**GROUP – 8**

**R0-Gowtham R1-Sujash R2-Sai Sree R3-Keerthi**

**R4-Aditya R5-Suneel R6-Sumanth R7-Surya**

**AIM: -** Write a Program to implement the following types of Lists

I) Singly linked list II) Doubly linked list

**SINGLY LINKED LIST**

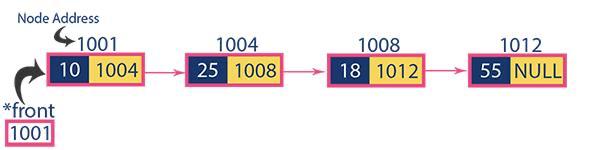
Single Linked list is a sequence of elements in which every element has link to its next element in the sequence.

In a Single linked list, the individual element is called as "NODE”.

Every Node contains two fields: Data Field and the Next Field.

Data Field—stores data

Next Field—stores address of next node



OPERATIONS ON SINGLE LINKED LIST :-

* Insertion
* Deletion
* Display

**Inserting At Beginning of the list**

* Step 1 - Create anewNode with given value.
* Step 2 - Check whether list is **Empty (head == NULL)**
* Step 3 - If it is **Empty** then, set newNode→next = NULL and head = newNode.
* Step 4 - If it is **Not Empt**y then, **set newNode→next = head and head = newNode**.

# **Inserting At End of the list**

* Step 1 - Create a newNode with given value and **newNode → next** as **NULL**.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**).
* Step 3 - If it is **Empty** then, set **head** = **newNode**.
* Step 4 - If it is **Not Empty** then, define a node pointer temp and initialize with head.
* Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).
* Step 6 - Set **temp → next** = **newNode**.

# **Inserting At Specific location in the list (After a Node)**

* Step 1 - Create a newNode with given value.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step3- If it is **Empty** then, set **newNode → next** = **NULL** and **head** = **newNode**.
* Step 4 - If it is **Not Empty** then, define a node pointer temp and initialize with head.
* Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until **temp1 → data** is equal to location, here location is the node value after which we want to insert the newNode).
* Step 6 - Every time check whether temp is reached to last node or not. If it is reached to last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp** to next node.
* Step 7 - Finally, Set '**newNode → next** = **temp → next**' and '**temp → next** = **newNode**'

# **Deleting from Beginning of the list**

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is **Not Empty** then, define a Node pointer 'temp**'** and initialize with head.
* Step 4 - Check whether list is having only one node (**temp → next** == **NULL**)
* Step 5 - If it is **TRUE** then set **head** = **NULL** and delete temp.
* Step 6 - If it is **FALSE** then set **head** = **temp → next**, and delete temp.

# **Deleting from End of the list**

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is **Not Empty** then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with **head**.
* Step 4 - Check whether list has only one Node (**temp1 → next** == **NULL**)
* Step 5 - If it is **TRUE**. Then, set **head** = **NULL** and delete **temp1**. And terminate the function.
* Step 6 - If it is **FALSE**. Then, set '**temp2 = temp1** ' and move temp1 to its next node. Repeat the same until it reaches to the last node in the list. (until **temp1 → next** == **NULL**)
* Step 7 - Finally, Set **temp2 → next** = **NULL** and delete temp1.

# **Deleting a Specific Node from the list**

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is **Not Empty** then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
* Step 4 - Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set '**temp2 = temp1**' before moving the 'temp1' to its next node.
* Step 5 - If it is reached to the last node then display **'Given node not found in the list! Deletion not possible!!!'**. And terminate the function.
* Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
* Step 7 - If list has only one node and that is the node to be deleted, then set **head** = **NULL** and delete temp1 (free(temp1)).
* Step 8 - If list contains multiple nodes, then check whether temp1 is the first node in the list (**temp1 == head**).
* Step 9 - If temp1 is the first node then move the **head** to the next node (**head = head → next**) and delete temp1.
* Step 10 - If temp1 is not first node then check whether it is last node in the list (**temp1 → next == NULL**).
* Step 11 - If temp1 is last node then set **temp2 → next** = **NULL** and delete temp1 (free(temp1)).
* Step 12 - If temp1 is not first node and not last node then set **temp2 → next** = **temp1 → next** and delete temp1 (free(temp1)).

