Linked list

Creating a linked list

```
class node:
    def __init__(self,data):
        self.data=data  # create a node
        self.next=None

class linkedlist:
    def __init__(self):  # initializing the header
        self.head=None

llist=linkedlist()
llist.head=node(30)
second=node(20)
third=node(10)
llist.head.next=second
second.next=third
```

Printing a linked list

```
class node:
    def __init__(self,data):
        self.data=data  # create a node
        self.next=None

class linkedlist:
    def __init__(self):  # initializing the header
        self.head=None
    def print_ll(self):
        if self.head==None:
            print("list is empty")
        else:
            t=self.head
            while(t!=None):
                print(t.data)
                t=t.next

llist=linkedlist()
llist.head=node(30)
second=node(20)
third=node(10)
fouth=node(100)
llist.head.next=second
second.next=third
third.next=fouth
```

```
llist.print_ll()
```

```
30
20
10
100
```

insertion at beginning of linked list

```
class Node:
    def __init__(self,data):
        self.data=data  # create a node
        self.next=None

class linkedlist:
    def __init__(self):  # initializing the header
        self.head=None
    def print_ll(self):
        if self.head==None:
            print("list is empty")
        else:
            t=self.head
            while(t!=None):
                print(t.data)
                t=t.next
    def add_beg(self,data): # function for adding node at begening of 
the linked list
        new_node=Node(data)
        new_node.next=self.head # saving the address of header in new 
node
        self.head=new_node  #saving the address of new_node in 
self.header


llist=linkedlist()
llist.head=Node(30)
second=Node(20)
third=Node(10)
fouth=Node(100)
llist.head.next=second
second.next=third
third.next=fouth

llist.add_beg(39) # adding new node

llist.print_ll()
```

```
39
30
20
10
100
```

insertion at the end of linked list

```python
class node:
    def __init__(self,data):
        self.data=data  # create a node
        self.next=None

class linkedlist:
    def __init__(self):  # initializing the header
        self.head=None
    def print_ll(self):
        if self.head==None:
            print("list is empty")
        else:
            t=self.head
            while(t!=None):
                print(t.data)
                t=t.next
    def add_end(self,data): # function for adding node at the end of
the linked list
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.next=new_node

llist=linkedlist()
llist.head=node(30)
second=node(20)
third=node(10)
fouth=node(100)
llist.head.next=second
second.next=third
third.next=fouth
llist.add_end(50)  # inseting element at the end of linked list
llist.print_ll()
```

```
30
20
10
100
50
```

Insertion After Node

```python
class node:
    def __init__(self,data):
        self.data=data  # create a node
        self.next=None

class linkedlist:
    def __init__(self):  # initializing the header
        self.head=None
    def print_ll(self):
        if self.head==None:
            print("list is empty")
        else:
            t=self.head
            while(t!=None):
                print(t.data)
                t=t.next

    def after_node(self,data,x):    # function for adding node after
the selected node in linkedlist
        new_node=node(data)
        temp=self.head
        while(temp!=None):
            if(temp.data==x):
                break
            temp=temp.next
        if temp is None:
            print("value not found")
        else:
            new_node.next=temp.next
            temp.next=new_node

llist=linkedlist()
llist.head=node(30)
second=node(20)
third=node(10)
fouth=node(100)
llist.head.next=second
second.next=third
third.next=fouth
llist.after_node(50,20) #insertion after the given node(which is 20)
and new node (50) will be added after that
llist.print_ll()
```

```
30
20
50
10
100
```

insertion before the given node in linked list

```
class node:
    def __init__(self,data):
        self.data=data  # create a node
        self.next=None

class linkedlist:
    def __init__(self):  # initializing the header
        self.head=None
    def print_ll(self):  # function for printing the linked list
        if self.head==None:
            print("list is empty")
        else:
            t=self.head
            while(t!=None):
                print(t.data)
                t=t.next
    def before_node(self,data,x): # function for adding node before
the selected node in linkedlist
        if self.head==None:
            print("linked list is empty")
            return
        if self.head.data==x:
            new_node=node(data)
            new_node.next=self.head
            self.head=new_node

        temp=self.head
        while(temp.next!=None):
            if temp.next.data==x:
                break
            temp=temp.next
        if temp.next is None:
            print("Node not found")
        else:
            new_node=node(data)
            new_node.next=temp.next
            temp.next=new_node

llist=linkedlist()
llist.head=node(30)
second=node(20)
#third=node(10)
#fouth=node(100)
llist.head.next=second
#second.next=third
#third.next=fouth
llist.before_node(60,20)
llist.print_ll()
```

```
30
60
20
```

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None

class linkedlist:
    def __init__(self):
        self.head=None
    def print1(self):
        if (self.head==None):
            print("The linked list is Empty")
        else:
            t=self.head
            while(t!=None):
                print(t.data)
                t=t.next

    def add_beg(self,data):
        new_node=Node(data)
        new_node.next=self.head
        self.head=new_node

    def add_end(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.next=new_node


llist=linkedlist()
llist.head=Node(40)
second=Node(60)
third=Node(50)
fourth=Node(110)
fifth=Node(190)
llist.head.next=second
second.next=third
third.next=fourth
fourth.next=fifth
llist.add_beg(50)
llist.add_end(46)
llist.print1()
```

```
50
40
60
50
110
190
46
```

Reverse of Linked List

```
class node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def printll(self):
        if self.head==None:
            print("Linked list in empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def reverse_ll(self):
        prev=None
        temp=self.head
        while(temp!=None):
            next=temp.next
            temp.next=prev
            prev=temp
            temp=next
        self.head=prev

ll=linkedlist()
ll.head=node(10)
second=node(20)
third=node(30)
fourth=node(40)
ll.head.next=second
second.next=third
third.next=fourth
ll.printll()
ll.reverse_ll()
print("The reverse linked list is")
ll.printll()
```

