NAME: MOAZZAM FAROOQUI

TASK: DSA ASSIGNMENT#01

INSTRUCTOR: SAMIA MASOOD AWAN

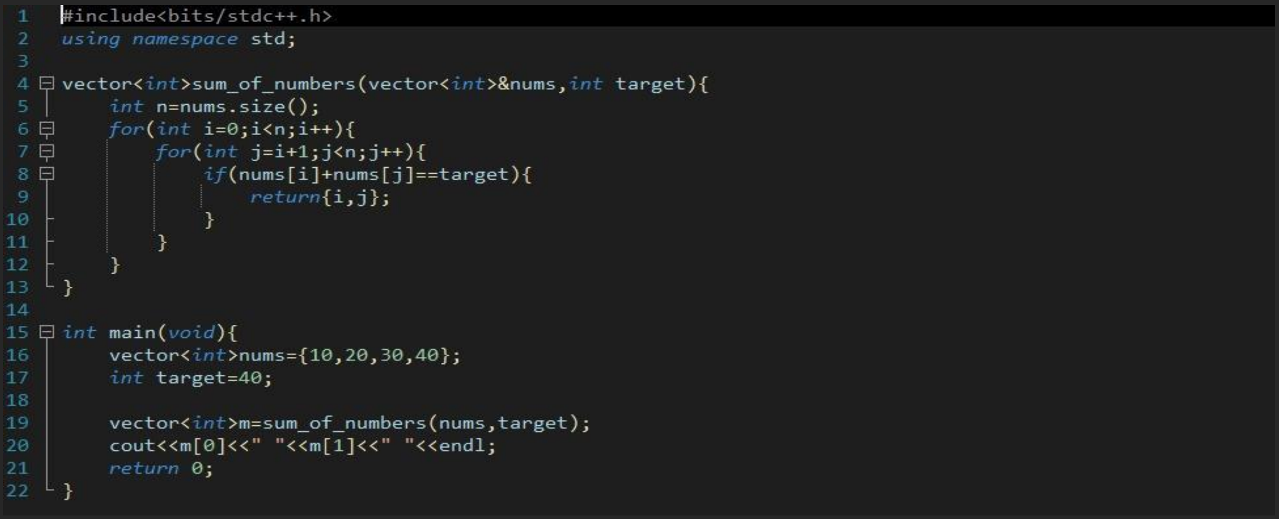
COURSE CODE: CT-159

ROLL NO: CT-24068

Question 1:

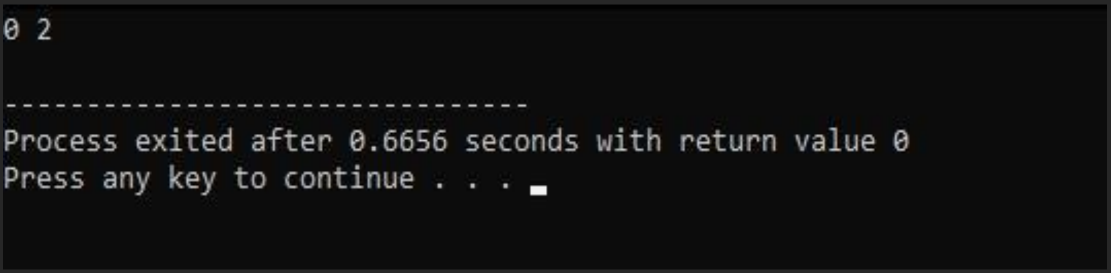
Given an array of integers and a target sum, find two numbers in the array that add up to the target sum. Return their indices. You can assume there will be exactly one so

SOURCE CODE:



```cpp
#include<bits/stdc++.h>
using namespace std;

vector<int>sum_of_numbers(vector<int>&nums,int target){
    int n=nums.size();
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(nums[i]+nums[j]==target){
                return{i,j};
            }
        }
    }
}

int main(void){
    vector<int>nums={10,20,30,40};
    int target=40;

    vector<int>m=sum_of_numbers(nums,target);
    cout<<m[0]<<" "<<m[1]<<" "<<endl;
    return 0;
}
```

OUTPUT:

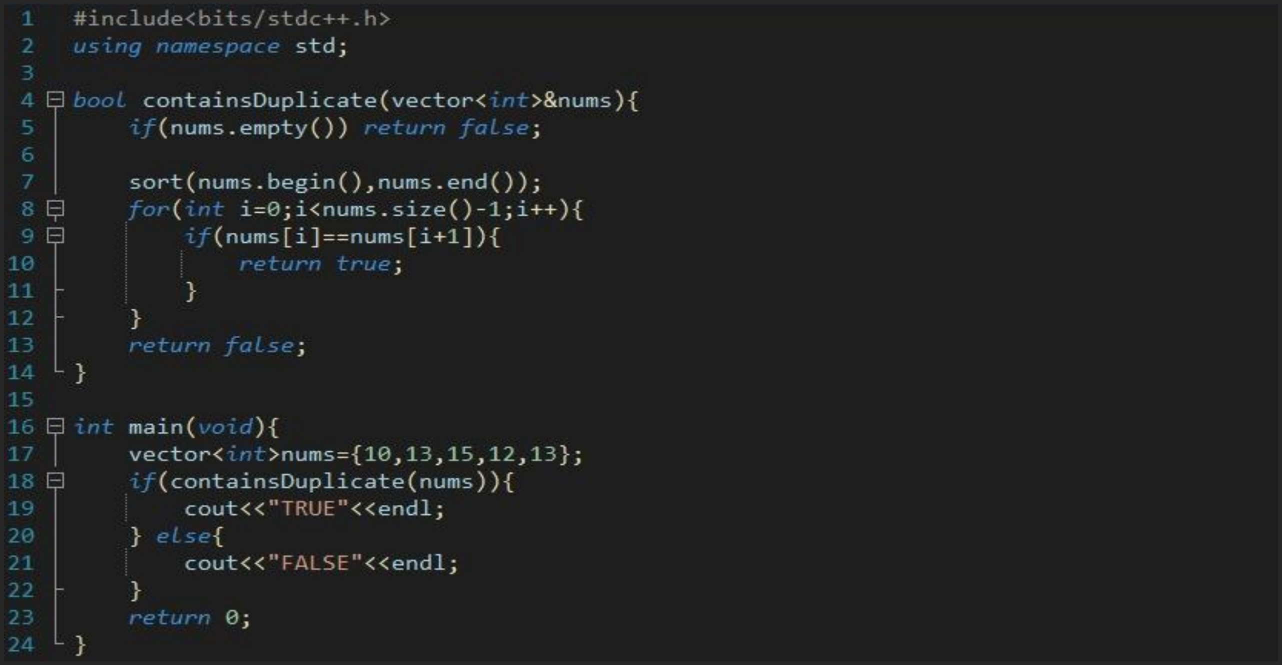


```
0 2

--------------------------------
Process exited after 0.6656 seconds with return value 0
Press any key to continue . . .
```

