Singly/Single Linked List

- Node
- head
- data
- ref/address #### Operations in LL
- Adding/Inserting
- Deleting/Removing
- Traversal
- Adding/Inserting(Insert at Begining,End and In between)
- Deleting(delete first node,last node and In between)

In [31]:

```
# Crearting a Node
class Node:
    def __init__(self,data):
        self.data=data
        self.ref=None
# Creating Linked List
class LinkedList:
    def __init__(self):
        self.head=None
# printing or traversal Operation
    def Traversal(self):
        if self.head is None:# self.head==None
            print("Linked List is Empty!!!")
        else:
            h=self.head
            while h is not None:
                print(h.data,"----->",end=" ")
                h=h.ref
# Inserting or Adding element at begin
    def add_begin(self,data):
        new_node=Node(data) # creating a new node
        new_node.ref=self.head # head ref is pointing to new ref
        self.head=new_node # link to new node from head
    def add_end(self,data):
        new_node=Node(data) # Creating a new Node
        if self.head is None: # Before adding element at end,first check wether the LL
 is empty or not?
            self.head=new_node # new_ node will be assigned to head
        else:
            h=self.head
            while h.ref is not None: # Moving to last node
                h=h.ref
            h.ref=new_node # new node address is stored in last node ref
# In -between Insertion
# After a Node ---insert an element
    def add_after(self,data,x):
        h=self.head
        while h is not None:
            if x==h.data:
                break
            h=h.ref
        if h is None:
            print("Node is Not Present in Linked List!!!")
        else:
            new_node=Node(data) # Creating a new node for ninsertion
            new_node.ref=h.ref # New node ref is pointing to the next node
            h.ref=new_node # new_node ref will copied prev(x) node ref
# Before a node ----insert an element
    def add_before(self,data,x):
        if self.head is None:
            print("Linked List is Empty You can't Insert!!!")
            return
        if self.head.data==x:
            new_node=Node(data) # creating a new node
            new_node.ref=self.head # head ref is pointing to new ref
            self.head=new_node # link to new node from head
            return
        h=self.head
        while h.ref is not None:
            if h.ref.data==x:
                break
            h=h.ref
        if h.ref is None:
            print("Node is Not Found to insert an element!!!")
        else:
            new_node=Node(data)
            new_node.ref=h.ref
            h.ref=new_node
# Deletion ---delete an first node
    def del_begin(self):
        if self.head is None:
            print("Linked List is Empty you can't delete!!!!")
        else:
            self.head=self.head.ref
# Delete a node at end
    def del_end(self):
        if self.head is None:
            print("Linked List is Empty you can't delete!!!!")
        else:
            h=self.head
            while h.ref.ref is not None:
                h=h.ref
            h.ref=None
# delete  by a value
    def del_value(self,x):
        if self.head is None:
            print("Linked List is Empty you can't delete!!!!")
            return
        if x==self.head.data:
            self.head=self.head.ref
        else:
            h=self.head
            while h.ref is not None:
                if x==h.ref.data:
                    break
                h=h.ref
            if h.ref is None:
                print("Node is Not Found!!!")
            else:
                h.ref=h.ref.ref







ll=LinkedList()
ll.add_begin(20)
ll.add_begin(10)
ll.add_end(100)
ll.add_after(1000,10)
ll.add_after(2000,20)
ll.add_before(500,10)
ll.add_before(999,1000)
ll.del_begin()
ll.del_end()
ll.del_value(20)
ll.Traversal()
```

```
10 -----> 999 -----> 1000 -----> 2000 ----->
```

Double/Double Linked List

A Doubly Linked List (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly linked list.

```
In [ ]:
```

In [6]:

```
# A linked list node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

# Class to create a Doubly Linked List
class DoublyLinkedList:

    # Constructor for empty Doubly Linked List
    def __init__(self):
        self.head = None

    # Given a reference to the head of a list and an
    # integer, inserts a new node on the front of list
    def push(self, new_data):

        # 1. Allocates node
        # 2. Put the data in it
        new_node = Node(new_data)

        # 3. Make next of new node as head and
        # previous as None (already None)
        new_node.next = self.head

        # 4. change prev of head node to new_node
        if self.head is not None:
            self.head.prev = new_node

        # 5. move the head to point to the new node
        self.head = new_node

    # Given a node as prev_node, insert a new node after
    # the given node
    def insertAfter(self, prev_node, new_data):

        # 1. Check if the given prev_node is None
        if prev_node is None:
            print ("the given previous node cannot be NULL")
            return

        # 2. allocate new node
        # 3. put in the data
        new_node = Node(new_data)

        # 4. Make net of new node as next of prev node
        new_node.next = prev_node.next

        # 5. Make prev_node as previous of new_node
        prev_node.next = new_node

        # 6. Make prev_node ass previous of new_node
        new_node.prev = prev_node

        # 7. Change previous of new_nodes's next node
        if new_node.next is not None:
            new_node.next.prev = new_node

    # Given a reference to the head of DLL and integer,
    # appends a new node at the end
    def append(self, new_data):

        # 1. Allocates node
        # 2. Put in the data
        new_node = Node(new_data)

        # 3. This new node is going to be the last node,
        # so make next of it as None
        new_node.next = None

        # 4. If the Linked List is empty, then make the
        # new node as head
        if self.head is None:
            new_node.prev = None
            self.head = new_node
            return

        # 5. Else traverse till the last node
        last = self.head
        while(last.next is not None):
            last = last.next

        # 6. Change the next of last node
        last.next = new_node

        # 7. Make last node as previous of new node
        new_node.prev = last

        return

    # This function prints contents of linked list
    # starting from the given node
    def printList(self, node):

        print( "\nTraversal in forward direction")
        while(node is not None):
            print (" % d" %(node.data), )
            last = node
            node = node.next

        print ("\nTraversal in reverse direction")
        while(last is not None):
            print (" %d " %(last.data), )
            last = last.prev

# Driver program to test above functions

# Start with empty list
llist = DoublyLinkedList()

# Insert 6. So the list becomes 6->None
llist.append(6)

# Insert 7 at the beginning.
# So linked list becomes 7->6->None
llist.push(7)
```

```
# Insert 1 at the beginning.
# So linked list becomes 1->7->6->None
llist.push(1)

# Insert 4 at the end.
# So linked list becomes 1->7->6->4->None
llist.append(4)

# Insert 8, after 7.
# So linked list becomes 1->7->8->6->4->None
llist.insertAfter(llist.head.next, 8)

print( "Created DLL is: ", )
llist.printList(llist.head)
```

```
Created DLL is: 

Traversal in forward direction
  1
  7
  8
  6
  4

Traversal in reverse direction
 4
 6
 8
 7
 1
```

In [ ]:

```

```