```
10
20
30
40
The reverse linked list is
40
30
20
10
```

Deletion of beginning node in linked list

```python
class node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def printll(self):
        if self.head==None:
            print("Linked list in empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def delete_beg(self):  #function for delection of first node in 
linkedlist
        if self.head==None:
            print("linked list is empty")
        else:
            self.head=self.head.next

ll=linkedlist()
ll.head=node(56)
second=node(86)
#third=node(67)
ll.head.next=second
#second.next=third
ll.printll()
ll.delete_beg()
print("linked list after deletion at beginning")
ll.printll()
```

```
56
86
linked list after deletion at beginning
86
```

Deletion of end node of linked list

```
class node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def printll(self):
        if self.head==None:
            print("Linked list in empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def del_atend(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp.next.next!=None):
                temp=temp.next
            temp.next=None
ll=linkedlist()
ll.head=node(10)
second=node(20)
ll.head.next=second
ll.printll()
print("linked list after deletion of end node")
ll.del_atend()
ll.printll()
```

```
10
20
linked list after deletion of end node
10
```

Deletion by value in linked list - done by me

```
class node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def printll(self):
        if self.head==None:
            print("Linked list in empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def del_byvalue(self,x):
        if self.head==None:
            print("Linked list is empty")

        else:
            temp=self.head
            while(temp.next.data!=x):
                temp=temp.next
            temp.next=temp.next.next

ll=linkedlist()
ll.head=node(10)
second=node(20)
third=node(30)
ll.head.next=second
second.next=third
ll.printll()
ll.del_byvalue(20)
print("linked list after deletion of node")
ll.printll()
```

```
10
20
30
linked list after deletion of node
10
30
```

Deletion by value of node in linkedlist

```
class node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def printll(self):
        if self.head==None:
            print("Linked list in empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def del_byvalue(self,x):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp.next!=None):
                if temp.next.data==x:
                    break
                temp=temp.next
            if temp.next==None:
                print("Node of value {} is not found".format(x))
            else:
                temp.next=temp.next.next

ll=linkedlist()
ll.head=node(10)
second=node(20)
third=node(30)
ll.head.next=second
second.next=third
ll.printll()
ll.del_byvalue(20)
#ll.del_byvalue(50) # as 50 is not present in linked list it will
print not found
print("linked list after deletion of node")
ll.printll()
```

```
10
20
30
linked list after deletion of node
10
30
```

Creating a linked list using function in linked list class

```
class Node:
    def __init__(self):
        self.data=None
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None

    def printll(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def create(self):
        self.head=None

        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break



ll=linkedlist()
ll.create()
ll.printll()
```

```
Enter the value: 4
Enter the char(Y:yes and N:no): y
Enter the value: 12
Enter the char(Y:yes and N:no): y
Enter the value: 43
Enter the char(Y:yes and N:no): n
4
12
43
```

Doubly linked list

Doubly linked list is a two way linked list - three blocks are prev,data,next

creating a doubly linked list

```
class Node:
    def __init__(self):
        self.prev=None
        self.data=None
        self.next=None

class doubly_linked_list:
    def __init__(self):
        self.head=None

    def printll(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next

    def create(self):
        self.head=None
        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
                new_node.prev=temp
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break

ll=doubly_linked_list()
ll.create()
ll.printll()
```

```
Enter the value: 4
Enter the char(Y:yes and N:no): y
Enter the value: 6
Enter the char(Y:yes and N:no): 7
Enter the value: 8
Enter the char(Y:yes and N:no): y
Enter the value: 8
Enter the char(Y:yes and N:no): n
4
6
8
8
```

insertion at beg in doubly linked list

```
class Node:
    def __init__(self):
        self.prev=None
        self.data=None
        self.next=None

class doubly_linked_list:
    def __init__(self):
        self.head=None

    def printll(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def create(self):
        self.head=None
        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
                new_node.prev=temp
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break
    def insert_atbeg(self,data):
        new_node=Node()
        new_node.data=data
        if self.head==None:
            self.head=new_node
        else:
            new_node.next=self.head
            self.head.prev=new_node
            self.head=new_node

# when the linkedlist is empty
print("insertion at beginning when the linked list is empty")
ll=doubly_linked_list()
ll.insert_atbeg(34)
ll.printll()

# when the linked list is not empty
print("the linkedlist is ")
ll1=doubly_linked_list()
ll1.create()
ll1.printll()
print("linked list after insertion at beginning")
ll1.insert_atbeg(10)
ll1.printll()
```

```
insertion at beginning when the linked list is empty
34
the linkedlist is
Enter the value: 45
Enter the char(Y:yes and N:no): y
Enter the value: 90
Enter the char(Y:yes and N:no): n
45
90
linked list after insertion at beginning
10
45
90
```

Insertion at the end of doubly linked list

```python
class Node:
    def __init__(self):
        self.prev=None
        self.data=None
        self.next=None

class doubly_linked_list:
    def __init__(self):
        self.head=None

    def printll(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def create(self):
        self.head=None
        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
                new_node.prev=temp
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break
    def insert_atend(self,data):
        new_node=Node()
        new_node.data=data
        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            new_node.prev=temp
            temp.next=new_node

ll=doubly_linked_list()
print("when the linked list is empty")
ll.insert_atend(78)
ll.printll()

ll1=doubly_linked_list()
ll1.create()
print("the linked list is: ")
ll1.printll()

ll1.insert_atend(900)
print("the linked list after insertion at end: ")

ll1.printll()
```

```
when the linked list is empty
78
Enter the value: 100
Enter the char(Y:yes and N:no): y
Enter the value: 500
Enter the char(Y:yes and N:no): y
Enter the value: 700
Enter the char(Y:yes and N:no): n
the linked list is:
100
500
700
the linked list after insertion at end:
100
500
700
900
```

Insertion after Node

