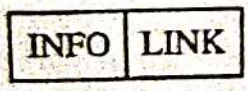
UNIT 4

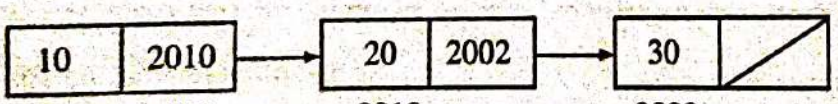
LINKED LIST

4.1 Overview of Linked list :

- Linked list is a linear data structure. It is a collection of Nodes where each nodes are connecte via link. In Linked List each node having at least two parts. First part represents value of th node and second part represents an address of the next node. If Next node is not present the it contains None value.



- Consider Following Presentation of Linked List :



- In above presentation variable FIRST contains an address of first node in the linked list. If linke list is empty then value of FIRST is None.
- In Linked List nodes are logically adjacent but physically not adjacent to each other. Becau address of first node is 2000, address of second node is 2010 and address of third node is 200
- Thus in Linked List memory is not allocated sequentially to each node.
- In Python Node can be represented using Class as shown below :

```
class Node:
    def __init__(self, info):
        self.info=info
        self.link = None
```

- Now you can simply create a new node by creating an object of class Node as shown below :

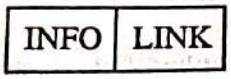
```
New_Node = Node(10) #Here 10 represents value of Node
```

4.2 Types of Linked List :

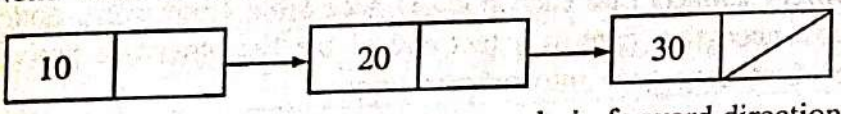
- There are four types of linked list:

(1) Singly Linked List

- Singly Linked List is a collection of variable number of nodes in which each node consists of two parts. First part contains a value of the node and second part contains an address of the next node.
- Following figure represents a structure of the node.



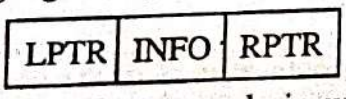
- Consider Following example in which Singly Linked List consist of three nodes. Each node having INFO part and LINK part.
- INFO part contains value of the node.
- LINK part contains address of the next node in the linked list. If next node is not present then LINK part contains None value. Thus LINK part of the last node always contains None value to indicate end of the list.



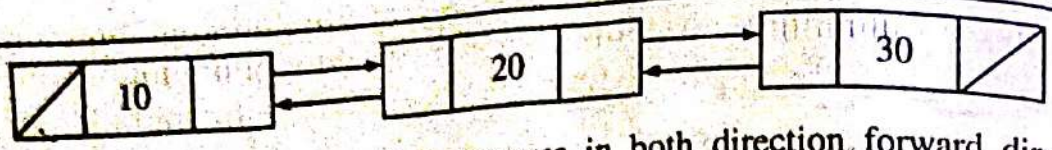
- In Singly Linked List we can traverse only in forward direction because LINK part contains address of the next node in the list. It is not possible to traverse in backward direction in Singly Linked List.

(2) Doubly Linked List

- Doubly Linked List is a collection of variable number of nodes in which each node consists of three parts. First part contains an address of previous node, second part contains a value of the node and third part contains an address of the next node.
- Following figure represents a structure of the node.



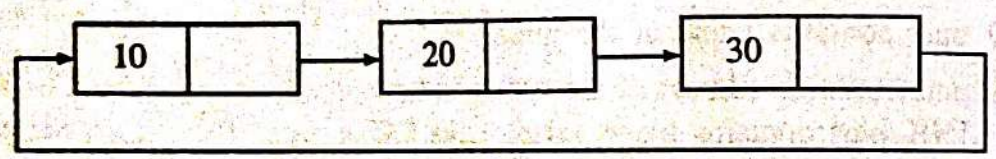
- Consider Following example in which Doubly Linked List consist of three nodes. Each node having LPTR part, INFO part and RPTR part.
- INFO part contains value of the node.
- LPTR part contains an address of the previous node in the linked list. If previous node is not present then LPTR part contains None value. Thus LPTR part of the first node always contains None value.
- RPTR part contains an address of the next node in the linked list. If next node is not present then RPTR part contains None value. Thus RPTR part of the last node always contains None value to indicate end of the list.



- In Doubly Linked List we can traverse in both direction forward direction as well backward direction because LPTR part contains an address of the previous node in RPTR part contains an address of the next node the list.
- However Doubly Linked List having two pointers it occupies more memory as compa to Singly Linked List.

