**DOUBLY AND CIRCULAR**

**LINKED LIST**

**EXPT NO: 4**   **DATE: 19/11/21**

**AIM**

**1)** Write a program to implement double linked list and perform following operations.

a. Create a doubly linked list  
b. Display doubly linked list  
c. Add to empty list  
d. Add at the beginning  
e. Add at the end  
f. Add after node  
g. Add before node  
h. Delete a node

**2)** Write a program to implement circular linked list and perform following operations.

a. Create a circular linked list  
b. Display circular linked list  
c. Add to empty list  
d. Add at the beginning  
e. Add at the end  
f. Add after node  
g. Add before node  
h. Delete a node

**3)** Write a program create a circular linked list in which info part of each node contains a digit of a given number. The digits should be stored in reverse order i.e., the least significant digit is stored in first node and most significant digit in the last node. Write a function to add two consecutive numbers from the circular linked list and store it in doubly linked list.

**4)** Write a program to find whether a doubly linked list is palindrome or not.

**THEORY**

In experiment 1 we saw that singly linked lists contain only one link. We could move only in one direction because each node has the address of the next node only.

We have another data structure known as the **Doubly Linked List** or a **Two-way** list. Here each node as 2 pointers. One pointer points to the next node while the other pointer points to the previous node.

**Structure for a typical doubly linked List.**

struct node

{

struct node \*prev;

int data;

struct node \*nxt;

};

The pointer prev holds the address of the previous node and next will contain the address of the next node in the list. Hence, we can move in any direction at any time.

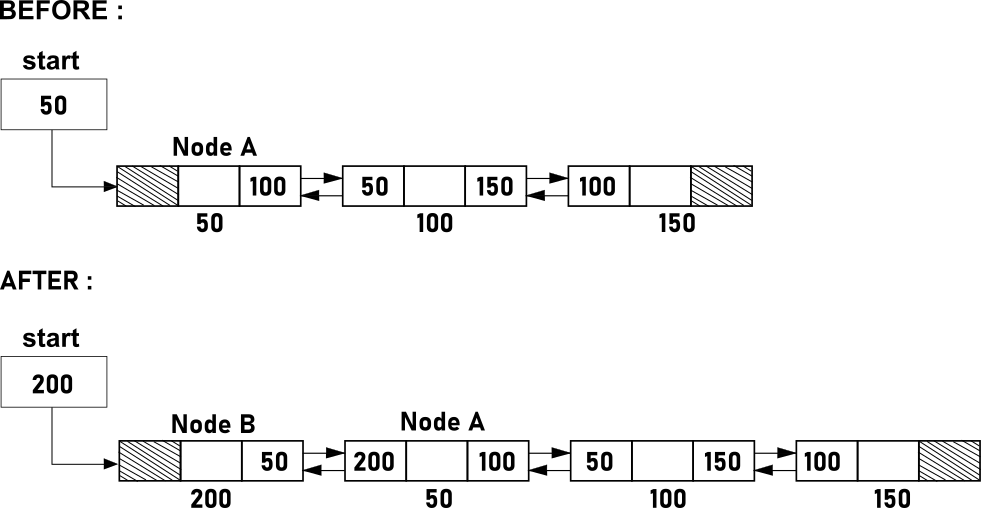
**Doubly Linked List illustration**

**Insertion in a Doubly Linked List**

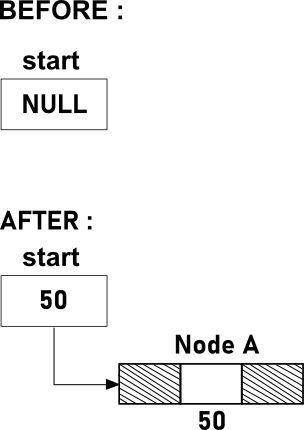
Four cases of insertion in a Doubly Linked List.

1. Insertion at the beginning of the list.
2. Insertion in an empty list.
3. Insertion at the end of the list.
4. Insertion in between the node.

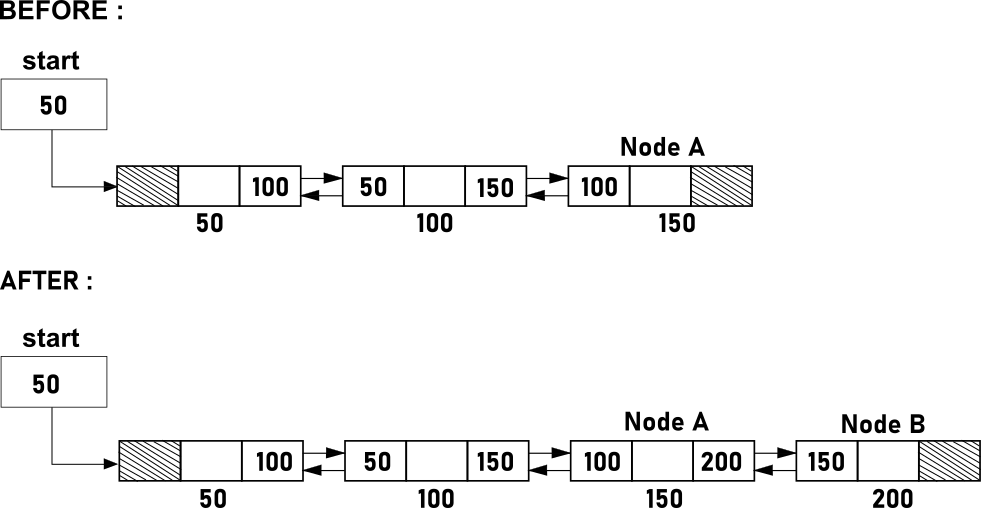
**Insertion at the beginning of the list**



