ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Фундаментали податочни структури
- НИЗИ и динамички ЛИСТИ-

АЛГОРИТМИ И
ПОДАТОЧНИ СТРУКТУРИ

- предавања -

Содржина

- Низи
- Операции со низи
- Динамички поврзани листи

Низи

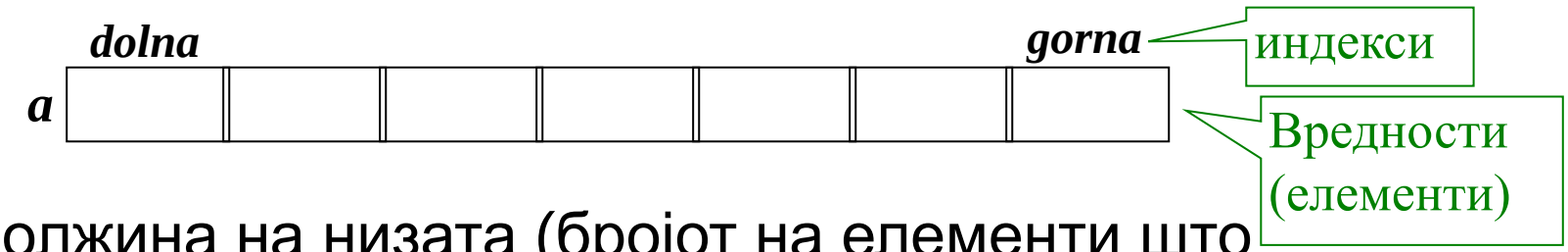
- последователно множество на мемориски локации
- множество на подредени парови

(индекс, вредност)

- при што за секое појавување на индекс, постои соодветна вредност асоцирана за тој индекс

Низи

- Индексите се единствени и фиксирани за секоја вредност.
- Индексите се во ранг од ***dolna*** граница до ***gorna*** граница:



- Должина на низата (бројот на елементи што содржат вредности) е вредност која се задава на почеток кога се формира низата
- Секоја вредност на низата може да се пристапи, со користење на нејзиниот индекс, во $O(1)$ време.

Низи

- Индексите се единствени и фиксирани за секоја вредност.
- Индексите се во ранг од ***dolna*** граница до ***gorna*** граница:

- Должина на низата (бројот на елементи што содржат вредности) е вредност која се задава на почеток кога се формира низата
- Секоја вредност на низата може да се пристапи, со користење на нејзиниот индекс, во $O(1)$ време.

ВАЖНО: Секогаш се специфицира максималниот индекс дозволен во низата

Низи

- Својства:
  - Низите чуваат повеќе елементи од ист тип под заедничко име
  - Елементите во низата може да се пристапуваат по случаен редослед, користејќи го индексот
  - Меморијата за низата е предефинирана, нема потреба од дополнителен меморсики простор
  - Низите се статички структури, нивната големина е финксна и не може да се промени после декларација

Низи

- Функции (операции) кои се изведуваат над низата се:
  - Изминување
  - Додавање
  - Бришење
  - Пребарување
  - Сортирање
- Анализа на перформансите за секоја од операциите

Изминување

- Изминување на низа е процес на посетување на секој од елементите по еднаш
- Изминувањето е потребно кога:
  - Се бројат елементите на низа
  - Се печатат елементите на низата
  - Се пресметува сума на елементите на низата
  - Итн ...

Вметнување

- Вметнувањето е процес на вклучување еден или повеќе нови елементи во низата
- Вметнување елемент во низа може да се изведе:
  - На почеток на низата
  - На било кој индекс во низата
  - На крај на низата

Вметнување

- Проблем: за дадена низа *a*[*left*…*right*], да се вметне вредност *val* во *a*[ins]. Ако е потребно, да се поместат вредностите надесно, за да се овозможи простор

(Претпоставуваме дека $left \leq ins \leq right$.)

Вметнување

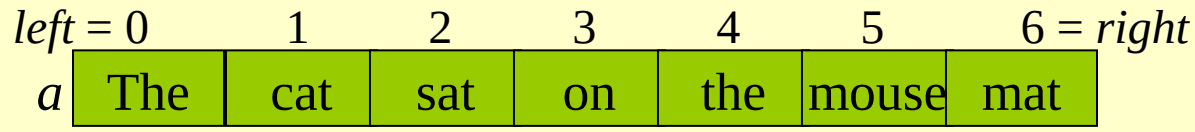
Вметнување

- Анимација:

Вметнување

- Анимација:

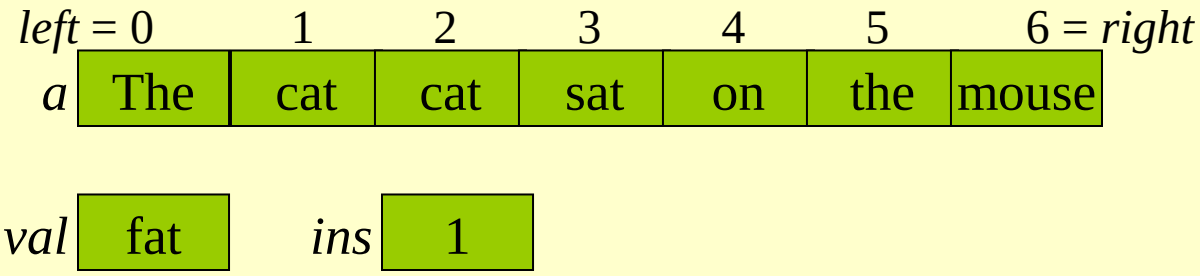
1. копирај го a[ins…right–1] во a[ins+1…right].
2. копирај го val во a[ins].
3. заврши.



Вметнување

- Анимација:

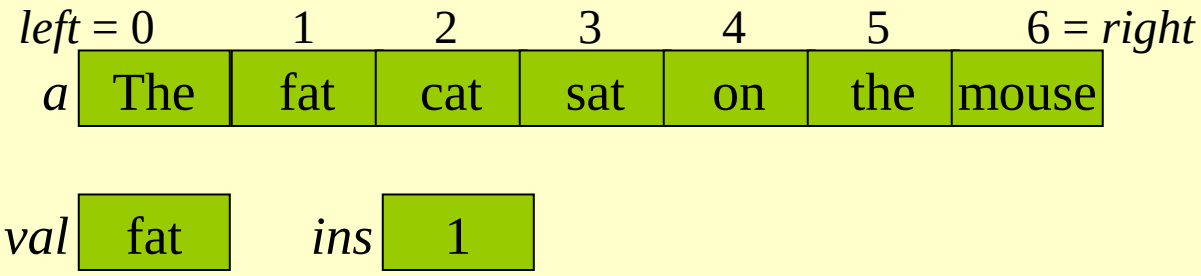
1. копирај го a[ins…right–1] во a[ins+1…right].
2. копирај го val во a[ins].
3. заврши.



Вметнување

- Анимација:

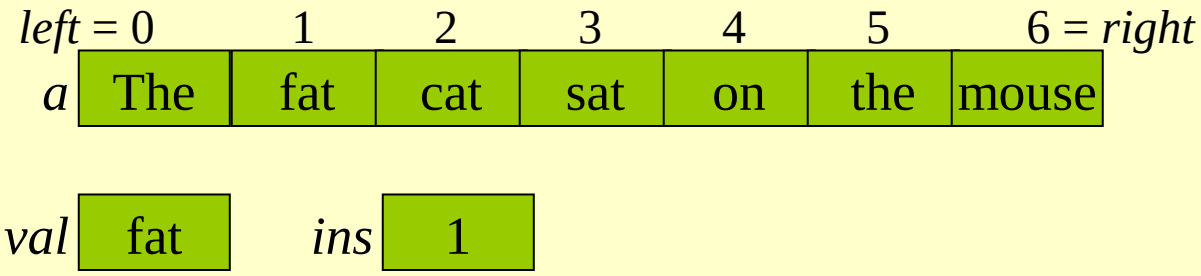
1. копирај го a[ins…right–1] во a[ins+1…right].
2. копирај го val во a[ins].
3. заврши.



Вметнување

- Анимација:

1. копирај го a[ins…right–1] во a[ins+1…right].
2. копирај го val во a[ins].
3. заврши.



Анализа за вметнувањето

- Анализа (се бројат копирањата (доделувањата)):

  Нека $n = right - left + 1$ е должината на низата.

  Чекор 2 изведува само едно копирање.

  Чекор 1 се спроведува од 0 до $n-1$ копирања, во просек $(n-1)/2$ копирања.