# **Displaying a Single Linked List**

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!!'** and terminate the function.
* Step 3 - If it is **Not Empty** then, define a Node pointer 'temp' and initialize with **head**.
* Step 4 - Keep displaying **temp → data** with an arrow (**--->**) until temp reaches to the last node
* Step 5 - Finally display **temp → data** with arrow pointing to **NULL** (**temp → data ---> NULL**).

**Program :**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Node

{

public:

int info;

Node\* next;

};

class List:public Node

{

Node \*first,\*last;

public:

List()

{

first=NULL;

last=NULL;

}

void create();

void insert();

void delet();

void display();

void search();

};

void List::create()

{

Node \*temp;

temp=new Node;

int n;

cout<<"\nEnter an Element:";

cin>>n;

temp->info=n;

temp->next=NULL;

if(first==NULL)

{

first=temp;

last=first;

}

else

{

last->next=temp;

last=temp;

}

}

void List::insert()

{

Node \*prev,\*cur;

prev=NULL;

cur=first;

int count=1,pos,ch,n;

Node \*temp=new Node;

cout<<"\nEnter an Element:";

cin>>n;

temp->info=n;

temp->next=NULL;

cout<<"\nINSERT AS\n1:FIRSTNODE\n2:LASTNODE\n3:IN BETWEEN FIRST&LAST NODES";

cout<<"\nEnter Your Choice:";

cin>>ch;

switch(ch)

{

case 1:

temp->next=first;

first=temp;

break;

case 2:

last->next=temp;

last=temp;

break;

case 3:

cout<<"\nEnter the Position to Insert:";

cin>>pos;

while(count!=pos)

{

prev=cur;

cur=cur->next;

count++;

}

if(count==pos)

{

prev->next=temp;

temp->next=cur;

}

else

cout<<"\nNot Able to Insert";

break;

}

}

void List::delet()

{

Node \*prev=NULL,\*cur=first;

int count=1,pos,ch;

cout<<"\nDELETE\n1:FIRSTNODE\n2:LASTNODE\n3:IN BETWEEN FIRST&LAST NODES";

cout<<"\nEnter Your Choice:";

cin>>ch;

switch(ch)

{

case 1:

if(first!=NULL)

{

cout<<"\nDeleted Element is "<<first->info;

first=first->next;

}

else

cout<<"\nNot Able to Delete";

break;

case 2:

while(cur!=last)

{

prev=cur;

cur=cur->next;

}

if(cur==last)

{

cout<<"\nDeleted Element is: "<<cur->info;

prev->next=NULL;

last=prev;

}

else

cout<<"\nNot Able to Delete";

break;

case 3:

cout<<"\nEnter the Position of Deletion:";

cin>>pos;

while(count!=pos)

{

prev=cur;

cur=cur->next;

count++;

}

if(count==pos)

{

cout<<"\nDeleted Element is: "<<cur->info;

prev->next=cur->next;

}

else

cout<<"\nNot Able to Delete";

break;

}

}

void List::display()

{

Node \*temp=first;

if(temp==NULL)

{

cout<<"\nList is Empty";

}

while(temp!=NULL)

{

cout<<temp->info;

cout<<"-->";

temp=temp->next;

}

cout<<"NULL";

}

void List::search()

{

int value,pos=0;

bool flag=false;

if(first==NULL)

{

cout<<"List is Empty";

return;

}

cout<<"Enter the Value to be Searched:";

cin>>value;

Node \*temp;

temp=first;

while(temp!=NULL)

{

pos++;

if(temp->info==value)

{

flag=true;

cout<<"Element"<<value<<"is Found at "<<pos<<" Position";

return;

}

temp=temp->next;

}

if(!flag)

{

cout<<"Element "<<value<<" not Found in the List";

