

Question5.cpp

```
1 // <---Lab 04 - Doubly and Circular Linked List-->
2 /*5. Break the above-created circular linked list into two halves.*/
3
4
5
6 #include<iostream>
7 using namespace std;
8
9 class node {
10 public:
11     int data;
12     node* next;
13     node(int value) {
14         data=value;
15         next=NULL;
16     }
17     node(int value,node* nxt) {
18         data=value;
19         next=nxt;
20     }
21 };
22
23 class CLL{
24     node* head=NULL;
25     int nodecount=0;
26 public:
27     void setlist(node* h,int n){
28         head=h;
29         nodecount=n;
30     }
31     void makehalves(CLL &obj){
32         int middleposition = (int)nodecount/2;    //Amount for 1st half
33         int secondhalf = nodecount-middleposition; //Amount for 2nd half
34         int count=0;
35         node* temp=head;
36         node* prev=temp;
37         while(temp->next!=head&&count<middleposition) { // prev is last element of first half,
temp is head of second half
38             prev=temp;
39             temp=temp->next;
40             count++;
41         }
42         prev->next=head; //detach prev from remaining half, circular link back to start
43         node* secondhead=temp;
44         while(temp->next!=head){
45             temp=temp->next;
46         }
47         temp->next=secondhead;
48         obj.setlist(secondhead,secondhalf);
49     }
50
51     void appendNode(int value){ //insert at end of list / tail
52         if(head==NULL) { //Check if Linked List Empty.
```

```
53         node* n=new node(value,n);
54         head=n;
55         head->next=head;
56         nodecount++;
57         return;
58     }
59     else{
60         node* temp=head;
61         while(temp->next!=head) {
62             temp=temp->next;
63         }
64         node* n=new node(value,head);
65         temp->next=n;
66         nodecount++;
67     }
68 }
69 void prependNode(int value){ //insert at start of list / head
70     if(head==NULL) { //Check if Linked List Empty.
71         node* n=new node(value,n);
72         head=n;
73         head->next=head;
74         nodecount++;
75         return;
76     }
77     node* n=new node(value,head);
78     node* temp=head;
79     while(temp->next!=head) {
80         temp=temp->next;
81     }
82     temp->next=n;
83     head=n;
84     nodecount++;
85 }
86 void insertNodeAfter(int posvalue,int value){ // insert at position (i+1)
87     if(head==NULL) { //Check if Linked List Empty.
88         cout<<"Empty List, adding at Head.\n";
89         node* n=new node(value,n);
90         head=n;
91         nodecount++;
92         return;
93     }
94     if(posvalue>nodecount-1){
95         cout<<"Position more than nodes in list, Inserting at tail.\n";
96         appendNode(value);
97         return;
98     }
99     int count=0;
100    node* temp=head;
101    while(temp->next!=head&&count<posvalue) {
102        temp=temp->next;
103        count++;
104    }
105    node* n=new node(value,temp->next);
106    temp->next=n;
107 }
108 void deleteathead(){
```

```
109     node* temp=head;
110     while(temp->next=head){
111         temp=temp->next;
112     }
113     temp->next=head->next;
114 }
115 void deleteattail(){
116     node* temp=head;
117     while(temp->next->next=head){