(3) Circular Singly Linked List

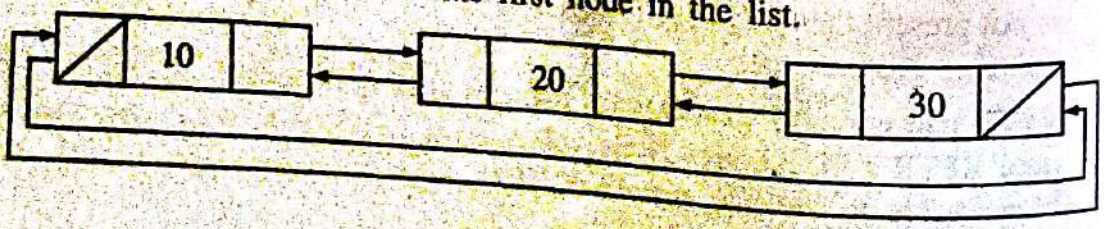
- Circular Singly Linked List is a collection of variable number of nodes in which each n consists of two parts. First part contains a value of the node and second part contains address of the next node such that LINK part of the last node contains an address of first node.
- Consider Following example in which Circular Singly Linked List consist of three noc Each node having INFO part and LINK part.
- INFO part contains value of the node.
- LINK part contains address of the next node in the linked list. If next node is not pres then LINK part contains an address of the first node. Thus LINK part of the last n always contains an address of the first node in the list.



- In Circular Singly Linked List each node is accessible from every node in the list. I we have to take necessary care to detect end of the list otherwise processing of Circ Singly Linked List may goes into infinite loop.

(4) Circular Doubly Linked List

- Circular Doubly Linked List is a collection of variable number of nodes in which e node consists of three parts. First part contains an address of previous node, second p contains a value of the node and third part contains an address of the next node such t RPTR part of the last node contains an address of the first node and LPTR part of first node contains an address of the last node in the list.
- Consider Following example in which Circular Doubly Linked List consist of three nod Each node having LPTR part, INFO part and RPTR part.
- INFO part contains value of the node.
- LPTR part contains an address of the previous node in the linked list. If previous n is not present then LPTR part contains an address of the last node. Thus LPTR part the first node always contains an address of the last node in the list.
- RPTR part contains an address of the next node in the linked list. If next node is not pres then RPTR part contains an address of the first node. Thus RPTR part of the last n always contains an address of the first node in the list.



❖ In Circular Doubly Linked List each node is accessible from every node in the list. But we have to take necessary care to detect end of the list otherwise processing of Circular Doubly Linked List may goes into infinite loop.

4.3 Basic operations on singly linked list :

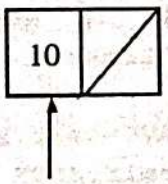
4.3.1 Insertion of a New Node at the beginning of Linked List :

❖ In order to insert a New Node at the beginning of the Linked List we have to follows the below steps:

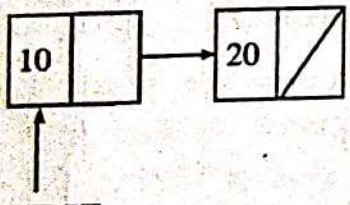
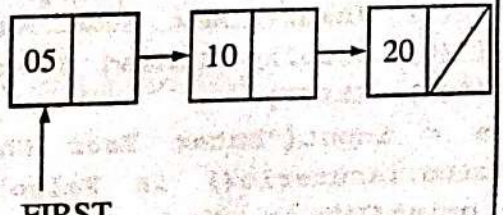
(1) First we have to Create a New Node.

(2) After creating new node we have to check weather linked list is empty or not. We have two possibilities:

(A) Linked List is empty (FIRST=None). In this case the newly created node becomes the first node of linked list.

Before Insertion	After Insertion
FIRST = None	NEW  FIRST

(B) Linked List is not empty (FIRST ‴ None). In this case we can insert new node at the beginning of linked list. Now new node becomes the first node.

Before Insertion	After Insertion
FIRST ≠ None 	NEW_Node 

❖ **Algorithm to insert new node at the beginning of Linked List**

Step 1 : New_Node ← Node(Value)

Step 2 : If self.FIRST is None then

```
        self.FIRST=New_Node
    else
        New_Node.link = self.FIRST
        self.FIRST=New_Node
```

❖ **Program to Insert New Node at the Beginning of Linked List.**

```
class Node:
    def __init__(self,info):
        self.info=info
        self.link=None
class SinglyLinkedList:
    def __init__(self):
        self.FIRST = None
        self.PRED = No
        self.SAVE = No
    def InsertBegining(self,New_Node):
        if(self.FIRST is None):
            self.FIRST=New_No
        el
            New_Node.link=self.FIRS
            self.FIRST=New_No
    def display(self):
        if(self.FIRST is None):
            print("Linked List is Empty"
        el
            self.SAVE=self.FIRS
            while(self.SAVE is not None)
                if(self.SAVE.link is not None
                    print("{} ->".format(self.SAVE.info),end=" "
                else:
                    print("{} ".format(self.SAVE.info))
                self.SAVE=self.SAVE.l
SList = SinglyLinkedList()
while(True):
    print("Select Your Option")
    print("(1) Insert New Node At Beginning")
    print("(2) Display Linked List")
    print("(3) EXIT")
    choice = input("Enter Your Choice:")
    if(choice.isnumeric() is False):
        print("Wrong Choice Entered")
    elif(int(choice)==1):
        Value = input("Enter Element:")
        New_Node = Node(Value)
        SList.InsertBegining(New_Node)
    elif(int(choice)==2):
        SList.display()
    elif(int(choice)==3):
        break
    else:
        print("Wrong Choice Entered")
```

