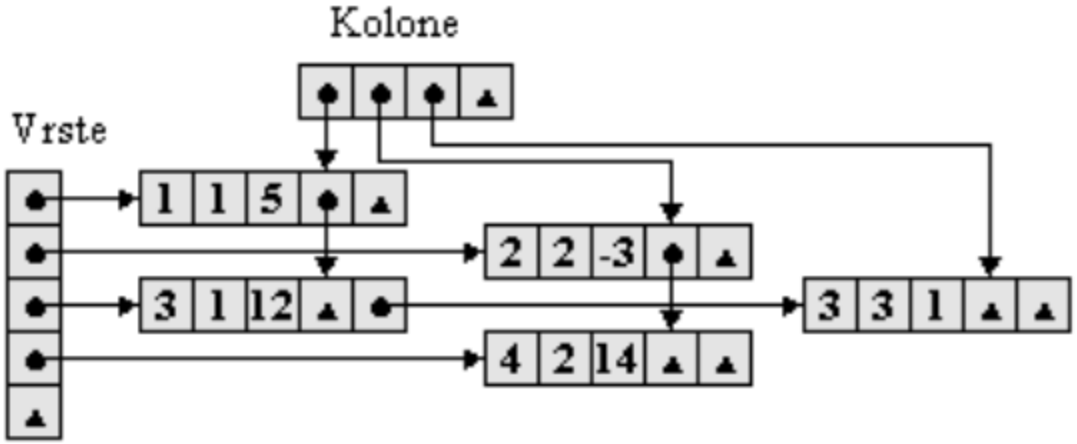
Lančana reprezentacija teksta



Lančana reprezentacija retko posednute matrice



$$M = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 \\ 12 & 0 & 1 & 0 \\ 0 & 14 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

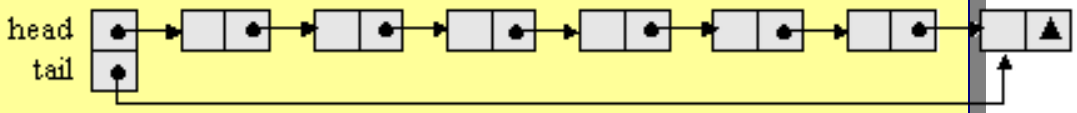


Klasa čvora jednostruko ulančane liste

```
template <class T>
class SLLNode
{
public:
      T info;
      SLLNode<T>* next;          // pokazivac na sledbenika
public:
            SLLNode() { next = NULL; };
            SLLNode(T i) { info = i; next = NULL; };
            SLLNode(T i, SLLNode<T>* n) {
                        info = i; next = n;
             };
            ~SLLNode() { };
            T print() {return info;};
            bool isEqual(T el) {return el == info};

};
```

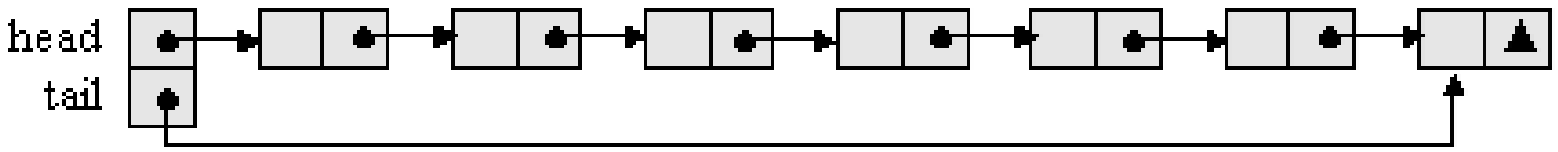
Zaglavlje klase jednostruko ulančane liste (SLL)

```
template <class T>
class SLList
{
protected:
      SLLNode<T> *head, *tail; // pokazivaci pocetka i kraja liste
public:
     SLList() { head = tail = NULL; }
     ~SLList();
      bool isEmpty() { return head == NULL; }
     void addToHead(T el);
     void addToTail(T el);
     T deleteFromHead();
     T deleteFromTail();
     SLLNode<T>* findNodePtr(T el);
     SLLNode<T>* getHead() {return head;}
     SLLNode<T>* getNext(SLLNode<T>* ptr) ;
     T getHeadEl();
     T getNextEl(T el) ;
     void printAll();
     bool isInList(T el);
     void deleteEl(T el);
};
```



