ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Fundamental data structures - Arrays and dynamical lists-

second part

Algorithms and data structures

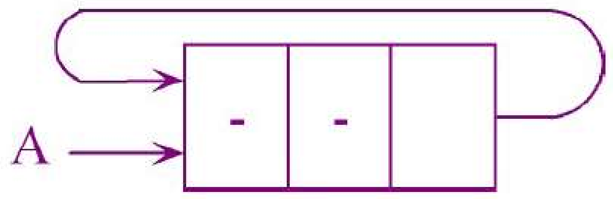
- lectures -

Circular linked lists

- The circular linked lists could give complex algorithms in terms of their traversal that contains several iterations
- The SLL complex algorithms problem with beginning and end can be solved with the use of circular linked lists
- Representation: There is no need of a null poiner for the end of the list. A new special node - **head** is introduced which is the beginning and the end of the list

Circular linked lists

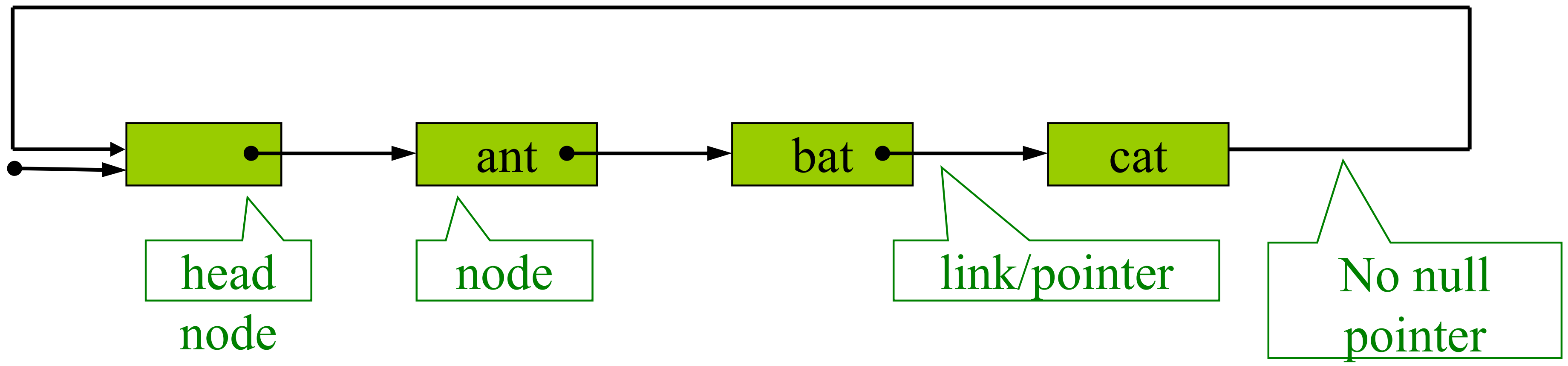
- An empty circular list is not specified with a null pointer, but with a node with an empty info field, and the pointer points to the same node itself



- The circular lists can be deleted in fixed time

Circular linked lists

- A circular linked list traversal is done until the head node is visited, from where the beginning is



- The biggest usage is in operating systems processes scheduling

Circular linked lists

- In general case the operations as insertions of an element or deletion of an element are similar as in a single linked list

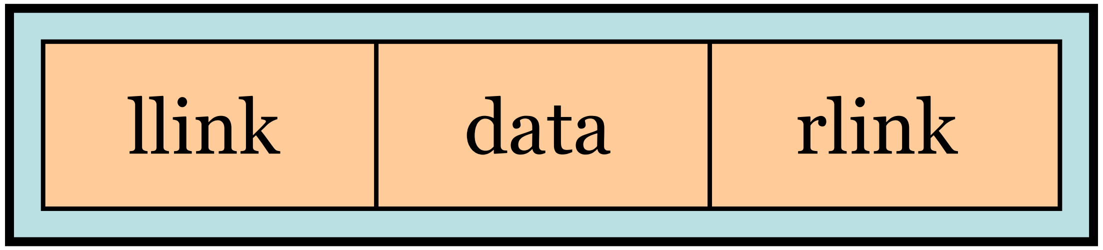
- Homework for implementation

Double linked lists

- **Linked lists problem: Predecessor of a node search for operations of insertion or deletion of a node!**
- One direction in the way of traversal in the data structure
- **Problem solution: Introducing of a double direction for traversal in the data structure!**

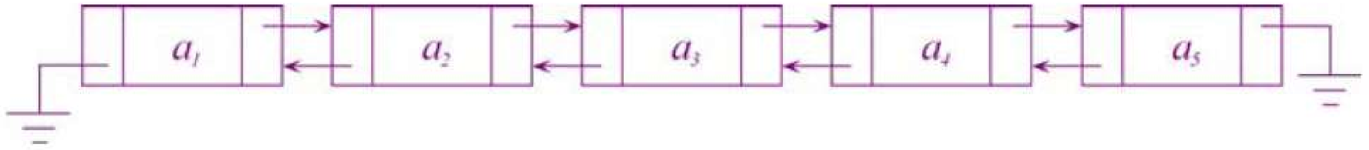
Double linked lists

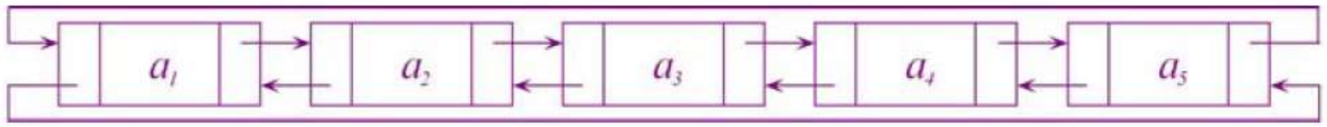
- Representation: The node is consisted of a *data* field and **two pointers** - *rlink* pointer towards the following node and *llink* pointer towards the previous list node

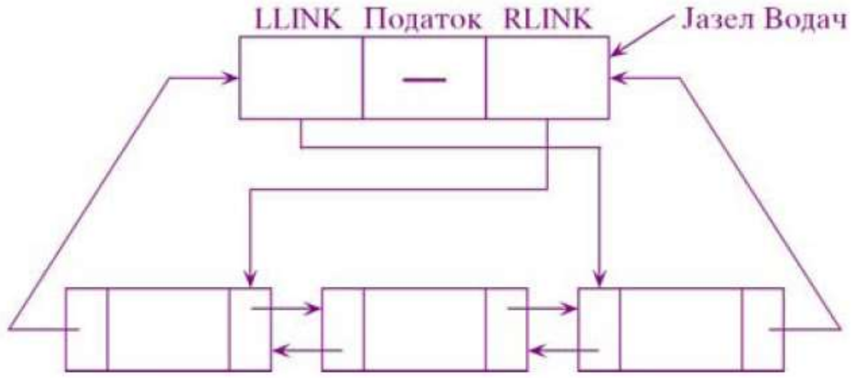


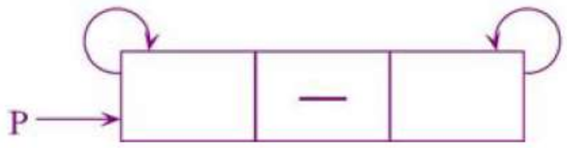
Double linked lists

- Double linked lists examples representations:







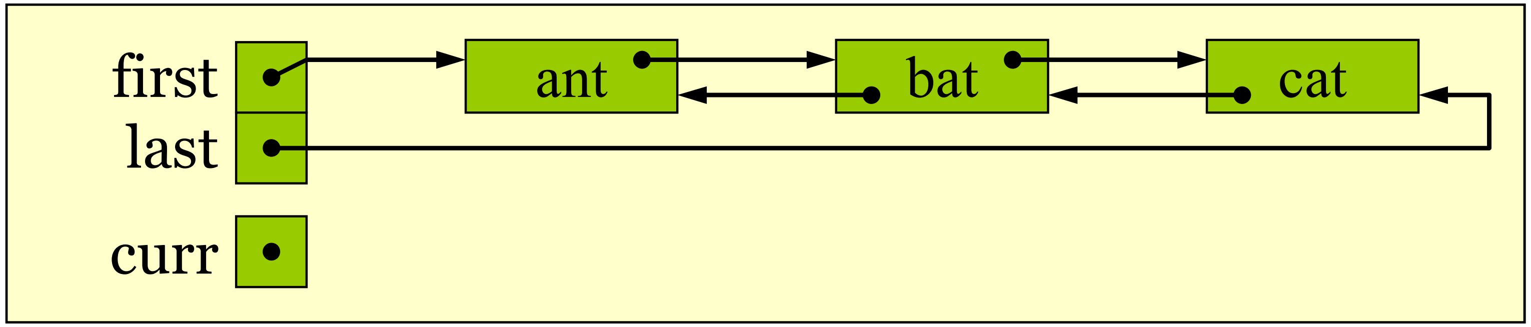


Double linked lists

- Double linked list traversal

```
public void printLastToFirst () {
// Print all elements in this DLL, in last-to-first order.
   for (DLLNode curr = this.last;
            curr != null; curr = curr.pred)
        System.out.println(curr.element);
}
```

- Animation:



Double linked lists usage

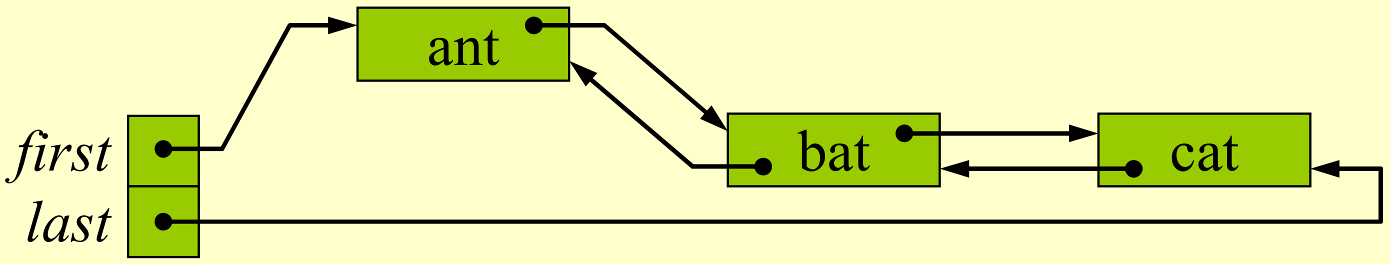
- DLL node insertion algorithm:
  1. a node that is not currently in use in the available memory space is selected
  2. the appropriate value is assigned to the node info field
  3. the value of the rlink field is filled with the address of the node that should be the successor of the new node, and the value of the llink field with the predecessor of the new node
  4. the value of the rlink field of the node that will be the predecessor of the new node should be changed with the address of the new node and the value of the llink field of the node that will be the successor of the new node should be changed with the address of the new node

Double linked lists usage

- Animation (insertion before first list node):

Insert *elem* in a given DLL at the beginning:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put successor of node *ins* to be *first* and *first* to be *ins*
3. Let *succ* be the successor of node *ins*
4. Put the predecessor of node *succ* to be *ins*.
5. End.

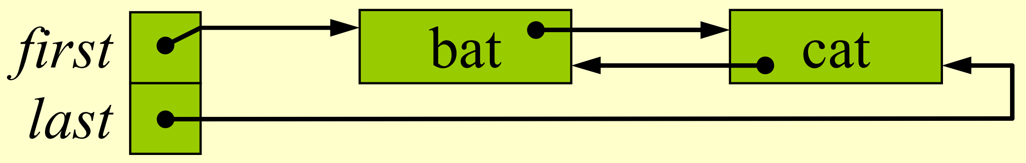


Double linked lists usage

- Animation (insertion after last list node):

```
Insert elem in given DLL at the end:
1.  Put node ins to be new node with info elem and predecessor and successor null.
2.  Put predecessor of node ins to be last and last to be ins

3.  Let succ be the predecessor of ins
4.  Put successor of node succ to be ins
5.  End.
```



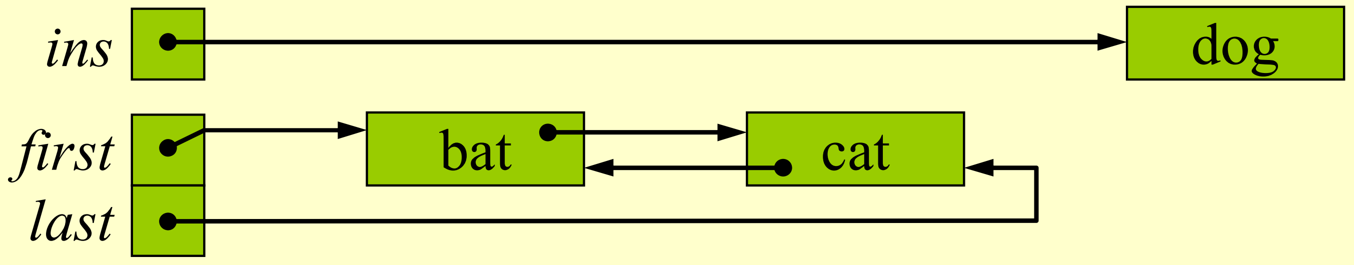
Double linked lists usage

- Animation (insertion after last list node):

Insert *elem* in given DLL at the end:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *last* and *last* to be *ins*