```
class Node:
    def __init__(self):
        self.prev=None
        self.data=None
        self.next=None
```

```
class doubly_linked_list:
    def __init__(self):
        self.head=None

    def printll(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def create(self):
        self.head=None
        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
                new_node.prev=temp
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break
    def insert_after(self,data,x): #function for insertion after node 
of data=x
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                if temp.data==x:
                    break
                temp=temp.next
            if temp==None:
                print("Node not found")
            else:
                new_node=Node()
                new_node.data=data
                new_node.next=temp.next
                new_node.prev=temp
                if temp.next!=None:
                    temp.next.prev=new_node
                temp.next=new_node
```

```
ll=doubly_linked_list()
ll.create()
print("the linked list is :")
ll.printll()

print("linked list after insertion after node")
ll.insert_after(99,40)
ll.printll()
```

```
Enter the value: 10
Enter the char(Y:yes and N:no): y
Enter the value: 20
Enter the char(Y:yes and N:no): 40
Enter the value: 40
Enter the char(Y:yes and N:no): y
Enter the value: 50
Enter the char(Y:yes and N:no): n
the linked list is :
10
20
40
50
linked list after insertion after node
10
20
40
99
50
```

Insertion before Node

```
class Node:
    def __init__(self):
        self.prev=None
        self.data=None
        self.next=None

class doubly_linked_list:
    def __init__(self):
        self.head=None

    def printll(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def create(self):
        self.head=None
        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
                new_node.prev=temp
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break
    def insert_before(self,data,x): #function for insertion before
node of data=x
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                if temp.data==x:
                    break
                temp=temp.next
            if temp==None:
                print("Node not found")
            else:
                new_node=Node()
                new_node.data=data
                new_node.next=temp
                new_node.prev=temp.prev
                if temp.prev!=None:
                    temp.prev.next=new_node

                temp.prev=new_node



ll=doubly_linked_list()
ll.create()
print("the linked list is :")
ll.printll()

print("linked list after insertion before node")
ll.insert_before(99,40)
ll.printll()
```

```
Enter the value: 10
Enter the char(Y:yes and N:no): y
Enter the value: 40
Enter the char(Y:yes and N:no): y
Enter the value: 50
Enter the char(Y:yes and N:no): n
the linked list is :
10
40
50
linked list after insertion before node
10
99
40
50
```

Deletion at beginning in doubly linkedlist

```python
class Node:
    def __init__(self):
        self.prev=None
        self.data=None
        self.next=None

class doubly_linked_list:
    def __init__(self):
        self.head=None

    def printll(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def create(self):
        self.head=None
        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
                new_node.prev=temp
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break
    def del_atbeg(self):
        if self.head==None:
            print("Linked list is empty")
            return
        if self.head.next==None:
            self.head=None
        else:
            self.head=self.head.next
            self.head.prev=None


ll=doubly_linked_list()
print("when linked list is empty")
ll.del_atbeg()


ll1=doubly_linked_list()
ll1.create()
print("linked list is :")
ll1.printll()
ll1.del_atbeg()
print("linked list after deletion at beginning:")
ll1.printll()
```

```
when linked list is empty
Linked list is empty
Enter the value: 4
Enter the char(Y:yes and N:no): y
Enter the value: 5
Enter the char(Y:yes and N:no): y
Enter the value: 6
Enter the char(Y:yes and N:no): y
Enter the value: 8
Enter the char(Y:yes and N:no): n
linked list is :
4
5
6
8
linked list after deletion at beginning:
5
6
8
```

deletion at end in linked list

```
class Node:
    def __init__(self):
        self.prev=None
        self.data=None
        self.next=None

class doubly_linked_list:
    def __init__(self):
        self.head=None
    def printll(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def create(self):
        self.head=None
        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
                new_node.prev=temp
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break

    def del_atend(self):
        if self.head==None:
            print("Linked list is empty")
            return
        if self.head.next==None:
            self.head=None
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.prev.next=None

ll=doubly_linked_list()
ll.create()
print("The linked list is:")
ll.printll()
ll.del_atend()
print("the linked list after deletion at end:")
ll.printll()
```

```
Enter the value: 4
Enter the char(Y:yes and N:no): y
Enter the value: 5
Enter the char(Y:yes and N:no): y
Enter the value: 34
Enter the char(Y:yes and N:no): y
Enter the value: 90
Enter the char(Y:yes and N:no): n
The linked list is:
4
5
34
90
the linked list after deletion at end:
4
5
34
```

deletion by value of node in linked list