**Insertion in an empty list**



**Insertion at the end of the list**

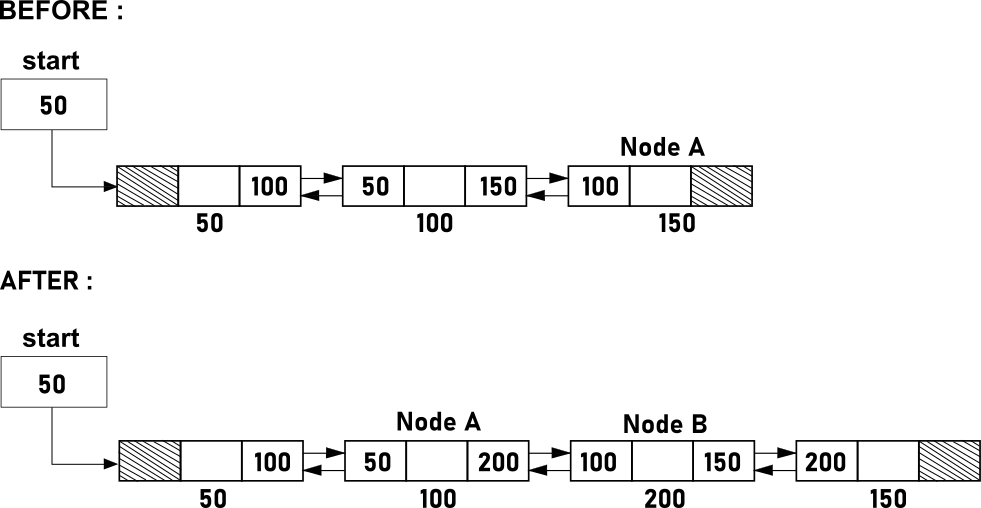


**Insertion in between the node**

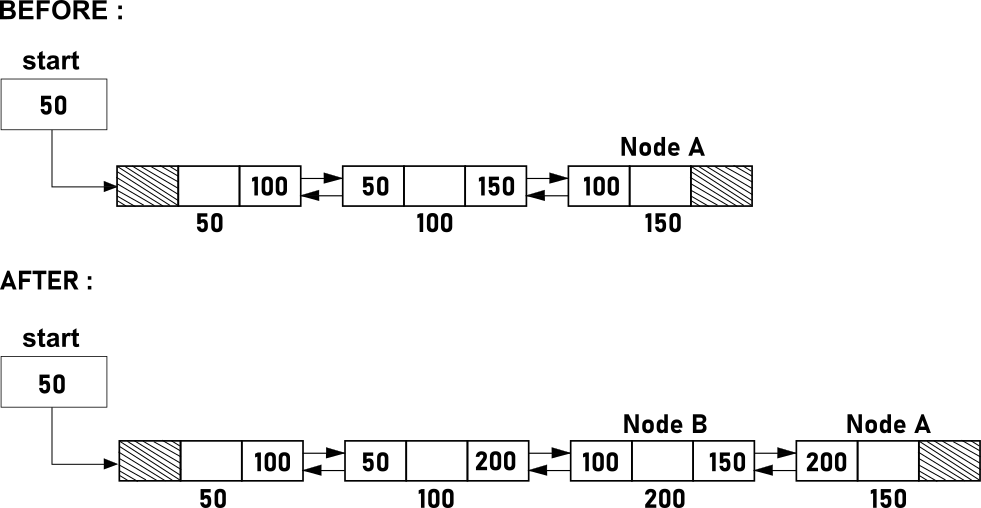
**Further divided into two more cases**

1. Insertion after a node
2. Insertion before a node

**Insertion after a node**



**Insertion before a node**

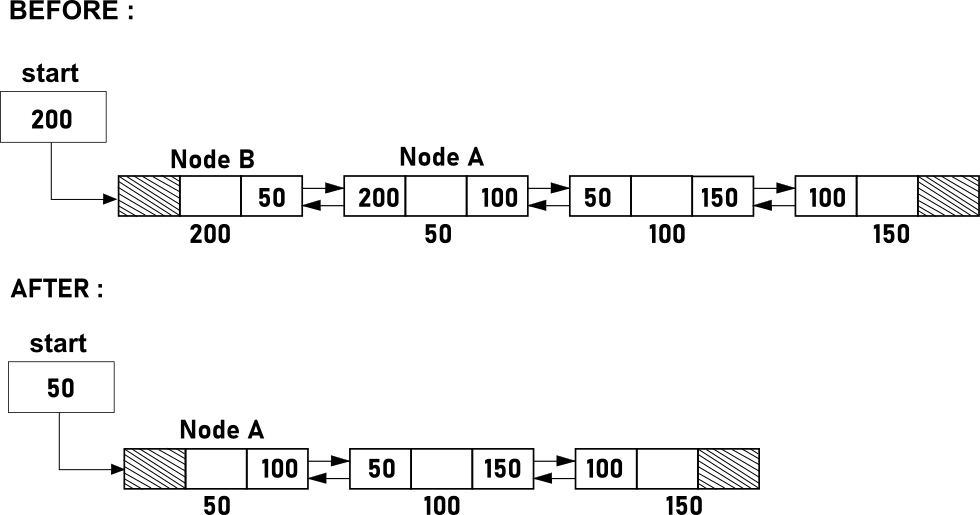


**Deletion in a Doubly Linked List**

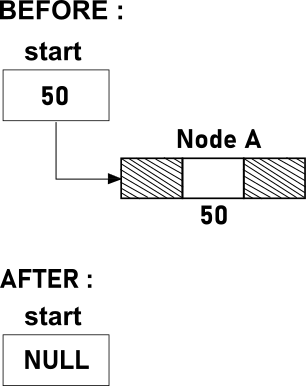
As in single linked list, here also the node is first logically removed by rearranging the pointers and then it is physically removed nu calling the function free(). The four cases for deletion are as follows.

1. Deletion of the first node.
2. Deletion of the only node.
3. Deletion in between the nodes.
4. Deletion at the end of the node.