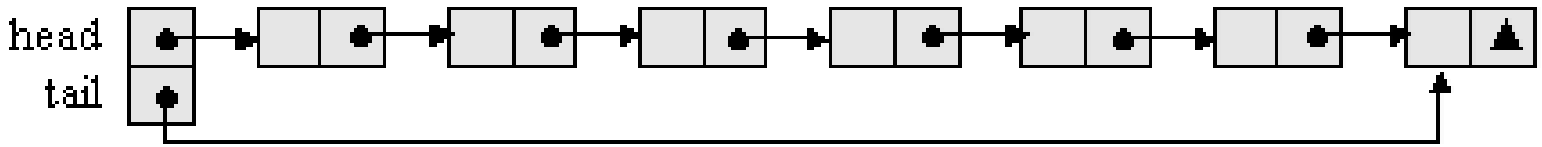
Destruktor i metod za štampanje SLL

```
template <class T>
SLList<T>::~SLList()
{
        while(!isEmpty()){
                T tmp = deleteFromHead();
        }
}

template <class T>
void SLList<T>::printAll() {
    for (SLLNode<T>* tmp = head; tmp != NULL; tmp = tmp->next)
                cout << tmp->print() << " ";
}
```



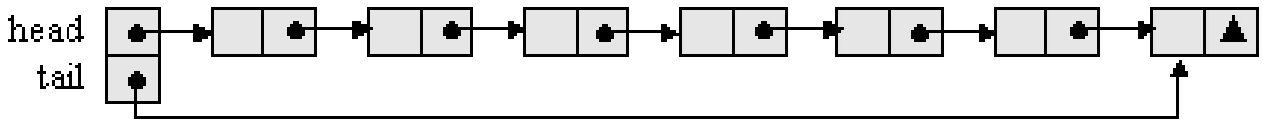
Dodavanje elementa

```
template <class T>
void SLList<T>::addToHead(T el) {
      head = new SLLNode<T>(el, head);
      if (tail == NULL) tail = head;
}

template <class T>
void SLList<T>::addToTail(T el) {
      if (!isEmpty()) {
            tail->next = new SLLNode<T>(el);
            tail = tail->next;
      }
      else
            head = tail = new SLLNode<T>(el);
}
```

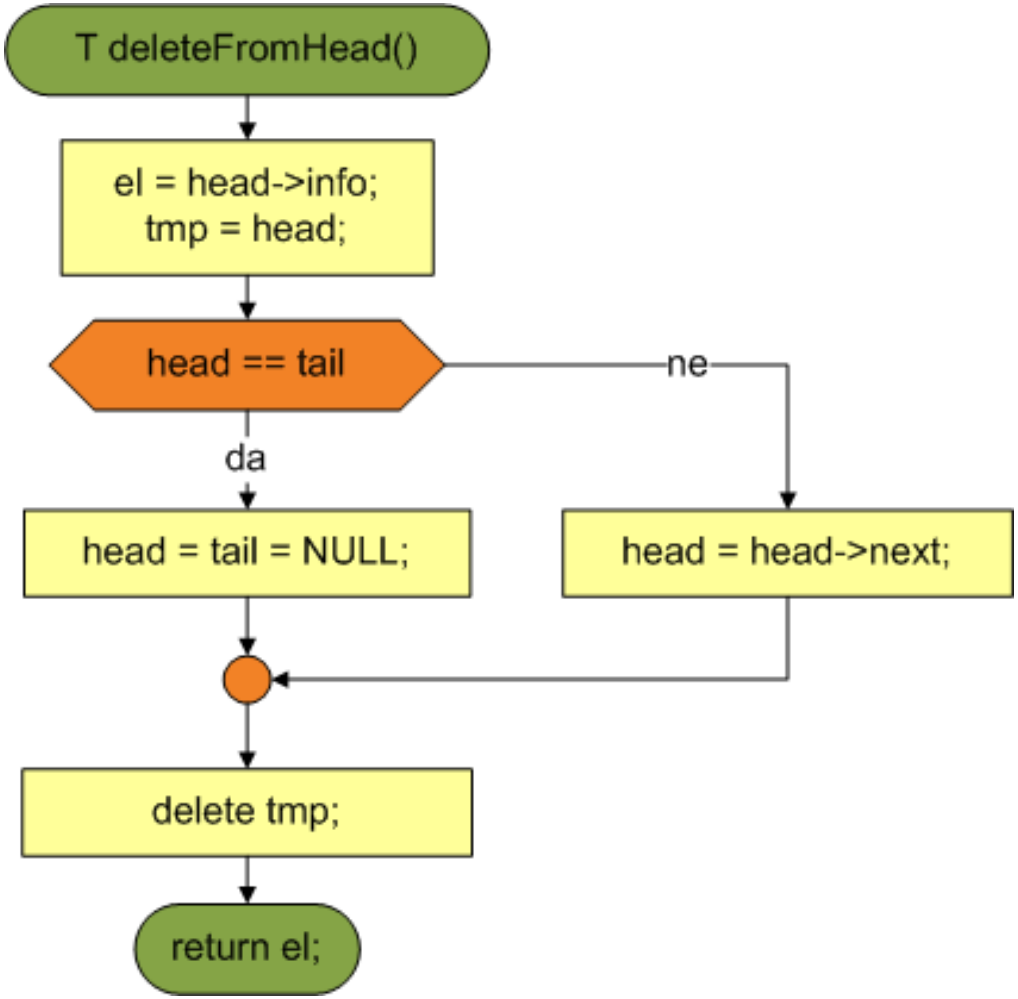


Brisanje elementa sa početka

```
template <class T>
T SLList<T>::deleteFromHead()
{
    if( isEmpty() )
        throw new SBPException("Lista je prazna!");
    T el = head->info;
    SLLNode<T>* tmp = head;
    if (head == tail)
        head = tail = NULL;
    else
        head = head->next;
    delete tmp;
    return el;
}
```