```python
class Node:
    def __init__(self):
        self.prev=None
        self.data=None
        self.next=None

class doubly_linked_list:
    def __init__(self):
        self.head=None

    def printll(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def create(self):
        self.head=None
        while True:
            new_node=Node()
            new_node.data=int(input("Enter the value: "))
            if self.head==None:
                self.head=new_node
            else:
                temp.next=new_node
                new_node.prev=temp
            temp=new_node
            ch=input("Enter the char(Y:yes and N:no): ")
            if (ch=="N" or ch=="n"):
                break
    def del_atvalue(self,x):
        if self.head==None:
            print("Linked list is empty")
            return
        if self.head.next==None:
            if self.head.data==x:
                self.head=None
            else:
                print("Node not found")
            return
        if self.head.data==x:
            self.head=self.head.next
            self.head.prev=None
            return
        temp=self.head
        while(temp.next!=None):
            if temp.data==x:
                break
            temp=temp.next
        if temp.data!=x:
                print("Node of value {} is not found".format(x))
                return
        else:
            if temp.data==x and temp.next==None:
                temp.prev.next=None
                temp.prev=None
            else:
                temp.prev.next=temp.next
                temp.next.prev=temp.prev
print("Case 1:  when the linked list is empty")
ll=doubly_linked_list()
ll.del_atvalue(10)

print("case 2: when there is only one node")
ll1=doubly_linked_list()
ll1.create()
print("linked list is :")
ll1.printll()
ll1.del_atvalue(10)
print('Linked list after deletion')
ll1.printll()

print("case 3 when the value is found at first node")
ll2=doubly_linked_list()
ll2.create()
print("the linked list is :")
ll2.printll()
ll2.del_atvalue(10)
print("linked list after deletion:")
ll2.printll()

print("case 4 : when the is the mid of linked list")
ll3=doubly_linked_list()
```

```
ll3.create()
print("linked list is :")
ll3.printll()
ll3.del_atvalue(40)
print("linked list after deletion ")
ll3.printll()

print("case 5: when the value is at end of the linked list: ")
ll4=doubly_linked_list()
ll4.create()
print("the linked list is :")
ll4.printll()
ll4.del_atvalue(50)
print("the linked list after deletion :")
ll4.printll()
```

```
Case 1:  when the linked list is empty
Linked list is empty
case 2: when there is only one node
Enter the value: 10
Enter the char(Y:yes and N:no): n
linked list is :
10
Linked list after deletion
Linked list is empty
case 3 when the value is found at first node
Enter the value: 10
Enter the char(Y:yes and N:no): y
Enter the value: 20
Enter the char(Y:yes and N:no): y
Enter the value: 30
Enter the char(Y:yes and N:no): n
the linked list is :
10
20
30
linked list after deletion:
20
30
case 4 : when the is the mid of linked list
Enter the value: 10
Enter the char(Y:yes and N:no): y
Enter the value: 20
Enter the char(Y:yes and N:no): y
Enter the value: 30
Enter the char(Y:yes and N:no): y
Enter the value: 40
Enter the char(Y:yes and N:no): y
Enter the value: 50
Enter the char(Y:yes and N:no): n
linked list is :
10
20
30
40
50
linked list after deletion
10
20
30
50
case 5: when the value is at end of the linked list:
Enter the value: 10
Enter the char(Y:yes and N:no): y
Enter the value: 30
Enter the char(Y:yes and N:no): y
Enter the value: 40
Enter the char(Y:yes and N:no): y
Enter the value: 50
Enter the char(Y:yes and N:no): n
the linked list is :
10
30
40
50
the linked list after deletion :
10
30
40
```

```python
ll4=doubly_linked_list()
ll4.create()
print("the linked list is :")
ll4.printll()
ll4.del_atvalue(60)
print("the linked list after deletion :")
ll4.printll()
```

```
Enter the value: 10
Enter the char(Y:yes and N:no): t
Enter the value: 50
Enter the char(Y:yes and N:no): n
the linked list is :
10
50
Node of value 60 is not found
the linked list after deletion :
```

```
10
50
```

```python
ll4=doubly_linked_list()
ll4.create()
print("the linked list is :")
ll4.printll()
ll4.del_atvalue(60)
print("the linked list after deletion :")
ll4.printll()
```

```
Enter the value: 10
Enter the char(Y:yes and N:no): y
Enter the value: 50
Enter the char(Y:yes and N:no): y
Enter the value: 60
Enter the char(Y:yes and N:no): n
the linked list is :
10
50
60
the linked list after deletion :
10
50
```

```python
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
        self.prev=None
class Llist:
    def __init__(self):
        self.head=None
        self.tail=None
    def add(self,data):
        nnode=Node(data)
        if self.head:
            current=self.head
            while current.next:
                current=current.next
            current.next=nnode
            nnode.prev=current.next
            nnode.next=None
        else:
            self.head=self.tail=nnode
            nnode.next=nnode.prev=None
    def search(self,s):
        current=self.head
        count=1
        while current:
            if s==current.data:
                print("Node is present in the list at the 
position :",count)
                break
            else:
                count+=1
                current=current.next
        else:
            print("Node is not present in the list")
    def display(self):
        if self.head:
            current=self.head
            while current:
                print(current.data)
                current=current.next
        else:
            print("List is empty!!!")
l=Llist()
while True:
    n=int(input("Select a operation:\n1.Insertion\n2.Searching\
n3.Display\n4.Quit\t"))
    if n==1:
        d=int(input("Enter Element "))
        l.add(d)
    elif n==2:
        s=int(input("Enter Element to Search "))
        l.search(s)
    elif n==3:
        l.display()
    elif n==4:
        break
    else:
        print("Invalid Input!!!")
```

```
Select a operation:
1.Insertion
2.Searching
3.Display
4.Quit     4
```

Circular linked list

creating a circular list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class cll:
    def __init__(self):
        self.head=None
    def create(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
            new_node.next=new_node
        else:
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            temp.next=new_node
            new_node.next=self.head
    def display(self):
        if self.head==None:
            print("cll is empty")
        else:
            temp=self.head
            print(temp.data)
            while(temp.next!=self.head ):
                temp=temp.next
                print(temp.data,end=" ")


cl=cll()
cl.create(40)
cl.create(50)
cl.create(50)
cl.create(50)
cl.create(50)
cl.create(10)

cl.display()
```

```
40
50 50 50 50 10
```