Question 2:

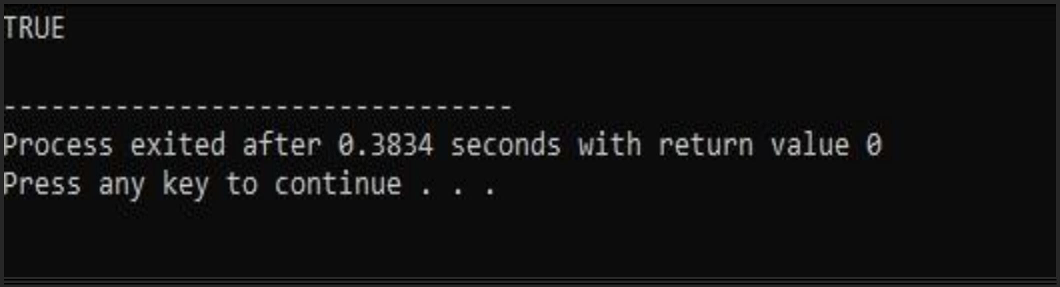
Create a function that takes a list of integers and returns True if the list contains duplicate elements, otherwise return False.

SOURCE CODE:



```cpp
#include<bits/stdc++.h>
using namespace std;

bool containsDuplicate(vector<int>&nums){
    if(nums.empty()) return false;

    sort(nums.begin(),nums.end());
    for(int i=0;i<nums.size()-1;i++){
        if(nums[i]==nums[i+1]){
            return true;
        }
    }
    return false;
}

int main(void){
    vector<int>nums={10,13,15,12,13};
    if(containsDuplicate(nums)){
        cout<<"TRUE"<<endl;
    } else{
        cout<<"FALSE"<<endl;
    }
    return 0;
}
```

OUTPUT:

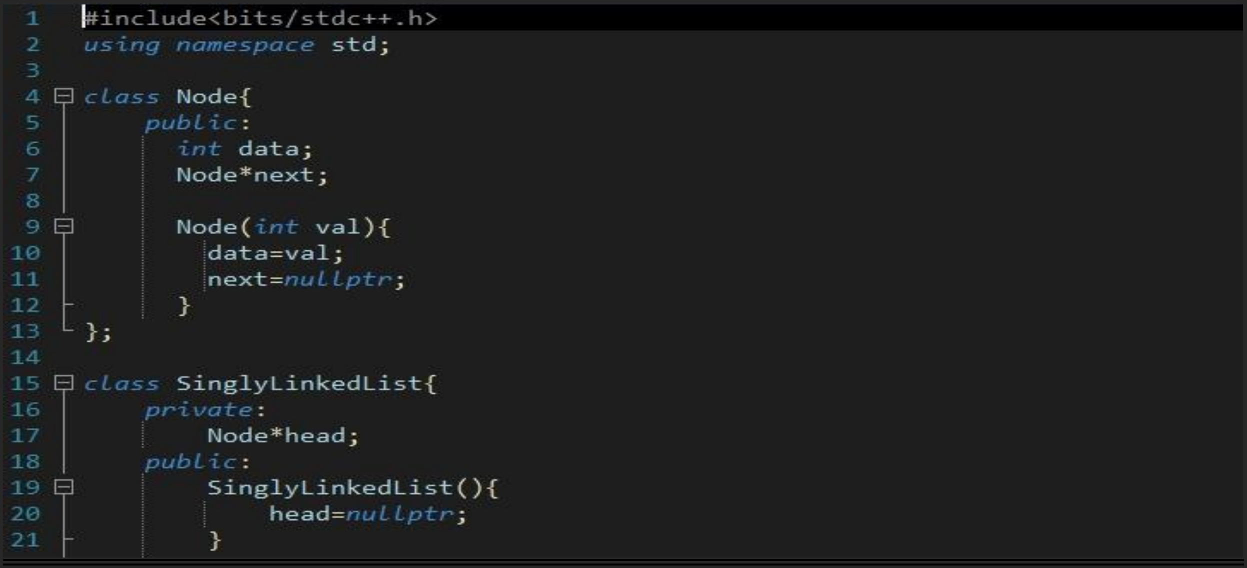


```
TRUE


--------------------------------
Process exited after 0.3834 seconds with return value 0
Press any key to continue . . .
```