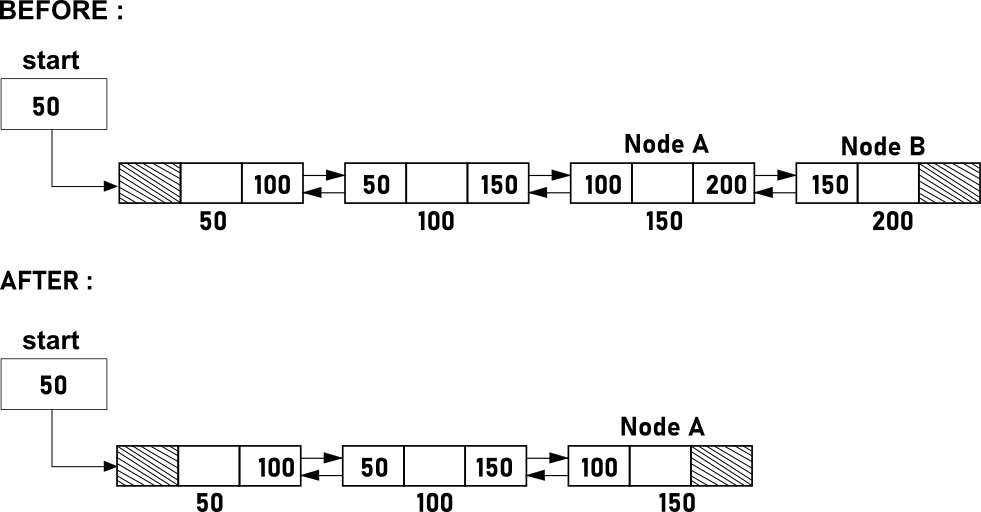
**Deletion of the first node**



**Deletion of the only node**



**Deletion at the end**

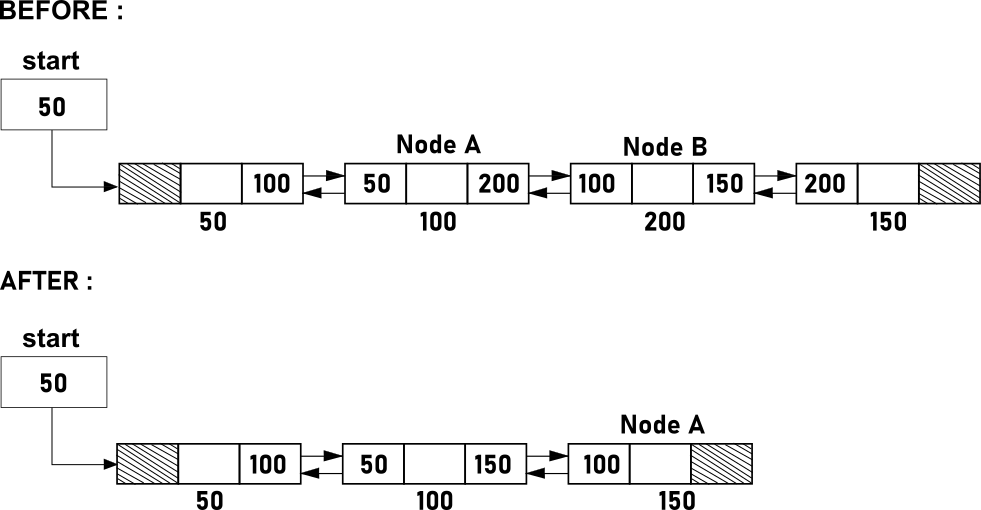


**Deletion in between the nodes**

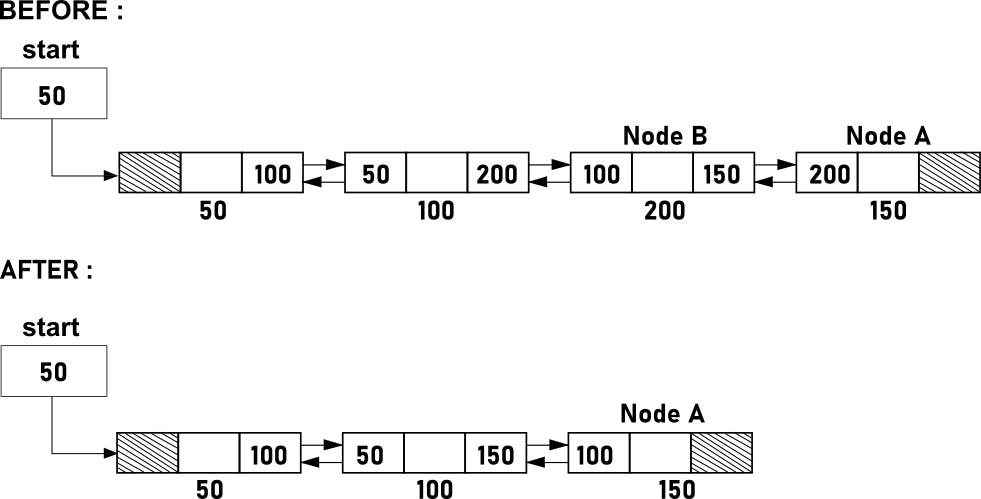
**Further divided into two more cases**

1. Deletion after a node
2. Deletion before a node

**Deletion after a node**



**Deletion before a node**



**Circular Linked List**

In a single linked list, for accessing any node of the list, start traversing from the first node. if we are at any node in the middle of the list, then it is not possible to access nodes that precede the given nodes. This problem can be solved by slightly altering the structure of the linked list. In a single Linked List, link part of the last node holds NULL. If we utilize this to point to the fist node then we can have some advantages. The structure thus formed is known as a Circular Linked List.

Each node has a successor and all the nodes form a ring. Now we can access any node of the linked list without going back and starting traversal again from the first node.

We take an external pointer that pints to the last node of the list. If we have a pointer last pointing to the last node, then last->link will point to the first node.

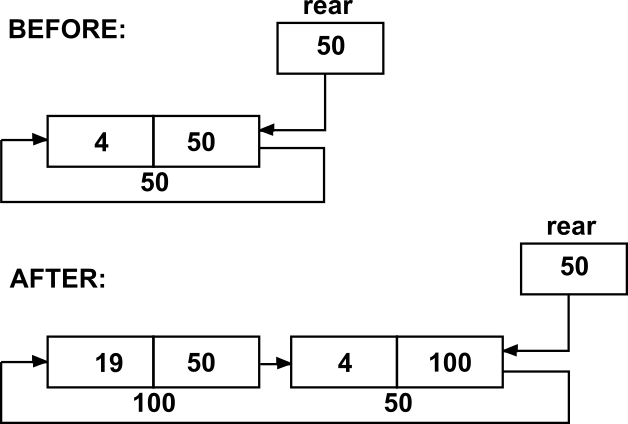
If the circular linked list is empty the last pointer is NULL. And if it has only one element then it points to its own-self

**Insertion in a Circular Linked List**

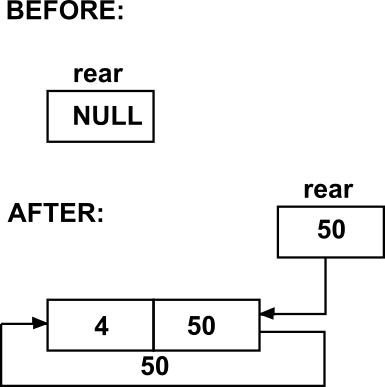
Four cases of insertion in a Doubly Linked List.

1. Insertion at the beginning of the list.
2. Insertion in an empty list.
3. Insertion at the end of the list.
4. Insertion in between the node.

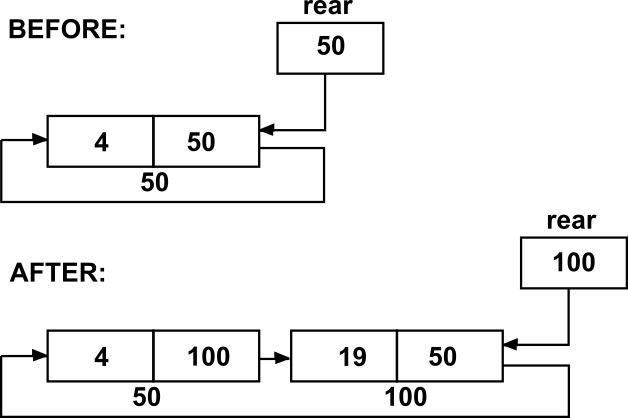
**Insertion at the beginning of the list**



