Lecture 7
Doubly Linked List

Dr. Sara S. Elhishi

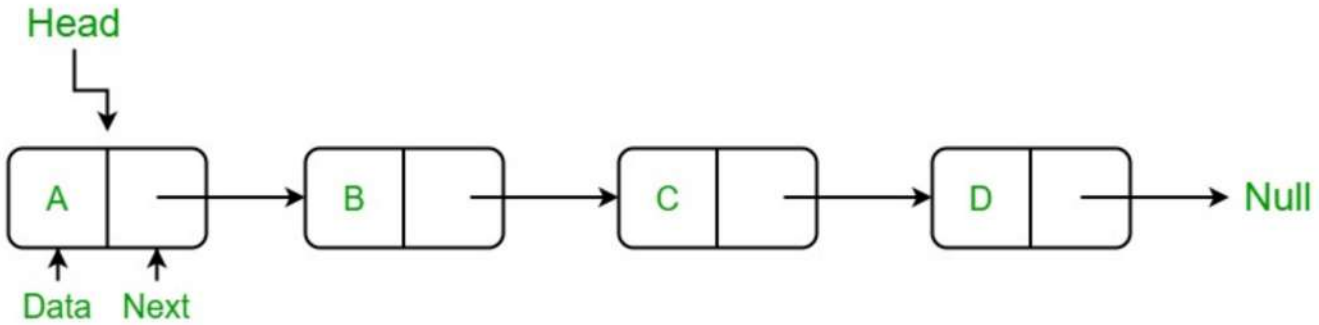
Information Systems Dept.

Mansoura University, Egypt.

Sara_shaker2008@mans.edu.eg

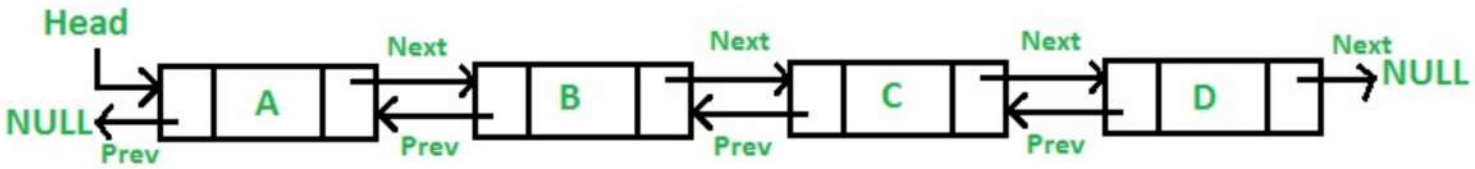
Recap

- Single Linked List; knows its head and next of each link.
- Double-ended Ended List; knows its head, its last, and next of each link.
- Both lists provide a utility to move forward, but not backward.



Doubly Linked List

- Doubly linked list enables applications to move forward and backwards through the list.
- Both links terminate in a null pointer.
- Each instance of link insertion or deletion requires handling four links.
- Consider it as an upgrade of the traditional Single Linked List.



Link Class

```python
# Link class of the DoublyLinkedList.py module
import LinkedList

class Link(LinkedList.Link):
    def __init__(self, datum, next=None, previous=None):
        self.__data = datum
        self.__next = next
        self.__previous = previous

    # Accessors
    def getData(self): return self.__data
    def getNext(self): return self.__next
    def getPrevious(self): return self.__previous

    def setData(self, d): self.__data = d

    def setNext(self, link):
        if link is None or isinstance(link, Link):
            self.__next = link
        else:
            raise Exception('Next link must be Link or None')

    def setPrevious(self, link):
        if link is None or isinstance(link, Link):
            self.__previous = link
        else:
            raise Exception('Next link must be Link or None')

    def isFirst(self): return self.__previous is None

```

DoublyLinkedList Class

```
import Link
import LinkedList

def identity(x): return x

class DoublyLinkedList(LinkedList.LinkedList):
    def __init__(self):
        self.__first = None
        self.__last = None

    def getFirst(self): return self.__first
    def getLast(self): return self.__last

    def setFirst(self, link):
        if link is None or isinstance(link, Link.Link):  # Check type
            self.__first = link
            if(self.__last is None or link is None): # if list is empty or truncated
                self.__last = link                   # update both links
        else:
            raise Exception('First link must be Link or None')

    def setLast(self, link):
        if link is None or isinstance(link, Link.Link):  # Check type
            self.__last = link
            if(self.__first is None or link is None): # if list is empty or truncated
                self.__first = link                   # update both links
        else:
            raise Exception('First link must be Link or None')
```