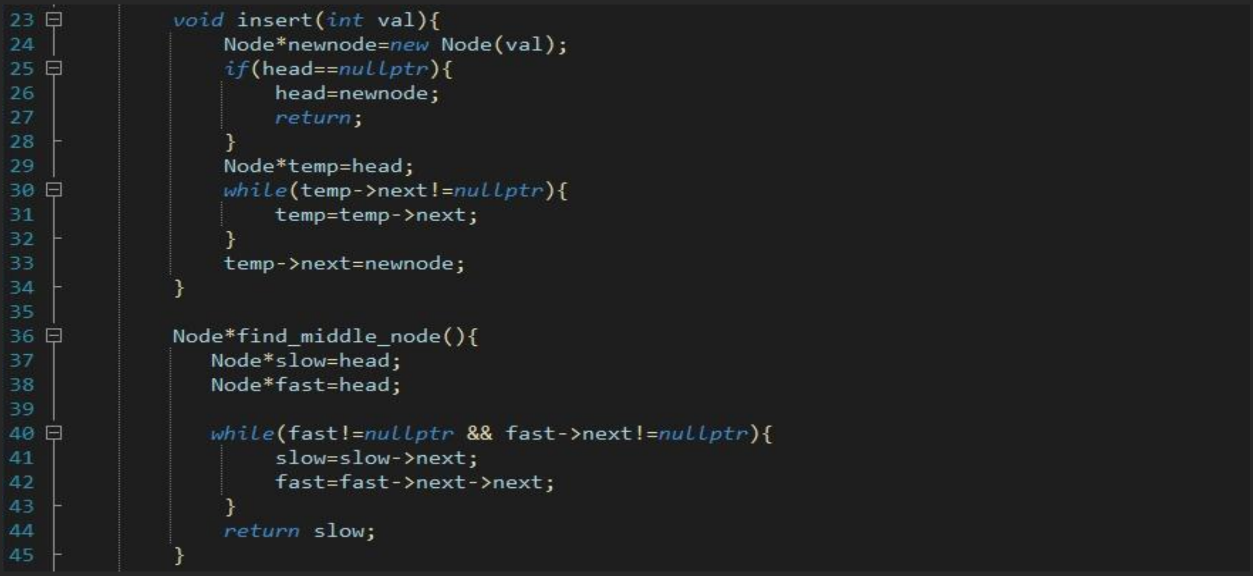
Question 3:

Write a program to find the middle node of a singly linked list.

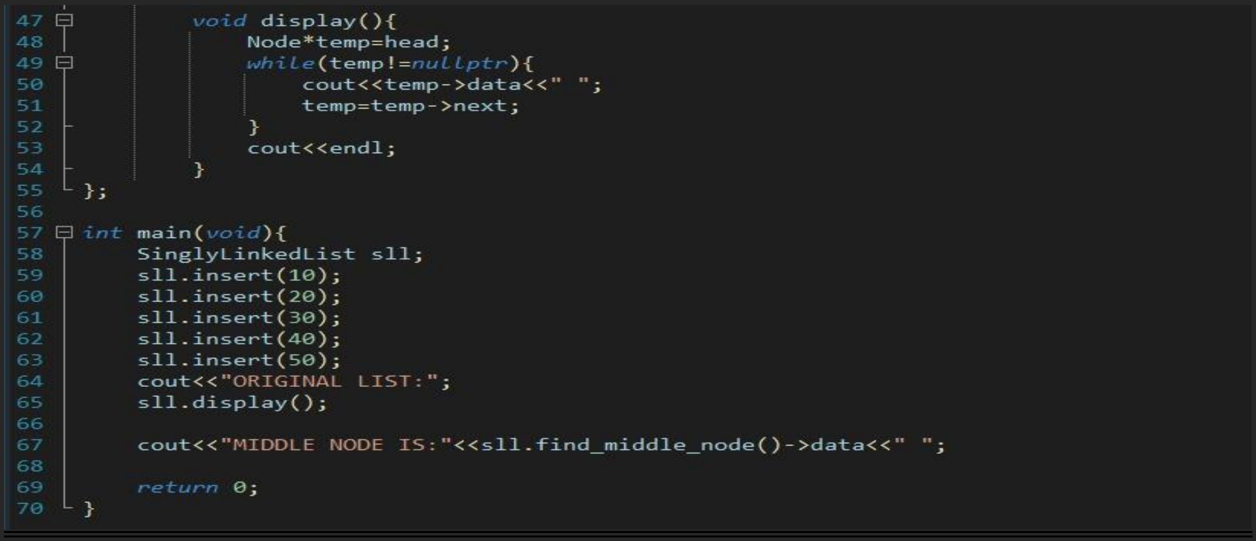
SOURCE CODE:



```cpp
#include<bits/stdc++.h>
using namespace std;

class Node{
    public:
        int data;
        Node*next;

        Node(int val){
          data=val;
          next=nullptr;
        }
};

class SinglyLinkedList{
    private:
        Node*head;
    public:
        SinglyLinkedList(){
            head=nullptr;
        }
```