118         temp=temp->next;
119     }
120     temp->next=head;
121 }
122 void deleteNodeByKey(int value){ // delete by value
123     node* temp=head;
124     node* prev=temp;
125     while(temp->next!=head&&temp->data!=value) {
126         prev=temp;
127         temp=temp->next;
128     }
129     if(temp->data==value){
130         prev->next=temp->next; //skip temp (i.e delete)
131         delete temp;
132     }
133     if(temp->next==head){
134         cout<<"Value not in Linked List.\n";
135         return;
136     }
137 }
138 void updateNodeByKey(int value){ // update by value
139     node* temp=head;
140     while(temp->next!=head&&temp->data!=value) {
141         node* prev=temp;
142         temp=temp->next;
143     }
144     if(temp->data==value){
145         cout<<"Enter new value: ";
146         cin>>temp->data;
147     }
148     if(temp->next==head){
149         cout<<"Value not in Linked List.\n";
150         return;
151     }
152 }
153 void print(){
154     node* temp=head;
155     cout<<"[HEAD] ";
156     if(head!=NULL){
157         cout<<temp->data<<" | "<<temp->next<<" -> ";
158         temp=temp->next;
159         while(temp!=head) {
160             cout<<temp->data<<" | "<<temp->next<<" -> ";
161             temp=temp->next;
162         }
163         cout<<"head [TAIL]"<<endl;
164     }
```

```

165         else{
166             cout<<"NULL [TAIL]"<<endl;
167         }
168     }
169 };
170
171 int main(){
172     CLL list,list2;
173     float input=0;
174     int value;
175     while(input!=0.5) {
176         cout<<"-----\n";
177         cout<<"CURRENT LINKED LIST:\n";
178         list.print();
179         cout<<"-----\n";
180         cout<<"What would you like to do with the linked list?\n";
181         cout<<"1. Insert\t2. Delete\t3. Update\t4. Halve the Linked List [Halving will end the
program]\nEnter 0.5 to Exit\n[Anything else will default to Delete]\n";
182         cin>>input;
183         if(input==1) {
184             cout<<"Enter Value to insert: ";
185             cin>>value;
186             cout<<"Where to Insert in Linked List?\n";
187             cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will default
to Insertion at Head]\n";
188             cin>>input;
189             if(input == 2){
190                 list.appendNode(value);
191             }
192             else if(input == 3){
193                 int pos;
194                 cout<<"Enter the Position to insert After: ";
195                 cin>>pos;
196                 list.insertNodeAfter(pos,value);
197             }
198             else{
199                 list.prependNode(value);
200             }
201
202         }
203         else if(input==0.5){
204             break;
205         }
206         else if(input==3){
207             cout<<"Enter the Value to Update: ";
208             cin>>value;
209             list.updateNodeByKey(value);
210         }
211         else if(input==4){
212             cout<<"Halving list"<<endl;
213             list.makehalves(list2);
214             cout<<"First Half list-----"<<endl;
215             list.print();
216             cout<<"Second Half list-----"<<endl;
217             list2.print();
218             return 0;
219         }