  Просечен број на копирања =

  $$(n - 1)/2 + 1 = n/2 + 1/2$$

  Сложеноста на алгоритамот е $\boldsymbol{O(n)}$.

Анализа за вметнувањето

- Колкава е сложеноста за најдобриот и за најлошиот случај?
- Најдобар случај – вметнување елемент на крај од низата
  - Нема потреба од извршување на чекор 1
  - Се извршува само чекор 2, односно само 1 копирање

  Сложеноста на алгоритамот е O(1).
- Најлош случај – вметнување елемент на почеток на низа
  - Чекор 1 се спроведува од 0 до n-1 и врши n копирања
  - Чекор 2 изведува само 1 копирање
  - Вкупно n+1 копирања

  Сложеноста на алгоритамот е O(n).

Бришење

- Бришење елемент во низа е процес на отстранување на конкретен елемент и реорганизација на низата
- Бришењето исто така може да се изведе на различни начини:
  - Бришење на почеток на низата
  - Бришење на крај на низата
  - Бришење на било кој индекс од низата

Бришење

- Проблем: за дадена низа *a*[*left*…*right*], да се избрише вредност *val* во *a*[del]. Ако е потребно, да се поместат вредностите налево, за да се пополни празнината

  (Претпоставуваме дека $left \leq del \leq right$.)

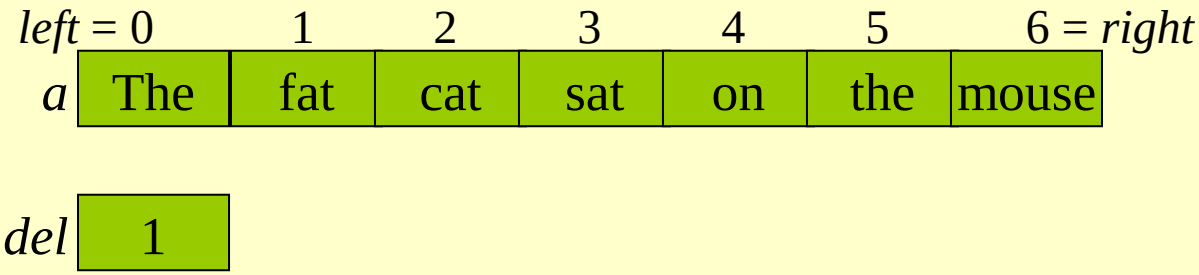
Бришење

- Анимација:

Бришење

- Анимација:

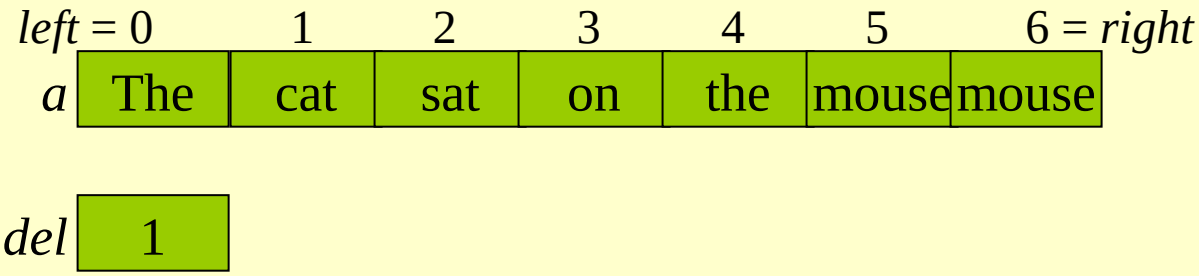
1. копирај a[del+1…right] во a[del…right–1].
2. ослободи го a[right]
3. заврши.



Бришење

- Анимација:

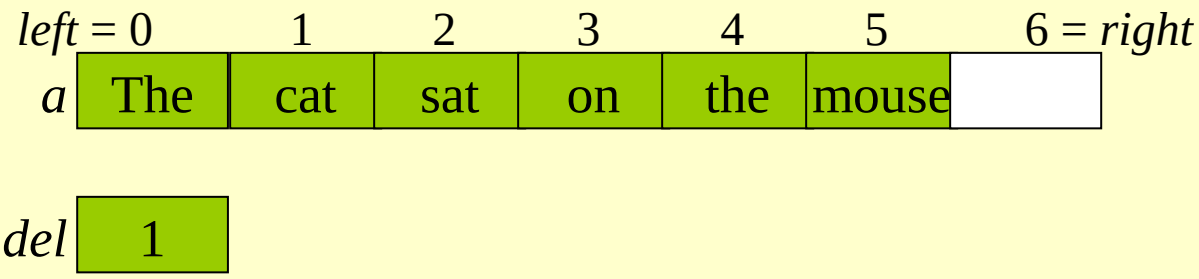
1. копирај a[del+1…right] во a[del…right–1].
2. промени го right=right-1
3. заврши.



Бришење

- Анимација:

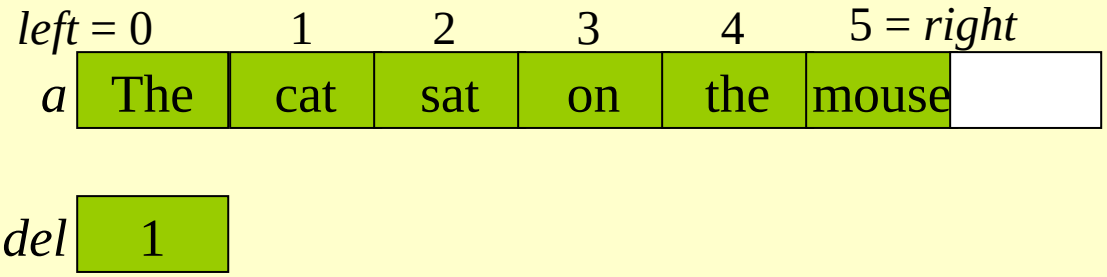
1. копирај a[del+1…right] во a[del…right–1].
2. промени го right = right-1
3. заврши.



Бришење

- Анимација:

1. копирај a[del+1…right] во a[del…right–1].
2. промени го right = right-1
3. заврши.



Анализа за бришењето

- Анализа (се бројат копирањата (доделувањата))

  Нека $n = right - left + 1$ е должина на низата.

  Чекор 1 се изведува помеѓу 0 и $n-1$ копирања.

  Просечен број копии =

  $$(n - 1)/2 = n/2 - 1/2$$

  Сложеноста на алгоритамот е $\boldsymbol{O(n)}$.

Анализа за бришењето

- Колкава е сложеноста за најдобриот и за најлошиот случај?
- Најдобар случај – бришење елемент на крај од низата
  - Нема потреба од извршување на чекор 1
  - Се извршува само чекор 2, една операција

  Сложеноста на алгоритамот е O(1).
- Најлош случај – бришење елемент на почеток
  - Чекор 1 се спроведува од 1 до n-1 и врши n-1 копирања
  - Чекор 2 изведува само 1 операција
  - Вкупно n–1+1 операции

  Сложеноста на алгоритамот е O(n).

Пребарување

- Пребарување елемент во низа е процес на наоѓање конкретен елемент во низата од вредности.
- Во овој процес се одлучува дали клучниот елемент по кој пребаруваме е присутен во низата или не.
- Во општ случај сложеноста на алгоритамот е O(n).
  - Специфичен случај е кога низата е сортирана и сложеноста е O(logn) (подоцна во материјалот)

Сортирање

- Сортирање на низа е процес во кој елементите се подредуваат според некој кориснички дефиниран редослед. На пример: нумерички, алфабетски и сл.
- Стандардно сортирачкиот процес се прави во растечки редослед.

