Question2.cpp

```cpp
// <---Lab 04 - Doubly and Circular Linked List--->
/*
2. Create two doubly link lists, say L and M . List L should contain all even elements from 2 to
10 and list M should contain all odd elements from 1 to 9. Create a new list N by
concatenating list L and M.
*/



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

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

```
165         void deleteAtTail() {
166             if(head==NULL) {
167                 cout<<"Empty Linked List, Returning"<<endl;
168                 return;
169             }
170             node* temp=head;
171             while(temp->nextnode!=NULL) {
172                 temp=temp->nextnode;
173             }
174             node* todelete=temp;
175             temp=temp->prevnode;
176             temp->nextnode=NULL;
177             delete todelete;
178             nodecount--;
179         }
180         void concatlist(DLL &obj){
181             node* temp=obj.head;
182             while(temp!=NULL){
183                 insertAttail(temp->data);
184                 temp=temp->nextnode;
185             }
186         }
187 };
188 
189 int main(){
190     DLL l,m;
191     for(int i=2;i<11;i+=2){  //Initializing L and M with evens and odds respectively
192         l.insertAttail(i);
193         m.insertAttail(i-1);
194     }
195     cout<<"-----------LIST L (Evens)-------------------"<<endl;
196     l.display();    //Showing L for surity of values
197     cout<<"-----------LIST M (odds)------------------------"<<endl;
198     m.display();    //Showing M for surity of values
199     cout<<"-----------------------------------"<<endl;
200     DLL n;      //Creates empty linked list
201     n.concatlist(l);  //concats L into N
202     n.concatlist(m);  //concats M into N
203     cout<<"-----------LIST N after concatenating L & M-------------------------"<<endl;
204     n.display();
205 }
```