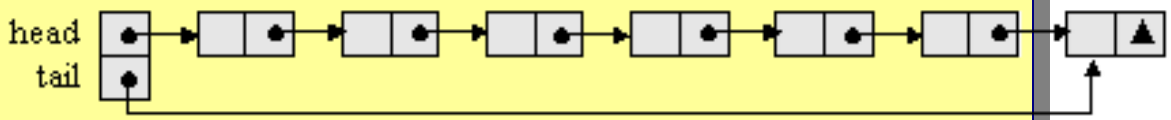


Brisanje elementa sa početka

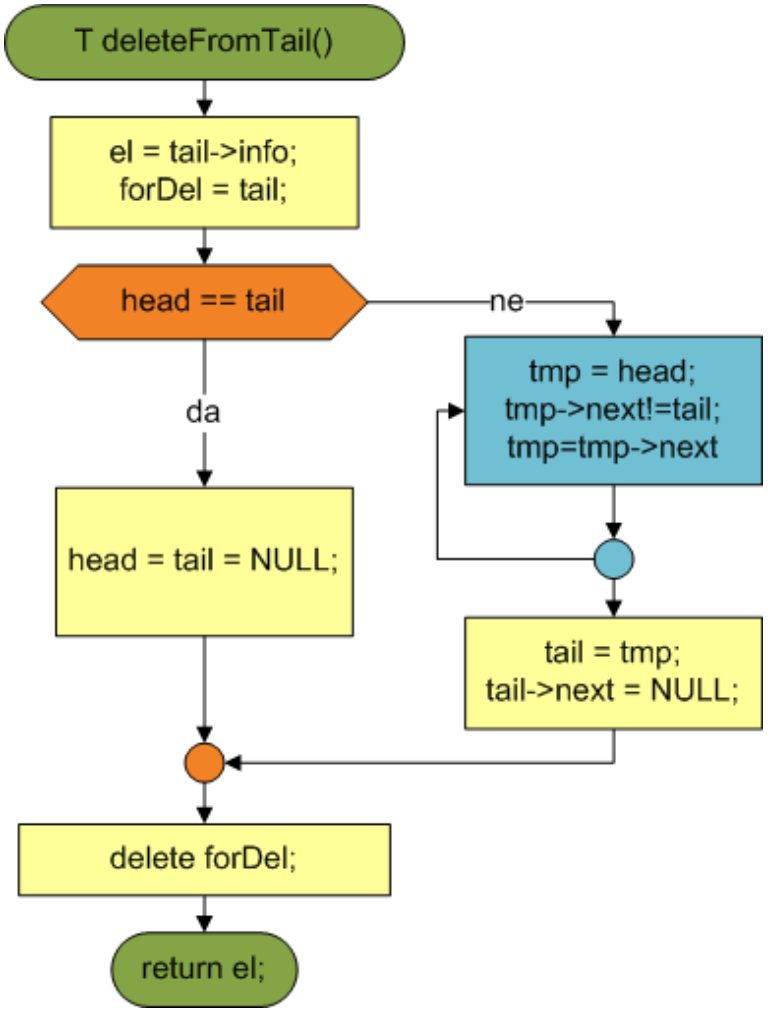


Brisanje elementa sa kraja

```
template <class T>
T SLList<T>::deleteFromTail()
{
    if( isEmpty() )
        throw new SBPException("Lista je prazna!");
     T el = tail->info;
     SLLNode<T>* forDel = tail;
     if (head == tail)
        head = tail = NULL;
     else
     {
        SLLNode<T>* tmp;
        for (tmp = head; tmp->next!=tail; tmp=tmp->next);
        tail = tmp;
        tail->next = NULL;
     }
     delete forDel;
     return el;
}
```