**Insertion in an empty list**



**Insertion at the end of the list**

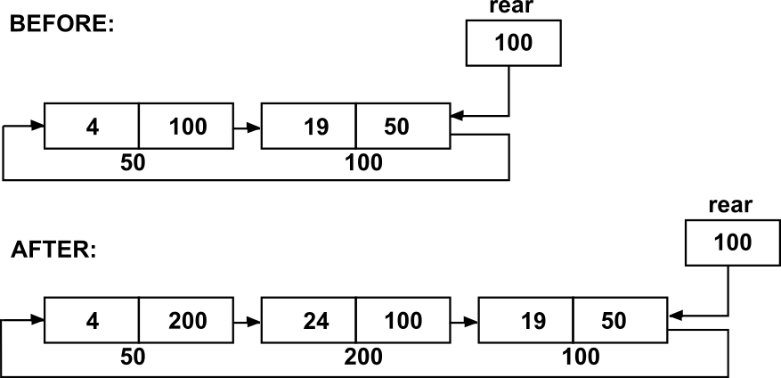


**Insertion in between the node**

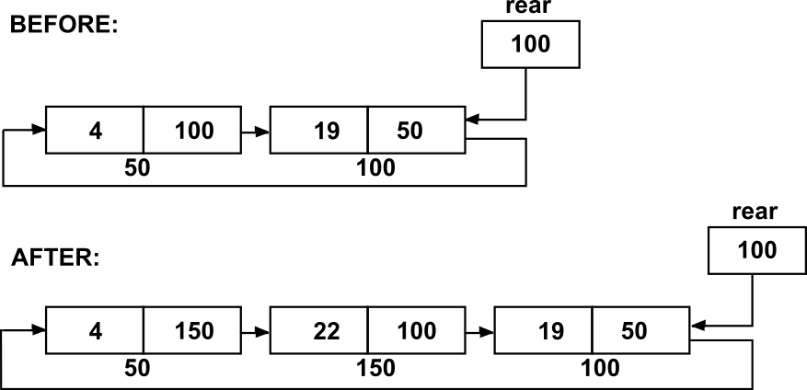
**Further divided into two more cases**

1. Insertion after a node
2. Insertion before a node

**Insertion after a node** (Insert 24 after 4)



**Insertion before a node** (Insert 22 before 19)

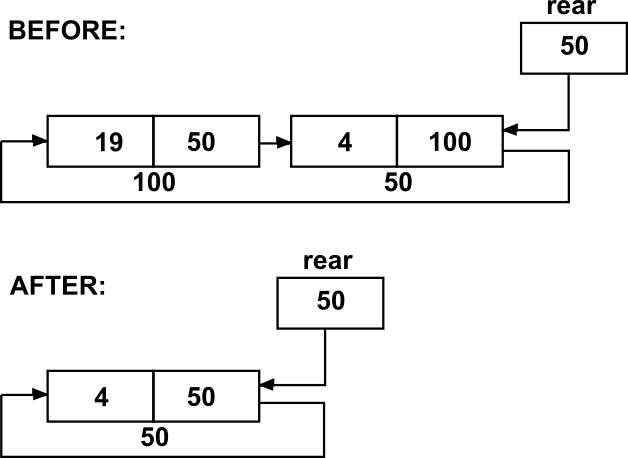


**Deletion in a Circular Linked List**

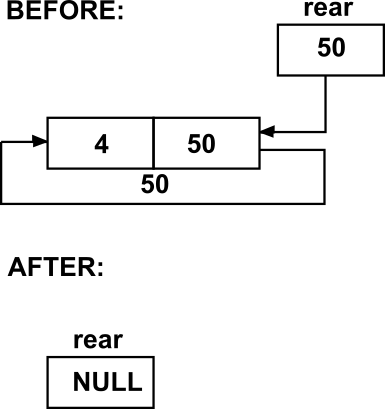
As in single linked list, here also the node is first logically removed by rearranging the pointers and then it is physically removed nu calling the function free(). The four cases for deletion are as follows.

1. Deletion of the first node.
2. Deletion of the only node.
3. Deletion in between the nodes.
4. Deletion at the end of the node.

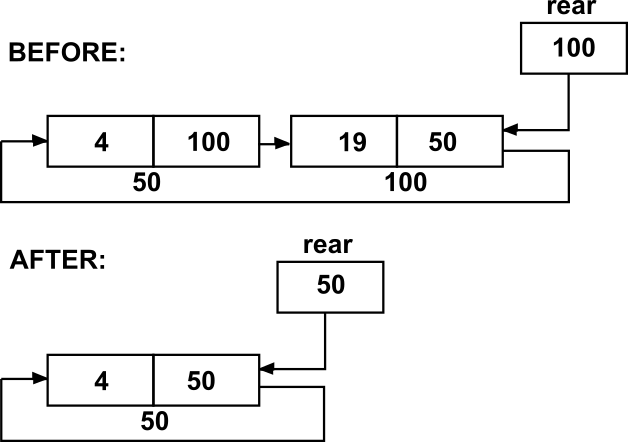
**Deletion of the first node**



**Deletion of the only node**



**Deletion at the end**

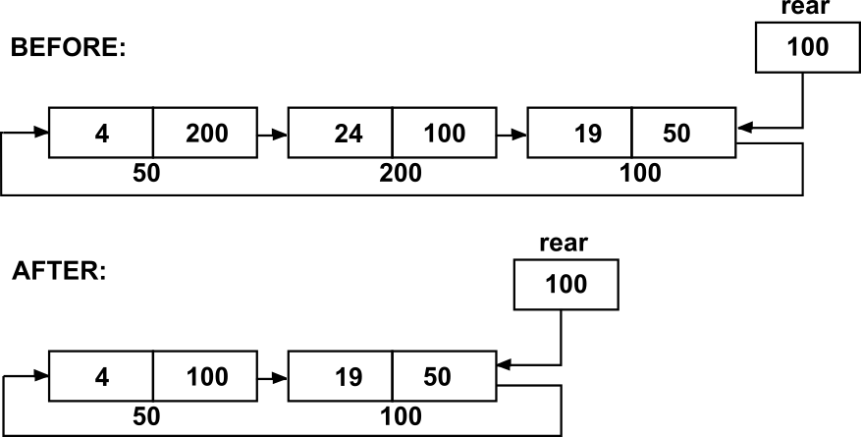


**Deletion in between the nodes**

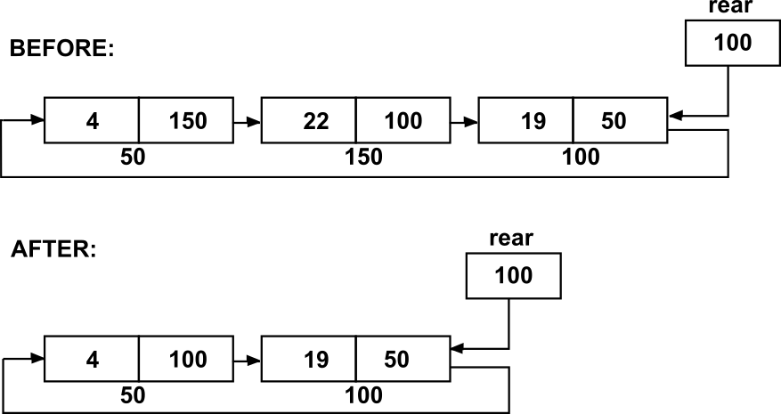
**Further divided into two more cases**

1. Deletion after a node
2. Deletion before a node

**Deletion after a node** (Delete Node After 4)



**Deletion after a node** (Delete Node Before 19)



**Application Programs Design Approach**

**Question 4C:**

1. Input a Number from the user
2. Extract the digits of the number and store them simultaneously in a Circular Linked List.
3. Now declare 2 pointers, the first pointer will point to the first node while the second pointer will point to the simultaneous next node.
4. Now add the contents of the first node and second node and then insert it into the doubly linked list. Now move both pointers to the next node and repeat the same till second pointer reaches the last node.