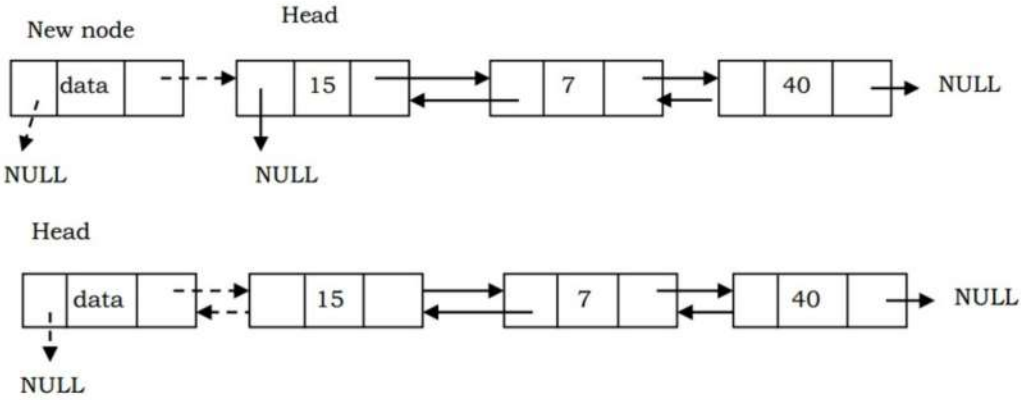
Traverse Backwards

- Since we inherit the LinkedList class, we can apply traversing forward using the **traverse** method implemented before.
- We need only to implement a method to traverse the list from last to first.

```
def traverseBackwards(self, func=print):
    link = self.getLast()                # start with the last link
    while link is not None:              # Keep going until no more links
        func(link)                       # Apply print
        link = link.getPrevious()        # Move Backwards
```

Insertion

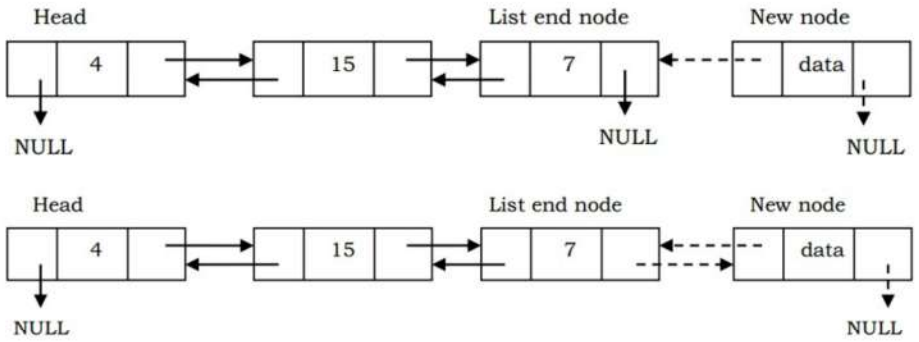
Insert at Beginning




```
# Insert new datum at the strat of the list
def insertFirst(self, datum):
    link = Link.Link(datum, next = self.getFirst())
    if self.isEmpty():      # if list is still empty
        self.setLast(link)  # insert at last (and first)
    else:                   #otherwise, first link in list now has link before
        self.getFirst().setPrevious(link)   # set link before current first
        self.setFirst(link)         # update first link

# override parent class
insert = insertFirst
```

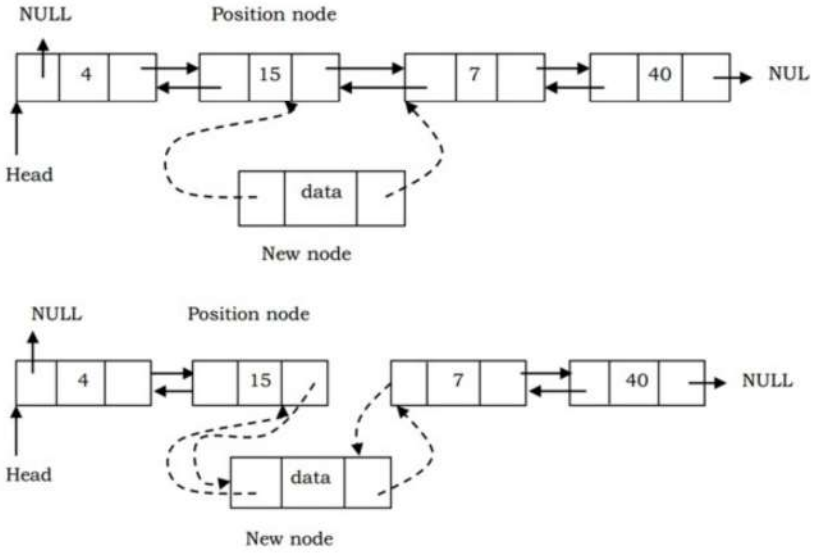
Insert at End




```
def insertLast(self, datum):
    link = Link.Link(datum, previous = self.getLast())
    if self.isEmpty():
        self.setFirst(link)     # insert link as first (and Last)
    else:
        self.getLast().setNext(link)    # set link after current last
        self.setLast(link)      # update last link
```