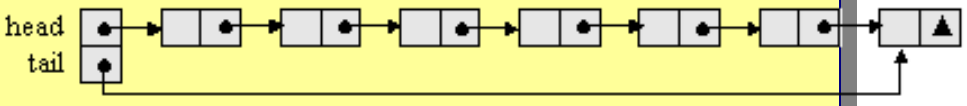


Brisanje elementa sa kraja



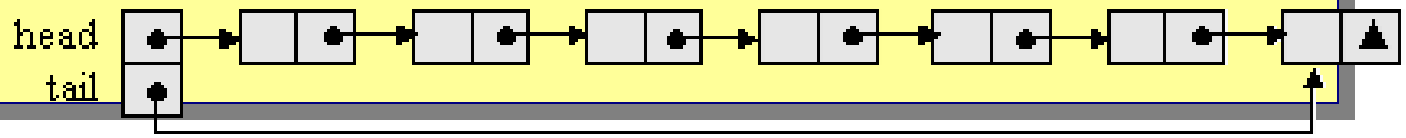
Brisanje elementa

```
template <class T>
void SLList<T>::deleteEl(T el)
{
        f( isEmpty() )
                throw new SBPException("Lista je prazna!");

        if (head == tail && head->info.isEqual(el))
        {
                delete head;
                head = tail = NULL;
        }

        else if (el == head->info)
        {
                SLLNode<T> *tmp = head;
                head = head->next;
                delete tmp;
        }
...
```

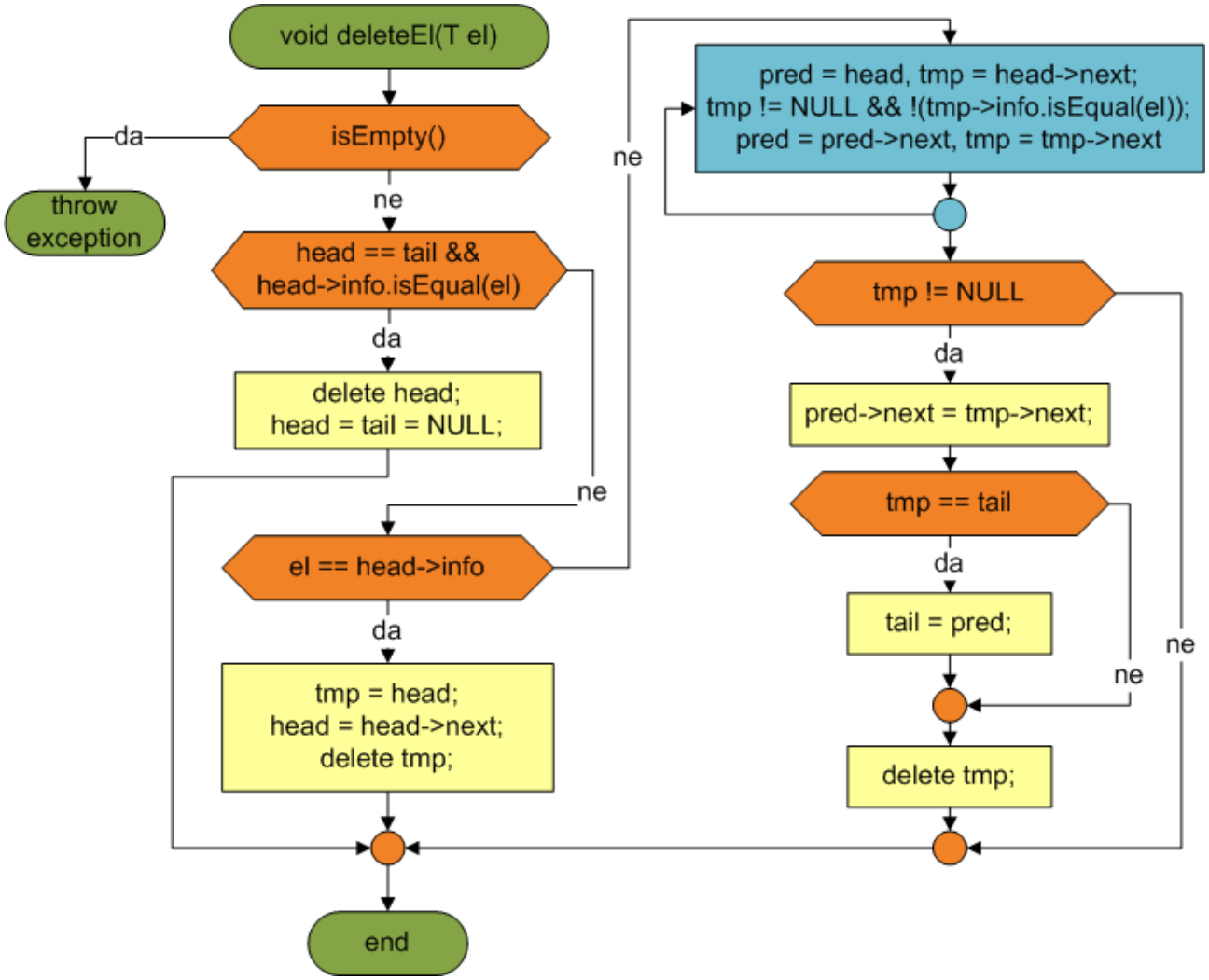


Brisanje elementa - nastavak

```
else {
        SLLNode<T> *pred, *tmp;

        for (pred = head, tmp = head->next;
              tmp != NULL && !(tmp->info.isEqual(el));
              pred = pred->next, tmp = tmp->next);

        if (tmp != NULL)
        {
                pred->next = tmp->next;
                if (tmp == tail)
                        tail = pred;
                delete tmp;
        }
    }
}
```