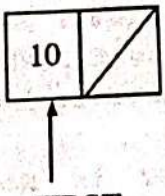
4.3.2 Insertion of a New Node at the end of Linked List :

❖ In order to insert a New Node at the end of the Linked List we have to follows the below steps:

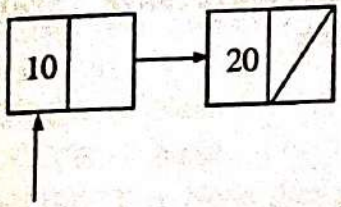
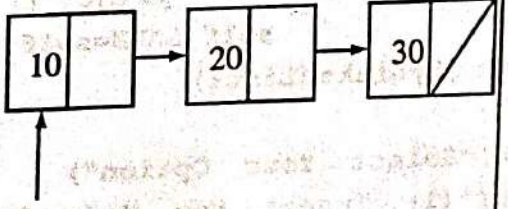
(1) First we have to Create a new node.

(2) After creating new node we have to check weather linked list is empty or not. We have two possibilities:

(A) Linked List is empty (FIRST=None). In this case the newly created node becomes the first node of linked list.

Before Insertion	After Insertion
FIRST = None	NEW 

(B) Linked List is not empty (FIRST "" None). In this case we have to traverse from first node to last node in the list and insert new node at the end of linked list.

Before Insertion	After Insertion
FIRST ≠ None 	NEW 

❖ **Algorithm to insert new node at the end of Linked List**

Step 1 : New_Node ← Node(Value)

Step 2 : If self.FIRST is None then

self.FIRST=New_Node

Else

self.SAVE ← self.FIRST

Repeat while self.SAVE.link is not None

self.SAVE ← self.SAVE.link

self.SAVE.link ← New_Node

❖ **Program to Insert New Node at the End of Linked List.**

```
class Node:
    def __init__(self,info):
```

```
            self.info=info
            self.link=None
class SinglyLinkedList:
    def __init__(self):
        self.FIRST = None
        self.PRED = None
        self.SAVE = None
    def InsertEnd(self,New_Node):
        if(self.FIRST is None):
            self.FIRST=New_Node
        else:
            self.SAVE=self.FIRST
            while (self.SAVE.link is not None):
                self.SAVE=self.SAVE.link
            self.SAVE.link=New_Node
    def display(self):
        if(self.FIRST is None):
            print("Linked List is Empty")
        else:
            self.SAVE=self.FIRST
            while(self.SAVE is not None):
                if(self.SAVE.link is not None):
                    print("{} ->".format(self.SAVE.info),end=" ")
                else:
                    print("{} ".format(self.SAVE.info))
                self.SAVE=self.SAVE.link
SList = SinglyLinkedList()
while(True):
    print("Select Your Option")
    print("(1) Insert New Node At End")
    print("(2) Display Linked List")
    print("(3) EXIT")
    choice = input("Enter Your Choice:")
    if(choice.isnumeric() is False):
        print("Wrong Choice Entered")
    elif(int(choice)==1):
        Value = input("Enter Element:")
        New_Node = Node(Value)
        SList.InsertEnd(New_Node)
    elif(int(choice)==2):
        SList.display()
    elif(int(choice)==3):
        break
    else:
        print("Wrong Choice Entered")
```

4.3.3 Insertion of a New Node after a given node in Linked List :

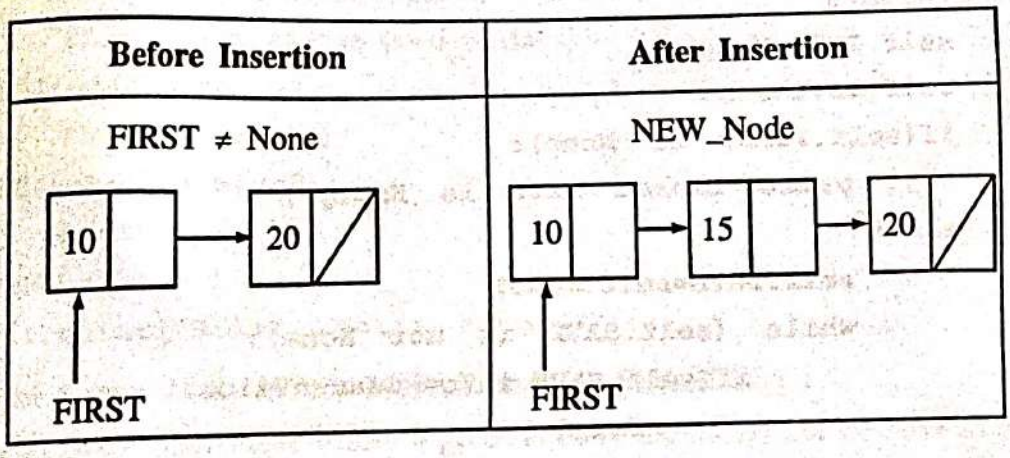
❖ In order to insert a new node after given node in the linked list we have to follows the below steps :

(1) First we have to create a new node.

(2) After creating new node we have to check weather linked list is empty or not. We have two possibilities:

(A) Linked List is empty (FIRST=None). Hence list is empty, specified node is not found in the linked list. In this case we cannot insert new node after given node.