Insertion at beginning in circular linked list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class cll:
    def __init__(self):
        self.head=None
    def create(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
            new_node.next=new_node
        else:
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            temp.next=new_node
            new_node.next=self.head
    def display(self):
        if self.head==None:
            print("cll is empty")
        else:
            temp=self.head
            print(temp.data)
            while(temp.next!=self.head ):
                temp=temp.next
                print(temp.data,end=" ")

    def add_beg(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
            new_node.next=new_node
        else:
            new_node.next=self.head
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            temp.next=new_node
            self.head=new_node


cl=cll()
cl.create(40)
cl.create(50)

cl.display()
print()
print("circular linked list after insertion at beginning ")
cl.add_beg(20)
cl.display()
```

```
40
50
circular linked list after insertion at beginning
20
40 50
```

Insertion at end of the circular linked list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
```

```python
class cll:
    def __init__(self):
        self.head=None
    def create(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
            new_node.next=new_node
        else:
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            temp.next=new_node
            new_node.next=self.head
    def display(self):
        if self.head==None:
            print("cll is empty")
        else:
            temp=self.head
            print(temp.data)
            while(temp.next!=self.head ):
                temp=temp.next
                print(temp.data,end=" ")

    def add_end(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
            new_node.next=self.head
            return
        else:
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            temp.next=new_node
            new_node.next=self.head


cl=cll()
cl.create(40)
cl.create(50)
cl.create(50)
cl.create(50)
cl.create(50)
cl.create(10)

cl.display()
print()
print("circular linked list after insertion at end")
```

```
cl.add_end(178)
cl.display()
```

```
40
50 50 50 50 10
circular linked list after insertion at end
40
50 50 50 50 10 178
```

deletion at beginning in circular linked list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class cll:
    def __init__(self):
        self.head=None
    def create(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
            new_node.next=new_node
        else:
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            temp.next=new_node
            new_node.next=self.head
    def display(self):
        if self.head==None:
            print("cll is empty")
        else:
            temp=self.head
            print(temp.data)
            while(temp.next!=self.head ):
                temp=temp.next
                print(temp.data,end=" ")

    def del_atbeg(self):
        if self.head==None:
            print("circular linked list is empty")
            return
        if self.head.next==self.head:
            self.head=None
            return
        else:
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            self.head=self.head.next
            temp.next=self.head


cl=cll()
cl.create(40)
cl.create(50)
cl.create(50)
cl.create(50)
cl.create(50)
cl.create(10)

cl.display()
print()
print("linked list after deletion at beg")
cl.del_atbeg()
cl.display()
```

```
40
50 50 50 50 10
linked list after deletion at beg
50
50 50 50 10
```

Deletion at end in circular linked list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class cll:
    def __init__(self):
        self.head=None
    def create(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
            new_node.next=new_node
        else:
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            temp.next=new_node
            new_node.next=self.head
    def display(self):
        if self.head==None:
            print("cll is empty")
        else:
            temp=self.head
            print(temp.data)
            while(temp.next!=self.head ):
                temp=temp.next
                print(temp.data,end=" ")
        print()

    def del_atend(self):
        if self.head==None:
            print("circular linked list is empty")
            return
        if self.head.next==self.head:
            self.head=None
            return
        else:
            temp=self.head
            while(temp.next.next!=self.head):
                temp=temp.next
            temp.next=self.head


cl=cll()
cl.create(40)
cl.create(50)
cl.create(50)
cl.create(50)
cl.create(50)
cl.create(10)


print("circular linked")
cl.display()
cl.del_atend()
print("after deletion at the end")
cl.display()
```

```
circular linked
40
50 50 50 50 10
after deletion at the end
40
50 50 50 50
```

Deletion at position in circular linked list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class cll:
    def __init__(self):
        self.head=None
    def create(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
            new_node.next=new_node
        else:
            temp=self.head
            while(temp.next!=self.head):
                temp=temp.next
            temp.next=new_node
            new_node.next=self.head
    def display(self):
        if self.head==None:
            print("cll is empty")
        else:
            temp=self.head
            print(temp.data)
            while(temp.next!=self.head ):
                temp=temp.next
                print(temp.data,end=" ")

    def del_atposition(self,position):
        temp=self.head
        count=0
        while(temp.next!=self.head):
            count+=1
            temp=temp.next
        if (position>1 and position >count):
            print("Invalid position")
            return
        """elif(position==):
            while(temp.next.next!=self.head):
                temp=temp.next
            temp.next=self.head"""

        elif(position ==1):
            if self.head.next==self.head:
                self.head=None
            else:
                while(temp.next!=self.head):
                    temp=temp.next
                self.head=self.head.next
                temp.next=self.head
        else:
            i=1
            while(i<position):
                temp=temp.next
                i+=1
            temp.next=temp.next.next
```

```
cl=cll()
cl.create(40)
cl.create(60)
cl.create(90)
cl.create(80)
cl.create(30)
cl.create(10)

cl.display()
print()
print("circular linked list after deletion at position ")
cl.del_atposition(4)
cl.display()
```