```

```
220     else{
221         cout<<"Where to Delete from Linked List?\n";
222         cout<<"1. At head\t2. At tail\t3. Delete a specific Value\n[Any other value will
default to Deletion from Head]\n";
223         cin>>input;
224         if(input == 2){
225             list.deleteattail();
226         }
227         else if(input == 3){
228             int pos;
229             cout<<"Enter the Value to Delete: ";
230             cin>>value;
231             list.deleteNodeByKey(value);
232         }
233         else{
234             list.deleteathead();
235         }
236     }
237 }
238 }
239 }
```

Question1.cpp

```
1  // <---Lab 04 - Doubly and Circular Linked List--->
2  // 1. Create a doubly link list and perform the mentioned tasks.
3  // a. Insert a new node at the end of the list.
4  // b. Insert a new node at the beginning of list.
5  // c. Insert a new node at given position.
6  // d. Delete any node.
7  // e. Print the complete doubly link list.
8
9
10 #include<iostream>
11 using namespace std;
12
13 class node {
14     public:
15         int data;
16         node* nextnode; //to point to node after it
17         node* prevnode; //to point to node before it
18         node() {
19             data=0;
20             nextnode=NULL;
21             prevnode=NULL;
22         }
23         node(int value) {
24             data=value;
25             nextnode=NULL;
26             prevnode=NULL;
27         }
28         node(int value, node* nn, node* pn) {
29             data=value;
30             nextnode=nn;
31             prevnode=pn;
32         }
33 };
34
35 class DLL {
36     int nodecount=0;
37     node* head=NULL;
38     public:
39     void insertAttail(int value) {
40         if(head==NULL) { // if list was empty
41             node* n=new node(value);
42             head=n;
43             nodecount++;
44             return;
45         }
46         node* temp=head;
47         while(temp->nextnode!=NULL) {
48             temp=temp->nextnode;
49         }
50         node* n=new node(value,NULL,temp);
51         temp->nextnode=n;
52         nodecount++;
53     }
```

```

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 };
181
182 int main() {
183     DLL list;
184     float input=0;
185     int value;
186     while(input!=0.5) {
187         cout<<"-----\n";
188         cout<<"CURRENT LINKED LIST:\n";
189         list.display();
190         cout<<"-----\n";
191         cout<<"What would you like to do with the linked list?\n";
192         cout<<"1. Insert\t2. Delete\nEnter 0.5 to Exit\n[Anything else will default to Delete]\n";
193         cin>>input;
194         if(input==1) {
195             cout<<"Enter Value to insert: ";
196             cin>>value;
197             cout<<"Where to Insert in Linked List?\n";
198             cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will default
to Insertion at Head]\n";
199             cin>>input;
200             if(input == 2){
201                 list.insertAttail(value);
202             }
203             else if(input == 3){
204                 int pos;
205                 cout<<"Enter the Position to insert into: ";
206                 cin>>pos;
207                 list.insertAtPos(pos,value);
208             }
209             else{
210                 list.insertAthead(value);
211             }
212         }
213         else if(input==0.5){
214             break;
215         }
216         else{
217             cout<<"Where to Delete from Linked List?\n";
218             cout<<"1. At head\t2. At tail\t3. At specified Position\t4. Delete a specific
Value\n[Any other value will default to Deletion from Head]\n";

```

```
220         cin>>input;
221         if(input == 2){
222             list.deleteAtTail();
223         }
224         else if(input == 3){
225             int pos;
226             cout<<"Enter the Position to Delete from: ";
227             cin>>pos;
228             list.deleteAtPos(pos);
229         }
230         else if(input == 4){
231             int pos;
232             cout<<"Enter the Value to Delete: ";
233             cin>>value;
234             list.deletion(value);
235         }
236         else{
237             list.deleteAtHead();
238         }
239     }
240 }
241 }
242 }
```

Question2.cpp

```
1 // <---Lab 04 - Doubly and Circular Linked List--->
2 /*
3 2. Create two doubly link lists, say L and M . List L should contain all even elements from 2 to
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
8
9
10 #include<iostream>
11 using namespace std;
12
13 class node {
14     public:
15         int data;
16         node* nextnode; //to point to node after it
17         node* prevnode; //to point to node before it
18         node() {
19             data=0;
20             nextnode=NULL;
21             prevnode=NULL;
22         }
23         node(int value) {
24             data=value;
25             nextnode=NULL;
26             prevnode=NULL;
27         }
28         node(int value, node* nn, node* pn) {
29             data=value;
30             nextnode=nn;
31             prevnode=pn;
32         }
33 };
34
35 class DLL {
36     int nodecount=0;
37     node* head=NULL;
38     public:
39     void insertAttail(int value) {
40         if(head==NULL) { // if array was empty
41             node* n=new node(value);
42             head=n;
43             nodecount++;
44             return;
45         }
46         node* temp=head;
47         while(temp->nextnode!=NULL) {
48             temp=temp->nextnode;
49         }
50         node* n=new node(value,NULL,temp);
51         temp->nextnode=n;
52         nodecount++;
53     }
```

```

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); //concat L into N
202     n.concatlist(m); //concat M into N
203     cout<<"-----LIST N after concatenating L & M-----"<<endl;
204     n.display();
205 }