(B) Linked List is not empty (FIRST ≠ None). In this case we have to traverse from first node to last node in the list until given node is found. If node is found in the list then we can insert new node after that node otherwise we cannot insert new node after given node. In below figure new node is inserted after node 10.



❖ **Algorithm to insert new node after given node**

```
Step 1 : New_Node ← Node(Value)
         found ← 0
         self.PRED ← None
         self.SAVE ← None
Step 2 : if self.FIRST is None then
                print("Linked List is Empty")
                return
Step 3 : self.SAVE←self.FIRST
Step 4 : Repeat while self.SAVE is not None
                if self.SAVE.info=SearchValue then
                        found=1
                        break
                else
                        self.PRED←self.SAVE
                        self.SAVE←self.SAVE.link
Step 5 : if found=1 then
                New_Node.link←self.SAVE.link
                self.SAVE.link←New_Node
         else:
                print("Node Not Found")
```

❖ Program to Insert New Node After Given Node in Linked List.

```
class Node:
    def __init__(self,info):
        self.info=info
        self.link=None
class SinglyLinkedList:
    def __init__(self):
        self.FIRST = None
        self.PRED = None
        self.SAVE = None
    def InsertAfter(self,New_Node,SearchValue):
        found=0
        self.PRED=None
        self.SAVE=None
        if(self.FIRST is None):
            print("Linked List is Empty")
        else:
            self.SAVE=self.FIRST
            while (self.SAVE is not None):
                if(self.SAVE.info==SearchValue):
                    found=1
                    break
                else:
                    self.PRED=self.SAVE
                    self.SAVE=self.SAVE.link
            if(found==1):
                New_Node.link=self.SAVE.link
                self.SAVE.link=New_Node
            else:
                print("Node Not Found")
    def display(self):
        if(self.FIRST is None):
            print("Linked List is Empty")
        else:
            self.SAVE=self.FIRST
            while(self.SAVE is not None):
                if(self.SAVE.link is not None):
                    print("{} ->".format(self.SAVE.info),end="
                else:
                    print("{} ".format(self.SAVE.info))
                    self.SAVE=self.SAVE.link
```

```
SList = SinglyLinkedList()
while(True):
    print("Select Your Option")
    print("(1) Insert New Node After Given Node")
    print("(2) Display Linked List")
    print("(3) EXIT")
    choice = input("Enter Your Choice:")
    if(choice.isnumeric() is False):
        print("Wrong Choice Entered")
    elif(int(choice)==1):
        Value = input("Enter Element:")
        New_Node = Node(Value)
        SList.InsertAfter(New_Node)
    elif(int(choice)==2):
        SList.display()
    elif(int(choice)==3):
        break
    else:
        print("Wrong Choice Entered")
```

4.3.4 Insertion of a New Node before a given node in Linked List :

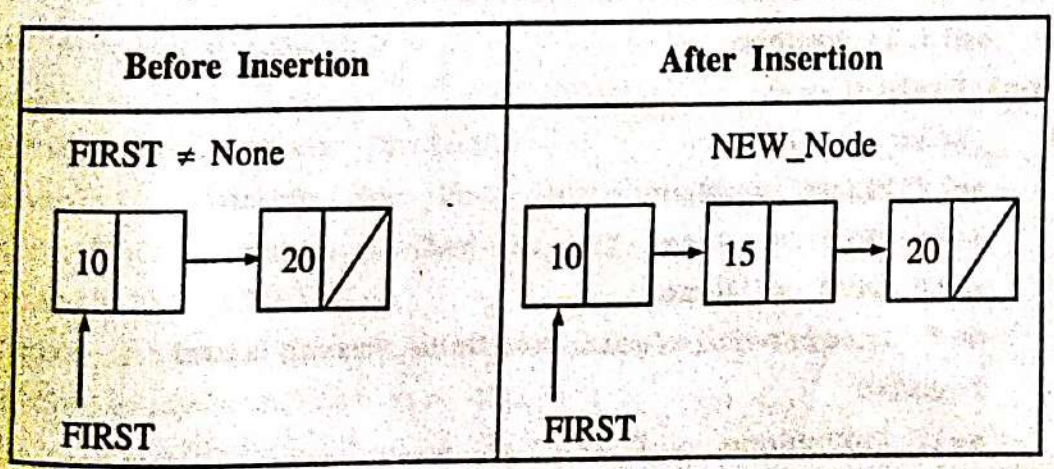
- In order to insert a new node before a given node in the linked list we have to follows the below steps :

(1) First we have to Create a new node.

(2) After creating new node we have to check weather linked list is empty or not. We have two possibilities:

(A) Linked List is empty (FIRST=None). Hence list is empty, specified node is not found in the linked list. In this case we cannot insert new node before given node.

(B) Linked List is not empty (FIRST ≠ None). In this case we have to traverse from first node to last node in the list until given node is found. If node is found in the list then we can insert new node before that node otherwise we cannot insert new node before given node. In below figure new node is inserted before node 20.



