Question2.cpp

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

// Q2. 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++;
        }
        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<0){
                cout<<"Position less than 0, Inserting at head.\n";
                insertAthead(value);
                return;
            }
            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;
                }
                temp=temp->nextnode;
            }
            if(temp->nextnode==NULL){
                deleteAtTail();
                return;
            }
            node* todelete=temp;
            temp->prevnode->nextnode=temp->nextnode;
            temp->nextnode->prevnode=temp->prevnode;
            delete todelete;
            nodecount--;
        }
        void deleteAtPos(int pos) {
            if(pos<0){
                cout<<"Position less than zero, INVALID. Returning..."<<endl;
                return;
            }
            if(pos==0){
                deleteAtHead();
                return;
            }
            else if(pos==nodecount-1){
                deleteAtTail();
                return;
            }
            if(pos>nodecount-1){
                cout<<"Invalid Position, Returning"<<endl;
                return;
            }
            if(head==NULL) {
                cout<<"Empty Linked List, Returning"<<endl;
                return;
            }
            int count=0;
            node* temp=head;
            while(temp->nextnode!=NULL && count<pos-1) {
                temp=temp->nextnode;
                count++;
            }
            node* todelete=temp->nextnode;
            temp->nextnode=temp->nextnode->nextnode;
            temp->nextnode->prevnode=temp;
            delete todelete;
            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--;
        }
        void concatlist(DLL &obj){
            node* temp=obj.head;
            while(temp!=NULL){
                insertAttail(temp->data);
                temp=temp->nextnode;
            }
        }
};

int main(){
    DLL l,m;
    for(int i=2;i<11;i+=2){  //Initializing L and M with evens and odds respectively
        l.insertAttail(i);
        m.insertAttail(i-1);
    }
    cout<<"-----------LIST L (Evens)-------------------"<<endl;
    l.display();    //Showing L for surity of values
    cout<<"-----------LIST M (odds)------------------------"<<endl;
    m.display();    //Showing M for surity of values
    cout<<"------------------------------------"<<endl;
    DLL n;      //Creates empty linked list
    n.concatlist(l);  //concats L into N
    n.concatlist(m);  //concats M into N
    cout<<"-----------LIST N after concatenating L & M------------------------"<<endl;
    n.display();
}
```