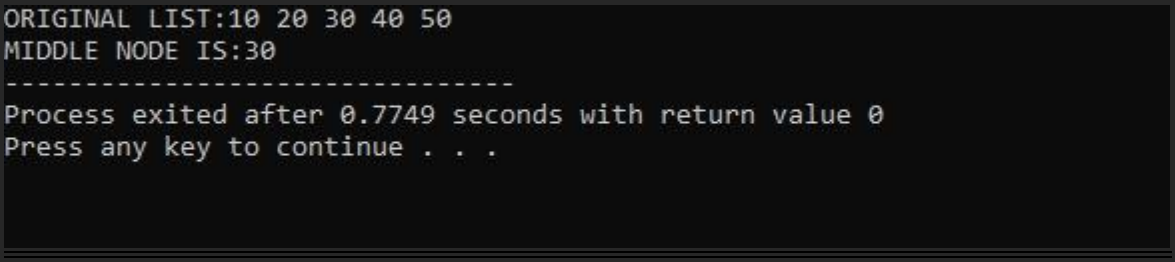


```cpp
        void insert(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                return;
            }
            Node*temp=head;
            while(temp->next!=nullptr){
                temp=temp->next;
            }
            temp->next=newnode;
        }

        Node*find_middle_node(){
           Node*slow=head;
           Node*fast=head;

           while(fast!=nullptr && fast->next!=nullptr){
                slow=slow->next;
                fast=fast->next->next;
            }
            return slow;
        }
```



```cpp
        void display(){
            Node*temp=head;
            while(temp!=nullptr){
                cout<<temp->data<<" ";
                temp=temp->next;
            }
            cout<<endl;
        }
};

int main(void){
    SinglyLinkedList sll;
    sll.insert(10);
    sll.insert(20);
    sll.insert(30);
    sll.insert(40);
    sll.insert(50);
    cout<<"ORIGINAL LIST:";
    sll.display();

    cout<<"MIDDLE NODE IS:"<<sll.find_middle_node()->data<<" ";

    return 0;
}
```

OUTPUT:

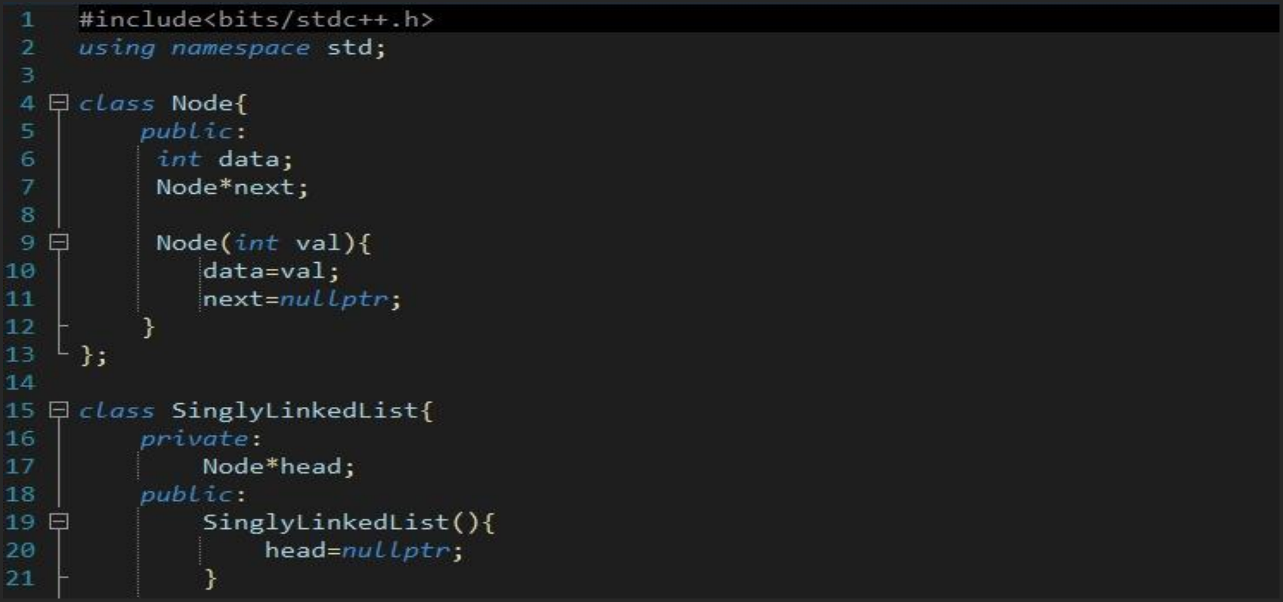


```
ORIGINAL LIST:10 20 30 40 50
MIDDLE NODE IS:30
--------------------------------
Process exited after 0.7749 seconds with return value 0
Press any key to continue . . .
```