```
  File "<ipython-input-5-b2153c237fce>", line 43
    elif(position ==1):
    ^
SyntaxError: invalid syntax
```

Code tantra questions

question 1

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None

    def insert(self,data):
        n=Node(data)
        if self.head==None:
            self.head=n
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.next=n
    def reverse(self):
        prev=None
        temp=self.head
        while(temp!=None):
            next=temp.next
            temp.next=prev
            prev=temp
            temp=next
        self.head=prev

    def display(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
ll=linkedlist()
while True:
    a=int(input("Select a Operation: 1.Insertion 2.Display 3.Quit "))

    if a==1:
        b=int(input("Enter element "))
        ll.insert(b)
    elif a==2:
        ll.reverse()
        print("The Inserted elements at the front end are :")
        ll.display()
    elif a==3:
        break
    else:
        print("Invalid Option!!!")
```

```
Select a Operation: 1.Insertion 2.Display 3.Quit 1
Enter element 23
Select a Operation: 1.Insertion 2.Display 3.Quit 1
Enter element 34
Select a Operation: 1.Insertion 2.Display 3.Quit 4
Invalid Option!!!
Select a Operation: 1.Insertion 2.Display 3.Quit 1
Enter element 54
Select a Operation: 1.Insertion 2.Display 3.Quit 2
The Inserted elements at the front end are :
54
34
23
Select a Operation: 1.Insertion 2.Display 3.Quit 3
```

```
a=int(input("Select operation \n1.insert \n2.delete \n3.revove \
n4.quit"))
print(a)
```

```
Select operation
1.insert
2.delete
3.revove
```

```
4.quit4
4

class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None

    def insert(self,data):
        n=Node(data)
        if self.head==None:
            self.head=n
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.next=n
    def reverse(self):
        prev=None
        temp=self.head
        while(temp!=None):
            next=temp.next
            temp.next=prev
            prev=temp
            temp=next
        self.head=prev

    def display(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def del_atposition(self,position):
        temp=self.head
        count=1
        while(temp.next!=None):
            count+=1
            temp=temp.next
        print(count)
        if (position>1 and position >count):
            print("Invalid position")
            return

        elif(position ==1):
            if self.head.next==None:
                self.head=None
            else:
                while(temp.next!=None):
                    temp=temp.next
                self.head=self.head.next
                temp.next=None
        else:
            temp=self.head
            for i in range(1,position-1):
                temp=temp.next
            temp.next=temp.next.next
```

```
ll=linkedlist()
ll.insert(10)
#ll.insert(20)
#ll.insert(80)
#ll.insert(40)
#ll.insert(50)
ll.display()
print("after deletion at position")
ll.del_atposition(2)
ll.display()
```

```
10
after deletion at position
1
Invalid position
10
```

```
n=2
for i in range(1,n):
    print(i)
```

```
1
```

```
# factorial
def fac(n):
    if n==0 or n==1:
        return 1
    else:
        return n*fac(n-1)
n=int(input("enter the number"))
fac(n)
```

```
enter the number4
```

```
24
```

```
# creating a linked list

class Node:
    def __init__(self,data):
        self.data=data
        self.next=None

class Linkedlist:
    def __init__(self):
        self.head=None

    def push(self,data):

        new_node=Node(data)

        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.next=new_node
    def display(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
ll=Linkedlist()
ll.push(10)
ll.push(30)
ll.display()
```

```
10
30
```

```
a=int(input("select an operation : \n1.Insert \n2.Deletion \n3.Display
\n4.quit"))
a
```

```
select an operation :
1.Insert
2.Deletion
3.Display
4.quit4
```

```
4
```

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None

    def insert(self,data):
        n=Node(data)
        if self.head==None:
            self.head=n
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.next=n
    def reverse(self):
        prev=None
        temp=self.head
        while(temp!=None):
            next=temp.next
            temp.next=prev
            prev=temp
            temp=next
        self.head=prev

    def display(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def search(self,x):
        #if self.head.data==x:
           # print("item found ")
            #return

        temp=self.head
        while(temp!=None):
            if temp.data==x:
                print("item found ")
                return
            temp=temp.next
        if temp==None:
            print("item not found")

ll=linkedlist()
```

```
while True:
    a=int(input("select an operation : \n1.Insert \n2.Search \
n3.Display \n4.quit"))

    if a==1:
        b=int(input("Enter element "))
        ll.insert(b)
    elif a==2:
        c=int(input("Enter the item to be search:"))
        ll.search(c)

    elif a==3:
        ll.reverse()
        ll.display()
    elif a==4:
        break
    else:
        print("Invalid Option!!!")
```

```
select an operation :
1.Insert
2.Search
3.Display
4.quit1
Enter element 10
select an operation :
1.Insert
2.Search
3.Display
4.quit2
Enter the item to be search:10
item found
select an operation :
1.Insert
2.Search
3.Display
4.quit2
Enter the item to be search:20
item not found
select an operation :
1.Insert
2.Search
3.Display
4.quit4
```

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None

    def insert(self,data):
        n=Node(data)
        if self.head==None:
            self.head=n
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.next=n
    def reverse(self):
        prev=None
        temp=self.head
        while(temp!=None):
            next=temp.next
            temp.next=prev
            prev=temp
            temp=next
        self.head=prev

    def display(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def search(self,x):
        if self.head.data==x:
            print("item found ")
            return

        temp=self.head
        while(temp!=None):
            if temp.data==x:
                print("item found ")
                return
            temp=temp.next
        if temp==None:
            print("item not found")


ll=linkedlist()
ll.insert(10)
ll.insert(20)
ll.insert(40)
ll.insert(50)
ll.display()
ll.search(70)
```

```
10
20
40
50
item not found
```

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None

    def insert(self,data):
        n=Node(data)
        if self.head==None:
            self.head=n
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            temp.next=n
    def reverse(self):
        prev=None
        temp=self.head
        while(temp!=None):
            next=temp.next
            temp.next=prev
            prev=temp
            temp=next
        self.head=prev

    def display(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def update(self,x,value):
        index=0
        temp=self.head
        while(temp!=None):
            if index==x:
                temp.data=value
            index+=1
            temp=temp.next
ll=linkedlist()
ll.insert(10)
ll.insert(40)
ll.insert(50)
ll.reverse()
ll.display()
print("after updation")
ll.update(5,99)
ll.display()
```