Insert at Specific Position

```
# Insertion and deletion in the middle
def insertAfter(self, goal, newDatum, key=identity):
    link = self.find(goal, key) # Find matching Link object
    if link is None: # If not found,
        return False # return failure
    if link.isLast(): # If matching Link is last,
        self.insertLast(newDatum) # then insert at end
    else:
        newLink = Link.Link( # Else build a new Link node with
            newDatum, # the new datum that comes just
            previous=link, # after the matching link and
            next=link.getNext()) # before the remaining list
        link.getNext().setPrevious( # Splice in reverse link
            newLink) # from link after matching link
        link.setNext(newLink) # Add newLink to list
    return True
```



Deletion

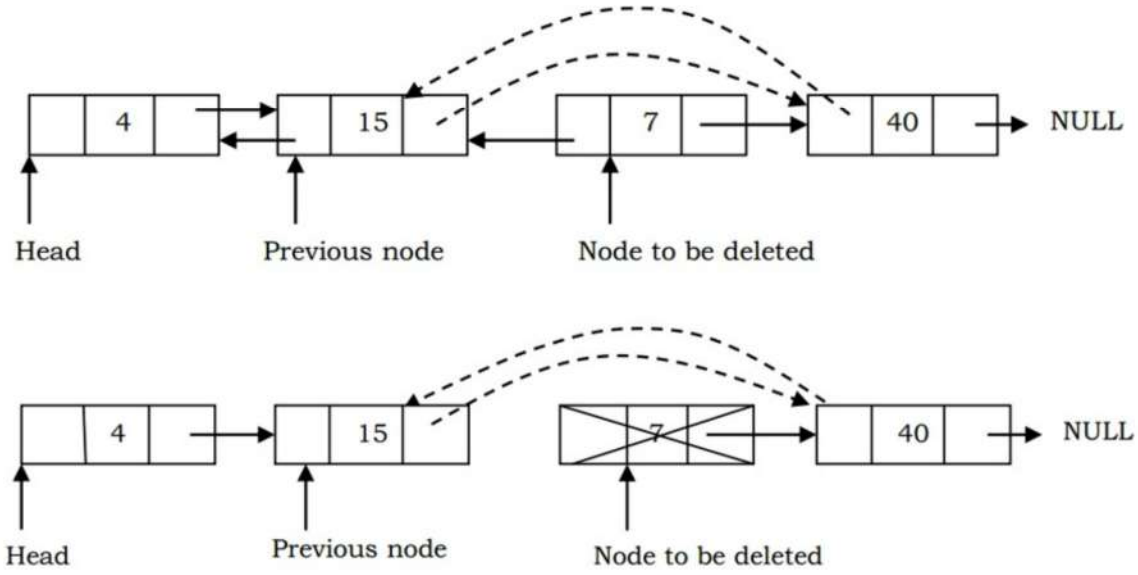
Delete at Beginning

```
def deleteFirst(self):
    if self.isEmpty():
        raise Exception('Cannot delete first of empty list')
    first = self.getFirst()      # Store first link
    self.setFirst(first.getNext())  # remove first, and advancec to its next
    if self.getFirst():             # If that leaves a link
        self.getFirst().setPrevious(None)   # update its predessor
```

Delete at End

```
def deleteLast(self):
    if self.isEmpty(): # If list is empty, raise exception
        raise Exception("Cannot delete last of empty list")
    last = self.getLast() # Store the last link
    self.setLast(last.getPrevious()) # Remove last, advance to prev
    if self.getLast(): # If that leaves a link in the list,
        self.getLast().setNext(None) # Update its successor
```

Delete at Specific Position



Delete at Specific Position

```
def delete(self, goal, key=identity):
    link = self.find(goal, key) # Find matching Link object
    if link is None: # If not found, raise exception
        raise Exception("Cannot find link to delete in list")
    if link.isLast(): # If matching Link is last,
        return self.deleteLast() # then delete from end
    elif link.isFirst(): # If matching Link is first,
        return self.deleteFirst() # then delete from front
    else: # Otherwise it's a middle link
        link.getNext().setPrevious(link.getPrevious()) # Set next link's previous
                                                       # to link preceding the match
        link.getPrevious().setNext(link.getNext()) # Set previous link's next
                                                   # to link following the match
    return link.getData() # Return deleted data item
```

Thanks