

Question2.cpp

```
1 // <----Lab 04 - Doubly and Circular Linked List---->
2
3 // Q2. Create two doubly link lists, say L and M . List L should contain all even elements
4 //      10 and list M should contain all odd elements from 1 to 9. Create a new list N by
5 //      concatenating list L and M.
6
7 #include<iostream>
8 using namespace std;
9
10 class node {
11     public:
12         int data;
13         node* nextnode;    //to point to node after it
14         node* prevnode;    //to point to node before it
15         node() {
16             data=0;
17             nextnode=NULL;
18             prevnode=NULL;
19         }
20         node(int value) {
21             data=value;
22             nextnode=NULL;
23             prevnode=NULL;
24         }
25         node(int value, node* nn, node* pn) {
26             data=value;
27             nextnode=nn;
28             prevnode=pn;
29         }
30 };
31
32 class DLL {
33     int nodecount=0;
34     node* head=NULL;
35     public:
36         void insertAttail(int value) {
37             if(head==NULL) { // if array was empty
38                 node* n=new node(value);
39                 head=n;
40                 nodecount++;
41                 return;
42             }
43             node* temp=head;
44             while(temp->nextnode!=NULL) {
45                 temp=temp->nextnode;
46             }
47             node* n=new node(value,NULL,temp);
48             temp->nextnode=n;
49             nodecount++;
50         }
51         void insertAthead(int value) {
52             node* n=new node(value,head,NULL);
```

```

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

```

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

```

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

```