```
50
40
10
after updation
50
40
10
```

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None

    def insert(self,data):
        n=Node(data)
        n.next=self.head
        self.head=n
    def reverse(self):
        prev=None
        temp=self.head
        while(temp!=None):
            next=temp.next
            temp.next=prev
            prev=temp
            temp=next
        self.head=prev

    def display(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def update(self,x,value):
        index=0
        temp=self.head
        while(temp!=None):
            if index==x:
                temp.data=value
            index+=1
            temp=temp.next
ll=linkedlist()
ll.insert(10)
ll.insert(40)
ll.insert(50)
ll.display()
ll.update(1,99)
ll.display()
```

```
50
40
10
50
99
10
```

Merging two sorted linked list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def insert(self,data):
        n=Node(data)
        n.next=self.head
        self.head=n
    def display(self):
        if self.head==None:
            print("linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data,end=" ")
                temp=temp.next
            print()
    def sort(self):
        index=None
        temp=self.head
        if self.head==None:
            return
        else:
            while(temp!=None):
                index=temp.next
                while(index!=None):
                    if temp.data>index.data:
                        temp.data,index.data=index.data,temp.data
                    index=index.next
                temp=temp.next
ll1=linkedlist()
ll2=linkedlist()
ll3=linkedlist()
a1=int(input("Enter number of elements in first list "))
while(a1>0):
    b1=int(input("enter element "))
    ll1.insert(b1)
    a1-=1
a2=int(input("Enter number of elements in second list "))
while(a2>0):
    b2=int(input("enter element "))
    ll2.insert(b2)
    a2-=1
print("linked list 1")
ll1.display()
print("linked list 2")
ll2.display()
def merge(l2):
    while(l2.head!=None):
        ll1.insert(l2.head.data)
        l2.head=l2.head.next
merge(ll2)
print("after merging")
#ll1.sort()
ll1.display()
```

```
Enter number of elements in first list 2
enter element 4
enter element 2
Enter number of elements in second list 3
enter element 8
enter element 1
enter element 5
linked list 1
2 4
linked list 2
5 1 8
after merging
8 1 5 2 4
```

```
class Node:
    def __init__(self,data):
        self.prev=None
        self.data=data
        self.next=None
```

```python
class doubly_linked_list:
    def __init__(self):
        self.head=None

    def display(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next

    def insert(self,data):
        new_node=Node(data)

        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            new_node.prev=temp
            temp.next=new_node

ll=doubly_linked_list()
while True:
    a=int(input("select Operation : \n1.Insertion \n2.Display \
n3.quit"))

    if a==1:
        b=int(input("Enter element "))
        ll.insert(b)
    elif a==2:
        print("Adding a node to the end of the list: ")
        ll.display()
    elif a==3:
        break
    else:
        print("Invalid Option!!!")
```

```
select Operation :
1.Insertion
2.Display
3.quit1
Enter element 6
select Operation :
1.Insertion
```

```
2.Display
3.quit1
Enter element 4
select Operation :
1.Insertion
2.Display
3.quit1
Enter element 8
select Operation :
1.Insertion
2.Display
3.quit2
Adding a node to the end of the list:
6
4
8
select Operation :
1.Insertion
2.Display
3.quit3
```

```
 def del_atbeg(self):
        if self.head==None:
            print("Linked list is empty")
            return
        if self.head.next==None:
            self.head=None
        else:
            self.head=self.head.next
            self.head.prev=None

class Node:
    def __init__(self,data):
        self.prev=None
        self.data=data
        self.next=None

class doubly_linked_list:
    def __init__(self):
        self.head=None

    def display(self):
        if self.head==None:
            print("List is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def insert(self,data):
        new_node=Node(data)

        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            new_node.prev=temp
            temp.next=new_node
    def delete(self):
        if self.head==None:
            print("Linked list is empty")
            return
        if self.head.next==None:
            self.head=None
        else:
            self.head=self.head.next
            self.head.prev=None

ll=doubly_linked_list()
while True:
    a=int(input("select Operation : \n1.Insertion \n2.Deletefromstart 
\n3.Display \n4.quit"))

    if a==1:
        b=int(input("Enter element "))
        ll.insert(b)
    elif a==2:
        ll.delete()
    elif a==3:

        ll.display()
    elif a==4:
        break
    else:
        print("Invalid Option!!!")
```

```
select Operation : 
1.Insertion 
2.Deletefromstart 
3.Display 
4.quit1
Enter element 23
select Operation : 
1.Insertion 
2.Deletefromstart 
3.Display 
```

```
4.quit1
Enter element 44
select Operation :
1.Insertion
2.Deletefromstart
3.Display
4.quit3
23
44
select Operation :
1.Insertion
2.Deletefromstart
3.Display
4.quit2
select Operation :
1.Insertion
2.Deletefromstart
3.Display
4.quit2
select Operation :
1.Insertion
2.Deletefromstart
3.Display
4.quit3
List is empty
select Operation :
1.Insertion
2.Deletefromstart
3.Display
4.quit4
```