3. Let *succ* be the predecessor of *ins*
4. Put successor of node *succ* to be *ins*
5. End.



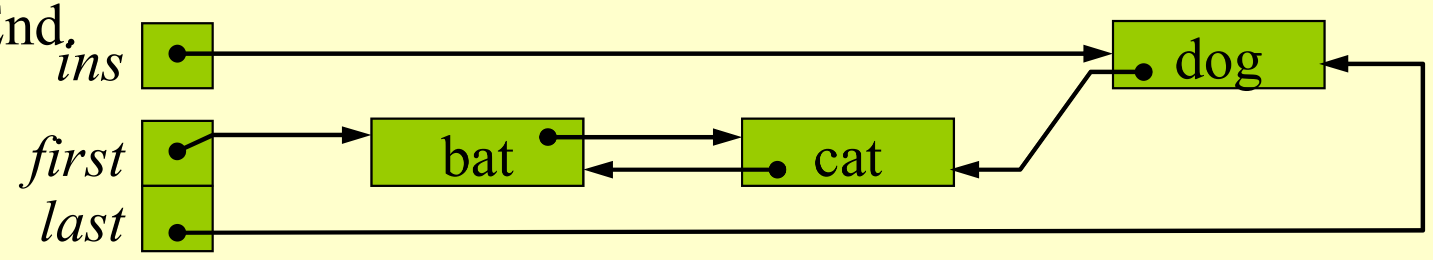
Double linked lists usage

- Animation (insertion after last list node):

Insert *elem* in given DLL at the end:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *last* and *last* to be *ins*

3. Let *succ* be the predecessor of *ins*
4. Put successor of node *succ* to be *ins*
5. End.

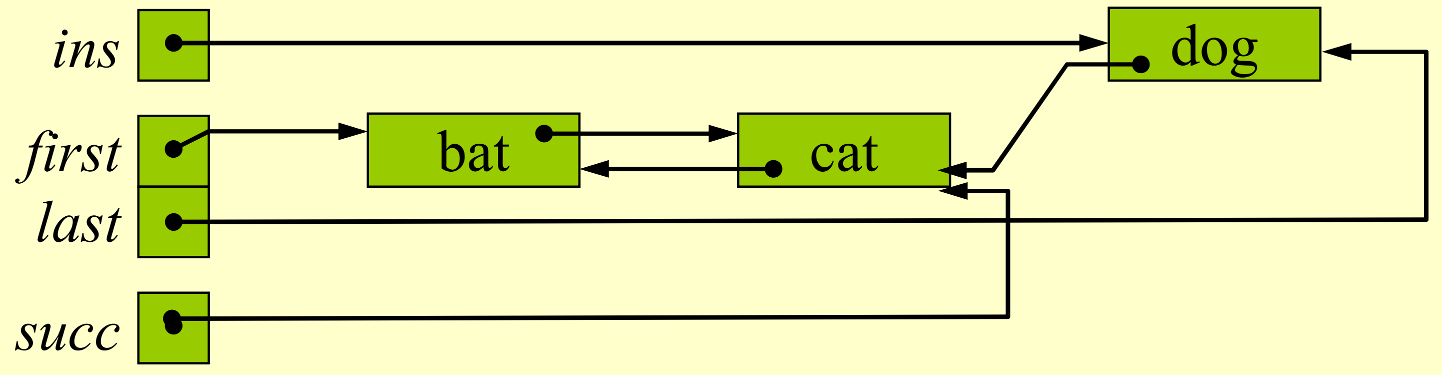


Double linked lists usage

- Animation (insertion after last list node):

Insert *elem* in given DLL at the end:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *last* and *last* to be *ins*
3. Let *succ* be the predecessor of *ins*
4. Put successor of node *succ* to be *ins*
5. End.

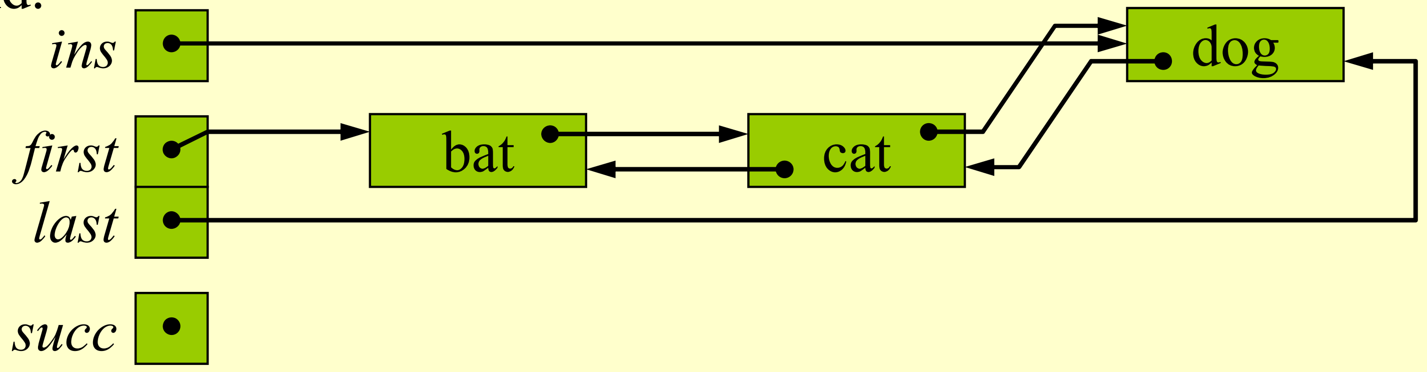


Double linked lists usage

- Animation (insertion after last list node):

Insert *elem* in given DLL at the end:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *last* and *last* to be *ins*
3. Let *succ* be the predecessor of *ins*
4. Put successor of node *succ* to be *ins*
5. End.

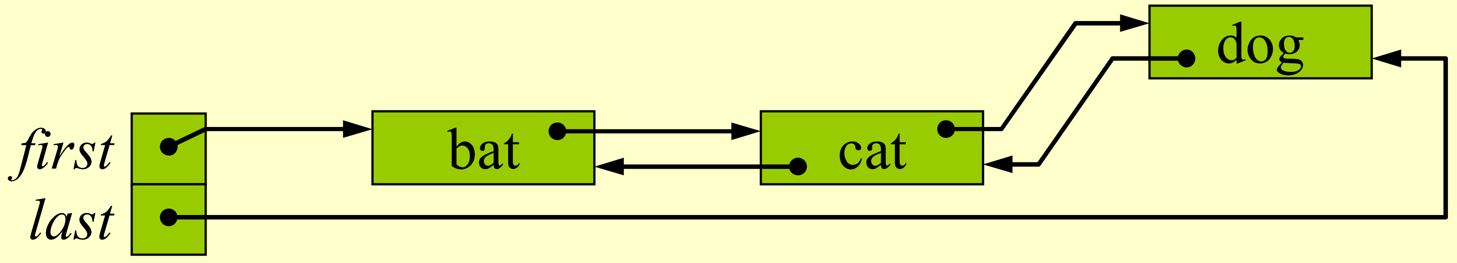


Double linked lists usage

- Animation (insertion after last list node):

Insert *elem* in given DLL at the end:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *last* and *last* to be *ins*
3. Let *succ* be the predecessor of *ins*
4. Put successor of node *succ* to be *ins*
5. End.