**Question 4D:**

1. Insert the elements into the doubly linked list
2. Now take 2 pointers P and Q
3. Let P pointer hold the address of the first node

while Q pointer hold the address of the last node

1. Now Compare the data of the P pointer and Q pointer, if the data doesn’t match to each other then report that it isn’t a palindrome doubly linked list, else if all the checks report a match of data then declare that it is a palindrome doubly linked list.
2. Now move the P pointer to the immediate next node and move the R pointer the immediate previous node and repeat step 4.

**ALGORITHMS**

**1)**

Void create(int n) Void addatbeg()

1. Declare int i

2. if n==0

1.return

3. addatemp()

4. for(i=1;i<n;i++)

1. addatend()

1. struct node \*t=(struct node\*)malloc(sizeof(struct node))

2. Output t->data

3. t->prev=NULL

4. t->next=start

5. start->prev=t

6. start=t

Void display() Void addatend()

1. struct node\*t=start

2. if (start==NULL)

1. Output Empty List

2. return

3. while(t!=NULL)

1. Output t->data

2. t=t->next

1. struct node \*p,\*t=(struct node\*)malloc(sizeof(struct node))

2. t->data=x,p=start

3. while(p->next!=NULL

P=p->next

4. p->next=t

5. t->prev=p

6. t->next=NULL

void addafter() void del(int data)

1. struct node\*p=start

2. struct node \*temp=(struct node\*)malloc(sizeof(struct node))

3. temp->data=item

4. while(p!=NULL)

1. if(p->data==item)

1. temp->next=p->next

2. temp->prev=p

3. if(p->next!=NULL)

1. p->next->prev=temp

2. p->next=temp

3. return

2. p=p->next

5 . Output item not found

1. Declare int data

2. struct node\*p

3. if(start==NULL)

1. Output List is empty

2. return

4. Output &data

5. if(start->next==NULL)

1. if(start->data==data)

1. free(start)

2. start=NULL

3. return

2. Output data not found

3. return

6. if(start->data==data)

1. p=start

2. start=start->next

3. start->prev=NULL

4. free(p)

5. return

7. p=start->next

8. while(p->next!=NJLL)

1. if(p->data==data)

1. p->next->prev=p->prev

2. p->prev->next=p->next

3. free(p)

4. return

2. p=p->next

9. if(p->data==data)

1. p->prev->next=NULL

2. free(p)

3. return

10. Output data not found

void addbefore(int item, int x)

1. Declare int item

2. if(start==NULL)

1. Output List is Empty

2. return

3. struct node\*p=start

4. if(start->data==item)

1. addatbeg()

2. return

5. while(p!=NULL)

1. if(p->data==item)

1. struct node\*t=(struct node\*)malloc(sizeof(struct node))

2. t->data=x

3. t->next=p

4. t->prev=p->prev

5. p->prev->next=t

6. p->prev=t

7. return

2.p=p->next

3. Output item not found

void addempty(int x)

1. struct node \*t=(struct node\*)malloc(sizeof(struct node))

2. t->data=item

3. t->prev=NULL

4. t->next=NULL

5. start=t

**2)**

Void create(int n,int x) Void addafter(int item ,int x)

1. Declare int item

2. struct node\*p=last->next

3. do

1. if(p->data==item)

1. Struct node\*t=(struct node\*)malloc(sizeof(struct node))

2. t->data=x

3. t->next=p->next

4. p->next=t

5. if(p==last)

1. last=t

6. return

2. p=p->next

4. while(p!=last->next)

5. Output Item not found

1. Declare int i

2. if(n==0)

1.return

3. addatemp(x)

4. for(i=1;i<n;i++)

1. addatend(x)

Void display()

1. if(last==NULL)

1. Output empty list

2. return

2. struct node\*t=last->next

3. do

1. Output t->data

2. t= t->next

4. while(t!=last->next)

Void addbefore(int item, int x()

Void addatemp(int x)

1. Declare int item

2. if(last==NULL)

1. Output List is empty

2. return

3. struct node\*p=last->next

4. if(last->next->data==item

1. addatbeg()

2. return

5. do

1. if(p->next->data==item)

1. Struct node\*t=(struct node\*)malloc(sizeof(struct node))

2. t->data=x

3. t->next=p->next

4. p->next=t

5. return

2. p=p->next

6. while(p->next!=last->next)

7t. Output item not found

1. Struct node\*t=(struct node\*)malloc(sizeof(struct node))

2. t->data=x

3. t->next=t

4. last=t

Void addatend(int item)

1. Struct node\*t=(struct node\*)malloc(sizeof(struct node))

2. t->data=item

3. t->next=last->next

4. last->next=t

5. last=t

void delnode)

1. Declare int data

2. struct node\*t,\*p

3. if(last==NULL)

1. Output list is empty

2. return

4. input data

5. if(last->next==last)

1. free(last)

2. last==NULL

3. return

6. if(last->next->data==data)

1. t=last->next

2. last->next=t->next

3. free(t)

4. return

7. p=last->next

8. while(p->next!=last)

1.if(p->next->data==data)

1. t=p->next

2. p->next=t->next

3. free(t)

4. return()

2. p=p->next

9. if(last->data==data)

1. last=p

2. t=p->next

3. p->next=t->next

4. free(t)

5. return

10. Output data not found

**CODE**

**1)**

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

int data;

struct node \*next;

};

struct node\* addAfter(struct node \*start,int item,int data)

{

struct node \*temp,\*p;

temp=(struct node\*)malloc(sizeof(struct node));

temp->data=data;

p=start;

while(p!=NULL)

{

if(p->data==item)

{

temp->prev=p;

temp->next=p->next;

if(p->next!=NULL)

p->next->prev=temp;

p->next=temp;

return start;

}

p=p->next;

}

}

struct node \*del(struct node \*start,int data)

{

struct node\*temp;

if(start==NULL)

{

printf("LIST IS EMPTY\n");

return start;

}

if(start->next==NULL)

{

if(start->data==data)

{

temp=start;

start=NULL;

free(temp);

return start;

}

else

{

printf("ELEMENT NOT FOUND\n");

return start;

}

}

if(start->data==data)

{

temp=start;

start=start->next;

start->prev=NULL;

free(temp);

return start;

}

temp=start->next;

while(temp->next!=NULL)

{

if(temp->data==data)

{ temp->prev->next=temp->next;

temp->next->prev=temp->prev;

free(temp);

return start;

}

temp=temp->next;

}

if(temp->data==data)

{

temp->prev->next=NULL;

free(temp);

return start;

}

printf("THE ELEMENT IS NOT FOUND\n");

return start;

}

struct node \*addBefore(struct node \*start, int item,int data)

{

struct node \*temp,\*p;

if(start==NULL)

{

printf("THE LIST IS EMPTY\n");

return start;

}

if(start->data==item)

{

temp=(struct node \*)malloc(sizeof(struct node));

temp->data=data;

temp->prev=NULL;

temp->next= start;

start->prev=temp;

start=temp;

return start;

}

p=start;

while(p!=NULL)

{

if(p->data==item)

{

temp=(struct node \*)malloc(sizeof(struct node));

temp->data=data;

temp->prev=p->prev;

temp->next=p;

p->prev->next=temp;

p->prev=temp;

return start;

}

p=p->next;

}

printf("ELEMENT IS NOT FOUND\n");

