uc3m | Universidad Carlos III de Madrid

Unit 2
Doubly linked list

Data Structures and Algorithms

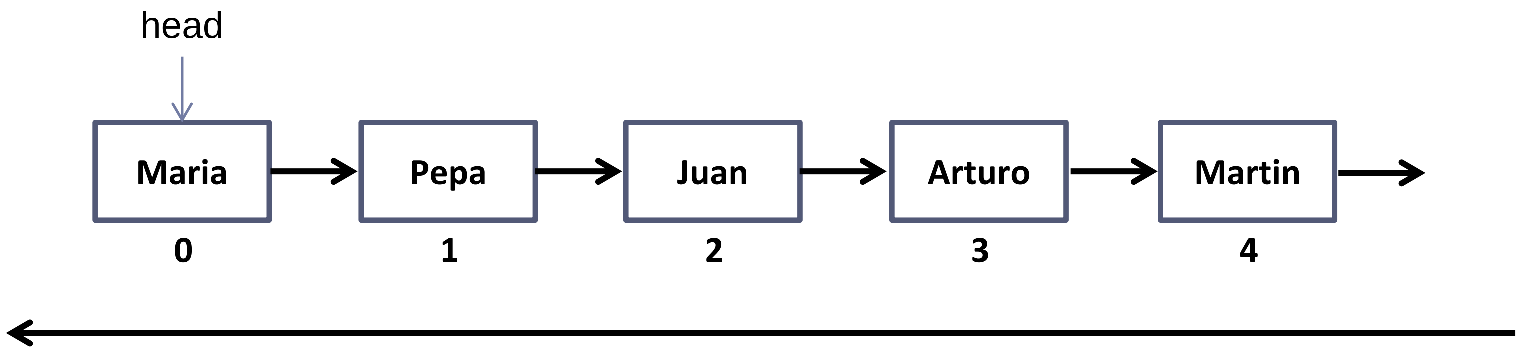
Linear ADTs

- Stack ADT
- Queue ADT
- Singly Linked List ADT
- **Doubly Linked List ADT**

How to improve the access to the nodes?

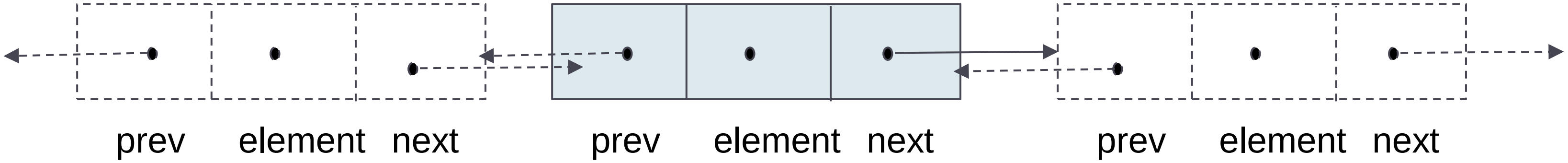
Objectives:

- To make easy searching an element that is in the **last positions**.
- To make easy performing **reverse traversing** in the list.



How to improve the access to the nodes?

- In addition to the element and the reference **next**, a node has an additional reference to the previous node (**prev**).
- Each node points forward to the **next** node and backward to the **previous** node.
- It allows visiting the list from left to right, and also in reverse.



Doubly Linked List ADT

Some possible operation are:

List(): creates a new list.

addFirst(L,e): add the element e at the beginning of the list L.

addLast(L,e): add the element e at the tail of the list L.

removeFirst(L): removes the first element of the list L. It returns the element.

removeLast(L): removes the last element of the list L. It returns the element.

isEmpty(L): returns True if the list L is empty, False otherwise.

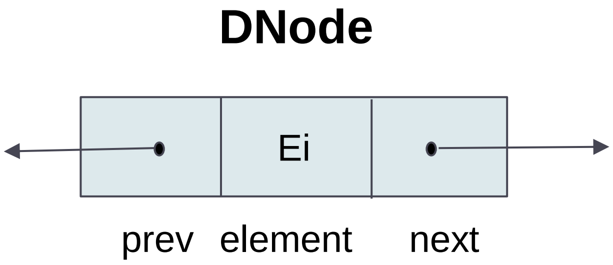
getAt(L,index): returns the element at the position index of the list L.