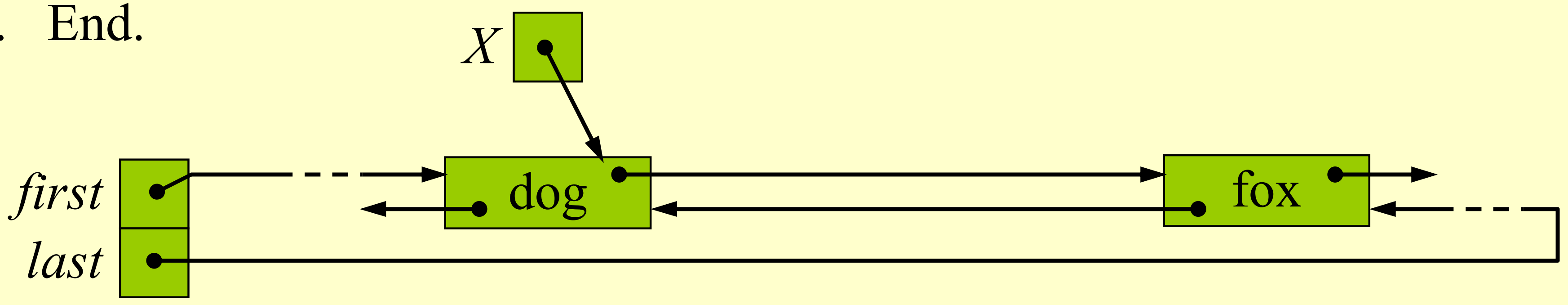


Double linked lists usage

- Animation (insertion between nodes):

Insert *elem* in a given DLL after node X:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *X*
3. Put successor of *ins* to be the successor of node *X*
3. Let *succ* be the successor of *ins*
4. Put predecessor of node *succ* to be *ins,* and successor od *X* to be *ins*
5. End.

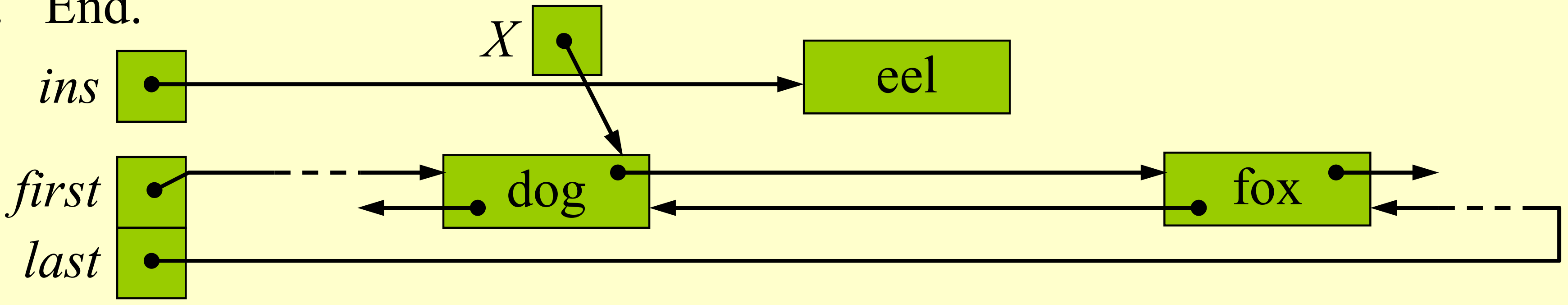


Double linked lists usage

- Animation (insertion between nodes):

Insert *elem* in a given DLL after node X:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *X*
3. Put successor of *ins* to be the successor of node *X*
3. Let *succ* be the successor of *ins*
4. Put predecessor of node *succ* to be *ins,* and successor od *X* to be *ins*
5. End.

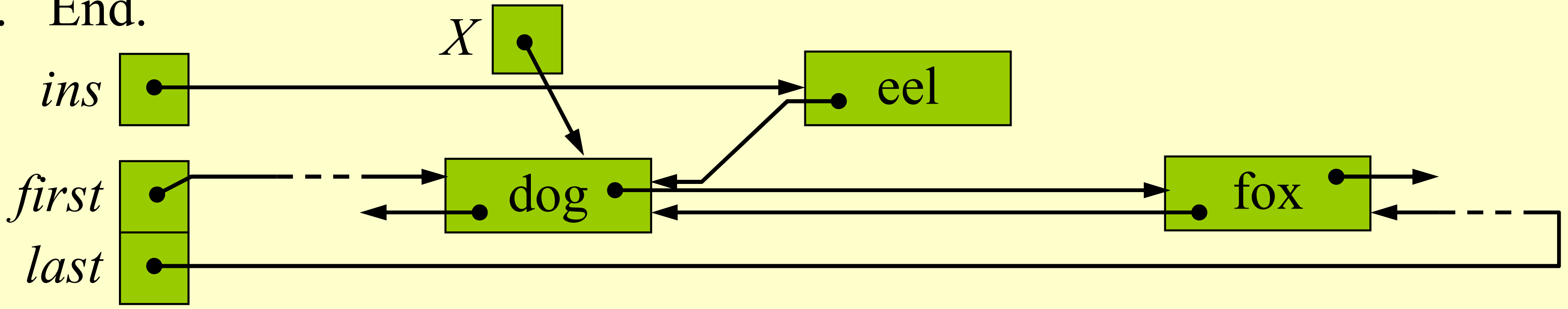


Double linked lists usage

- Animation (insertion between nodes):

Insert *elem* in a given DLL after node X:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *X*
3. Put successor of *ins* to be the successor of node *X*
3. Let *succ* be the successor of *ins*
4. Put predecessor of node *succ* to be *ins,* and successor od *X* to be *ins*
5. End.