N-димензионални низи

- Дефиниција за n-димензионална низа:

  Ако имаме *n+1* подредени парови на вредности, каде првите *n* елементи го сочинуваат индексот, а последниот елемент е вредноста асоцирана на тој индекс

  $$(idx_1, idx_2, \ldots, idx_n, value)$$

- Кога **n=2** тогаш имаме **матрица**

Матрици

$$\begin{bmatrix} -27 & 3 & 4 \\ 6 & 82 & -0.3 \\ 109 & -64 & 4 \\ 12 & 8 & 9 \\ 3.4 & 36 & 27 \end{bmatrix}$$

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

Матрици

$$\begin{bmatrix} -27 & 3 & 4 \\ 6 & 82 & -0.3 \\ 109 & -64 & 4 \\ 12 & 8 & 9 \\ 3.4 & 36 & 27 \end{bmatrix}$$

колона

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

Матрици

$$\begin{bmatrix} -27 & 3 & 4 \\ 6 & 82 & -0.3 \\ 109 & -64 & 4 \\ 12 & 8 & 9 \\ 3.4 & 36 & 27 \end{bmatrix}$$

колона

редица

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

Матрици

- Димензија на матрицата (mxn)
  - m редици
  - n колони
- Број на елементи m*n
- Ако m=n тогаш матрицата е квадратна
- Пристап до елемент во матрицата A[i][j]

Ретки (sparse) матрици

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

- Матрици кои имаат многу елементи со вредност 0 (нула)
- **Проблем: Трошат многу меморија!**

Ретки матрици

- Репрезентација:

$$(i,\ j,\ value)$$

  - Првиот елемент е димензијата на матрицата и бројот на ненултите елементи
  - Позиција и вредност на ненултите елементи

Ретки матрици

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

Ретки матрици

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

```
                1,   2,    3
               -----------
A(0,           6,   6,     8)
  (1,          1,   1,    15)
  (2,          1,   4,    22)
  (3,          1,   6,   -15)
  (4,          2,   2,    11)
  (5,          2,   3,     3)
  (6,          3,   4,    -6)
  (7,          5,   1,    91)
  (8,          6,   3,    28)
```

Ретки матрици

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

```
          1,   2,    3
        ------------
A(0,    6,   6,     8)
  (1,   1,   1,    15)
  (2,   1,   4,    22)
  (3,   1,   6,   -15)
  (4,   2,   2,    11)
  (5,   2,   3,     3)
  (6,   3,   4,    -6)
  (7,   5,   1,    91)
  (8,   6,   3,    28)
```

За дома: Да се предложат соодветни репрезентации на долно/горно триаголни матрици!

Потреба од динамички структури

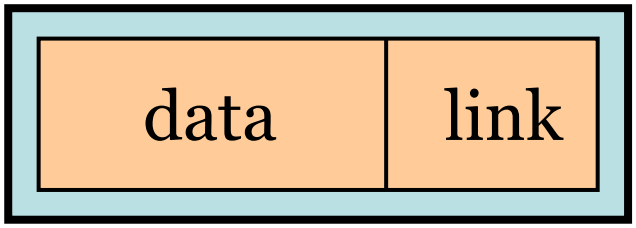
- Лошата страна на низите е тоа што не може да се промени нивната големина во време на извршување.
  - Програмата ќе ни падне кога сакаме да го додадеме n+1 виот елемент, а сме резервирале простор за n елементи
- Што е со динамичките низи?
  - автоматски растат во должина кога се обидуваме да направиме внес на нов елемент
  - При иницијализација се започнува со должина 1, после се дуплира должината во 2, односно од n во 2n секогаш кога ќе снемаме простор.
- Примери: vector во C++, ArrayList во Јава

Потреба од динамички структури

- Во општ случај велиме дека секој од n-те елементи ќе се помести во просек 2 пати, па според тоа сложеноста на работа со динамички низи е исто така O(n)
  - Исто како однапред да сме обезбедиле доволно меморија за сместување на сите елементи

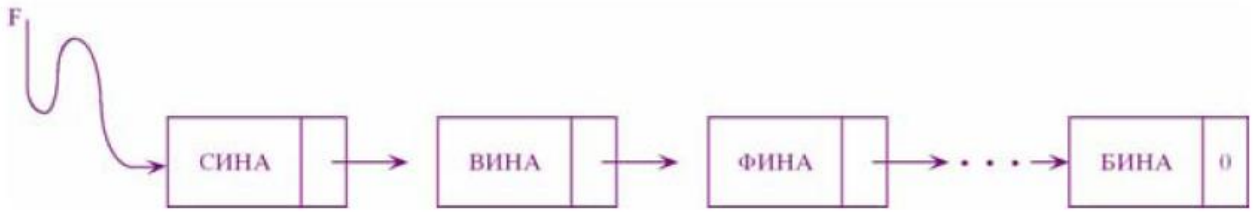
Едностранo поврзани листи

- Едностранo поврзана листа – Single Linked List SLL
- Подреденоста на елементите се запазува, меѓутоа нема потреба да има и мемориски континуитет
- Репрезентација: множество на подредени елементи, каде секој елемент е опишан со вредност на јазелот(теме) (*data*) и покажувач кон следниот јазел (*link*)



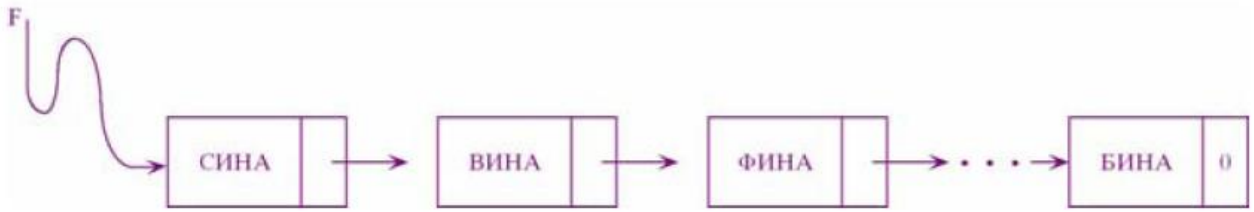
Едностранo поврзани листи

	Податок	Линк
1	ДИНА	4
2		
3	СИНА	11
4	МИНА	14
5		
6		
7	ШИНА	8
8	БИНА	0
9	ФИНА	1
10		
11	ВИНА	9
	⋮	⋮

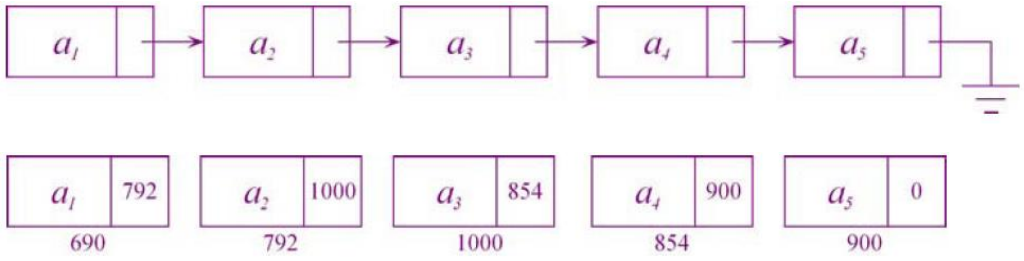


Едностранo поврзани листи

	Податок	Линк
1	ДИНА	4
2		
3	СИНА	11
4	МИНА	14
5		
6		
7	ШИНА	8
8	БИНА	0
9	ФИНА	1
10		
11	ВИНА	9
	⋮	⋮

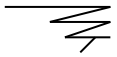


Друга репрезентација:



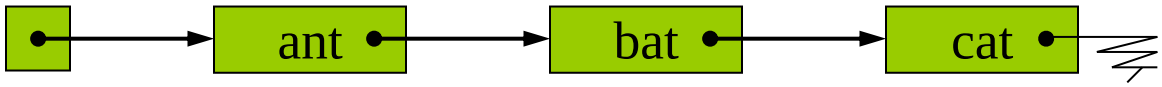
Едностранo поврзани листи

- Секој јазол (освен последниот) има **наследник,** и секој јазол (освен првиот) има **претходник**.
- **Должина** на листата е бројот на јазлите.
- Празна листа не содржи ниту еден јазол.
- Во поврзаните листи може:
  - Да се достапува/чита/брише до секој елемент (јазол).
  - Да се менуваат врските, а со тоа да се менува структурата на листата
    - Ова не е можно кај низите



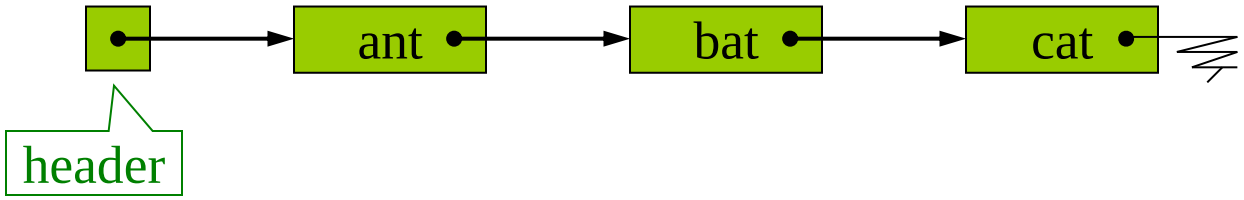
Едностранo поврзани листи

- Секој јазол (освен последниот) има **наследник,** и секој јазол (освен првиот) има **претходник**.
- **Должина** на листата е бројот на јазлите.
- Празна листа не содржи ниту еден јазол.
- Во поврзаните листи може:
  - Да се достапува/чита/брише до секој елемент (јазол).
  - Да се менуваат врските, а со тоа да се менува структурата на листата
    - Ова не е можно кај низите



