Question5.cpp

```cpp
// <---Lab 04 - Doubly and Circular Linked List--->
/*5. Break the above-created circular linked list into two halves.*/



#include<iostream>
using namespace std;

class node {
    public:
        int data;
        node* next;
        node(int value) {
            data=value;
            next=NULL;
        }
        node(int value,node* nxt) {
            data=value;
            next=nxt;
        }
};

class CLL{
    node* head=NULL;
    int nodecount=0;
    public:
    void setlist(node* h,int n){
        head=h;
        nodecount=n;
    }
    void makehalves(CLL &obj){
        int middleposition = (int)nodecount/2;     //Amount for 1st half
        int secondhalf = nodecount-middleposition;  //Amount for 2nd half
        int count=0;
        node* temp=head;
        node* prev=temp;
        while(temp->next!=head&&count<middleposition) {  // prev is last element of first half, temp is head of second half
            prev=temp;
            temp=temp->next;
            count++;
        }
        prev->next=head; //detach prev from remaining half, circular link back to start
        node* secondhead=temp;
        while(temp->next!=head){
            temp=temp->next;
        }
        temp->next=secondhead;
        obj.setlist(secondhead,secondhalf);
    }

    void appendNode(int value){ //insert at end of list / tail
        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         }
```

```
        else{
            cout<<"NULL [TAIL]"<<endl;
        }
    }
};

int main(){
    CLL list,list2;
    float input=0;
    int value;
    while(input!=0.5) {
        cout<<"-----------------------------------------------------\n";
        cout<<"CURRENT LINKED LIST:\n";
        list.print();
        cout<<"-----------------------------------------------------\n";
        cout<<"What would you like to do with the linked list?\n";
        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";
        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.appendNode(value);
            }
            else if(input == 3){
                int pos;
                cout<<"Enter the Position to insert After: ";
                cin>>pos;
                list.insertNodeAfter(pos,value);
            }
            else{
                list.prependNode(value);
            }

        }
        else if(input==0.5){
            break;
        }
        else if(input==3){
            cout<<"Enter the Value to Update: ";
            cin>>value;
            list.updateNodeByKey(value);
        }
        else if(input==4){
            cout<<"Halving list"<<endl;
            list.makehalves(list2);
            cout<<"First Half list--------------------------------"<<endl;
            list.print();
            cout<<"Second Half list-------------------------------"<<endl;
            list2.print();
            return 0;
        }
```

```
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

```cpp
// <---Lab 04 - Doubly and Circular Linked List--->
// 1. 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          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         }
214         else if(input==0.5){
215             break;
216         }
217         else{
218             cout<<"Where to Delete from Linked List?\n";
219             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";
```

```
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

```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 }
```

Question3.cpp

```cpp
// <---Lab 04 - Doubly and Circular Linked List--->
/*3. Using the above created list N, sort the contents of list N is descending order.*/

#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++;
        }
        void insertAthead(int value) {
            node* n=new node(value,head,NULL);
            if(head!=NULL){
                head->prevnode=n;
            }
            head=n;
```

```
            nodecount++;
        }
        void insertAtPos(int pos,int value) {
            if(pos>nodecount-1){
                cout<<"Position more than nodes in list, Inserting at tail.\n";
                insertAttail(value);
                return;
            }
            int count=0;
            node* temp=head;
            while(temp->nextnode!=NULL && count<pos-1) {
                temp=temp->nextnode;
                count++;
            }
            node* n=new node(value,temp->nextnode,temp);
            temp->nextnode=n;
            n->nextnode->prevnode=n;
            nodecount++;
        }
        void display() {
            node* temp=head;
            cout<<"HEAD | ";
            while(temp!=NULL) {
                cout<<" <--"<<temp->prevnode<<"  |  "<<temp->data<<"  |  "<<temp->nextnode<<"--> "
;
                temp=temp->nextnode;
            }
            cout<<"| TAIL"<<endl;
        }
//Assuming ANY node means any of the 4 types (head,tail,position,value)
        void deleteAtHead() {
            if(head==NULL) {
                cout<<"Empty Linked List, Returning"<<endl;
                return;
            }
            node* todelete=head;
            head=head->nextnode;
            head->prevnode=NULL;
            delete todelete;
            nodecount--;
        }
        void deletion(int value) {
            if(head==NULL) {
                cout<<"Empty Linked List, Returning"<<endl;
                return;
            }
            node* temp=head;
            if(head->data==value) {
                deleteAtHead();
                return;
            }
            while(temp->data!=value) {
                if(temp->nextnode==NULL) {
                    cout<<"Value not found, Returning\n";
                    return;
                }
```

```
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