```

Question3.cpp

```
1  // <---Lab 04 - Doubly and Circular Linked List-->
2  /*3. Using the above created list N, sort the contents of list N in descending order.*/
3
4  #include<iostream>
5  using namespace std;
6
7  class node {
8      public:
9          int data;
10         node* nextnode; //to point to node after it
11         node* prevnode; //to point to node before it
12         node() {
13             data=0;
14             nextnode=NULL;
15             prevnode=NULL;
16         }
17         node(int value) {
18             data=value;
19             nextnode=NULL;
20             prevnode=NULL;
21         }
22         node(int value, node* nn, node* pn) {
23             data=value;
24             nextnode=nn;
25             prevnode=pn;
26         }
27     };
28
29     class DLL {
30         int nodecount=0;
31         node* head=NULL;
32     public:
33         void insertAttail(int value) {
34             if(head==NULL) { // if array was empty
35                 node* n=new node(value);
36                 head=n;
37                 nodecount++;
38                 return;
39             }
40             node* temp=head;
41             while(temp->nextnode!=NULL) {
42                 temp=temp->nextnode;
43             }
44             node* n=new node(value,NULL,temp);
45             temp->nextnode=n;
46             nodecount++;
47         }
48         void insertAthead(int value) {
49             node* n=new node(value,head,NULL);
50             if(head!=NULL){
51                 head->prevnode=n;
52             }
53             head=n;
```

```

54         nodecount++;
55     }
56     void insertAtPos(int pos,int value) {
57         if(pos>nodecount-1){
58             cout<<"Position more than nodes in list, Inserting at tail.\n";
59             insertAttail(value);
60             return;
61         }
62         int count=0;
63         node* temp=head;
64         while(temp->nextnode!=NULL && count<pos-1) {
65             temp=temp->nextnode;
66             count++;
67         }
68         node* n=new node(value,temp->nextnode,temp);
69         temp->nextnode=n;
70         n->nextnode->prevnode=n;
71         nodecount++;
72     }
73     void display() {
74         node* temp=head;
75         cout<<"HEAD | ";
76         while(temp!=NULL) {
77             cout<<" <--"<<temp->prevnode<<" | "<<temp->data<<" | "<<temp->nextnode<<"--> ";
78             temp=temp->nextnode;
79         }
80         cout<<"| TAIL"<<endl;
81     }
82     //Assuming ANY node means any of the 4 types (head,tail,position,value)
83     void deleteAtHead() {
84         if(head==NULL) {
85             cout<<"Empty Linked List, Returning"<<endl;
86             return;
87         }
88         node* todelete=head;
89         head=head->nextnode;
90         head->prevnode=NULL;
91         delete todelete;
92         nodecount--;
93     }
94     void deletion(int value) {
95         if(head==NULL) {
96             cout<<"Empty Linked List, Returning"<<endl;
97             return;
98         }
99         node* temp=head;
100        if(head->data==value) {
101            deleteAtHead();
102            return;
103        }
104        while(temp->data!=value) {
105            if(temp->nextnode==NULL) {
106                cout<<"Value not found, Returning\n";
107                return;
108            }