Brisanje elementa

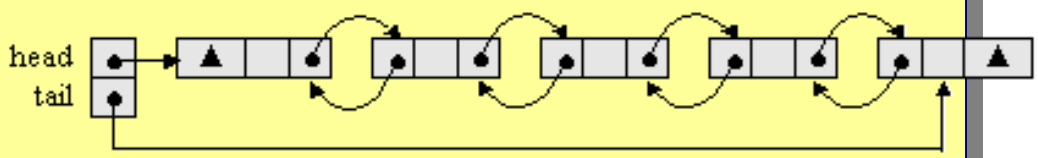


Čvor dvostruko ulančane list

```
template <class T>
class DLLNode
{
public:
     T info;
     DLLNode<T> *prev, *next;        // pokazivaci na susede

     DLLNode() { prev = NULL; next = NULL; };
     DLLNode(T el) { info = el; prev = NULL; next = NULL; };
     DLLNode(T el, DLLNode<T>* p, DLLNode<T>* n) {
          info = el; next = n; prev = p;
    }
    ~DLLNode() { };
    T print() {return info;};
    bool isEqual(T el) {return el == info};
} ;
```

Zaglavlje dvostruko ulančane liste (DLL)

```
template <class T>
class DLList
{
  protected:
        DLLNode<T> *head, *tail;      // pokazivaci pocetka i kraja liste
  public:
        DLList() { head = tail = NULL; };
        ~DLList();

      bool isEmpty() { return head == NULL; }
      void printAll();
      DLLNode<T>* findNodePtr(T el);
      DLLNode<T>* getHead() {return head;}
      DLLNode<T>* getNext(DLLNode<T>* ptr) ;
      T getHeadEl();
      T getNextEl(T el) ;
      bool isInList(T el);
      void deleteEl(T el);
      void addToHead(T el);
      void addToTail(T el);
      T deleteFromHead();
      T deleteFromTail();
};
```

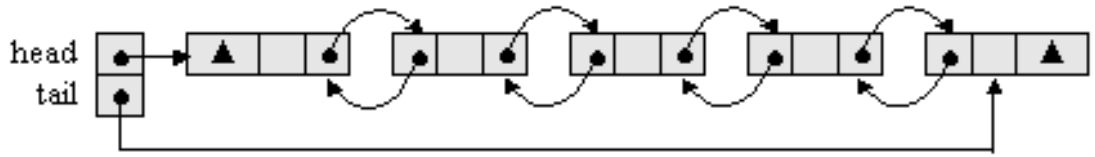


Destruktor i dodavanje el. u DLL

```
template <class T>
DLList<T>::~DLList()
{
        while(!isEmpty())  {
                T tmp = deleteFromHead();
        }
 }
template <class T>
void DLList<T>::addToHead(T el) {
        if (!isEmpty()) {
                head = new DLLNode<T>(el, NULL, head);
                head->next->prev = head;
        }
        else head = tail = new DLLNode<T>(el);
}
template <class T>
void DLList<T>::addToTail(T el) {
        if (!isEmpty()) {
                tail = new DLLNode<T>(el, tail, NULL);
                tail->prev->next = tail;
        }
        else head = tail = new DLLNode<T>(el);
}
```

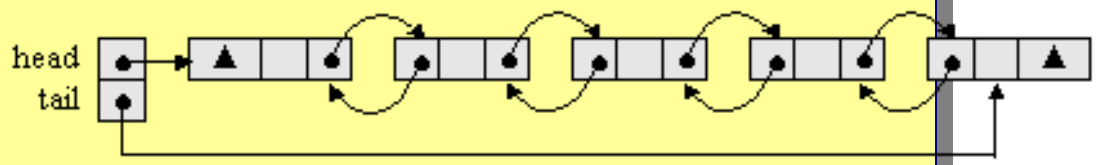
Brisanje elementa

```
template <class T>
void DLList<T>::deleteEl(T el)
{
        if (isEmpty()) return;
        if (head == tail && head->info.isEqual(el))
        {
                delete head;
                head = tail = NULL;
        }
        else if (el == head->info)
        {
                DLLNode<T> *tmp = head;
                head = head->next; head->prev = NULL;
                delete tmp;
        }
...
```