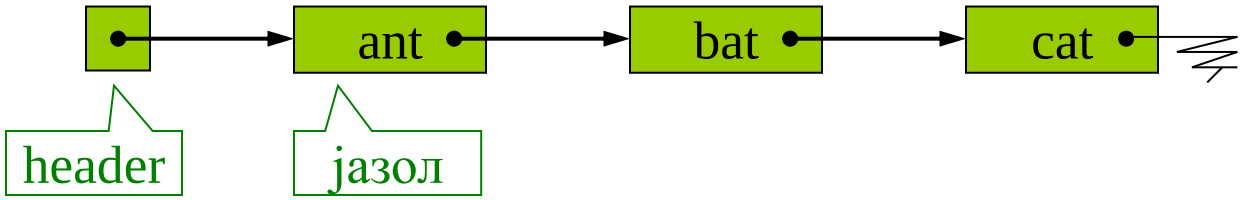
Едностранo поврзани листи

- Секој јазол (освен последниот) има **наследник,** и секој јазол (освен првиот) има **претходник**.
- **Должина** на листата е бројот на јазлите.
- Празна листа не содржи ниту еден јазол.
- Во поврзаните листи може:
  - Да се достапува/чита/брише до секој елемент (јазол).
  - Да се менуваат врските, а со тоа да се менува структурата на листата
    - Ова не е можно кај низите



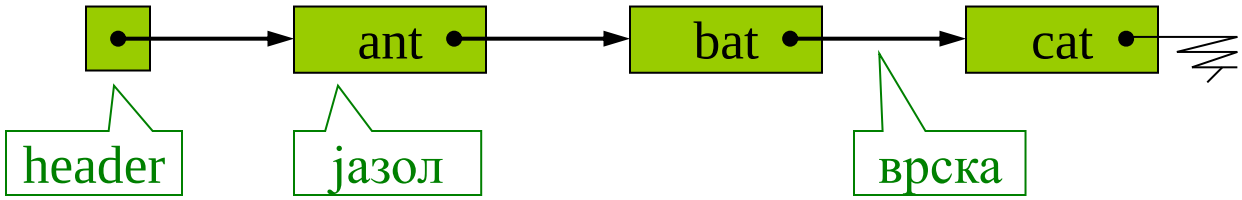
Едностранo поврзани листи

- Секој јазол (освен последниот) има **наследник,** и секој јазол (освен првиот) има **претходник**.
- **Должина** на листата е бројот на јазлите.
- Празна листа не содржи ниту еден јазол.
- Во поврзаните листи може:
  - Да се достапува/чита/брише до секој елемент (јазол).
  - Да се менуваат врските, а со тоа да се менува структурата на листата
    - Ова не е можно кај низите



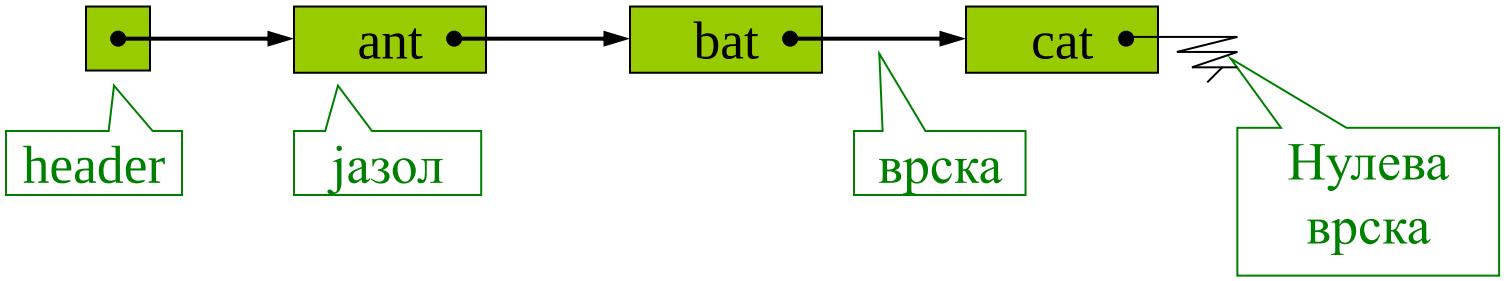
Едностранo поврзани листи

- Секој јазол (освен последниот) има **наследник,** и секој јазол (освен првиот) има **претходник**.
- **Должина** на листата е бројот на јазлите.
- Празна листа не содржи ниту еден јазол.
- Во поврзаните листи може:
  - Да се достапува/чита/брише до секој елемент (јазол).
  - Да се менуваат врските, а со тоа да се менува структурата на листата
    - Ова не е можно кај низите



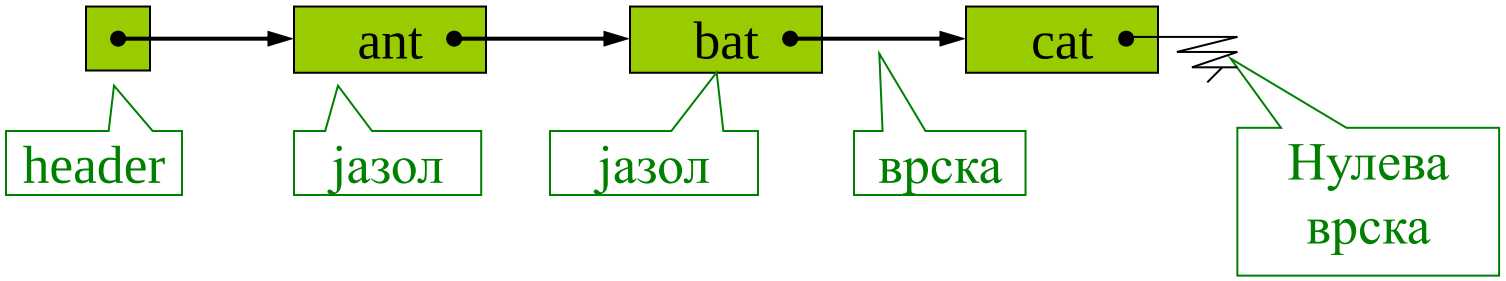
Едностранo поврзани листи

- Секој јазол (освен последниот) има **наследник,** и секој јазол (освен првиот) има **претходник**.
- **Должина** на листата е бројот на јазлите.
- Празна листа не содржи ниту еден јазол.
- Во поврзаните листи може:
  - Да се достапува/чита/брише до секој елемент (јазол).
  - Да се менуваат врските, а со тоа да се менува структурата на листата
    - Ова не е можно кај низите



Едностранo поврзани листи

- Секој јазол (освен последниот) има **наследник,** и секој јазол (освен првиот) има **претходник**.
- **Должина** на листата е бројот на јазлите.
- Празна листа не содржи ниту еден јазол.
- Во поврзаните листи може:
  - Да се достапува/чита/брише до секој елемент (јазол).
  - Да се менуваат врските, а со тоа да се менува структурата на листата
    - Ова не е можно кај низите



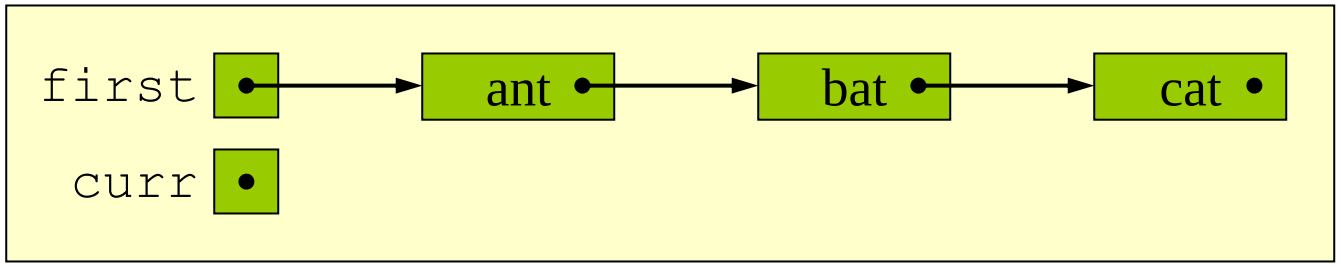
Едностранo поврзани листи

- Операции на SLL:
  - креирање празна листа
  - изминување на листата
  - вметнување елемент во листата
  - бришење елемент од листата
  - пронаоѓање елемент во листата
  - бришење на листата
  - и.т.н.