❖ **Algorithm to insert new node before given node :**

```
Step 1 :  New_Node □! Node(Value)
          found ← 0
          self.PRED ← None
          self.SAVE ← None
Step 2 :  if self.FIRST is None then
          print("Linked List is Empty")
          return
Step 3 :  self.SAVE←self.FIRST
Step 4 :  Repeat while self.SAVE is not None
          if self.SAVE.info=SearchValue then
                found=1
                break
          else
                self.PRED←self.SAVE
                self.SAVE←self.SAVE.link
Step 5 :  if found=1 then
                if self.PRED is None then
                      New_Node.link←s self.SAVE
                      self.FIRST←s New_Node
                else
                      self.PRED.link□ls New_Node
                      New_Node.link□!s self.SAVE
          else:
                print("Node Not Found")
```

❖ **Program to Insert New Node Before Given Node in Linked List.**

```
class Node:
      def __init__(self,info):
            self.info=info
            self.link=None
class SinglyLinkedList:
      def __init__(self):
            self.FIRST = None
            self.PRED = None
            self.SAVE = None
            def InsertBefore(self,New_Node,SearchValue):
            found=0
            self.PRED=None
            self.SAVE=None
```

```
            if(self.FIRST is None):
                print("Linked List is Empty")
            else:
                self.SAVE=self.FIRST
                while (self.SAVE is not None):
                    if(self.SAVE.info==SearchValue):
                        found=1
                        break
                    else:
                        self.PRED=self.SAVE
                        self.SAVE=self.SAVE.link
                if(found==1):
                    if(self.PRED is None):
                        New_Node.link=self.SAVE
                        self.FIRST=New_Node
                    else:
                        self.PRED.link=New_Node
                        New_Node.link=self.SAVE
                else:
                    print("Node Not Found")
    def display(self):
        if(self.FIRST is None):
            print("Linked List is Empty")
        else:
            self.SAVE=self.FIRST
            while(self.SAVE is not None):
                if(self.SAVE.link is not None):
                    print("{} ->".format(self.SAVE.info),end=" ")
                else:
                    print("{} ".format(self.SAVE.info))
                self.SAVE=self.SAVE.link
SList = SinglyLinkedList()
while(True):
    print("Select Your Option")
    print("(1) Insert New Node Before Given Node")
    print("(2) Display Linked List")
    print("(3) EXIT")
    choice = input("Enter Your Choice:")
    if(choice.isnumeric() is False):
        print("Wrong Choice Entered")
    elif(int(choice)==1):
```

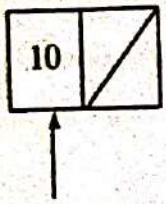
```
            Value = input("Enter Element:")
            New_Node = Node(Value)
            SList.InsertBefore(New_Node)
        elif(int(choice)==2):
            SList.display()
        elif(int(choice)==3):
            break
        else:
            print("Wrong Choice Entered")
```

4.3.5 Deleting the first node from a Linked List :

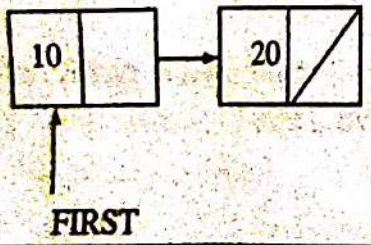
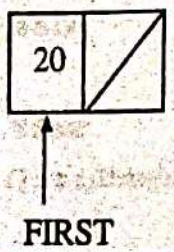
❖ In order to delete first node from linked list we have to consider three possibilities:

(A) List is Empty (FIRST = None). In this case we cannot delete node from linked list.

(B) There is only one node in the linked list (FIRST.link=None). In this case we can delete the first node and then linked list becomes empty (FIRST=None).

Before Deletion	After Deletion
	FIRST = None

(C) There are more than one nodes in the linked list. In this case we can delete the first node. After deleting the first node we have to move FIRST pointer to next node so that it can point to the newly first node in the linked list.

Before Deletion	After Deletion
	

❖ **Algorithm to delete first node from linked list**

Step 1 : if self.FIRST is None then
Write ("Linked List is Empty")
return

Step 2 : if self.FIRST.link is None then
Write (self.FIRST.info)
self.FIRST←None

```
else
        Write( self.FIRST.info)
        self.FIRST←self.FIRST.link
```

Program to Create Linked List and Delete First Node of Linked List.

```
class Node:
    def __init__(self,info):
        self.info=info
        self.link=None
class SinglyLinkedList:
    def __init__(self):
        self.FIRST = None
        self.PRED = None
        self.SAVE = None
    def InsertEnd(self,New_Node):
        if(self.FIRST is None):
            self.FIRST=New_Node
        else:
            self.SAVE=self.FIRST
            while (self.SAVE.link is not None):
                self.SAVE=self.SAVE.link
            self.SAVE.link=New_Node
    def DeleteFirst(self):
        if(self.FIRST is None):
            print("Linked List is Empty")
        elif(self.FIRST.link is None):
            print("Deleted Element is
{}".format(self.FIRST.info))
            self.FIRST=None
        else:
            print("Deleted Element is {}".format(self.FIRST.info))
            self.FIRST=self.FIRST.link
    def display(self):
        if(self.FIRST is None):
            print("Linked List is Empty")
        else:
            self.SAVE=self.FIRST
            while(self.SAVE is not None):
                if(self.SAVE.link is not None):
                    print("{} ->".format(self.SAVE.info),end=" ")
                else:
                    print("{} ".format(self.SAVE.info))
```