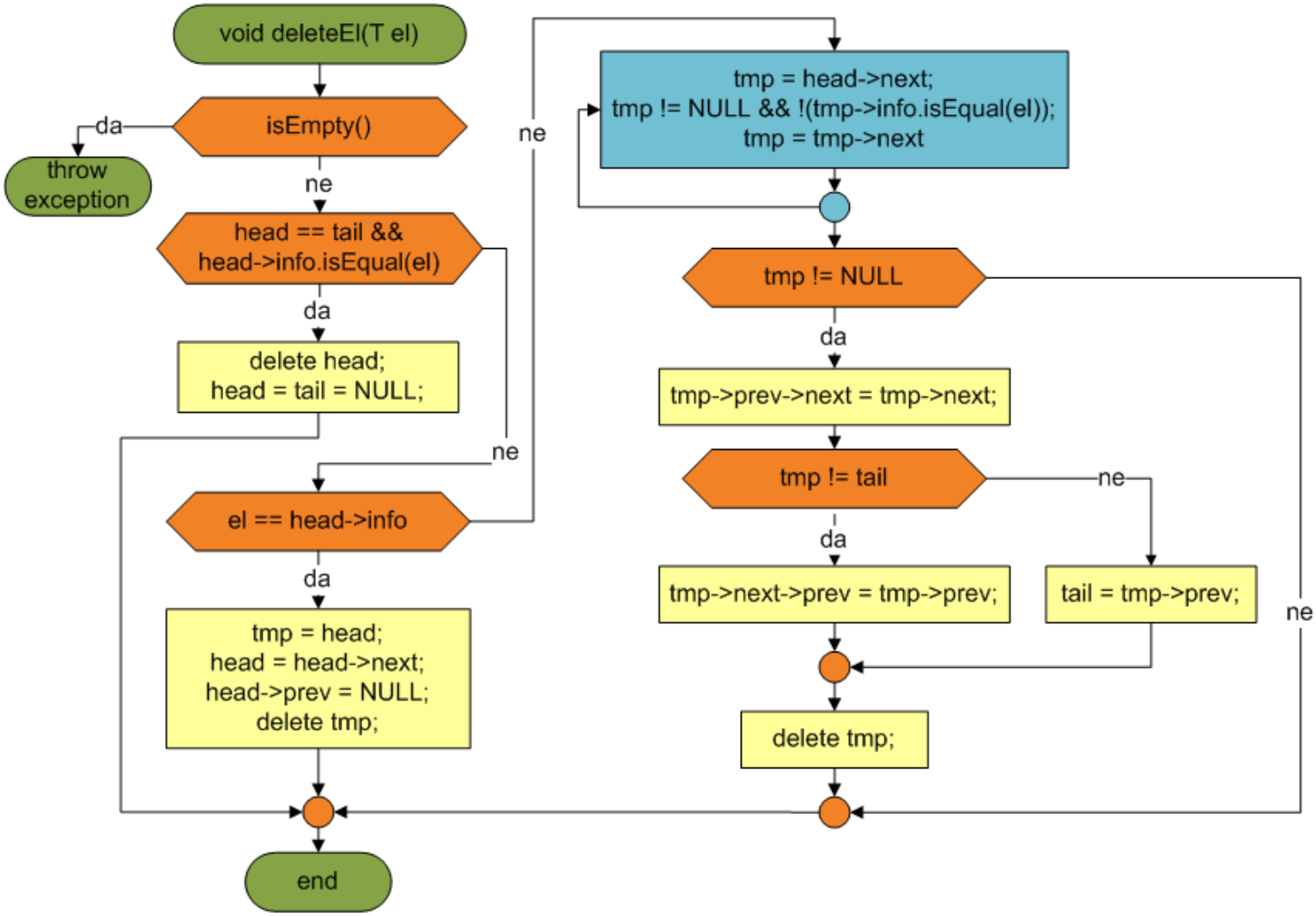


Brisanje elementa - nastavak

```
    else
    {
        DLLNode<T> *tmp;
        for (tmp = head->next;
            tmp != NULL && !(tmp->info.isEqual(el));
            tmp = tmp->next);

        if (tmp != NULL) {
                tmp->prev->next = tmp->next;
                if(tmp ->next )
                        tmp->next->prev = tmp->prev;
                if (tmp == tail)
                        tail = tmp->prev;
                delete tmp;
        }
    }
}
```