}

}

int main()

{

List l;

int ch;

while(1)

{

cout<<"\n\*\* MENU \*\*";

cout<<"\n1:CREATE\n2:INSERT\n3:DELETE\n4:SEARCH\n5:DISPLAY\n6:EXIT\n";

cout<<"\nEnter Your Choice:";

cin>>ch;

switch(ch)

{

case 1:

l.create();

break;

case 2:

l.insert();

break;

case 3:

l.delet();

break;

case 4:

l.search();

break;

case 5:

l.display();

break;

case 6:

return 0;

}

}

return 0;

}

**Sample Output :**

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:1

Enter an Element:45

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:5

45-->NULL

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:1

Enter an Element:2

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:2

Enter an Element:54

INSERT AS

1:FIRSTNODE

2:LASTNODE

3:IN BETWEEN FIRST&LAST NODES

Enter Your Choice:2

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:66

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:5

45-->2-->54-->NULL

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:3

DELETE

1:FIRSTNODE

2:LASTNODE

3:IN BETWEEN FIRST&LAST NODES

Enter Your Choice:1

Deleted Element is 45

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:1

Enter an Element:54

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:5

54-->NULL

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:2

Enter an Element:45

INSERT AS

1:FIRSTNODE

2:LASTNODE

3:IN BETWEEN FIRST&LAST NODES

Enter Your Choice:1

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:2

Enter an Element:89

INSERT AS

1:FIRSTNODE

2:LASTNODE

3:IN BETWEEN FIRST&LAST NODES

Enter Your Choice:2

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:5

45-->54-->89-->NULL

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:3

DELETE

1:FIRSTNODE

2:LASTNODE

3:IN BETWEEN FIRST&LAST NODES

Enter Your Choice:1

Deleted Element is 45

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:5

54-->89-->NULL

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:2

Enter an Element:65

INSERT AS

1:FIRSTNODE

2:LASTNODE

3:IN BETWEEN FIRST&LAST NODES

Enter Your Choice:1

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

5:DISPLAY

6:EXIT

Enter Your Choice:4

Enter the Value to be Searched:65

Element65is Found at 1 Position

\* MENU \*

1:CREATE

2:INSERT

3:DELETE

4:SEARCH

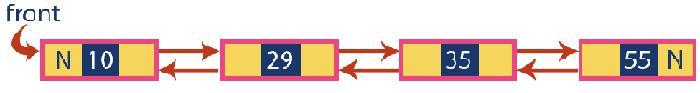
5:DISPLAY

6:EXIT

**DOUBLY LINKED LIST**

Doubly linked list is a sequence of elements in which every element has links to its previous element and its next element in the sequence.

In this list, every node has a link to its previous node and next node. So, we can traverse forward by using the next field and can traverse backward by using the previous field.

Every node in a doubly linked list contains three fields: - Data, a pointer to next node, and a pointer to previous node.

OPERATIONS ON DOUBLE LINKED LIST :-

* Insertion
* Deletion
* Display

# **Inserting At Beginning of the list**

* Step 1 - Create a newNode with given value and **newNode → previous** as **NULL**.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step 3 - If it is **Empty** then, assign **NULL** to **newNode → next** and **newNode** to **head**.
* Step 4 - If it is **not Empty** then, assign **head** to **newNode → next** and **newNode** to **head**.

# **Inserting At End of the list**

* Step 1 - Create a newNode with given value and **newNode → next** as **NULL**.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step 3 - If it is **Empty**, then assign **NULL** to **newNode → previous** and **newNode** to **head**.
* Step 4 - If it is **not Empty**, then, define a node pointer **temp** and initialize with **head**.
* Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).
* Step 6 - Assign **newNode** to **temp → next** and **temp** to **newNode → previous**.