Едностранo поврзани листи

- Изминување на едностранa листа:

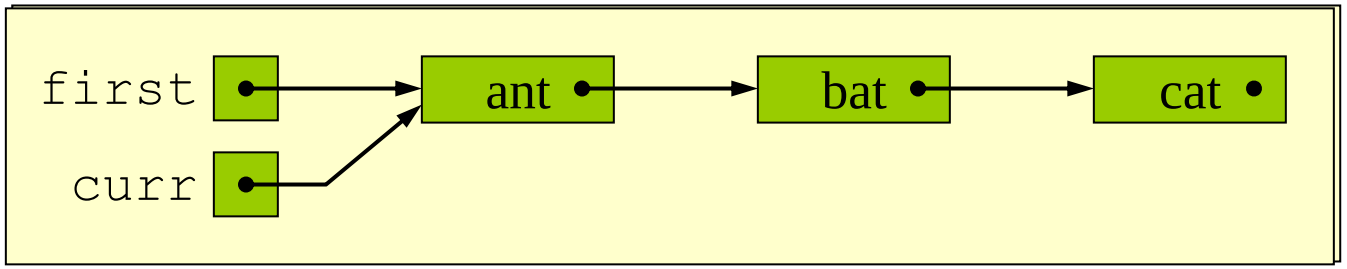
```
public void printFirstToLast () {
// Print all elements in this SLL, in first-to-last order.
   for (SLLNode curr = this.first;
        curr != null; curr = curr.succ)
      System.out.println(curr.element);
}
```



Едностранo поврзани листи

- Изминување на едностранa листа:

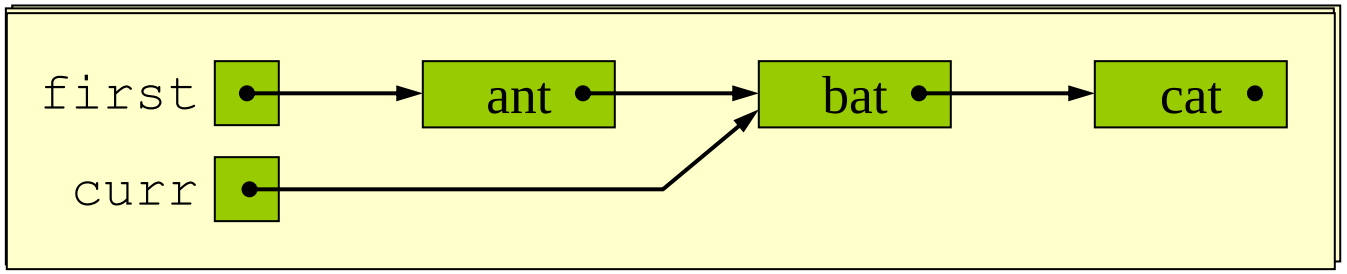
```
public void printFirstToLast () {
// Print all elements in this SLL, in first-to-last order.
   for (SLLNode curr = this.first;
        curr != null; curr = curr.succ)
      System.out.println(curr.element);
}
```



Едностранo поврзани листи

- Изминување на едностранa листа:

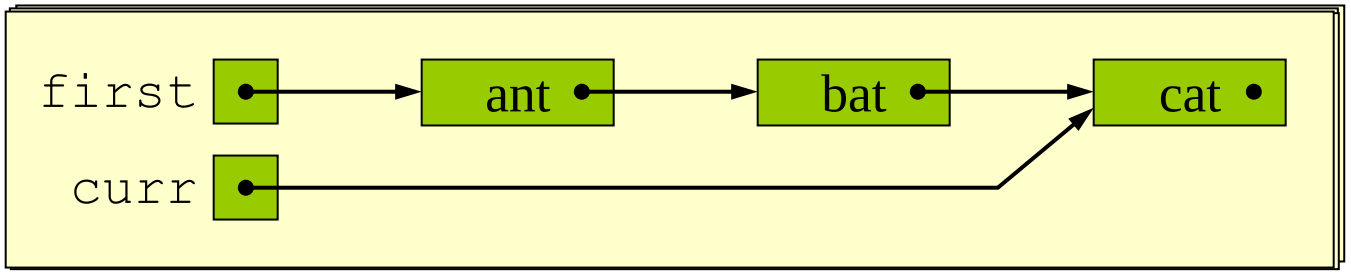
```
public void printFirstToLast () {
// Print all elements in this SLL, in first-to-last order.
   for (SLLNode curr = this.first;
        curr != null; curr = curr.succ)
      System.out.println(curr.element);
}
```



Едностранo поврзани листи

- Изминување на едностранa листа:

```
public void printFirstToLast () {
// Print all elements in this SLL, in first-to-last order.
   for (SLLNode curr = this.first;
        curr != null; curr = curr.succ)
      System.out.println(curr.element);
}
```



Едностранo поврзани листи

- Вметнување на јазел во SLL вклучува 4 случаи:
  1. Вметнување во празна листа
  2. Вметнување на почеток во непразна SLL
  3. Вметнување на крај во непразна SLL
  4. Вметнување измеѓу два јазли во непразна SLL

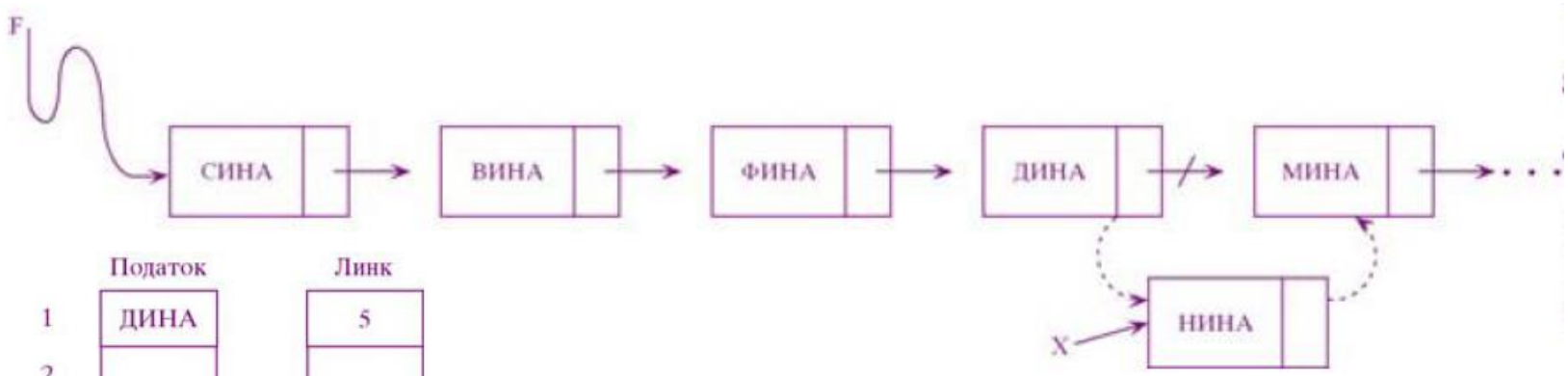
- При вметнување јазел, треба да се внимава на линковите на неговите претходници/следбеници на тој јазел

Едностранo поврзани листи

- Алгоритам за вметнување јазол во SLL:
  1. се избира јазел кој моментално не се користи во достапниот мемориски простор
  2. се доделува соодветната вредност на инфо полето од јазелот
  3. вредноста на линк полето се пополнува со адресата на јазелот кој треба да биде следбеник на новиот јазел
  4. вредноста на линк полето на јазелот кој ќе биде претходник на новиот јазел треба да се промени со адресата на новиот јазел