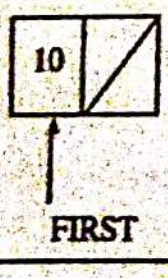
```
                self.SAVE=self.SAVE.link
SList = SinglyLinkedList()
while(True):
    print("Select Your Option")
    print("(1) Insert New Node At End")
    print("(2) Delete First Node")
    print("(3) Display Linked List")
    print("(4) EXIT")
    choice = input("Enter Your Choice:")
    if(choice.isnumeric() is False):
        print("Wrong Choice Entered")
    elif(int(choice)==1):
        Value = input("Enter Element:")
        New_Node = Node(Value)
        SList.InsertEnd(New_Node)
    elif(int(choice)==2):
        SList.DeleteFirst()
    elif(int(choice)==3):
        SList.display()
    elif(int(choice)==4):
        break
    else:
        print("Wrong Choice Entered")
```

4.3.6 Deleting the last node from a Linked List :

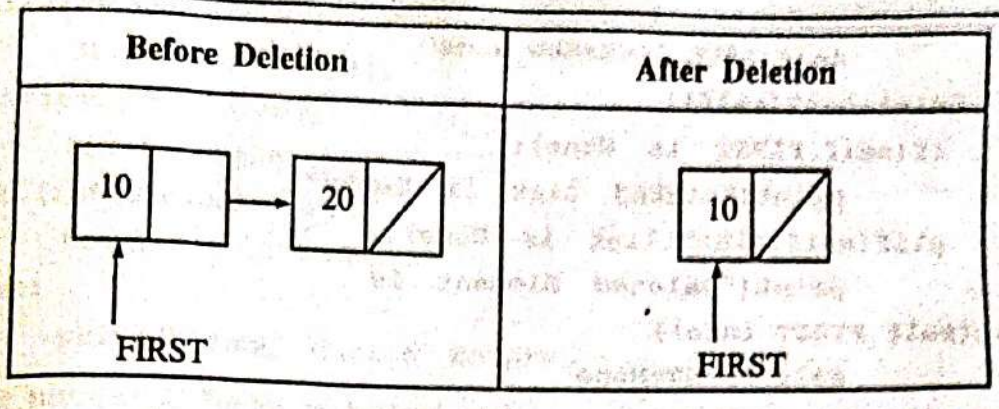
- In order to delete first node from linked list we have to consider three possibilities:

(A) List is Empty (FIRST = None). In this case we cannot delete node from linked list.

(B) There is only one node in the linked list (FIRST.link=None). In this case we can delete node and then linked list becomes empty (FIRST=None).

Before Deletion	After Deletion
	FIRST = None

(C) There are more than one nodes in the linked list. In this case we have to traverse from node to last node and then delete the last node. After deleting the last node we have to NULL value in the LINK part of the previous node.



❖ **Algorithm to delete last node from linked list :**

Step 1 : if self.FIRST is None then
Write ("Linked List is Empty")
return

Step 2 : if self.FIRST.link is None then
Write (self.FIRST.info)
self.FIRST←None
else
self.SAVE←self.FIRST
repeat while self.SAVE.link is not None
self.PRED←self.SAVE
self.SAVE←self.SAVE.link
Write (self.SAVE.info)
self.PRED.link←None

❖ **Program to Create Linked List and Delete Last Node of Linked List.**

```
class Node:
    def __init__(self,info):
        self.info=info
        self.link=None
class SinglyLinkedList:
    def __init__(self):
        self.FIRST = None
        self.PRED = None
        self.SAVE = None
    def InsertEnd(self,New_Node):
        if(self.FIRST is None):
            self.FIRST=New_Node
        else:
            self.SAVE=self.FIRST
            while (self.SAVE.link is not None):
                self.SAVE=self.SAVE.link
```

```
                self.SAVE.link=New_Node
        def DeleteLast(self):
            if(self.FIRST is None):
                print("Linked List is Empty")
            elif(self.FIRST.link is None):
                print("Deleted Element is
{}".format(self.FIRST.info))
                self.FIRST=None
            else:
                self.SAVE=self.FIRST
                while(self.SAVE.link is not None):
                    self.PRED=self.SAVE
                    self.SAVE=self.SAVE.link
                print("Deleted Element is
{}".format(self.SAVE.info))
                self.PRED.link=None
        def display(self):
            if(self.FIRST is None):
                print("Linked List is Empty")
            else:
                self.SAVE=self.FIRST
                while(self.SAVE is not None):
                    if(self.SAVE.link is not None):
                        print("{} ->".format(self.SAVE.info),end=" ")
                    else:
                        print("{} ".format(self.SAVE.info))
                    self.SAVE=self.SAVE.link
SList = SinglyLinkedList()
while(True):
    print("Select Your Option")
    print("(1) Insert New Node At End")
    print("(2) Delete Last Node")
    print("(3) Display Linked List")
    print("(4) EXIT")
    choice = input("Enter Your Choice:")
    if(choice.isnumeric() is False):
        print("Wrong Choice Entered")
    elif(int(choice)==1):
        Value = input("Enter Element:")
        New_Node = Node(Value)
        SList.InsertEnd(New_Node)
```

```
        elif(int(choice)==2):
                SList.DeleteLast()
        elif(int(choice)==3):
                SList.display()
        elif(int(choice)==4):
                break
        else:
                print("Wrong Choice Entered")
```