# **Inserting At Specific location in the list (After a Node)**

* Step 1 - Create a newNode with given value.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step 3 - If it is **Empty** then, assign **NULL** to both **newNode → previous** & **newNode → next** and set **newNode** to **head**.
* Step 4 - If it is **not Empty** then, define two node pointers temp1 & temp2 and initialize temp1 with head.
* Step 5 - Keep moving the temp1 to its next node until it reaches to the node after which we want to insert the newNode (until **temp1 → data** is equal to location, here location is the node value after which we want to insert the newNode).
* Step 6 - Every time check whether temp1 is reached to the last node. If it is reached to the last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the temp1 to next node.
* Step 7 - Assign **temp1 → next** to **temp2**, **newNode** to **temp1 → next**, **temp1** to **newNode → previous**, **temp2** to **newNode → next** and **newNode** to **temp2 → previous**.

# **Deleting from Beginning of the list**

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is not Empty then, define a Node pointer 'temp' and initialize with head.
* Step 4 - Check whether list is having only one node (**temp → previous** is equal to **temp → next**)
* Step 5 - If it is **TRUE**, then set **head** to **NULL** and delete **temp** (Setting **Empty** list conditions)
* Step 6 - If it is **FALSE**, then assign **temp → next** to **head**, **NULL** to **head → previous** and delete **temp**.

# **Deleting from End of the list**

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty**, then display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is not Empty then, define a Node pointer **'temp'** and initialize with **head**.
* Step 4 - Check whether list has only one Node (**temp → previous** and **temp → next** both are **NULL**)
* Step 5 - If it is **TRUE**, then assign **NULL** to **head** and delete **temp**. And terminate from the function. (Setting **Empty** list condition)
* Step 6 - If it is **FALSE**, then keep moving **temp** until it reaches to the last node in the list. (until **temp → next** is equal to **NULL**)
* Step 7 - Assign **NULL** to **temp → previous → next** and delete **temp**.

# **Deleting a Specific Node from the list**

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is not Empty, then define a Node pointer 'temp' and initialize with head.
* Step 4 - Keep moving the temp until it reaches to the exact node to be deleted or to the last node.
* Step 5 - If it is reached to the last node, then display **'Given node not found in the list! Deletion not possible!!!'** and terminate the function.
* Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
* Step 7 - If list has only one node and that is the node which is to be deleted then set **head** to **NULL** and delete **temp** (**free(temp)**).
* Step 8 - If list contains multiple nodes, then check whether temp is the first node in the list (**temp == head**).
* Step 9 - If temp is the first node, then move the head to the next node (**head = head → next**), set **head** of **previous** to **NULL** (**head → previous = NULL**) and delete **temp**.
* Step 10 - If temp is not the first node, then check whether it is the last node in the list (**temp → next == NULL**).
* Step 11 - If temp is the last node then set **temp** of **previous** of **next** to **NULL** (**temp → previous → next = NULL**) and delete **temp** (**free(temp**)).
* Step 12 - If **temp** is not the first node and not the last node, then set **temp** of **previous** of **next** to **temp** of **next** (**temp → previous → next = temp → next**), **temp** of **next** of **previous** to **temp** of **previous** (**temp → next → previous = temp → previous**) and delete **temp** (**free(temp)**).

# **Displaying a Double Linked List**

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty**, then display **'List is Empty!!!'** and terminate the function.
* Step 3 - If it is not Empty, then define a Node pointer 'temp' and initialize with head.
* Step 4 - Display **'NULL <--- '**.
* Step 5 - Keep displaying **temp → data** with an arrow (**<===>**) until temp reaches to the last node
* Step 6 - Finally, display **temp → data** with arrow pointing to **NULL** (**temp → data ---> NULL**).