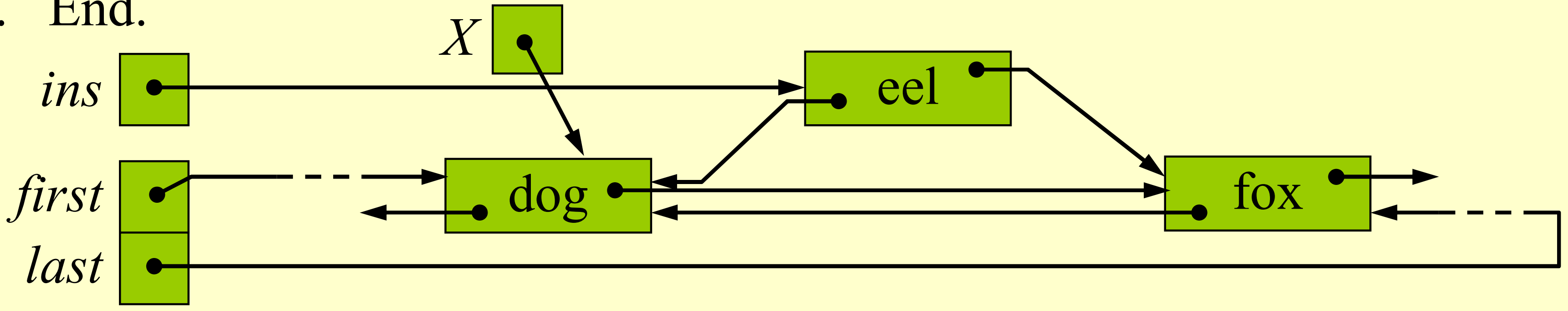


Double linked lists usage

- Animation (insertion between nodes):

Insert *elem* in a given DLL after node X:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *X*
3. Put successor of *ins* to be the successor of node *X*
3. Let *succ* be the successor of *ins*
4. Put predecessor of node *succ* to be *ins,* and successor od *X* to be *ins*
5. End.

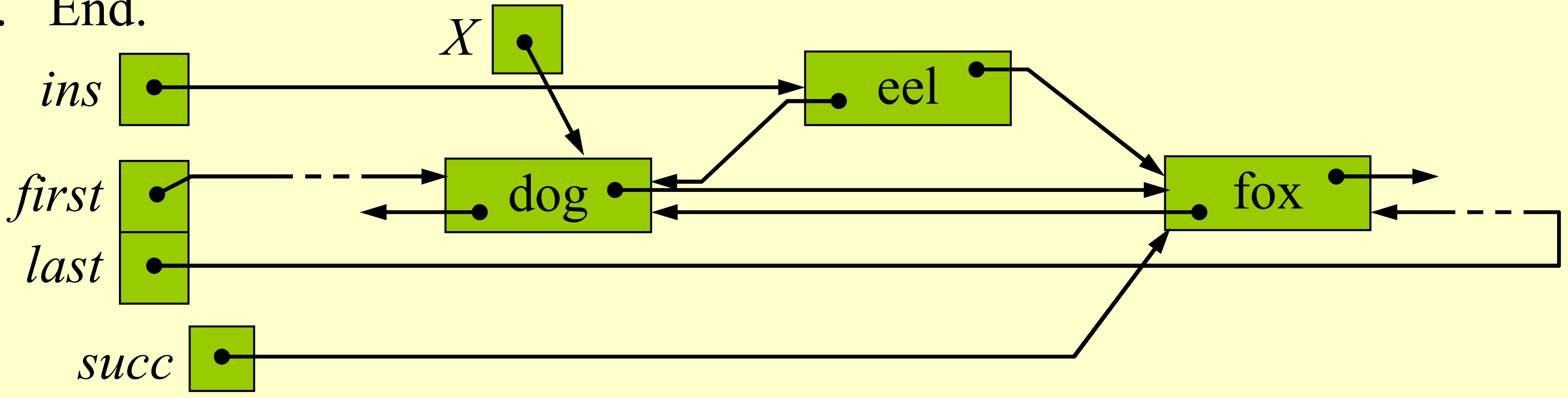


Double linked lists usage

- Animation (insertion between nodes):

Insert *elem* in a given DLL after node X:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *X*
3. Put successor of *ins* to be the successor of node *X*
3. Let *succ* be the successor of *ins*
4. Put predecessor of node *succ* to be *ins,* and successor od *X* to be *ins*
5. End.

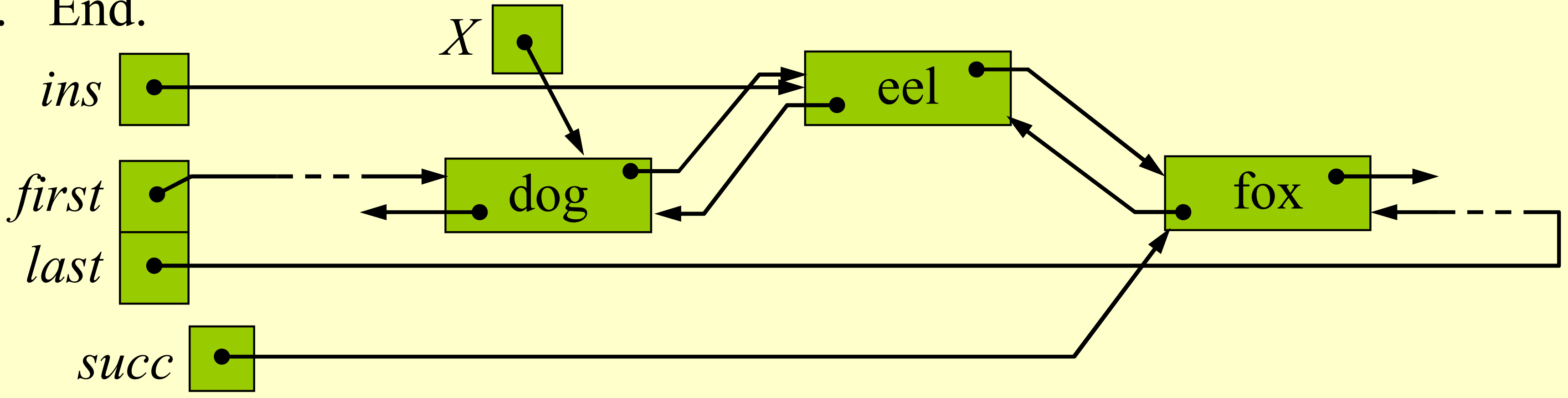


Double linked lists usage

- Animation (insertion between nodes):

Insert *elem* in a given DLL after node X:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *X*
3. Put successor of *ins* to be the successor of node *X*
3. Let *succ* be the successor of *ins*
4. Put predecessor of node *succ* to be *ins,* and successor od *X* to be *ins*
5. End.