4.3.7 Count the number of Nodes in Linked List :

- In order to count number of nodes in linked list we have to traverse from first node to last node in the list.
- **Algorithm to count number of nodes in linked list**

Step 1 : count←0
self.SAVE←self.FIRST

Step 2 : repeat while self.SAVE is not None
self.SAVE=self.SAVE.link
count=count+1

Step 3 : return count

- **Program to Create Linked List and Count Number of Nodes in Linked List.**

```
class Node:
    def __init__(self,info):
        self.info=info
        self.link=None
class SinglyLinkedList:
    def __init__(self):
        self.FIRST = None
        self.PRED = None
        self.SAVE = None
    def InsertEnd(self,New_Node):
        if(self.FIRST is None):
            self.FIRST=New_Node
        else:
            self.SAVE=self.FIRST
            while (self.SAVE.link is not None):
                self.SAVE=self.SAVE.link
            self.SAVE.link=New_Node
    def CountNode(self):
        count=0
        self.SAVE=self.FIRST
        while(self.SAVE is not None):
```

```
                self.SAVE=self.SAVE.link
                count=count+1
            return count
        def display(self):
            if(self.FIRST is None):
                print("Linked List is Empty")
            else:
                self.SAVE=self.FIRST
                while(self.SAVE is not None):
                    if(self.SAVE.link is not None):
                        print("{} ->".format(self.SAVE.info),end=" ")
                    else:
                        print("{} ".format(self.SAVE.info))
                    self.SAVE=self.SAVE.link
SList = SinglyLinkedList()
while(True):
    print("Select Your Option")
    print("(1) Insert New Node At End")
    print("(2) Count Node In Linked List")
    print("(3) Display Linked List")
    print("(4) EXIT")
    choice = input("Enter Your Choice:")
    if(choice.isnumeric() is False):
        print("Wrong Choice Entered")
    elif(int(choice)==1):
        Value = input("Enter Element:")
        New_Node = Node(Value)
        SList.InsertEnd(New_Node)
    elif(int(choice)==2):
        SList.CountNode()
    elif(int(choice)==3):
        SList.display()
    elif(int(choice)==4):
        break
    else:
        print("Wrong Choice Entered")
```

4.3.8 Search Node in Linked List :

- In order to search particular node in a linked list we have to traverse from first node to last node in a linked list and compare the value to be search against each node in a linked list. Whenever a node is found we set the flag to indicate successful search.
- **Algorithm to search a node in Linked List**

```
Step 1 :  found←0
          self.SAVE←self.FIRST
Step 2 :  repeat while self.SAVE is not None
                if self.SAVE.info = SearchValue then
                      found=1
                      break
                else
                      self.SAVE=self.SAVE.link
Step 3 :  if found=0 then
                write ("Node Not Found")
          else
                write("Node Found")
```

♦ **Program to Create Linked List and Search Node in Linked List.**

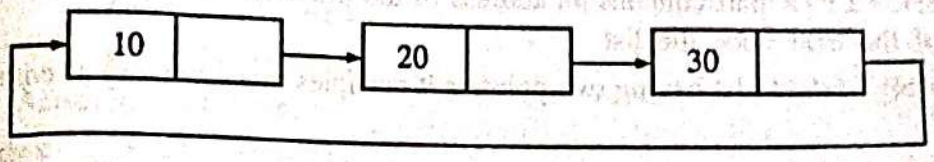
```
class Node:
    def __init__(self,info):
        self.info=info
        self.link=None
class SinglyLinkedList:
    def __init__(self):
        self.FIRST = None
        self.PRED = None
        self.SAVE = None
    def InsertEnd(self,New_Node):
        if(self.FIRST is None):
            self.FIRST=New_Node
        else:
            self.SAVE=self.FIRST
            while (self.SAVE.link is not None):
                self.SAVE=self.SAVE.link
            self.SAVE.link=New_Node
    def SearchNode(self,SearchValue):
        self.SAVE=self.FIRST
        found=0
        while(self.SAVE is not None):
            if(self.SAVE.info==SearchValue):
                found=1
                break
            else:
                self.SAVE=self.SAVE.link
        if(found==0):
```

```
                print("Node Not Found")
            else:
                print("Node Found")
        def display(self):
            if(self.FIRST is None):
                print("Linked List is Empty")
            else:
                self.SAVE=self.FIRST
                while(self.SAVE is not None):
                    if(self.SAVE.link is not None):
                        print("{} ->".format(self.SAVE.info),end=" ")
                    else:
                        print("{} ".format(self.SAVE.info))
                    self.SAVE=self.SAVE.link
SList = SinglyLinkedList()
while(True):
    print("Select Your Option")
    print("(1) Insert New Node At End")
    print("(2) Search Node In Linked List")
    print("(3) Display Linked List")
    print("(4) EXIT")
    choice = input("Enter Your Choice:")
    if(choice.isnumeric() is False):
        print("Wrong Choice Entered")
    elif(int(choice)==1):
        Value = input("Enter Element:")
        New_Node = Node(Value)
        SList.InsertEnd(New_Node)
    elif(int(choice)==2):
        SearchValue = input("Enter Search Value:")
        SList.SearchNode(SearchValue)
    elif(int(choice)==3):
        SList.display()
    elif(int(choice)==4):
        break
    else:
        print("Wrong Choice Entered")
```

4.4 Overview of circular linked list :

- Circular Singly Linked List is a collection of variable number of nodes in which each node consists of two parts. First part contains a value of the node and second part contains an address of the next node such that LINK part of the last node contains an address of the first node.

