**Question1.cpp**

```cpp
// <----Lab 04 - Doubly and Circular Linked List---->

// Q1. Create a doubly link list and perform the mentioned tasks.
// a. Insert a new node at the end of the list.
// b. Insert a new node at the beginning of list.
// c. Insert a new node at given position.
// d. Delete any node.
// e. Print the complete doubly link list.


#include<iostream>
using namespace std;

class node {
    public:
        int data;
        node* nextnode;   //to point to node after it
        node* prevnode;  //to point to node before it
        node() {
            data=0;
            nextnode=NULL;
            prevnode=NULL;
        }
        node(int value) {
            data=value;
            nextnode=NULL;
            prevnode=NULL;
        }
        node(int value, node* nn, node* pn) {
            data=value;
            nextnode=nn;
            prevnode=pn;
        }
};

class DLL {
        int nodecount=0;
        node* head=NULL;
    public:
        void insertAttail(int value) {
            if(head==NULL) {  // if list was empty
                node* n=new node(value);
                head=n;
                nodecount++;
                return;
            }
            node* temp=head;
            while(temp->nextnode!=NULL) {
                temp=temp->nextnode;
            }
            node* n=new node(value,NULL,temp);
            temp->nextnode=n;
            nodecount++;
```

```
54          }
55          void insertAthead(int value) {
56              node* n=new node(value,head,NULL);
57              if(head!=NULL){
58                  head->prevnode=n;
59              }
60              head=n;
61              nodecount++;
62          }
63          void insertAtPos(int pos,int value) {
64              if(pos<0){
65                  cout<<"Position less than 0, Inserting at head.\n";
66                  insertAthead(value);
67                  return;
68              }
69              if(pos>nodecount-1){
70                  cout<<"Position more than nodes in list, Inserting at tail.\n";
71                  insertAttail(value);
72                  return;
73              }
74              int count=0;
75              node* temp=head;
76              while(temp->nextnode!=NULL && count<pos-1) {
77                  temp=temp->nextnode;
78                  count++;
79              }
80              node* n=new node(value,temp->nextnode,temp);
81              temp->nextnode=n;
82              n->nextnode->prevnode=n;
83              nodecount++;
84          }
85          void display() {
86              node* temp=head;
87              cout<<"HEAD | ";
88              while(temp!=NULL) {
89                  cout<<" <--"<<temp->prevnode<<"  |  "<<temp->data<<"  |  "<<temp->nextnode<<"
    --> ";
90                  temp=temp->nextnode;
91              }
92              cout<<"| TAIL"<<endl;
93          }
94  //Assuming ANY node means any of the 4 types (head,tail,position,value)
95          void deleteAtHead() {
96              if(head==NULL) {
97                  cout<<"Empty Linked List, Returning"<<endl;
98                  return;
99              }
100             node* todelete=head;
101             head=head->nextnode;
102             head->prevnode=NULL;
103             delete todelete;
104             nodecount--;
105         }
106         void deletion(int value) {
107             if(head==NULL) {
108                 cout<<"Empty Linked List, Returning"<<endl;
```

```
109                 return;
110             }
111             node* temp=head;
112             if(head->data==value) {
113                 deleteAtHead();
114                 return;
115             }
116             while(temp->data!=value) {
117                 if(temp->nextnode==NULL) {
118                     cout<<"Value not found, Returning\n";
119                     return;
120                 }
121                 temp=temp->nextnode;
122             }
123             if(temp->nextnode==NULL){
124                 deleteAtTail();
125                 return;
126             }
127             node* todelete=temp;
128             temp->prevnode->nextnode=temp->nextnode;
129             temp->nextnode->prevnode=temp->prevnode;
130             delete todelete;
131             nodecount--;
132         }
133         void deleteAtPos(int pos) {
134             if(pos<0){
135                 cout<<"Position less than zero, INVALID. Returning..."<<endl;
136                 return;
137             }
138             if(pos==0){
139                 deleteAtHead();
140                 return;
141             }
142             else if(pos==nodecount-1){
143                 deleteAtTail();
144                 return;
145             }
146             if(pos>nodecount-1){
147                 cout<<"Invalid Position, Returning"<<endl;
148                 return;
149             }
150             if(head==NULL) {
151                 cout<<"Empty Linked List, Returning"<<endl;
152                 return;
153             }
154             int count=0;
155             node* temp=head;
156             while(temp->nextnode!=NULL && count<pos-1) {
157                 temp=temp->nextnode;
158                 count++;
159             }
160             node* todelete=temp->nextnode;
161             temp->nextnode=temp->nextnode->nextnode;
162             temp->nextnode->prevnode=temp;
163             delete todelete;
164             nodecount--;
```

```
        }
        void deleteAtTail() {
            if(head==NULL) {
                cout<<"Empty Linked List, Returning"<<endl;
                return;
            }
            node* temp=head;
            while(temp->nextnode!=NULL) {
                temp=temp->nextnode;
            }
            node* todelete=temp;
            temp=temp->prevnode;
            temp->nextnode=NULL;
            delete todelete;
            nodecount--;
        }
};

int main() {
    DLL list;
    float input=0;
    int value;
    while(input!=0.5) {
        cout<<"-----------------------------------------------------\n";
        cout<<"CURRENT LINKED LIST:\n";
        list.display();
        cout<<"-----------------------------------------------------\n";
        cout<<"What would you like to do with the linked list?\n";
        cout<<"1. Insert\t2. Delete\nEnter 0.5 to Exit\n[Anything else will default to Delete]\n";
        cin>>input;
        if(input==1) {
            cout<<"Enter Value to insert: ";
            cin>>value;
            cout<<"Where to Insert in Linked List?\n";
            cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will default to Insertion at Head]\n";
            cin>>input;
            if(input == 2){
                list.insertAttail(value);
            }
            else if(input == 3){
                int pos;
                cout<<"Enter the Position to insert into: ";
                cin>>pos;
                list.insertAtPos(pos,value);
            }
            else{
                list.insertAthead(value);
            }

        }
        else if(input==0.5){
            break;
        }
        else{
            cout<<"Where to Delete from Linked List?\n";
```

```
            cout<<"1. At head\t2. At tail\t3. At specified Position\t 4. Delete a specific Value\n[Any other value will default to Deletion from Head]\n";
            cin>>input;
            if(input == 2){
                list.deleteAtTail();
            }
            else if(input == 3){
                int pos;
                cout<<"Enter the Position to Delete from: ";
                cin>>pos;
                list.deleteAtPos(pos);
            }
            else if(input == 4){
                int pos;
                cout<<"Enter the Value to Delete: ";
                cin>>value;
                list.deletion(value);
            }
            else{
                list.deleteAtHead();
            }
        }

    }
}
```