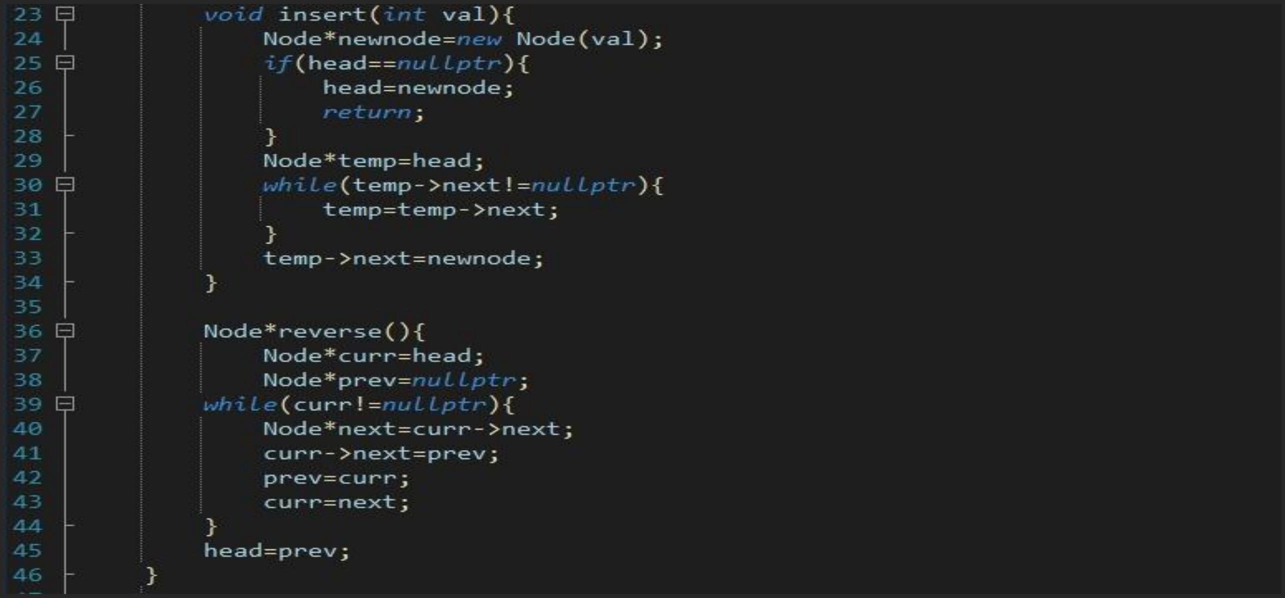
<u>Question 4:</u>

Implement a function to reverse a singly linked list. Your solution should update the pointers of the nodes and not just print the elements in reverse.

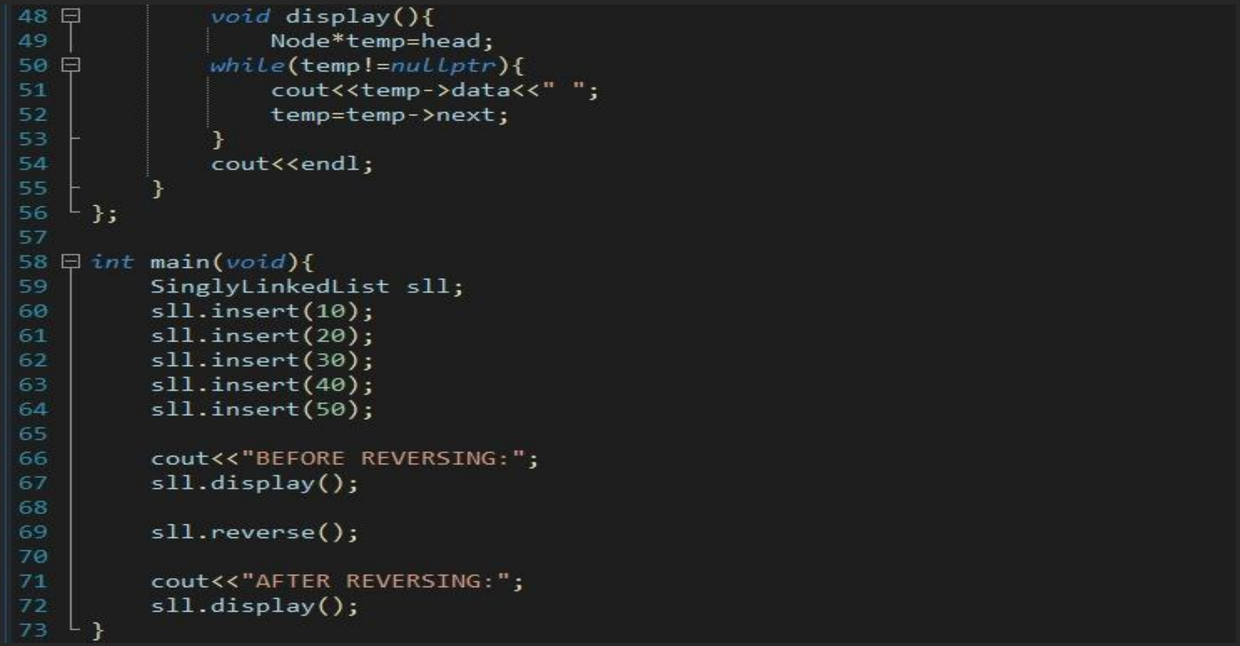
SOURCE CODE:



```
#include<bits/stdc++.h>
using namespace std;

class Node{
    public:
     int data;
     Node*next;

     Node(int val){
        data=val;
        next=nullptr;
    }
};

class SinglyLinkedList{
    private:
        Node*head;
    public:
        SinglyLinkedList(){
            head=nullptr;
        }
```

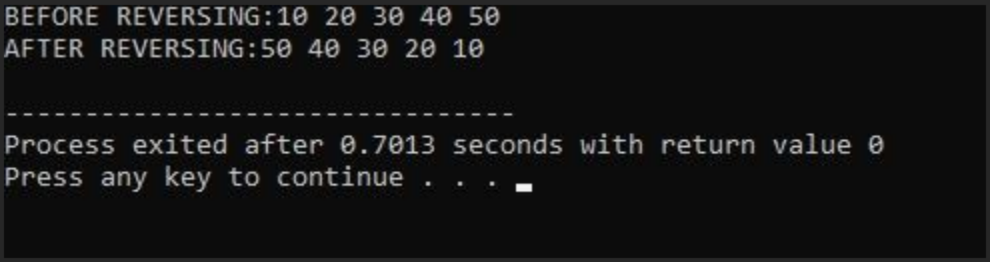


```
        void insert(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                return;
            }
            Node*temp=head;
            while(temp->next!=nullptr){
                temp=temp->next;
            }
            temp->next=newnode;
        }

        Node*reverse(){
            Node*curr=head;
            Node*prev=nullptr;
        while(curr!=nullptr){
            Node*next=curr->next;
            curr->next=prev;
            prev=curr;
            curr=next;
        }
        head=prev;
    }
```



```cpp
		void display(){
			Node*temp=head;
		while(temp!=nullptr){
			cout<<temp->data<<" ";
			temp=temp->next;
		}
		cout<<endl;
	}
};

int main(void){
	SinglyLinkedList sll;
	sll.insert(10);
	sll.insert(20);
	sll.insert(30);
	sll.insert(40);
	sll.insert(50);
	
	cout<<"BEFORE REVERSING:";
	sll.display();
	
	sll.reverse();
	
	cout<<"AFTER REVERSING:";
	sll.display();
}
```