```cpp
// <---Lab 04 - Doubly and Circular Linked List--->
/*4. Create a circular 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* next;
        node(int value) {
            data=value;
            next=NULL;
        }
        node(int value,node* nxt) {
            data=value;
            next=nxt;
        }
};

class CLL{
    node* head=NULL;
    int nodecount=0;
    public:
    void appendNode(int value){ //insert at end of list / tail
        if(head==NULL) {  //Check if Linked List Empty.
            node* n=new node(value,n);
            head=n;
            head->next=head;
            nodecount++;
            return;
        }
        else{
            node* temp=head;
            while(temp->next!=head) {
                temp=temp->next;
            }
            node* n=new node(value,head);
            temp->next=n;
            nodecount++;
        }
    }
    void prependNode(int value){ //insert at start of list / head
        if(head==NULL) {  //Check if Linked List Empty.
            node* n=new node(value,n);
            head=n;
            head->next=head;
```

```
            nodecount++;
            return;
        }
        node* n=new node(value,head);
        node* temp=head;
        while(temp->next!=head) {
            temp=temp->next;
        }
        temp->next=n;
        head=n;
        nodecount++;
    }
    void insertNodeAfter(int posvalue,int value){  // insert at position (i+1)
        if(head==NULL) {  //Check if Linked List Empty.
            cout<<"Empty List, adding at Head.\n";
            node* n=new node(value,n);
            head=n;
            nodecount++;
            return;
        }
        if(posvalue>nodecount-1){
            cout<<"Position more than nodes in list, Inserting at tail.\n";
            appendNode(value);
            return;
        }
        int count=0;
        node* temp=head;
        while(temp->next!=head&&count<posvalue) {
            temp=temp->next;
            count++;
        }
        node* n=new node(value,temp->next);
        temp->next=n;
    }
    void deleteathead(){
        node* temp=head;
        while(temp->next=head){
            temp=temp->next;
        }
        temp->next=head->next;
        delete head;
        head = temp->next;
    }
    void deleteattail(){
        node* temp=head;
        while(temp->next->next!=head){
            temp=temp->next;
        }
        node* todelete=temp->next;
        temp->next=head;
        delete todelete;
    }
    void deleteNodeByKey(int value){  // delete by value
        node* temp=head;
        node* prev=temp;
        while(temp->next!=head&&temp->data!=value) {
```

```
            prev=temp;
            temp=temp->next;
        }
        if(temp->data==value){
            prev->next=temp->next;  //skip temp (i.e delete)
            delete temp;
        }
        if(temp->next==head){
            cout<<"Value not in Linked List.\n";
            return;
        }
    }
    void updateNodeByKey(int value){  // update by value
        node* temp=head;
        while(temp->next!=head&&temp->data!=value) {
            node* prev=temp;
            temp=temp->next;
        }
        if(temp->data==value){
            cout<<"Enter new value: ";
            cin>>temp->data;
        }
        if(temp->next==head){
            cout<<"Value not in Linked List.\n";
            return;
        }
    }
    void print(){
        node* temp=head;
        cout<<"[HEAD] ";
        if(head!=NULL){
        cout<<temp->data<<"| "<<temp->next<<" -> ";
        temp=temp->next;
        while(temp!=head) {
            cout<<temp->data<<"| "<<temp->next<<" -> ";
            temp=temp->next;
        }
        cout<<"head [TAIL]"<<endl;
        }
        else{
            cout<<"NULL [TAIL]"<<endl;
        }
    }
};

int main(){
    CLL list;
    float input=0;
    int value;
    while(input!=0.5) {
        cout<<"-----------------------------------------------------\n";
        cout<<"CURRENT LINKED LIST:\n";
        list.print();
        cout<<"-----------------------------------------------------\n";
        cout<<"What would you like to do with the linked list?\n";
        cout<<"1. Insert\t2. Delete\t3.Update\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.appendNode(value);
            }
            else if(input == 3){
                int pos;
                cout<<"Enter the Position to insert After: ";
                cin>>pos;
                list.insertNodeAfter(pos,value);
            }
            else{
                list.prependNode(value);
            }

        }
        else if(input==0.5){
            break;
        }
        else if(input==3){
            cout<<"Enter the Value to Update: ";
            cin>>value;
            list.updateNodeByKey(value);
        }
        else{
            cout<<"Where to Delete from Linked List?\n";
            cout<<"1. At head\t2. At tail\t3. 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 Value to Delete: ";
                cin>>value;
                list.deleteNodeByKey(value);
            }
            else{
                list.deleteathead();
            }
        }

    }
}
```