- Consider Following example in which Circular Singly Linked List consist of three nodes. Each node having INFO part and LINK part.
- INFO part contains value of the node.
- LINK part contains address of the next node in the linked list. If next node is not present then LINK part contains an address of the first node. Thus LINK part of the last node always contains an address of the first node in the list.



- In Circular Singly Linked List each node is accessible from every node in the list. But we have to take necessary care to detect end of the list otherwise processing of Circular Singly Linked List may goes into infinite loop.

4.5 Difference between circular linked list and singly linked list :

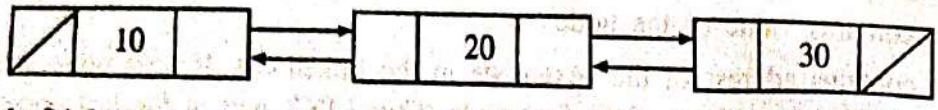
	Singly Linked List	Circular Linked List
1	In Singly Linked List LINK part of last node always contains None value.	In Circular Linked List LINK part of last node always contains an address of first node.
2	In Singly Linked List if we are at middle of the list and want to access the first node we cannot go backward in the list. Thus every node in the list is not accessible from given node.	In Circular linked list last node contains an address of the first node so every node in the list is accessible from given node.
3	In Singly Linked List LINK part of last node always contains None value so we can easily detect end of the list.	In Circular Linked List LINK part of last node always contains an address of first node. So we cannot easily detect end of the list.

4.6 Overview of doubly linked list :

- **Doubly Linked List**
- Doubly Linked List is a collection of variable number of nodes in which each node consists of three parts. First part contains an address of previous node, second part contains a value of the node and third part contains an address of the next node.
- Following figure represents a structure of the node.

- Consider Following example in which Doubly Linked List consist of three nodes. Each node having LPTR part, INFO part and RPTR part.
- INFO part contains value of the node.
- LPTR part contains an address of the previous node in the linked list. If previous node is not present then LPTR part contains NULL value. Thus LPTR part of the first node always contains None value.
- RPTR part contains an address of the next node in the linked list. If next node is not present

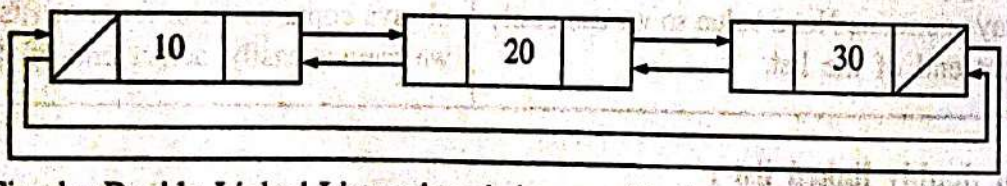
then RPTR part contains NULL value. Thus RPTR part of the last node always contains None value to indicate end of the list.



- In Doubly Linked List we can traverse in both direction forward direction as well as backward direction because LPTR part contains an address of the previous node in and RPTR part contains an address of the next node the list.
- However Doubly Linked List having two pointers it occupies more memory as compared to Singly Linked List.

Circular Doubly Linked List :

- Circular Doubly Linked List is a collection of variable number of nodes in which each node consists of three parts. First part contains an address of previous node, second part contains a value of the node and third part contains an address of the next node such that RPTR part of the last node contains an address of the first node and LPTR part of the first node contains an address of the last node in the list.
- Consider Following example in which Circular Doubly Linked List consist of three nodes. Each node having LPTR part, INFO part and RPTR part.
- INFO part contains value of the node.
- LPTR part contains an address of the previous node in the linked list. If previous node is not present then LPTR part contains an address of the last node. Thus LPTR part of the first node always contains an address of the last node in the list.
- RPTR part contains an address of the next node in the linked list. If next node is not present then RPTR part contains an address of the first node. Thus RPTR part of the last node always contains an address of the first node in the list.



- In Circular Doubly Linked List each node is accessible from every node in the list. But we have to take necessary care to detect end of the list otherwise processing of Circular Doubly Linked List may goes into infinite loop.

4.7 Applications of linked list :

- Linked lists are used as a building block for many other data structures, such as stacks and queues. Stack and Queue can be represented dynamically using Linked List.
- Linked List is used for representation and manipulation of polynomial equation.
- We can also use linked list to implement associative arrays.
- Linked List is widely used by compiler for the construction and maintenance of a linked dictionary which is commonly known as symbol table.
- Linked List is used in the multiple precision arithmetic.