}

struct node \*addBeg(struct node \*start,int data)

{

struct node \*temp;

temp=(struct node\*)malloc(sizeof(struct node));

temp->data=data;

temp->prev=NULL;

temp->next=start;

start->prev=temp;

start=temp;

return start;

}

struct node \*addEmpty(struct node \*start,int x)

{

struct node \*temp;

temp=(struct node\*)malloc(sizeof(struct node));

temp->data=x;

temp->next=NULL;

temp->prev=NULL;

start=temp;

return start;

}

struct node \*addEnd(struct node \*start, int x)

{

struct node \*temp,\*p;

temp=(struct node\*)malloc(sizeof(struct node));

temp->data=x;

p=start;

while(p->next!=NULL)

{

p=p->next;

}

p->next=temp;

temp->prev=p;

temp->next=NULL;

return start;

}

struct node \*createList(struct node \*start)

{

int i,data,n;

printf("ENTER NUMBER OF NODES\n");

scanf("%d",&n);

start=NULL;

if(n==0)

{

return start;

}

printf("ENTER THE DATA OF THE NODE: ");

scanf("%d",&data);

start=addEmpty(start,data);

for(i=2;i<=n;i++)

{

printf("ENTER DATA OF THE NODE: ");

scanf("%d",&data);

start=addEnd(start,data);

}

return start;

}

void display(struct node \*start)

{

struct node \*p;

p=start;

if(start==NULL)

{

printf("LINKED LIST EMPTY\n");

return;

}

printf("\n\n");

while(p!=NULL)

{

printf("%d ",p->data);

p=p->next;

}

printf("\n");

}

int main()

{

int choice,key,data,item;

struct node \*start;

while(1){

printf("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("1: CREATE A DOUBLY LINKEDLIST\n");

printf("2: DISPLAY LINKEDLIST\n");

printf("3: ADD TO EMPTY LIST\n");

printf("4: ADD AT BEGINNING\n");

printf("5: ADD AT THE END\n");

printf("6: ADD AFTER A NODE\n");

printf("7: ADD BEFORE A NODE\n");

printf("8: DELETE A NODE\n");

printf("0: PROGRAM END\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("ENTER THE CHOICE\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

start=createList(start);

break;

case 2:

display(start);

break;

case 3:

printf("ENTER NODE DATA: ");

scanf("%d",&item);

start=addEmpty(start,item);

break;

case 4:

printf("ENTER NODE DATA: ");

scanf("%d",&item);

start=addBeg(start,item);

break;

case 5:

printf("ENTER NODE DATA: ");

scanf("%d",&item);

start=addEnd(start,item);

break;

case 6:

printf("ENTER THE ELEMENT AFTER WHICH YOU WANT TO ADD A NEW ELEMENT\n");

scanf("%d",&key);

printf("ENTER THE DATA OF THE NEW NODE\n");

scanf("%d",&data);

start=addAfter(start,key,data);

break;

case 7:

printf("ENTER THE ELEMENT BEFORE WHICH YOU WANT TO ADD A NEW ELEMENT\n");

scanf("%d",&key);

printf("ENTER THE DATA OF THE NEW NODE\n");

scanf("%d",&data);

start=addBefore(start,key,data);

break;

case 8:

printf("ENTER THE ELEMENT TO BE DELETED\n");

scanf("%d",&key);

start=del(start,key);

break;

case 0:

return 0;

default:

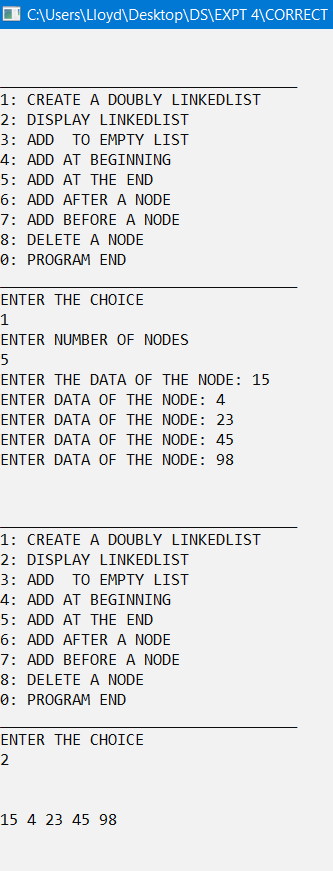
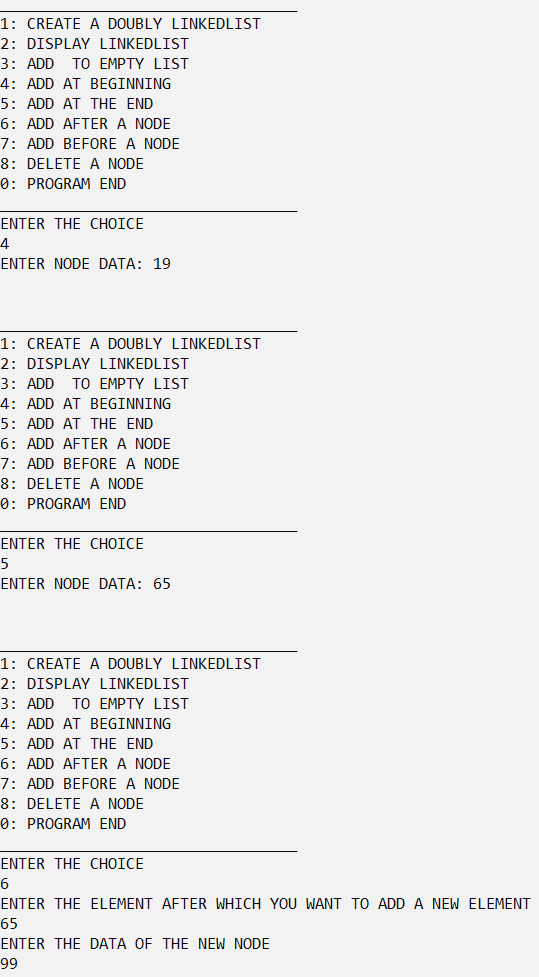
printf("INVLID OPTION\n");