**Program :**

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node

{

struct node \*prev\_node;

int info;

struct node \*next\_node;

};

struct node \*create\_list(struct node \*begin);

void display(struct node \*begin);

struct node \*addtoemptylist(struct node \*begin,int data\_element);

struct node \*addatbeglist(struct node \*begin,int data\_element);

struct node \*addatendlist(struct node \*begin,int data\_element);

struct node \*addafterlist(struct node \*begin,int data\_element,int item\_pos);

struct node \*addbeforelist(struct node \*begin,int data\_element,int item\_pos);

struct node \*deletenode(struct node \*begin,int data\_element);

struct node \*reverselist(struct node \*begin);

int main()

{

int option,data\_element,item\_pos;

struct node \*begin=NULL;

while(1)

{

cout<<"\n1.Create A New Doubly Linked List\n";

cout<<"2.Display the Doubly Linked List\n";

cout<<"3.Add to an Empty Doubly Linked List\n";

cout<<"4.Add at Starting of the Doubly Linked List\n";

cout<<"5.Add at Ending\n";

cout<<"6.Add After a Node\n";

cout<<"7.Add Before a Node\n";

cout<<"8.Delete a Node\n";

cout<<"9.Reverse the Doubly Linked List\n";

cout<<"10.Exit\n";

cout<<"Enter your option : ";

cin>>option;

switch(option)

{

case 1:

begin=create\_list(begin);

break;

case 2:

display(begin);

break;

case 3:

cout<<"Enter the element:";

cin>>data\_element;

begin=addtoemptylist(begin,data\_element);

break;

case 4:

cout<<"Enter the element:";

cin>>data\_element;

begin=addatbeglist(begin,data\_element);

break;

case 5:

cout<<"Enter the element:";

cin>>data\_element;

begin=addatendlist(begin,data\_element);

break;

case 6:

cout<<"Enter the element:";

cin>>data\_element;

cout<<"Enter the element after which to insert : ";

cin>>item\_pos;

begin=addafterlist(begin,data\_element,item\_pos);

break;

case 7:

cout<<"Enter the element: ";

cin>>data\_element;

cout<<"Enter the element before which to insert : ";

cin>>item\_pos;

begin=addbeforelist(begin,data\_element,item\_pos);

break;

case 8:

cout<<"Enter the element to be Deleted : ";

cin>>data\_element;

begin=deletenode(begin,data\_element);

break;

case 9:

begin=reverselist(begin);

break;

case 10:

exit(1);

default:

cout<<"Wrong option\n";

}

}

return 0;

}

struct node \*create\_list(struct node \*begin)

{

int i,n,data\_element;

cout<<"Enter the number of nodes : ";

cin>>n;

begin=NULL;

if(n==0)

return begin;

cout<<"Enter the element: ";

cin>>data\_element;

begin=addtoemptylist(begin,data\_element);

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

{

cout<<"Enter the element to be inserted : ";

cin>>data\_element;

begin=addatendlist(begin,data\_element);

}

return begin;

}

void display(struct node \*begin)

{

struct node \*p;

if(begin==NULL)

{

cout<<"List is empty\n";

return;

}

p=begin;

cout<<"List is :\n";

while(p!=NULL)

{

cout<<p->info<<" ";

p=p->next\_node;

}

cout<<"\n";

}

struct node \*addtoemptylist(struct node \*begin,int data\_element)

{

struct node \*temp;

temp=new struct node;

temp->info=data\_element;

temp->prev\_node=NULL;

temp->next\_node=NULL;

begin=temp;

return begin;

}

struct node \*addatbeglist(struct node \*begin,int data\_element)

{

if(begin==NULL)

{

cout<<"List is empty\n";

return begin;

}

struct node \*temp;

temp = new struct node;

temp->info=data\_element;

temp->prev\_node=NULL;

temp->next\_node=begin;

begin->prev\_node=temp;

begin=temp;

return begin;

}

struct node \*addatendlist(struct node \*begin,int data\_element)

{

if(begin==NULL)

{

cout<<"List is empty\n";

return begin;

}

struct node \*temp,\*p;

temp=new struct node;

temp->info=data\_element;

p=begin;

while(p->next\_node!=NULL)

p=p->next\_node;

p->next\_node=temp;

temp->next\_node=NULL;

temp->prev\_node=p;

return begin;

}

struct node \*addafterlist(struct node \*begin,int data\_element,int item\_pos)

{

struct node \*temp,\*p;

temp=new struct node;

temp->info=data\_element;

p=begin;

while(p!=NULL)