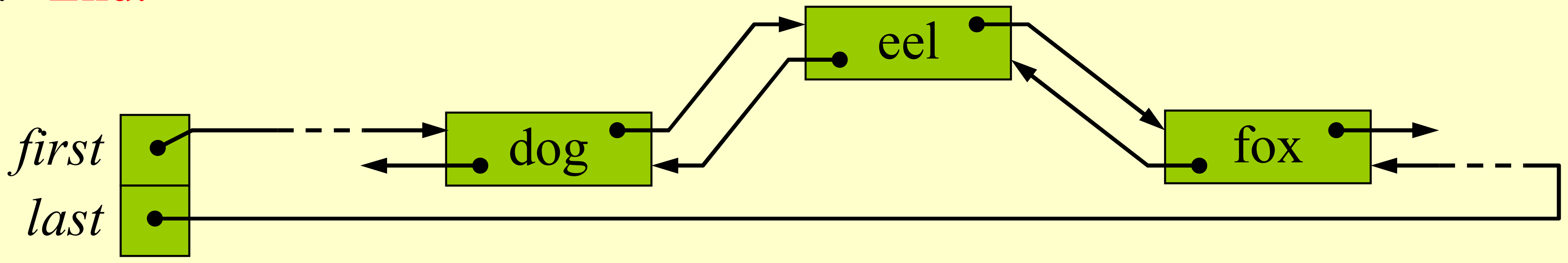


Double linked lists usage

- Animation (insertion between nodes):

Insert *elem* in a given DLL after node X:

1. Put node *ins* to be new node with info *elem* and predecessor and successor null.
2. Put predecessor of node *ins* to be *X*
3. Put successor of *ins* to be the successor of node *X*
3. Let *succ* be the successor of *ins*
4. Put predecessor of node *succ* to be *ins*, and successor od *X* to be *ins*
5. End.



Double linked lists usage

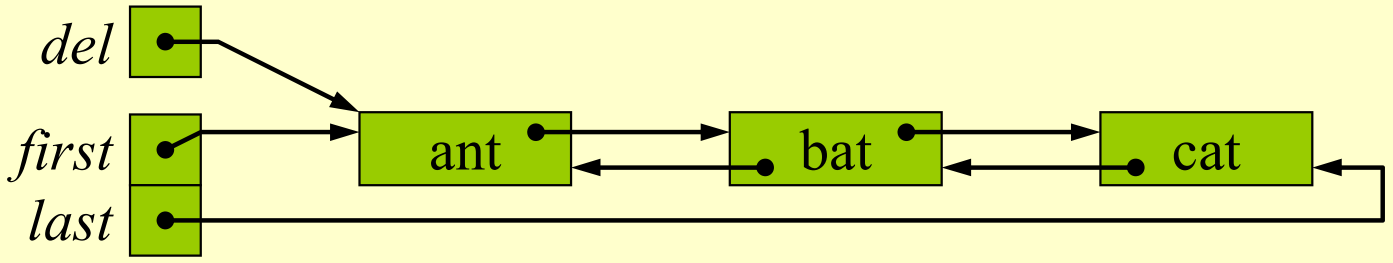
- DLL node deletion algorithm:
  1. an element that we want to delete is selected
  2. its predecessor and successor are found respectively
  3. the value of the rlink field of its predecessor is replaced by the address of the node that is its successor
  4. the value of its successor's llink field is changed with the address of the new predecessor

Double linked lists usage

- Animation (deletion of first (but not last) node)

Delete node *del* from a given DLL at the beginning:

1. Let *succ* be the successor of node *del*.
2. Put *first* to point to node *succ*.
3. Put successor of *succ* to be *null*
4. End.

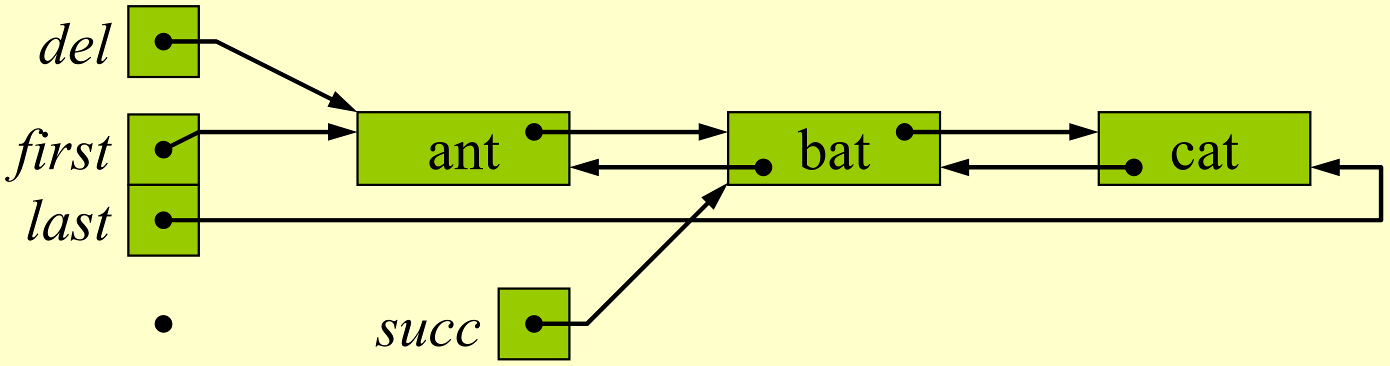


Double linked lists usage

- Animation (deletion of first (but not last) node)

Delete node *del* from a given DLL at the beginning:

1. Let *succ* be the successor of node *del*.
2. Put *first* to point to node *succ*.
3. Put successor of *succ* to be *null*
4. End.

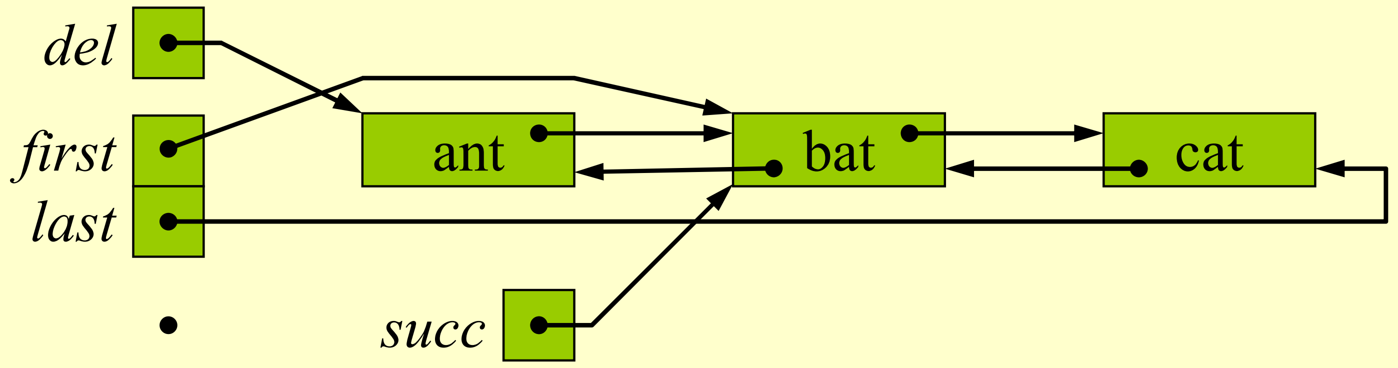


Double linked lists usage

- Animation (deletion of first (but not last) node)

Delete node *del* from a given DLL at the beginning:

1. Let *succ* be the successor of node *del*.
2. Put *first* to point to node *succ*.
3. Put successor of *succ* to be *null*
4. End.