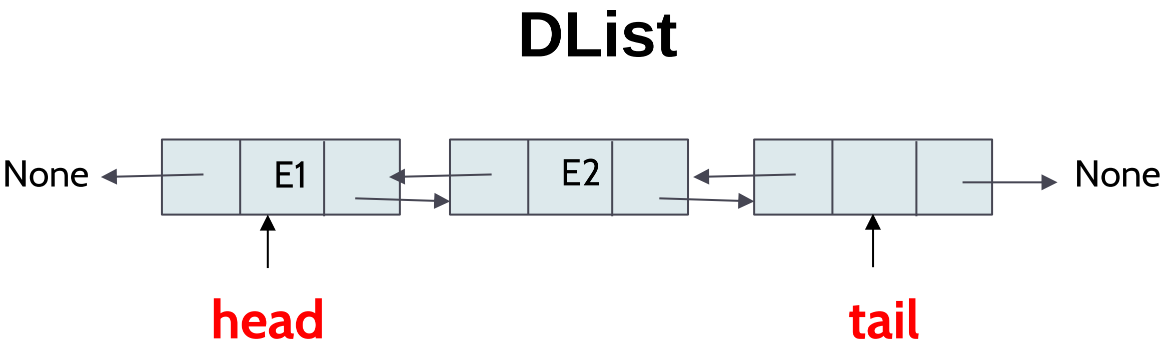
contains(L,e): returns the first index of e in the list L. If e does not exist, it returns -1.

insertAt(L,index,e): insert the element e at the position index of the list L.

removeAt(L,index): removes the element at the position index of the list L. It returns the element.

Doubly Linked List ADT

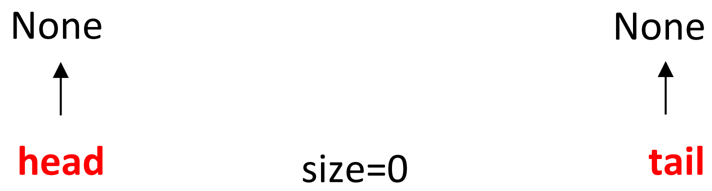




Doubly Linked List ADT

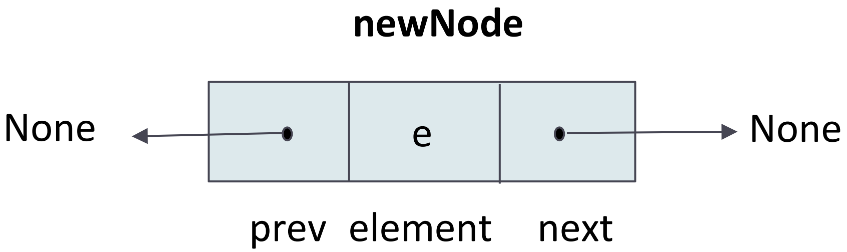
```
class DList:
  def __init__(self):
    """creates an empty list"""
    self.head=None
    self.tail=None
    self.size=0
```

l=DList()



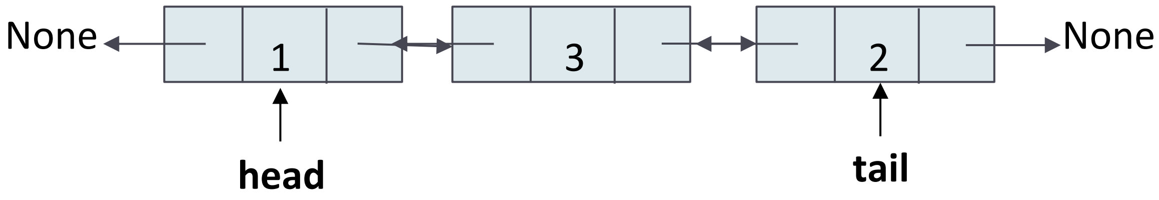
```
class DNode:
  def __init__(self, e, n=None, p=None ):
    self.element = e
    self.next = n
    self.prev = p
```

newNode=DNode(e)