```



```
109         temp=temp->nextnode;
110     }
111     if(temp->nextnode==NULL){
112         deleteAtTail();
113         return;
114     }
115     node* todelete=temp;
116     temp->prevnode->nextnode=temp->nextnode;
117     temp->nextnode->prevnode=temp->prevnode;
118     delete todelete;
119     nodecount--;
120 }
121 void deleteAtPos(int pos) {
122     if(pos==0){
123         deleteAtHead();
124         return;
125     }
126     else if(pos==nodecount-1){
127         deleteAtTail();
128         return;
129     }
130     if(pos>nodecount-1){
131         cout<<"Invalid Position, Returning"<<endl;
132         return;
133     }
134     if(head==NULL) {
135         cout<<"Empty Linked List, Returning"<<endl;
136         return;
137     }
138     int count=0;
139     node* temp=head;
140     while(temp->nextnode!=NULL && count<pos-1) {
141         temp=temp->nextnode;
142         count++;
143     }
144     node* todelete=temp->nextnode;
145     temp->nextnode=temp->nextnode->nextnode;
146     temp->nextnode->prevnode=temp;
147     delete todelete;
148     nodecount--;
149 }
150 void deleteAtTail() {
151     if(head==NULL) {
152         cout<<"Empty Linked List, Returning"<<endl;
153         return;
154     }
155     node* temp=head;
156     while(temp->nextnode!=NULL) {
157         temp=temp->nextnode;
158     }
159     node* todelete=temp;
160     temp=temp->prevnode;
161     temp->nextnode=NULL;
162     delete todelete;
163     nodecount--;
164 }
```

```

165     void concatlist(DLL &obj){
166         node* temp=obj.head;
167         while(temp!=NULL){
168             insertAttail(temp->data);
169             temp=temp->nextnode;
170         }
171     }
172     void sortlist(){          //sorting by data , bubble sort
173         node* temp=head;
174         while(temp->nextnode!=NULL) {
175             node* temp2=head;
176             while(temp2->nextnode!=NULL) {
177                 if(temp2->data<temp2->nextnode->data) {
178                     char tempchar = temp2->data;
179                     temp2->data = temp2->nextnode->data;
180                     temp2->nextnode->data = tempchar;
181                 }
182                 temp2=temp2->nextnode;
183             }
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
192         l.insertAttail(i);
193         m.insertAttail(i-1);
194     }
195     cout<<"-----LIST L (Evens)-----"<<endl;
196     l.display();
197     cout<<"-----LIST M (odds)-----"<<endl;
198     m.display();
199     cout<<"-----" <<endl;
200     DLL n;
201     n.concatlist(l);
202     n.concatlist(m);
203     cout<<"-----LIST N after concatenating L & M-----"<<endl;
204     n.display();
205
206     //Question 3 part starts here
207
208     cout<<"-----LIST N after Desc Sort-----"<<endl;
209     n.sortlist();          //calls sorting function
210     n.display();
211
212 }