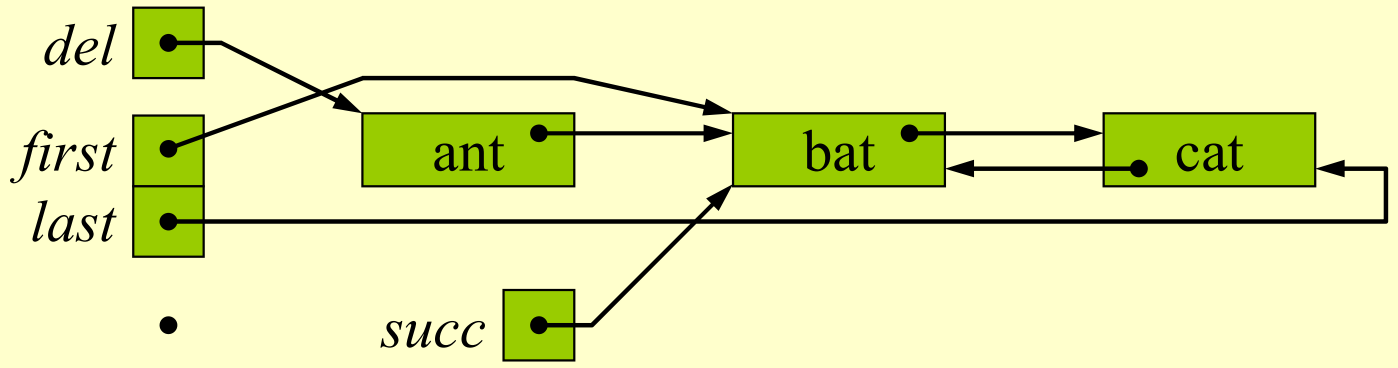


Double linked lists usage

- Animation (deletion of first (but not last) node)

Delete node *del* from a given DLL at the beginning:

1. Let *succ* be the successor of node *del*.
2. Put *first* to point to node *succ*.
3. Put successor of *succ* to be *null*
4. End.

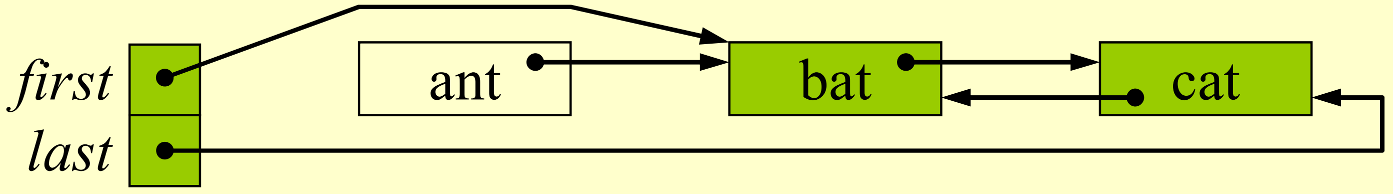


Double linked lists usage

- Animation (deletion of first (but not last) node)

Delete node *del* from a given DLL at the beginning:

1. Let *succ* be the successor of node *del*.
2. Put *first* to point to node *succ*.
3. Put successor of *succ* to be *null*
4. End.

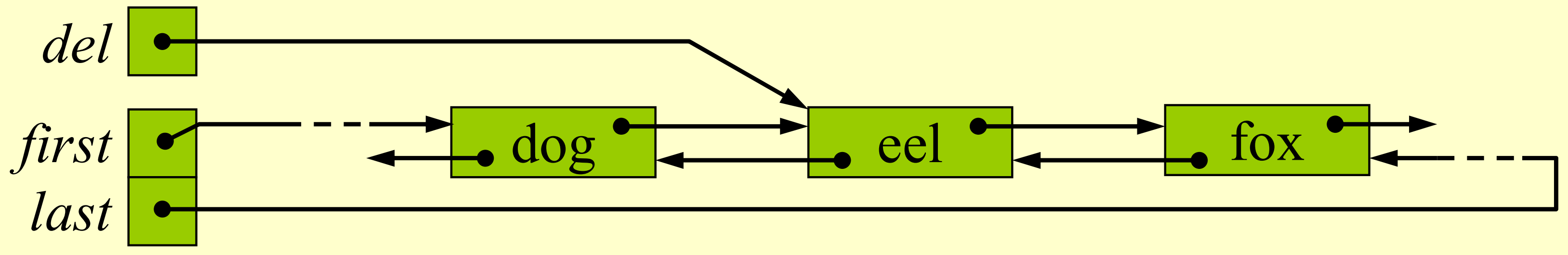


Double linked lists usage

- Animation (deletion of an inside list node):

Delete arbitrary node *del* from DLL:

1. Let *pred* and *succ* be nodes that represent the predecessor of node *del* and successor of node *del* respectively.
2. Put *succ* to be successor of node *pred*.
3. Put *pred* to be predecessor of node *succ*.
4. End.

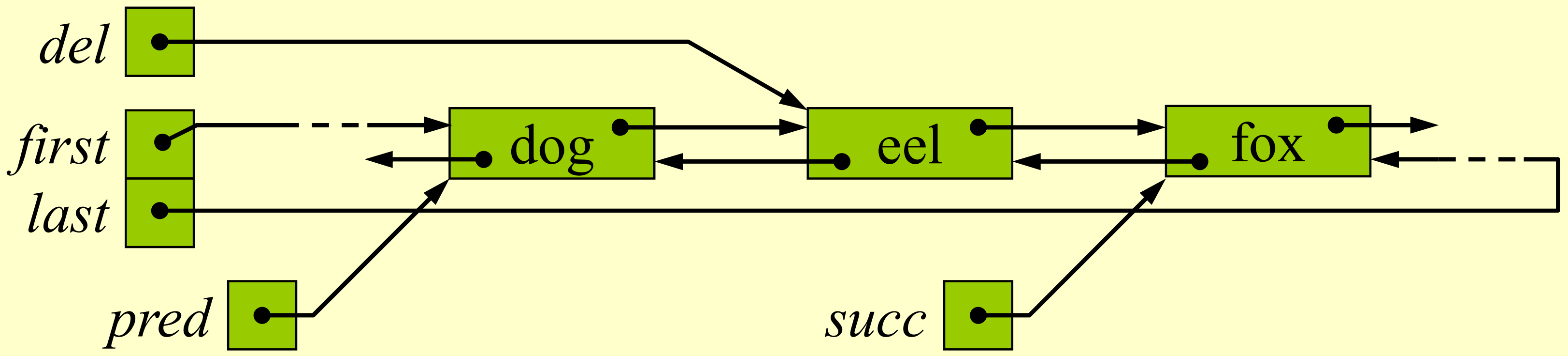


Double linked lists usage

- Animation (deletion of an inside list node):

Delete arbitrary node *del* from DLL:

1. Let *pred* and *succ* be nodes that represent the predecessor of node *del* and successor of node *del* respectively.
2. Put *succ* to be successor of node *pred.*
3. Put *pred* to be predecessor of node *succ*.
4. End.