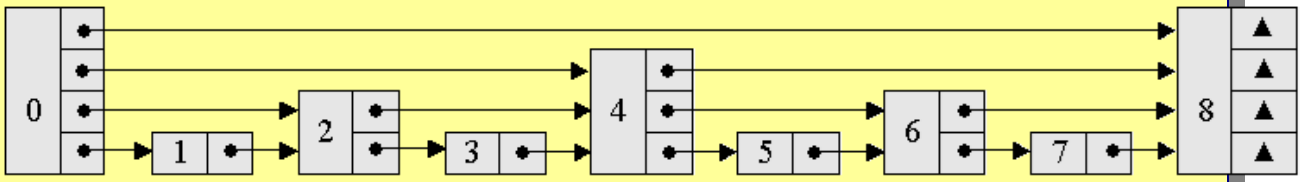


Brisanje elementa



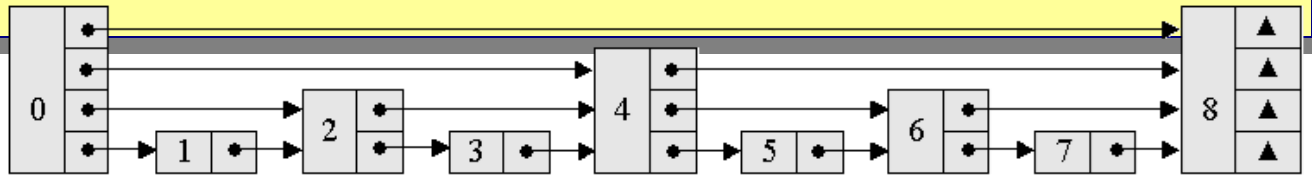
Čvor *skip* liste

```
template <class T>
class SkipListNode
{
public:
      T info;
      int level;
      SkipListNode<T>** link;

     SkipListNode() { level = 0; link = NULL; }

     SkipListNode(T i, int n){
           info = i;
           level = n;
           link = new SkipListNode<T>*[level];
           for (int j = 0; j < level; j++)  link[j] = NULL;
     }
           ~SkipListNode() { delete [] link; }
};
```



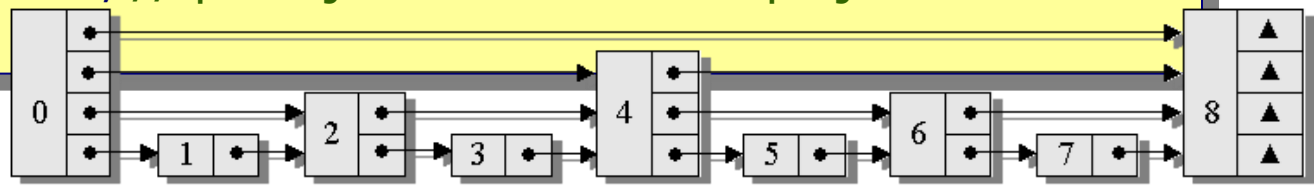
Zaglavlje *skip* liste

```
template <class T>
class SkipList
{
protected:
      int maxLevel;                 // maksimalni nivo cvora u listi
      SkipListNode<T>* root;        // pocetak liste
      long noElem;                  // broj cvorova u listi
public:
     SkipList(int i);
     ~SkipList();
     bool isEmpty() { return root == NULL; }
     bool isInList(T info);
     void skipListInsert (T info);
protected:
     void updateLinks(SkipListNode<T>* node);
};
```



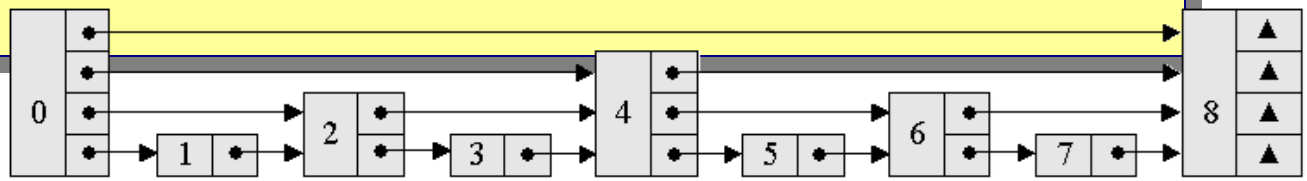
Traženje u *Skip*-listi

```
template <class T>
bool SkipList<T>::isInList(T info)
{
    SkipListNode<T> *curr, *prev;
    int lvl;
    curr = root;
    prev = NULL;
    for (lvl = maxLevel - 1; lvl >= 0; lvl--)
    {
        while (curr != NULL && curr->info < info)
        {
            prev = curr;    // ako je manji preci na sledeci cvor
            curr = curr->link[lvl];
        }
        if (curr != NULL && curr->info == info)
            return true; // postoji cvor sa tim info poljem
...
```