<u>OUTPUT:</u>

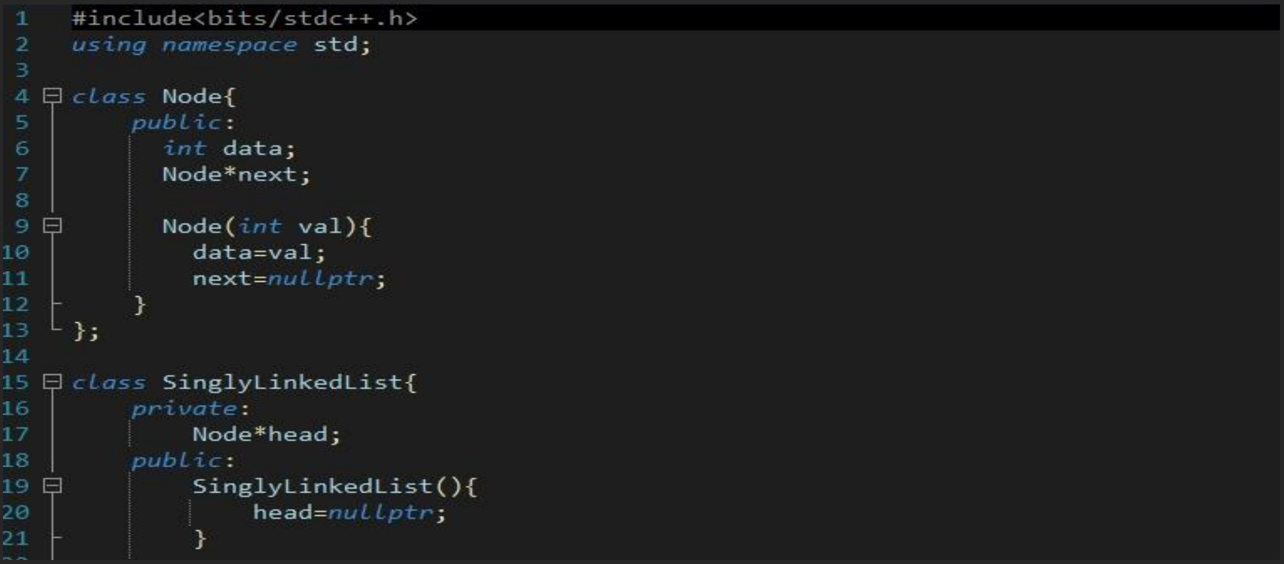


```
BEFORE REVERSING:10 20 30 40 50
AFTER REVERSING:50 40 30 20 10


--------------------------------
Process exited after 0.7013 seconds with return value 0
Press any key to continue . . .
```

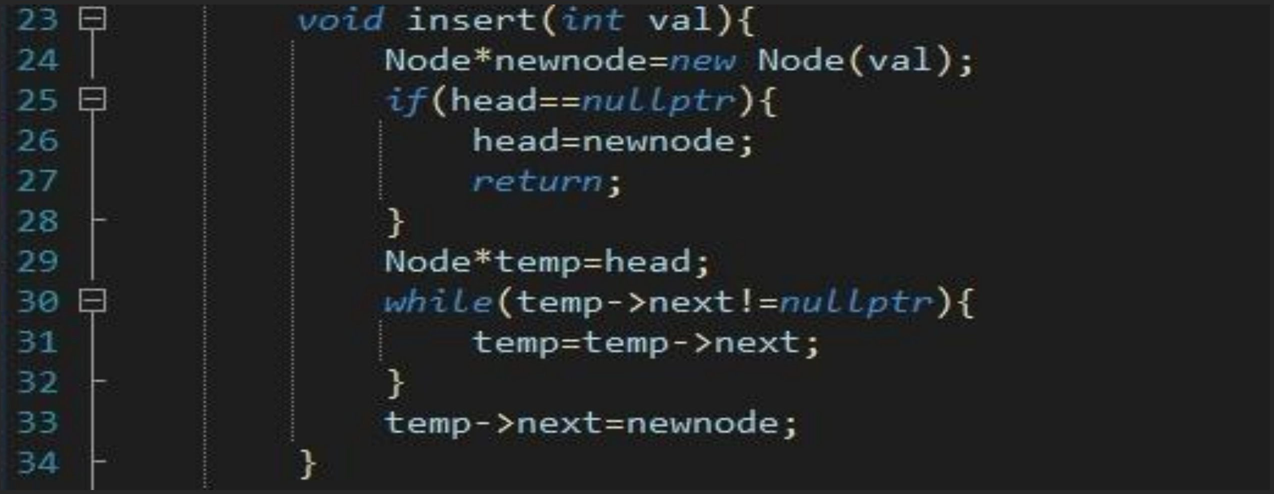
__Question 5:__

Given the heads of two sorted singly linked lists, merge the two lists into a single sorted list. The new list should be made by splicing together the nodes of the first two lists.