Едностранo поврзани листи

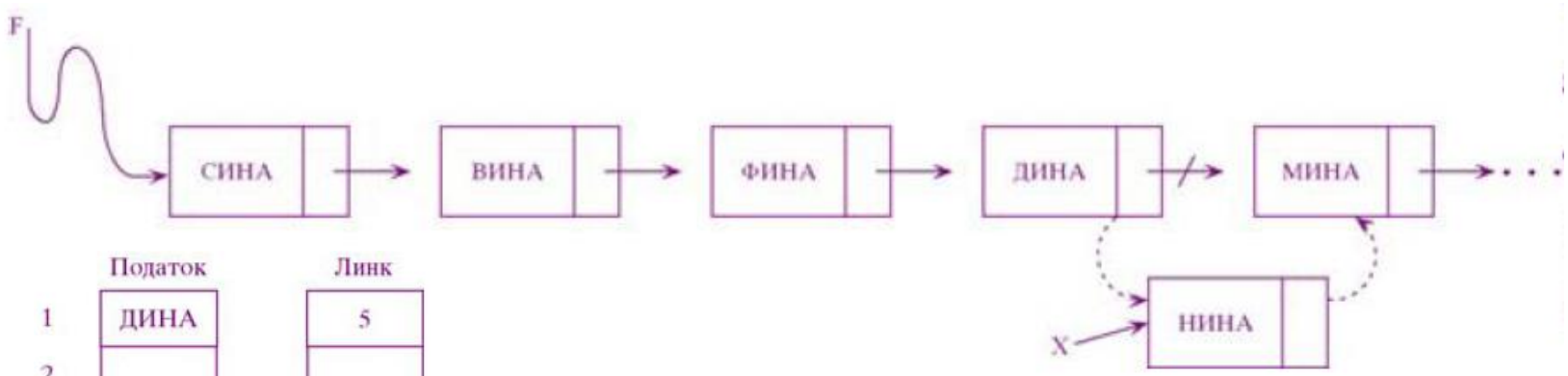
- Вметнување на јазел во листата:



	Податок	Линк
1	ДИНА	5
2		
3	СИНА	11
4	МИНА	14
5	НИНА	4
6		
7	ШИНА	8
8	БИНА	0
9	ФИНА	1
10		
11	ВИНА	9

Едностранo поврзани листи

- Вметнување на јазел во листата:



	Податок	Линк
1	ДИНА	5
2		
3	СИНА	11
4	МИНА	14
5	НИНА	4
6		
7	ШИНА	8
8	БИНА	0
9	ФИНА	1
10		
11	ВИНА	9

Предности на овој пристап:

- нема поместување на елементите за одржување на подреденоста
- нема потреба од декларација на максимално дозволената должина на низата

Користење на поврзаните листи

- Анимација (вметнување пред првиот јазол):

  Да се вметне *elem* во дадената SLL на почеток:

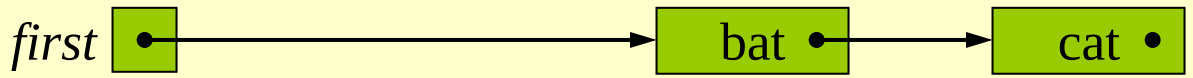
  1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
  2. Постави следбеник на јазолот *ins* да биде *first*.
  3. Постави го *first* да биде *ins*.
  4. Заврши.

Користење на поврзаните листи

- Анимација (вметнување пред првиот јазол):

Да се вметне *elem* во дадената SLL на почеток:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. Постави следбеник на јазолот *ins* да биде *first*.
3. Постави го *first* да биде *ins*.
4. Заврши.

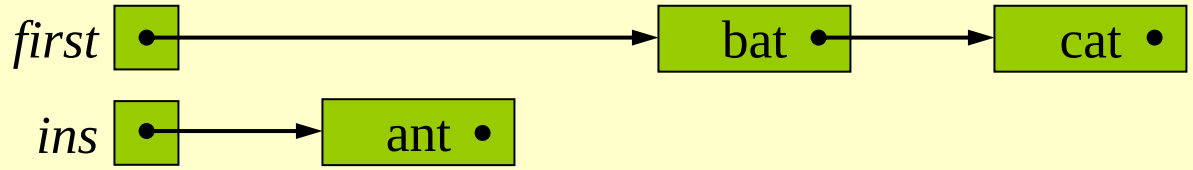


Користење на поврзаните листи

- Анимација (вметнување пред првиот јазол):

Да се вметне *elem* во дадената SLL на почеток:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. Постави следбеник на јазолот *ins* да биде *first*.
3. Постави го *first* да биде *ins*.
4. Заврши.

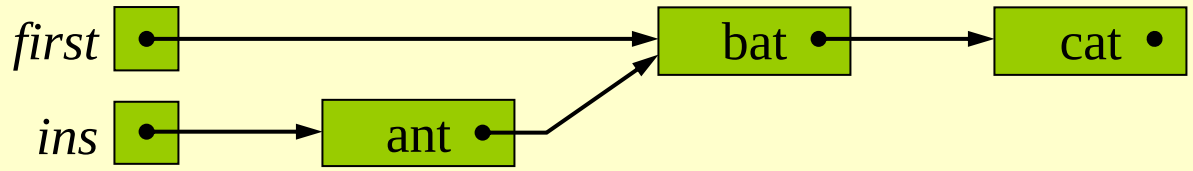


Користење на поврзаните листи

- Анимација (вметнување пред првиот јазол):

Да се вметне *elem* во дадената SLL на почеток:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. Постави следбеник на јазолот *ins* да биде *first*.
3. Постави го *first* да биде *ins*.
4. Заврши.

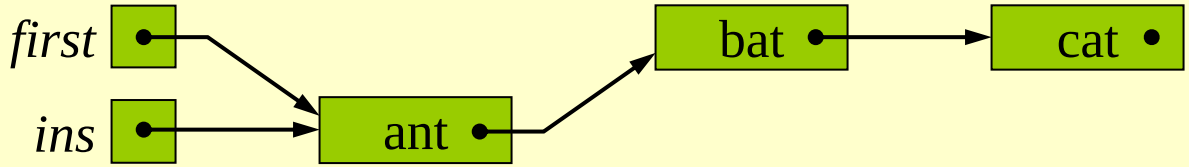


Користење на поврзаните листи

- Анимација (вметнување пред првиот јазол):

Да се вметне *elem* во дадената SLL на почеток:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. Постави следбеник на јазолот *ins* да биде *first*.
3. Постави го *first* да биде *ins*.
4. Заврши.

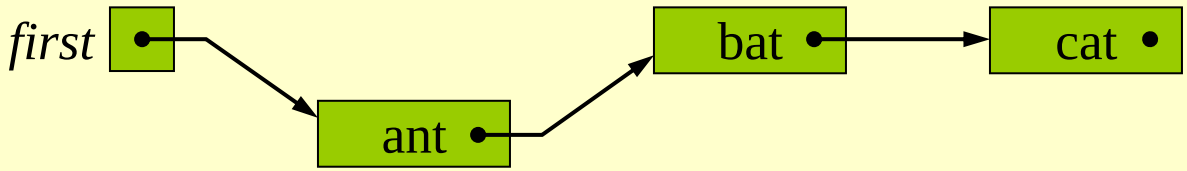


Користење на поврзаните листи

- Анимација (вметнување пред првиот јазол):

Да се вметне *elem* во дадената SLL на почеток:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. Постави следбеник на јазолот *ins* да биде *first*.
3. Постави го *first* да биде *ins*.
4. Заврши.



Користење на поврзаните листи

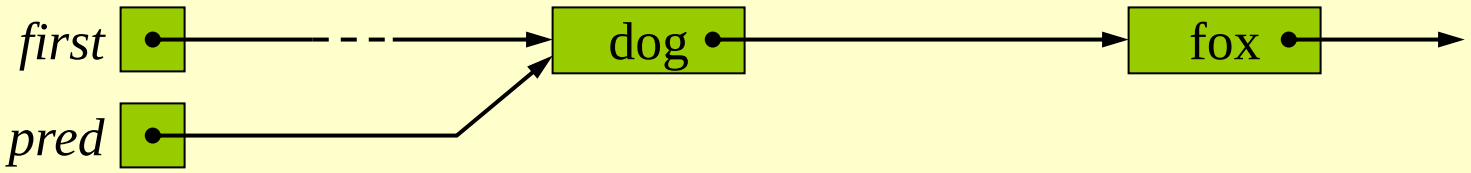
- Анимација (вметнување после средишен јазол):

Користење на поврзаните листи

- Анимација (вметнување после средишен јазол):

Да се вметне *elem* на дадена позиција во SLL, после јазолот *pred*:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. постави следбеник на јазолот *ins* да биде следбеникот на *pred*
3 постави го следбеникот на *pred* да биде *ins*
4. Заврши.