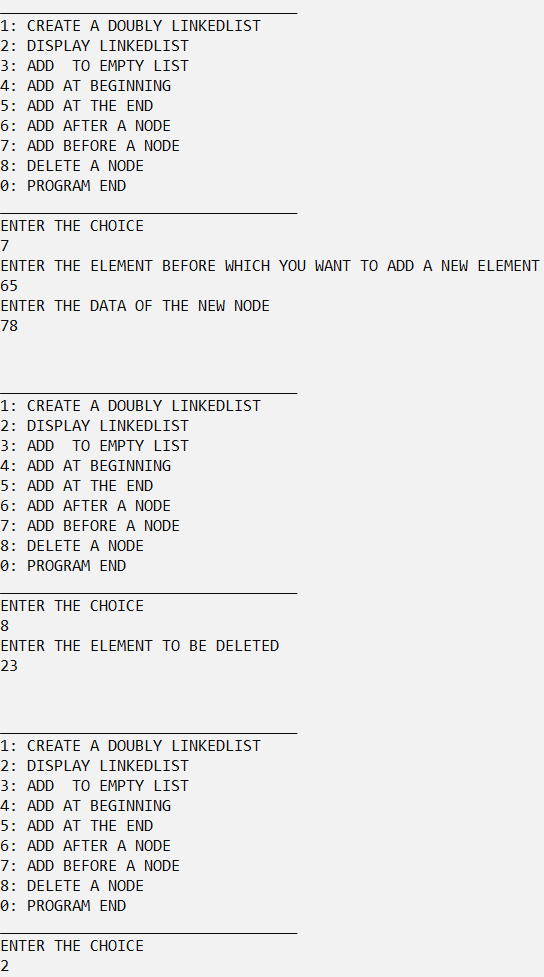
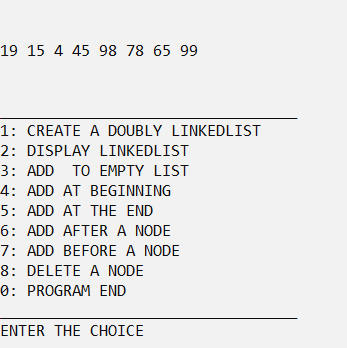
}

}

}

**OUTPUT**

****

****

**2)**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

}\*last=NULL;

void addatemp()

{

struct node \*t=(struct node\*)malloc(sizeof(struct node));

printf("ENTER THE DATA: ");

scanf("%d",&t->data);

t->next=t;

last=t;

}

void addatend()

{

struct node \*t=(struct node\*)malloc(sizeof(struct node));

printf("ENTER THE DATA: ");

scanf("%d",&t->data);

t->next=last->next;

last->next=t;

last=t;

}

void create(int n)

{

int i;

if(n==0)

return;

addatemp();

for(i=1;i<n;i++)

addatend();

}

void display()

{

if(last==NULL)

{

printf("EMPTY LIST\n");

return;

}

struct node \*t=last->next;

do

{

printf("%d ",t->data);

t=t->next;

}while(t!=last->next);

}

void addatbeg()

{

struct node \*t=(struct node\*)malloc(sizeof(struct node));

printf("ENTER THE DATA: ");

scanf("%d",&t->data);

t->next=last->next;

last->next=t;

}

void addafter()

{

int item;

struct node \*p=last->next;

printf("ENTER THE DATA: ");

scanf("%d",&item);

do

{

if(p->data==item)

{

struct node \*t=(struct node\*)malloc(sizeof(struct node));

printf("Enter the data: ");

scanf("%d",&t->data);

t->next=p->next;

p->next=t;

if(p==last)

last=t;

return;

}

p=p->next;

}while(p!=last->next);

printf("ELEMENT NOT FOUND");

}

void addbefore()

{

int item;

printf("ENTER THE ITEM: ");

scanf("%d",&item);

if(last==NULL)

{

printf("LIST IS EMPTY\n");

return;

}

struct node \*p=last->next;

if(last->next->data==item)

{

addatbeg();

return;

}

do

{

if(p->next->data==item)

{

struct node \*t=(struct node\*)malloc(sizeof(struct node));

printf("ENTER THE DATA: ");

scanf("%d",&t->data);

t->next=p->next;

p->next=t;

return;

}

p=p->next;

}while(p->next!=last->next);

printf("ELEMENT NOT FOUND");

}

void delnode()

{

int data;

struct node \*t, \*p;

if(last==NULL)

{

printf("LIST IS EMPTY");

return;

}

printf("ENTER THE DATA: ");

scanf("%d",&data);

if(last->next==last)

{

if(last->data==data)

{

free(last);

last=NULL;

return;

}

printf("DATA NOT FOUND\n");

return;

}

if(last->next->data==data)

{

t=last->next;

last->next=t->next;

free(t);

return;

}

p=last->next;

while(p->next!=last)

{

if(p->next->data==data)

{

t=p->next;

p->next=t->next;

free(t);

return;

}

p=p->next;

}

if(last->data==data)

{

last=p;

t=p->next;

p->next=t->next;

free(t);

return;

}

printf("DATA NOT FOUND\n");

}

int main()

{

int ch, n;

do

{

printf("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("1: CREATE A CIRCULAR LINKEDLIST\n");

printf("2: DISPLAY LINKEDLIST\n");

printf("3: ADD TO EMPTY LIST\n");

printf("4: ADD AT BEGINNING\n");

printf("5: ADD AT THE END\n");

printf("6: ADD AFTER A NODE\n");

printf("7: ADD BEFORE A NODE\n");

printf("8: DELETE A NODE\n");

printf("0: PROGRAM END\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("ENTER THE CHOICE\n");

scanf("%d",&ch);

printf("\n");

switch(ch)

{

case 1:

printf("ENTER HOW NODES YOU WANT: ");

scanf("%d",&n);

create(n);

break;

case 2:

display();

break;

case 3:

addatemp();

break;

case 4:

addatbeg();

break;

case 5:

addatend();

break;

case 6:

addafter();

break;

case 7:

addbefore();

break;

case 8:

delnode();

break;

case 9:

break;

default:

printf("INVALID CHOICE");

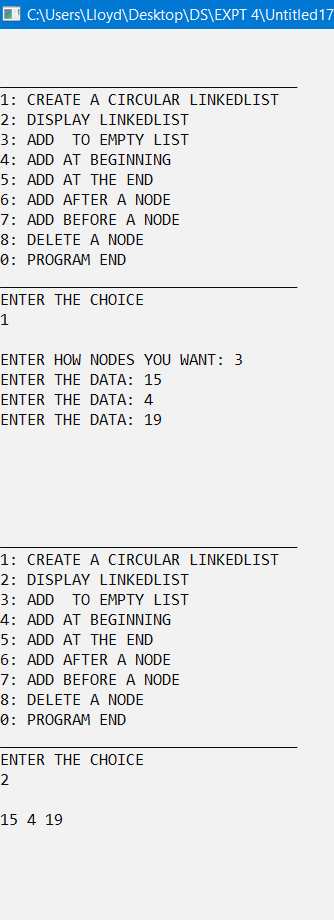
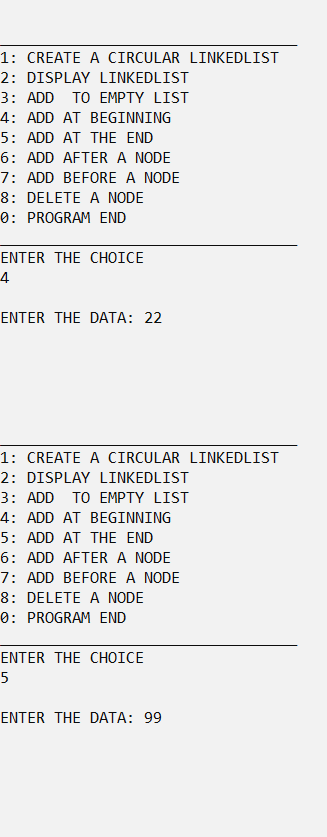
}