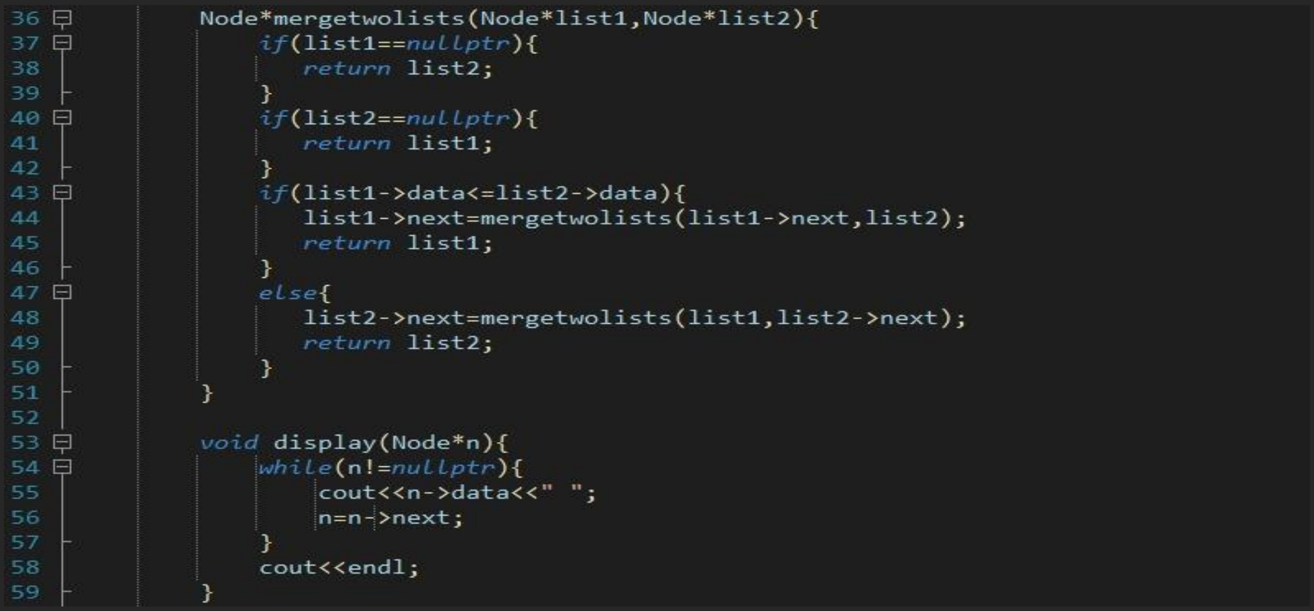
__SOURCE CODE:__



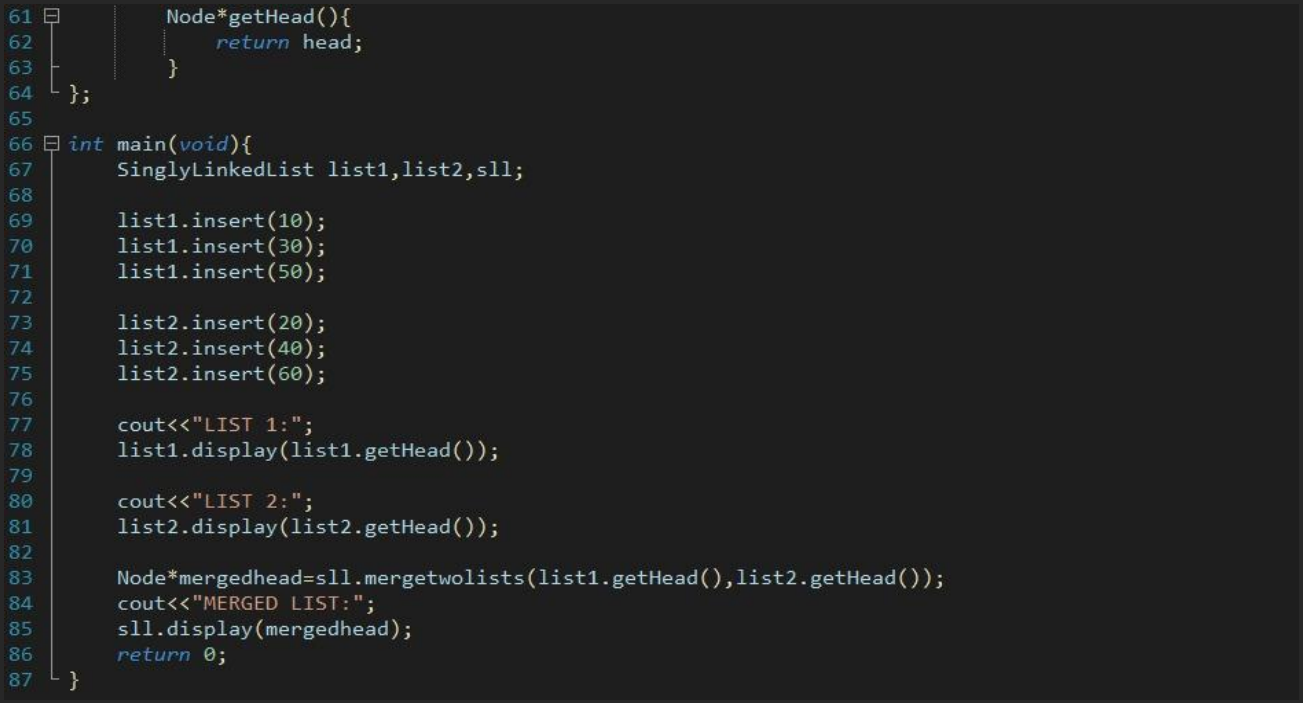
```
#include<bits/stdc++.h>
using namespace std;

class Node{
    public:
      int data;
      Node*next;

      Node(int val){
        data=val;
        next=nullptr;
    }
};

class SinglyLinkedList{
    private:
        Node*head;
    public:
        SinglyLinkedList(){
            head=nullptr;
        }
```



```
        void insert(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                return;
            }
            Node*temp=head;
            while(temp->next!=nullptr){
                temp=temp->next;
            }
            temp->next=newnode;
        }
```