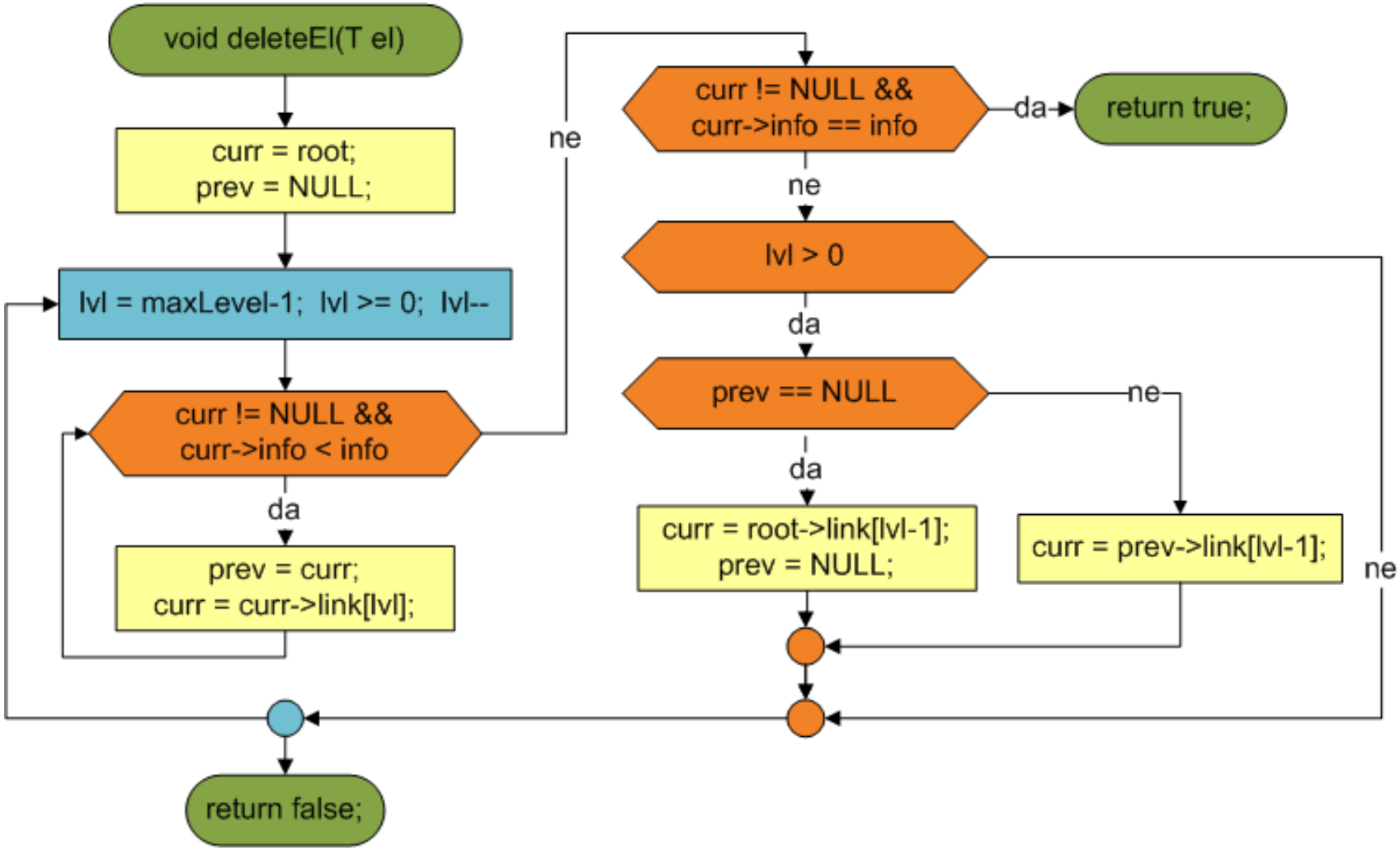


Traženje u *Skip*-listi

```
    if (lvl > 0)                    // preci na nizi nivo
    {
        if (prev == NULL)           // polazeci od root-a
        {
                curr = root->link[lvl-1];
        }
        else                        // ili pretnodnog cvora
        {
                curr = prev->link[lvl-1];
        }
    }
  }
  return false;
}
```



Traženje u *Skip*-listi



Zadaci za proveru razumevanja gradiva

- Projektovati klasu SLList1 za rad sa jednostruko ulančanom listom čiji čvorovi sadrže po dva stringa (prvi je “nepoznata” reč, a druga njeno objašnjenje) i implementirati sledeće funkcije:
  - ***explain*** *(nepoznata_rec)*, koja objašnjava zadatu reč tako što pronalazi njeno objašnjenje u listi i premešta pronađeni čvor na početak liste da bi ubrzala sledeće traženje te reči,
  - ***update*** *(druga_lista)*, koja u tekuću listu dodaje sve reči i odgovarajuća objašnjenja iz liste *druga_lista*, ako ne postoje u tekućoj listi
- Nacrtati algoritam za traženje u skip-listi.
- Odrediti funkciju složenosti traženja u skip listi f(n,l), gde je n-br.elemenata liste, a l-nivo liste (tj.max nivo čvora).
- Uporediti veličinu i brzinu pretrage skip-liste u odnosu na “običnu” jednostruko ulančanu listu.