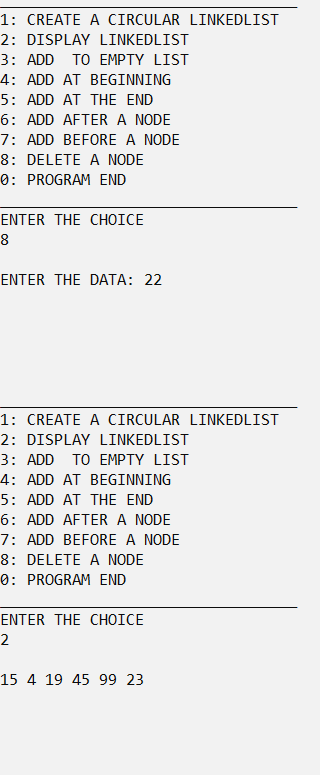
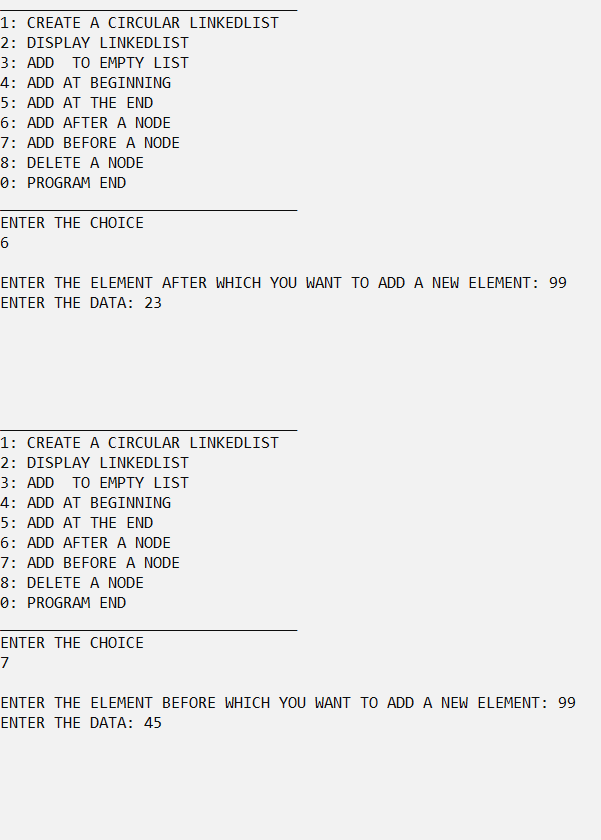
printf("\n\n\n");

}while(ch!=9);

}

**OUTPUT**

****

****

**3)**

#include<stdio.h>

#include<stdlib.h>

struct DLL

{

struct DLL \*prev;

int data;

struct DLL \*next;

}\*start=NULL;

struct CLL

{

int data;

struct CLL \*next;

}\*last=NULL;

void addtoCLL(int data)

{

struct CLL \*t=(struct CLL\*)malloc(sizeof(struct CLL));

t->data=data;

if(last==NULL)

{

t->next=t;

last=t;

}

else

{

t->next=last->next;

last->next=t;

last=t;

}

}

void addtoDLL(int data)

{

struct DLL \*t=(struct DLL\*)malloc(sizeof(struct DLL)), \*p=start;

t->data=data;

if(start==NULL)

{

t->prev=NULL;

t->next=NULL;

start=t;

}

else

{

while(p->next!=NULL)

p=p->next;

p->next=t;

t->prev=p;

t->next=NULL;

}

}

void add()

{

struct CLL \*p=last->next, \*q=p->next;

if(p==q)

{

addtoDLL(p->data);

return;

}

do

{

addtoDLL(p->data+q->data);

p=q->next;

q=p->next;

}while(p!=last->next && q!=last->next);

if(q==last->next)

{

addtoDLL(last->data);

}

}

void display()

{

struct DLL \*t=start;

printf("AFTER ADDING CONSECUTIVE NODES: ");

while(t!=NULL)

{

printf("%d ",t->data);

t=t->next;

}

}

int main()

{

long int n;

printf("ENTER THE NUMBER: ");

scanf("%d",&n);

while(n!=0)

{

addtoCLL(n%10);

n/=10;

}

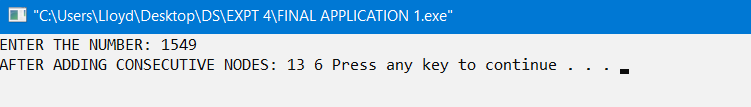
add();

display();

return 0;

}

**OUTPUT**

****

**4)**

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

int data;

struct node \*next;

};

struct node \*addEmpty(struct node \*start,int x)

{

struct node \*temp;

temp=(struct node\*)malloc(sizeof(struct node));

temp->data=x;

temp->next=NULL;

temp->prev=NULL;

start=temp;

return start;

}

struct node \*addEnd(struct node \*start, int x)

{

struct node \*temp,\*p;

temp=(struct node\*)malloc(sizeof(struct node));

temp->data=x;

p=start;

while(p->next!=NULL)

{

p=p->next;

}

p->next=temp;

temp->prev=p;

temp->next=NULL;

return start;

}

struct node \*createList(struct node \*start)

{

int i,data,n;

printf("ENTER NUMBER OF NODES\n");

scanf("%d",&n);

start=NULL;

if(n==0)

{

return start;

}

printf("ENTER THE DATA OF THE NODE: ");

scanf("%d",&data);

start=addEmpty(start,data);

for(i=2;i<=n;i++)

{

printf("ENTER DATA OF THE NODE: ");

scanf("%d",&data);

start=addEnd(start,data);

}

return start;

}

void display(struct node \*start)

{

struct node \*p,\*r;

if(start==NULL)

{

printf("LINKED LIST EMPTY\n");

return;

}

p=start;

r=start;

while(r->next!=NULL)

{

r=r->next;

}

while(p!=NULL)

{

if(p->data != r->data)

{

printf("DOUBLY LINKED LIST NOT PALLINDROME\n");

return;

}

;

p=p->next;

r=r->prev;

}

printf("DOUBLY LINKED LIST IS PALLINDROME\n");

printf("\n");

}

int main()

{

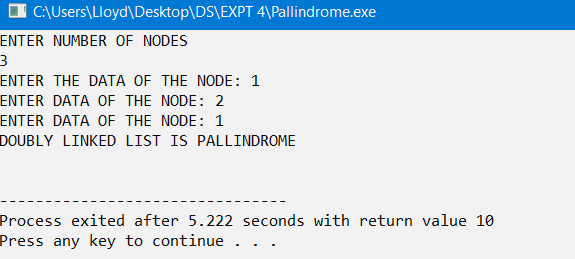
struct node \*start;

start=createList(start);

display(start);

}

**OUTPUT**

****

**CONCLUSION**

The given problem statements were successfully compiled and executed.

**LEARNINGS AND FINDINGS**

1. Concept of Doubly Linked List Data Structure

2. Concept of Circular Linked Lists

3. Implementations

4. Some applications of Doubly and Circular Linked Lists

Doubly Linked Lists eradicates the need to reverse a linked list in order to access the reverse form of the linked list since it as a bidirectional design.

Circular Linked Lists help in efficient traversing, since the last node points to the first node of the linked list.