{

if(p->info==item\_pos)

{

temp->prev\_node=p;

temp->next\_node=p->next\_node;

if(p->next\_node!=NULL)

p->next\_node->prev\_node=temp;

p->next\_node=temp;

return begin;

}

p=p->next\_node;

}

cout<<item\_pos<<" not present in the list\n\n";

return begin;

}

struct node \*addbeforelist(struct node \*begin,int data\_element,int item\_pos)

{

struct node \*temp,\*q;

if(begin==NULL )

{

cout<<"List is empty\n";

return begin;

}

if(begin->info==item\_pos)

{

temp = new struct node;

temp->info=data\_element;

temp->prev\_node=NULL;

temp->next\_node=begin;

begin->prev\_node=temp;

begin=temp;

return begin;

}

q=begin;

while(q!=NULL)

{

if(q->info==item\_pos)

{

temp=new struct node;

temp->info=data\_element;

temp->prev\_node=q->prev\_node;

temp->next\_node = q;

q->prev\_node->next\_node=temp;

q->prev\_node=temp;

return begin;

}

q=q->next\_node;

}

cout<<item\_pos<<" not present in the list\n";

return begin;

}

struct node \*deletenode(struct node \*begin,int data\_element)

{

struct node \*temp;

if(begin==NULL)

{

cout<<"List is empty\n";

return begin;

}

if(begin->next\_node==NULL)

if(begin->info==data\_element)

{

temp=begin;

begin=NULL;

delete(temp);

return begin;

}

else

{

cout<<"Element "<<data\_element<<" not found\n";

return begin;

}

if(begin->info==data\_element)

{

temp=begin;

begin=begin->next\_node;

begin->prev\_node=NULL;

delete(temp);

return begin;

}

temp=begin->next\_node;

while(temp->next\_node!=NULL )

{

if(temp->info==data\_element)

{

temp->prev\_node->next\_node=temp->next\_node;

temp->next\_node->prev\_node=temp->prev\_node;

delete(temp);

return begin;

}

temp=temp->next\_node;

}

if(temp->info==data\_element)

{

temp->prev\_node->next\_node=NULL;

delete(temp);

return begin;

}

cout<<"Element "<<data\_element<<" not found\n";

return begin;

}

struct node \*reverselist(struct node \*begin)

{

if(begin==NULL)

{

cout<<"List is empty\n";

return begin;

}

struct node \*p1,\*p2;

p1=begin;

p2=p1->next\_node;

p1->next\_node=NULL;

p1->prev\_node=p2;

while(p2!=NULL)

{

p2->prev\_node=p2->next\_node;

p2->next\_node=p1;

p1=p2;

p2=p2->prev\_node;

}

begin=p1;

cout<<"List reverselistd\n";

return begin;

}

**Sample Output** :

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 1

Enter the number of nodes : 5

Enter the element: 54

Enter the element to be inserted : 89

Enter the element to be inserted : 56

Enter the element to be inserted : 75

Enter the element to be inserted : 65

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 2

List is :

54 89 56 75 65

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 3

Enter the element:43

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 2

List is :

43

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 1

Enter the number of nodes : 5

Enter the element: 45

Enter the element to be inserted : 58

Enter the element to be inserted : 79

Enter the element to be inserted : 20

Enter the element to be inserted : 10

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 2

List is :

45 58 79 20 10

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 5

Enter the element:15

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 2

List is :

45 58 79 20 10 15

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 5

Enter the element:13

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 2

List is :

45 58 79 20 10 15 13

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 8

Enter the element to be Deleted : 79

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 2

List is :

45 58 20 10 15 13

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 9

List reverselistd

1.Create A New Doubly Linked List

2.Display the Doubly Linked List

3.Add to an Empty Doubly Linked List

4.Add at Starting of the Doubly Linked List

5.Add at Ending

6.Add After a Node

7.Add Before a Node

8.Delete a Node

9.Reverse the Doubly Linked List

10.Exit

Enter your option : 2

List is :

13 15 10 20 58 45