```python
class Node:
    def __init__(self,data):
        self.prev=None
        self.data=data
        self.next=None

class doublylinkedlist:
    def __init__(self):
        self.head=None

    def display(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def revdisplay(self):
        if self.head==None:
            print("List is empty")

        temp=self.head
        while(temp.next!=None):
            temp=temp.next
        self.head=temp
        while(temp!=None):
            print(temp.data)
            temp=temp.prev



    def insert(self,data):
        new_node=Node(data)

        if self.head==None:
            self.head=new_node
        else:
            new_node.next=self.head
            self.head.prev=new_node
            self.head=new_node

ll=doublylinkedlist()
a=int(input("Enter the no elements: "))
for i in range(a):
    b=int(input("enter element: "))
    ll.insert(b)
ll.display()
print("in reverse")
ll.revdisplay()
```

```
Enter the no elements: 3
enter element: 1
enter element: 2
enter element: 3
3
2
1
in reverse
1
2
3
```

```
class Node:
    def __init__(self,data):
        self.prev=None
        self.data=data
        self.next=None
```

```
class doubly_linked_list:
    def __init__(self):
        self.head=None

    def display(self):
        if self.head==None:
            print("List is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next

    def insert(self,data):
        new_node=Node(data)

        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            new_node.prev=temp
            temp.next=new_node
    def search(self,x):
        pos=1
        if self.head.data==x:
            print("item found at position : ",pos)
            return

        temp=self.head

        while(temp!=None):
            if temp.data==x:
                print("item found at positon : ",pos)
                return
            temp=temp.next
            pos+=1
        if temp==None:
            print("item not found")

ll=doubly_linked_list()
while True:
    a=int(input("select Operation : \n1.Insertion \n2.Search \
n3.Display \n4.quit"))

    if a==1:
        b=int(input("Enter element "))
        ll.insert(b)
    elif a==2:
        c=int(input("Enter the element to be for search "))
        ll.search(c)
    elif a==3:

        ll.display()
    elif a==4:
        break
    else:
        print("Invalid Option!!!")
```

```
select Operation :
1.Insertion
2.Search
3.Display
4.quit1
Enter element 2
select Operation :
1.Insertion
2.Search
3.Display
4.quit1
Enter element 4
select Operation :
1.Insertion
2.Search
3.Display
4.quit3
2
4
select Operation :
1.Insertion
2.Search
3.Display
4.quit2
Enter the element to be for search 2
item found at position :  1
select Operation :
1.Insertion
2.Search
3.Display
4.quit2
Enter the element to be for search 4
item found at positon :  2
select Operation :
1.Insertion
2.Search
3.Display
4.quit2
Enter the element to be for search 6
```

```
item not found
select Operation :
1.Insertion
2.Search
3.Display
4.quit4
```

```python
class Node:
    def __init__(self,data):
        self.prev=None
        self.data=data
        self.next=None

class doublylinkedlist:
    def __init__(self):
        self.head=None

    def display(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def revdisplay(self):
        if self.head==None:
            print("List is empty")

        temp=self.head
        while(temp.next!=None):
            temp=temp.next
        self.head=temp
        while(temp!=None):
            print(temp.data)
            temp=temp.prev



    def insertfront(self,data):
        new_node=Node(data)

        if self.head==None:
            self.head=new_node
        else:
            new_node.next=self.head
            self.head.prev=new_node
            self.head=new_node
    def insertend(self,data):
        new_node=Node(data)
        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            new_node.prev=temp
            temp.next=new_node

ll1=doublylinkedlist()
ll2=doublylinkedlist()
a1=int(input("Enter the no elements list1 : "))
for i in range(a1):
    b1=int(input("enter element: "))
    ll1.insertfront(b1)
a2=int(input("Enter the no elements list1 : "))
for i in range(a2):
    b2=int(input("enter element: "))
    ll2.insertfront(b2)
ll1.display()
ll2.display()
def merge(l2):
    while(l2.head!=None):
        ll1.insertend(l2.head.data)
        l2.head=l2.head.next
merge(ll2)
print("after merging")
#ll1.sort()
ll1.display()
```

```
Enter the no elements list1 : 2
enter element: 3
enter element: 1
Enter the no elements list1 : 2
enter element: 5
enter element: 4
1
3
4
5
after merging
1
3
4
5
```

```
class Node:
    def __init__(self,data):
        self.prev=None
```

```python
        self.data=data
        self.next=None

class doublylinkedlist:
    def __init__(self):
        self.head=None

    def display(self):
        if self.head==None:
            print("Linked list is empty")
        else:
            temp=self.head
            while(temp!=None):
                print(temp.data)
                temp=temp.next
    def revdisplay(self):
        if self.head==None:
            print("List is empty")

        temp=self.head
        while(temp.next!=None):
            temp=temp.next
        self.head=temp
        while(temp!=None):
            print(temp.data)
            temp=temp.prev



    def insertfront(self,data):
        new_node=Node(data)

        if self.head==None:
            self.head=new_node
        else:
            new_node.next=self.head
            self.head.prev=new_node
            self.head=new_node
    def insertend(self,data):
        new_node=Node(data)

        if self.head==None:
            self.head=new_node
        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next
            new_node.prev=temp
            temp.next=new_node
```

```python
ll1=doublylinkedlist()
ll2=doublylinkedlist()
a1=int(input("Enter the no elements list1 : "))
for i in range(a1):
    b1=int(input("enter element: "))
    ll1.insertfront(b1)
ll1.display()
a2=int(input("Enter the no elements list2 : "))
for i in range(a2):
    b2=int(input("enter element: "))
    ll2.insertfront(b2)

ll2.display()
def merge(l2):
    while(l2.head!=None):
        ll1.insertend(l2.head.data)
        l2.head=l2.head.next
merge(ll2)
print("after merging")
#ll1.sort()
ll1.display()
```

```
Enter the no elements list1 : 4
```