```

Question4.cpp

```
1 // <---Lab 04 - Doubly and Circular Linked List--->
2 /*4. Create a circular link list and perform the mentioned tasks.
3 a. Insert a new node at the end of the list.
4 b. Insert a new node at the beginning of list.
5 c. Insert a new node at given position.
6 d. Delete any node.
7 e. Print the complete doubly link list.*/
8
9
10 #include<iostream>
11 using namespace std;
12
13 class node {
14     public:
15         int data;
16         node* next;
17         node(int value) {
18             data=value;
19             next=NULL;
20         }
21         node(int value,node* nxt) {
22             data=value;
23             next=nxt;
24         }
25 };
26
27 class CLL{
28     node* head=NULL;
29     int nodecount=0;
30     public:
31     void appendNode(int value){ //insert at end of list / tail
32         if(head==NULL) { //Check if Linked List Empty.
33             node* n=new node(value,n);
34             head=n;
35             head->next=head;
36             nodecount++;
37             return;
38         }
39         else{
40             node* temp=head;
41             while(temp->next!=head) {
42                 temp=temp->next;
43             }
44             node* n=new node(value,head);
45             temp->next=n;
46             nodecount++;
47         }
48     }
49     void prependNode(int value){ //insert at start of list / head
50         if(head==NULL) { //Check if Linked List Empty.
51             node* n=new node(value,n);
52             head=n;
53             head->next=head;
```

```
54         nodecount++;
55         return;
56     }
57     node* n=new node(value,head);
58     node* temp=head;
59     while(temp->next!=head) {
60         temp=temp->next;
61     }
62     temp->next=n;
63     head=n;
64     nodecount++;
65 }
66 void insertNodeAfter(int posvalue,int value){ // insert at position (i+1)
67     if(head==NULL) { //Check if Linked List Empty.
68         cout<<"Empty List, adding at Head.\n";
69         node* n=new node(value,n);
70         head=n;
71         nodecount++;
72         return;
73     }
74     if(posvalue>nodecount-1){
75         cout<<"Position more than nodes in list, Inserting at tail.\n";
76         appendNode(value);
77         return;
78     }
79     int count=0;
80     node* temp=head;
81     while(temp->next!=head&&count<posvalue) {
82         temp=temp->next;
83         count++;
84     }
85     node* n=new node(value,temp->next);
86     temp->next=n;
87 }
88 void deleteathead(){
89     node* temp=head;
90     while(temp->next==head){
91         temp=temp->next;
92     }
93     temp->next=temp->next->next;
94     delete head;
95     head = temp->next;
96 }
97 void deleteattail(){
98     node* temp=head;
99     while(temp->next->next!=head){
100         temp=temp->next;
101     }
102     node* todelete=temp->next;
103     temp->next=temp->next->next;
104     delete todelete;
105 }
106 void deleteNodeByKey(int value){ // delete by value
107     node* temp=head;
108     node* prev=temp;
109     while(temp->next!=head&&temp->data!=value) {
```

```

110         prev=temp;
111         temp=temp->next;
112     }
113     if(temp->data==value){
114         prev->next=temp->next; //skip temp (i.e delete)
115         delete temp;
116     }
117     if(temp->next==head){
118         cout<<"Value not in Linked List.\n";
119         return;
120     }
121 }
122 void updateNodeByKey(int value){ // update by value
123     node* temp=head;
124     while(temp->next!=head&&temp->data!=value) {
125         node* prev=temp;
126         temp=temp->next;
127     }
128     if(temp->data==value){
129         cout<<"Enter new value: ";
130         cin>>temp->data;
131     }
132     if(temp->next==head){
133         cout<<"Value not in Linked List.\n";
134         return;
135     }
136 }
137 void print(){
138     node* temp=head;
139     cout<<"[HEAD] ";
140     if(head!=NULL){
141         cout<<temp->data<<" | "<<temp->next<<" -> ";
142         temp=temp->next;
143         while(temp!=head) {
144             cout<<temp->data<<" | "<<temp->next<<" -> ";
145             temp=temp->next;
146         }
147         cout<<"head [TAIL]"<<endl;
148     }
149     else{
150         cout<<"NULL [TAIL]"<<endl;
151     }
152 }
153 };
154
155 int main(){
156     CLL list;
157     float input=0;
158     int value;
159     while(input!=0.5) {
160         cout<<"-----\n";
161         cout<<"CURRENT LINKED LIST:\n";
162         list.print();
163         cout<<"-----\n";
164         cout<<"What would you like to do with the linked list?\n";
165         cout<<"1. Insert\t2. Delete\t3.Update\nEnter 0.5 to Exit\n[Anything else will default to

```

```

Delete]\n";
166     cin>>input;
167     if(input==1) {
168         cout<<"Enter Value to insert: ";
169         cin>>value;
170         cout<<"Where to Insert in Linked List?\n";
171         cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will default
to Insertion at Head]\n";
172         cin>>input;
173         if(input == 2){
174             list.appendNode(value);
175         }
176         else if(input == 3){
177             int pos;
178             cout<<"Enter the Position to insert After: ";
179             cin>>pos;
180             list.insertNodeAfter(pos,value);
181         }
182         else{
183             list.prependNode(value);
184         }
185     }
186 }
187 else if(input==0.5){
188     break;
189 }
190 else if(input==3){
191     cout<<"Enter the Value to Update: ";
192     cin>>value;
193     list.updateNodeByKey(value);
194 }
195 else{
196     cout<<"Where to Delete from Linked List?\n";
197     cout<<"1. At head\t2. At tail\t3. Delete a specific Value\n[Any other value will
default to Deletion from Head]\n";
198     cin>>input;
199     if(input == 2){
200         list.deleteattail();
201     }
202     else if(input == 3){
203         int pos;
204         cout<<"Enter the Value to Delete: ";
205         cin>>value;
206         list.deleteNodeByKey(value);
207     }
208     else{
209         list.deleteathead();
210     }
211 }
212 }
213 }
214 }

```