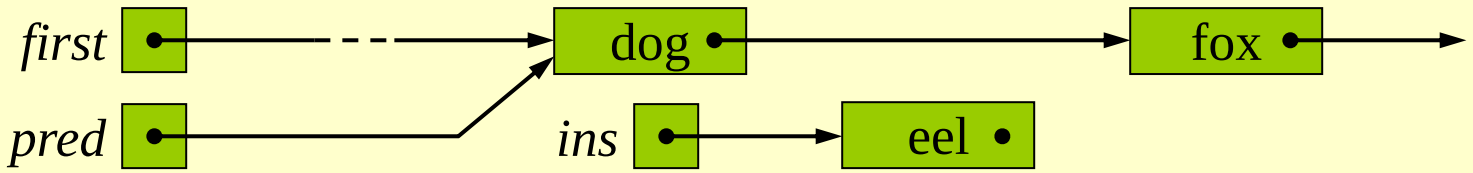


Користење на поврзаните листи

- Анимација (вметнување после средишен јазол):

Да се вметне *elem* на дадена позиција во SLL, после јазолот *pred*:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. постави следбеник на јазолот *ins* да биде следбеникот на *pred*
3 постави го следбеникот на *pred* да биде *ins*
4. Заврши.

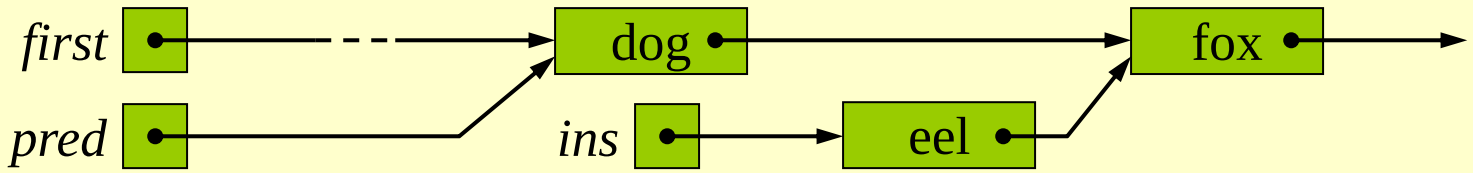


Користење на поврзаните листи

- Анимација (вметнување после средишен јазол):

Да се вметне *elem* на дадена позиција во SLL, после јазолот *pred*:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. постави следбеник на јазолот *ins* да биде следбеникот на *pred*
3. постави го следбеникот на *pred* да биде *ins*
4. Заврши.

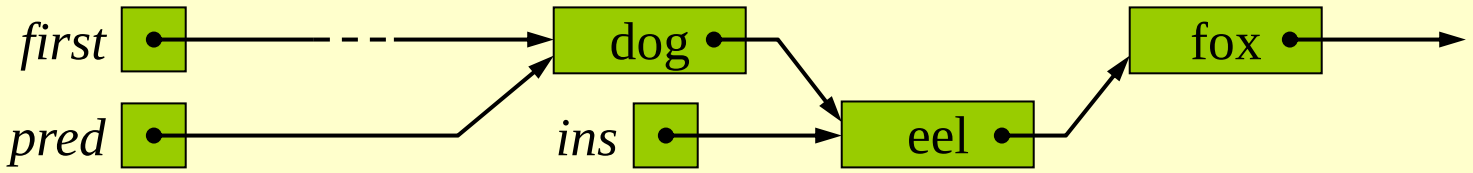


Користење на поврзаните листи

- Анимација (вметнување после средишен јазол):

Да се вметне *elem* на дадена позиција во SLL, после јазолот *pred*:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. постави следбеник на јазолот *ins* да биде следбеникот на *pred*
3. постави го следбеникот на *pred* да биде *ins*
4. Заврши.

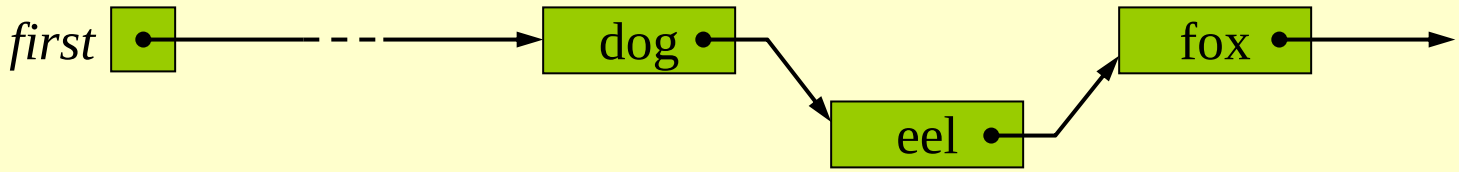


Користење на поврзаните листи

- Анимација (вметнување после средишен јазол):

Да се вметне *elem* на дадена позиција во SLL, после јазолот *pred*:

1. Направи го јазолот *ins* нов јазол со инфо *elem* и следбеник null.
2. постави следбеник на јазолот *ins* да биде следбеникот на *pred*
3 постави го следбеникот на *pred* да биде *ins*
4. Заврши.



Едностранo поврзани листи

- Бришење на јазел од листата:
  - се избира елемент кој претходи на елементот кој сакаме да го избришеме
  - вредноста на линк полето на претходникот треба да се промени со вредноста на адресата сместена во линк полето на јазелот кој се брише

Едностранo поврзани листи

- Бришење на јазел од листата:

Едностранo поврзани листи

- Бришењето јазел во SLL вклучува 4 случаи:
  1. Бришење од листа со единствен јазол
  2. Бришење на првиот јазол (но не и последен)
  3. Бришење на последниот јазол (но не и прв)
  4. Бришење средишен јазол

- При бришење јазел, треба да се внимава на линковите на неговите претходници/следбеници на тој јазел

Користење на поврзаните листи

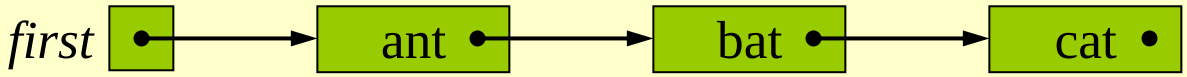
- Анимација (бришење на првиот јазол):

Користење на поврзаните листи

- Анимација (бришење на првиот јазол):

Да се избрише првиот јазол во дадена SLL:

1. Нека *succ* е следбеникот на првиот јазол
2. Го поставуваме следбеникот на јазолот *first* на *succ*
3. Заврши.

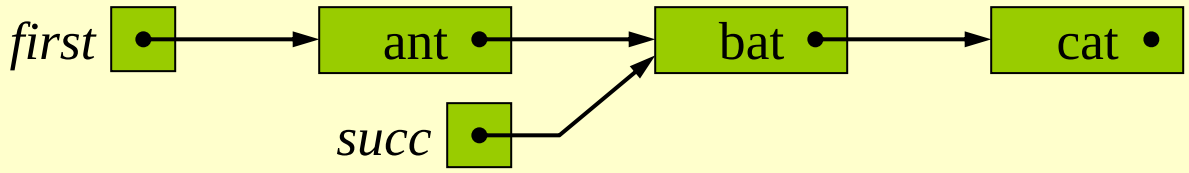


Користење на поврзаните листи

- Анимација (бришење на првиот јазол):

Да се избрише првиот јазол во дадена SLL:

1. Нека *succ* е следбеникот на првиот јазол
2. Го поставуваме следбеникот на јазолот *first* на *succ*
3. Заврши.

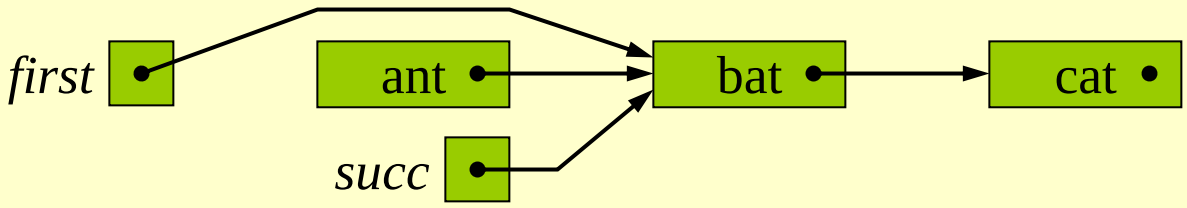


Користење на поврзаните листи

- Анимација (бришење на првиот јазол):

Да се избрише првиот јазол во дадена SLL:

1. Нека *succ* е следбеникот на првиот јазол
2. Го поставуваме следбеникот на јазолот *first* на *succ*
3. Заврши.