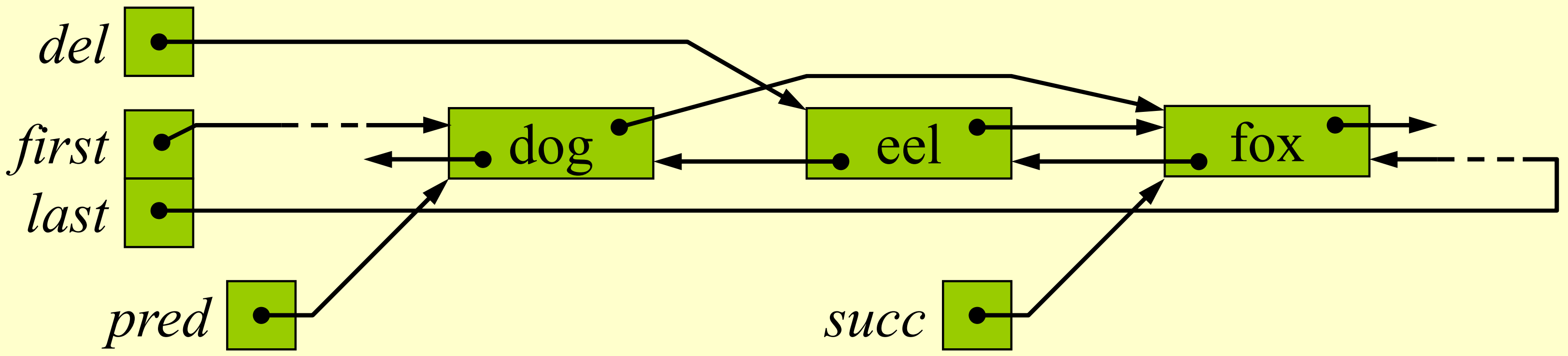


Double linked lists usage

- Animation (deletion of an inside list node):

Delete arbitrary node *del* from DLL:

1. Let *pred* and *succ* be nodes that represent the predecessor of node *del* and successor of node *del* respectively.
2. Put *succ* to be successor of node *pred.*
3. Put *pred* to be predecessor of node *succ*.
4. End.

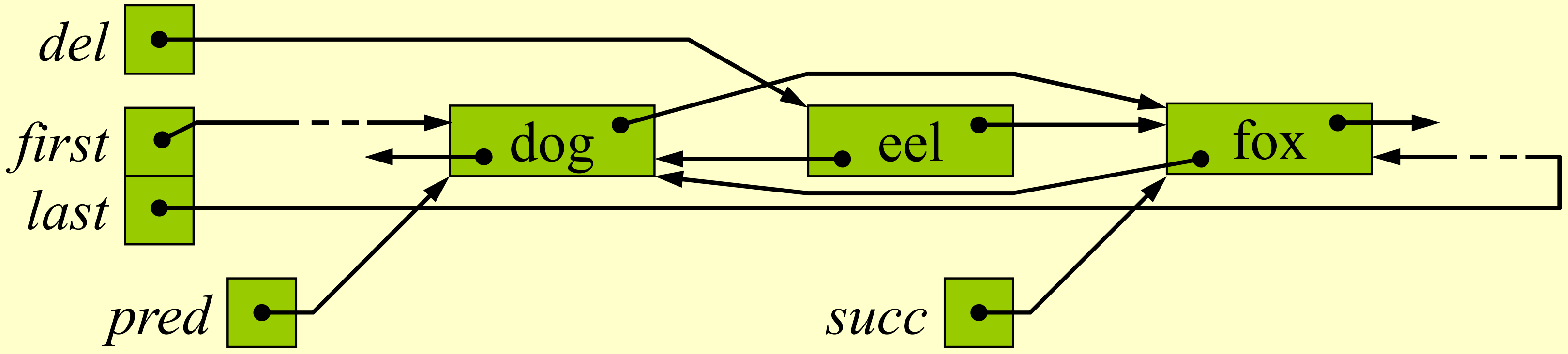


Користење на двојно поврзани листи

- Animation (deletion of an inside list node):

Delete arbitrary node *del* from DLL:

1. Let *pred* and *succ* be nodes that represent the predecessor of node *del* and successor of node *del* respectively.
2. Put *succ* to be successor of node *pred.*
3. Put *pred* to be predecessor of node *succ*.
4. End.

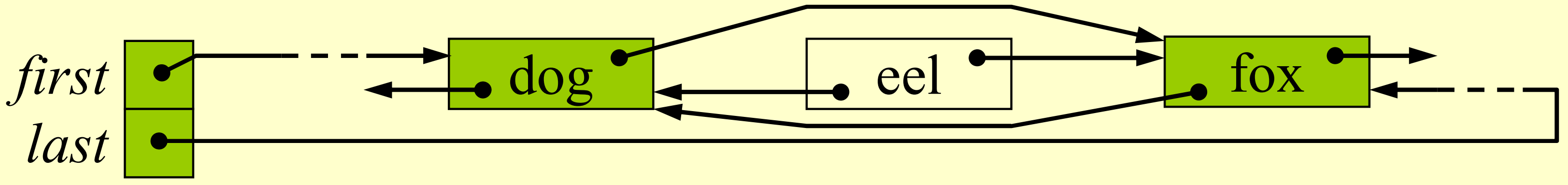


Double linked lists usage

- Animation (deletion of an inside list node):

Delete arbitrary node *del* from DLL:

1. Let *pred* and *succ* be nodes that represent the predecessor of node *del* and successor of node *del* respectively.
2. Put *succ* to be successor of node *pred*.
3. Put *pred* to be predecessor of node *succ*.
4. End.



Most common mistakes in working with dynamical memory allocation!

- Incorrect use of pointers in memory locations!

- The only correct operations with pointers towards nodes in lists can be:
  - checking if they are null
  - checking if they are equal to the value of other pointers (do they point to the same node)

Operations complexities

Operation	**Unsorted SLL**	**Unsorted DLL**
Search	$O(n)$	$O(n)$
Insertion	$O(1)$	$O(1)$
Deletion	$O(n)$	$O(1)$
Predecessor	$O(n)$	$O(1)$
Successor	$O(1)$	$O(1)$
Minimum	$O(n)$	$O(n)$
Maximum	$O(n)$	$O(n)$