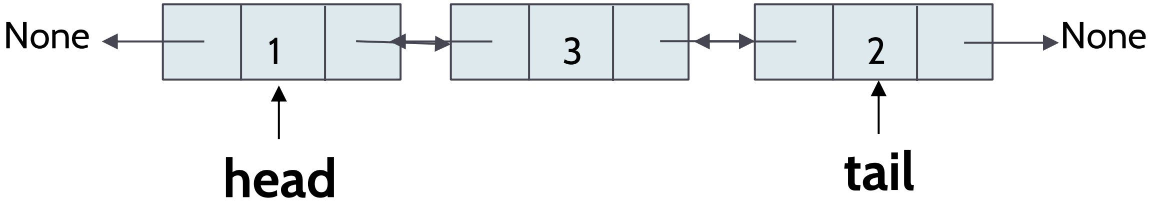
addFirst(e)

```
def addFirst(self,e):
   newNode=DNode(e)
   if self.isEmpty():
      self.head=newNode
      self.tail=newNode
   else:
      newNode.next=self.head
      self.head.prev=newNode
      self.head=newNode
   self.size=self.size+1
```



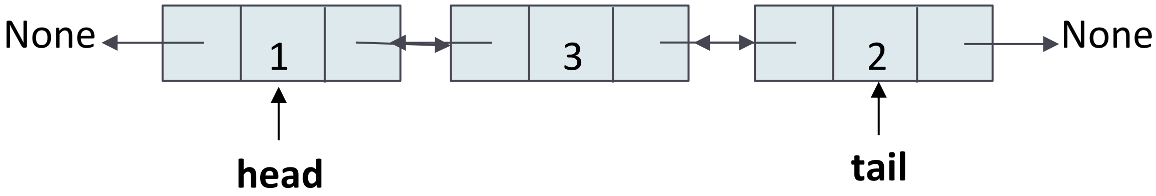
addLast(e)

```
def addLast(self,e):
  newNode=DNode(e)
  if self.isEmpty():
    self.head=newNode
    self.tail=newNode
  else:
    newNode.prev=self.tail
    self.tail.next=newNode
    self.tail=newNode
  self.size=self.size+1
```



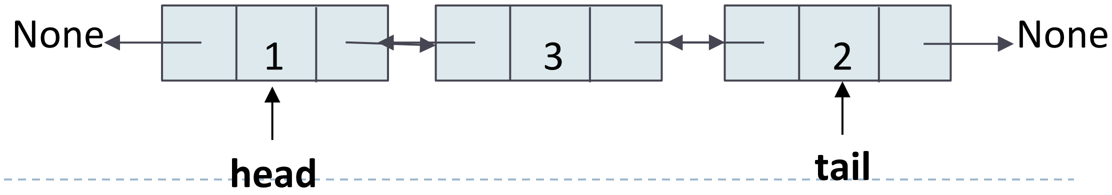
removeFirst()

```
def removeFirst(self):
    if self.isEmpty():
        print("Error: list is empty")
        return None
    result=self.head.element
    self.head= self.head.next
    if self.head is None:           #if the list has one node.
      self.tail=None
    else:
      self.head.prev = None
    self.size=self.size-1
return result
```



removeLast()

```
def removeLast(self):
  if self.isEmpty():
    print("Error: list is empty")
    return None
  result=self.tail.element
  self.tail= self.tail.prev
  if self.tail is None:            #if the list has one node.
    self.head=None
  else:
    self.tail.next = None
  self.size=self.size-1
  return result
```



Exercises for the lab class

- **getAt(L,index):** returns the element at the index position
- **contains(L,e):** returns the first index of the element in the list. If e does not exist, then it returns -1.
- **removeAt(L,index):** removes the element at the index position.
- **removeAll(L,e):** removes all the occurrences of e in the list.
- **show(L,op)**:
  - If op=0, it prints the elements of the list.
  - if op=1, it prints the elements of the list in reverse order (from right to left).

Checking palindrome words

A palindrome word is one that reads the same backward as forward. Examples:

Anna, Level, Civic, Madam, Noon.

- Implement a Python function that takes a word and returns true if it is palindrome, else false.
- In your solution, you **have to use a doubly linked list** where each node contains only one character of the input word.