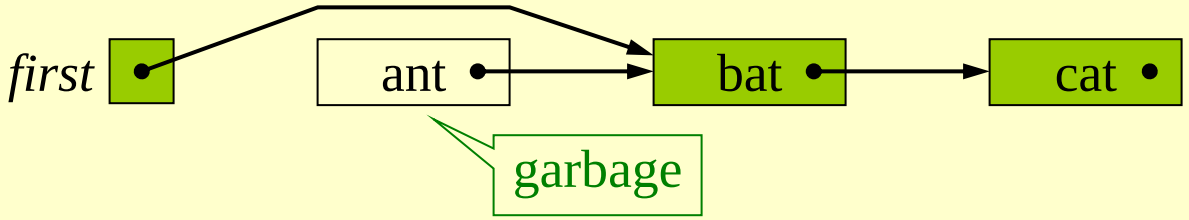


Користење на поврзаните листи

- Анимација (бришење на првиот јазол):

Да се избрише првиот јазол во дадена SLL:

1. Нека *succ* е следбеникот на првиот јазол
2. Го поставуваме следбеникот на јазолот *first* на *succ*
3. Заврши.



Користење на поврзаните листи

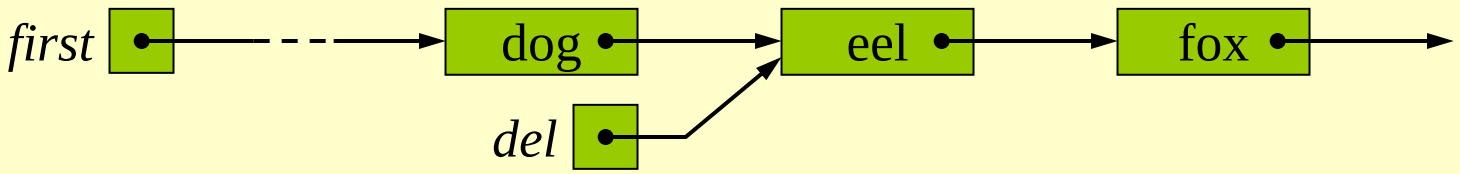
- Анимација (бришење на средишен или последен јазол)

Користење на поврзаните листи

- Анимација (бришење на средишен или последен јазол)

Да се избрише даден јазол *del* од SLL:

1. Нека *succ* е следбеник на јазолот *del*.
2. Во случај кога јазолот *del* не е *first*
   - 2.1. Нека *pred* е јазол претходник на *del*
   - 2.2. Постави го следбеникот на јазолот *pred* да покажува на *succ*
3. Заврши

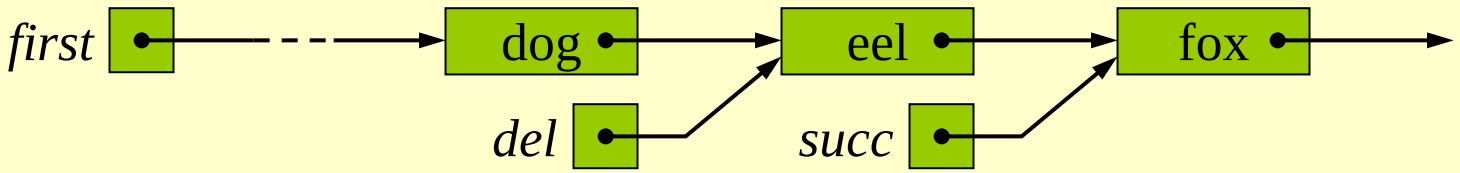


Користење на поврзаните листи

- Анимација (бришење на средишен или последен јазол)

Да се избрише даден јазол *del* од SLL:

1. Нека *succ* е следбеник на јазолот *del*.
2. Во случај кога јазолот *del* не е *first*
   - 2.1. Нека *pred* е јазол претходник на *del*
   - 2.2. Постави го следбеникот на јазолот *pred* да покажува на *succ*
3. Заврши

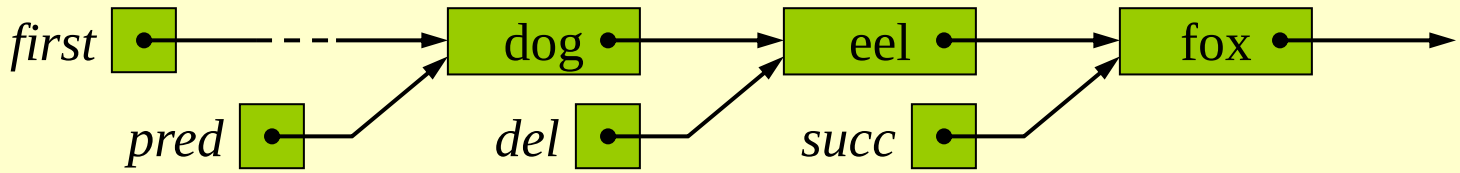


Користење на поврзаните листи

- Анимација (бришење на средишен или последен јазол)

Да се избрише даден јазол *del* од SLL:

1. Нека *succ* е следбеник на јазолот *del*.
2. Во случај кога јазолот *del* не е *first*
   - 2.1. Нека *pred* е јазол претходник на *del*
   - 2.2. Постави го следбеникот на јазолот *pred* да покажува на *succ*
3. Заврши

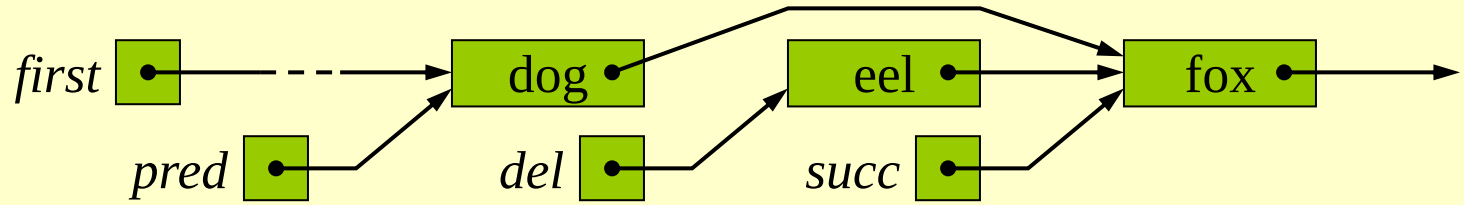


Користење на поврзаните листи

- Анимација (бришење на средишен или последен јазол)

Да се избрише даден јазол *del* од SLL:

1. Нека *succ* е следбеник на јазолот *del*.
2. Во случај кога јазолот *del* не е *first*
   - 2.1. Нека *pred* е јазол претходник на *del*
   - 2.2. Постави го следбеникот на јазолот *pred* да покажува на *succ*
3. Заврши

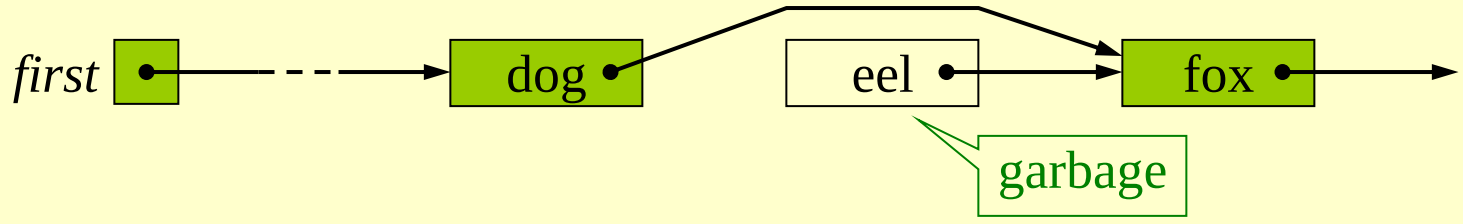


Користење на поврзаните листи

- Анимација (бришење на средишен или последен јазол)

Да се избрише даден јазол *del* од SLL:

1. Нека *succ* е следбеник на јазолот *del*.
2. Во случај кога јазолот *del* не е *first*
   - 2.1. Нека *pred* е јазол претходник на *del*
   - 2.2. Постави го следбеникот на јазолот *pred* да покажува на *succ*
3. Заврши



Предности и недостатоци

- Предности за користење SLL:
  - листите никогаш не можат да се наполнат и секогаш ќе има место за додавање на нови елементи (освен доколку не се наполни меморијата)
  - додавањето и бришењето на елемент е поедноставно отколку кај низите
- Недостатоци за користење SLL:
  - кај листите потребна е дополнителна меморија за чување на покажувачите (следбениците) кои реално не носат корисна информација
  - листите не овозможуваат ефикасен пристап до произволен елемент, туку треба да се измине цела листа за да се стигне до даден елемент