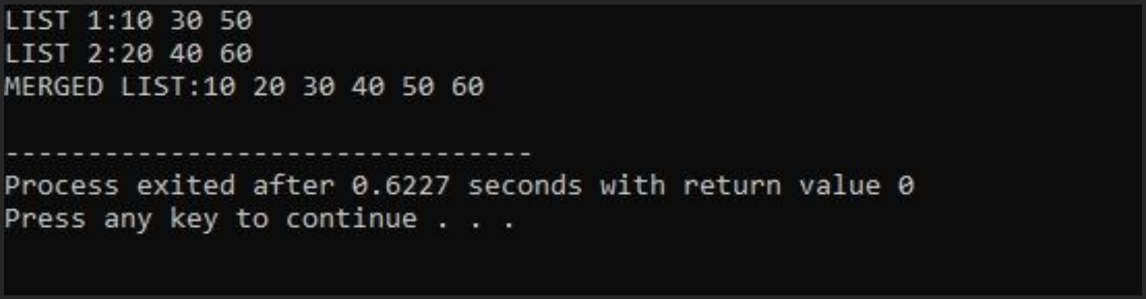


```cpp
        Node*mergetwolists(Node*list1,Node*list2){
            if(list1==nullptr){
                return list2;
            }
            if(list2==nullptr){
                return list1;
            }
            if(list1->data<=list2->data){
                list1->next=mergetwolists(list1->next,list2);
                return list1;
            }
            else{
                list2->next=mergetwolists(list1,list2->next);
                return list2;
            }
        }

        void display(Node*n){
            while(n!=nullptr){
                 cout<<n->data<<" ";
                 n=n->next;
            }
            cout<<endl;
        }
```



```cpp
        Node*getHead(){
            return head;
        }
};

int main(void){
    SinglyLinkedList list1,list2,sll;

    list1.insert(10);
    list1.insert(30);
    list1.insert(50);

    list2.insert(20);
    list2.insert(40);
    list2.insert(60);

    cout<<"LIST 1:";
    list1.display(list1.getHead());

    cout<<"LIST 2:";
    list2.display(list2.getHead());

    Node*mergedhead=sll.mergetwolists(list1.getHead(),list2.getHead());
    cout<<"MERGED LIST:";
    sll.display(mergedhead);
    return 0;
}
```

OUTPUT:

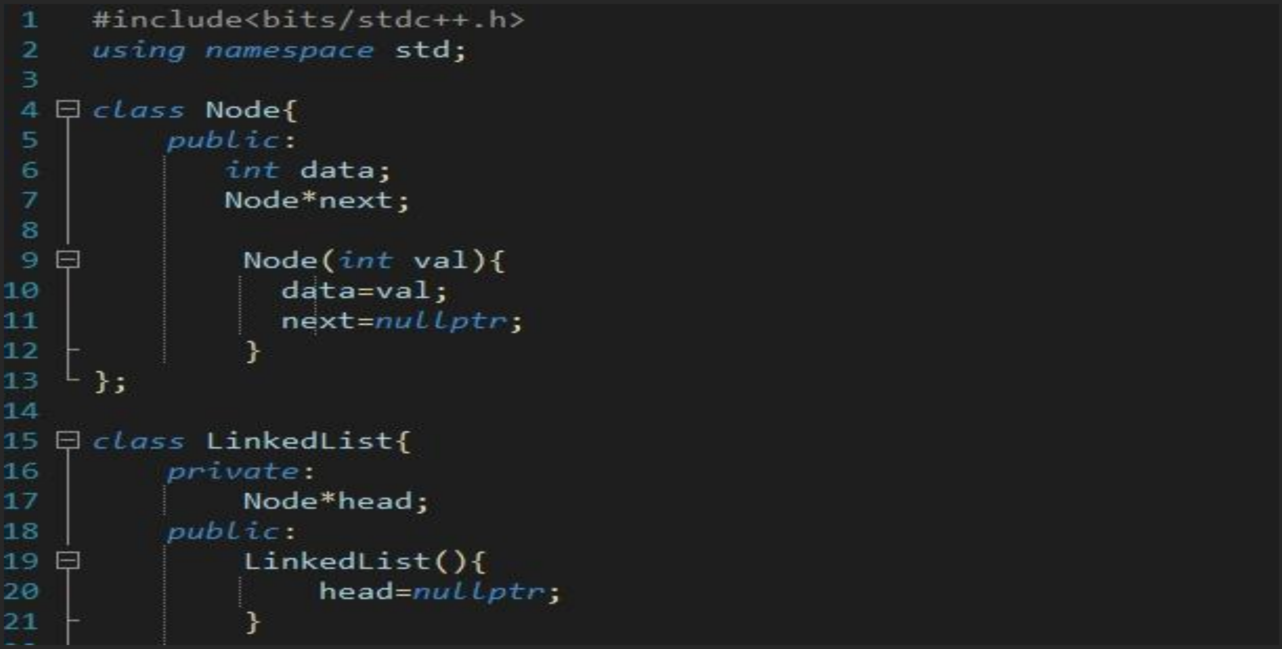


```
LIST 1:10 30 50
LIST 2:20 40 60
MERGED LIST:10 20 30 40 50 60

--------------------------------
Process exited after 0.6227 seconds with return value 0
Press any key to continue . . .
```

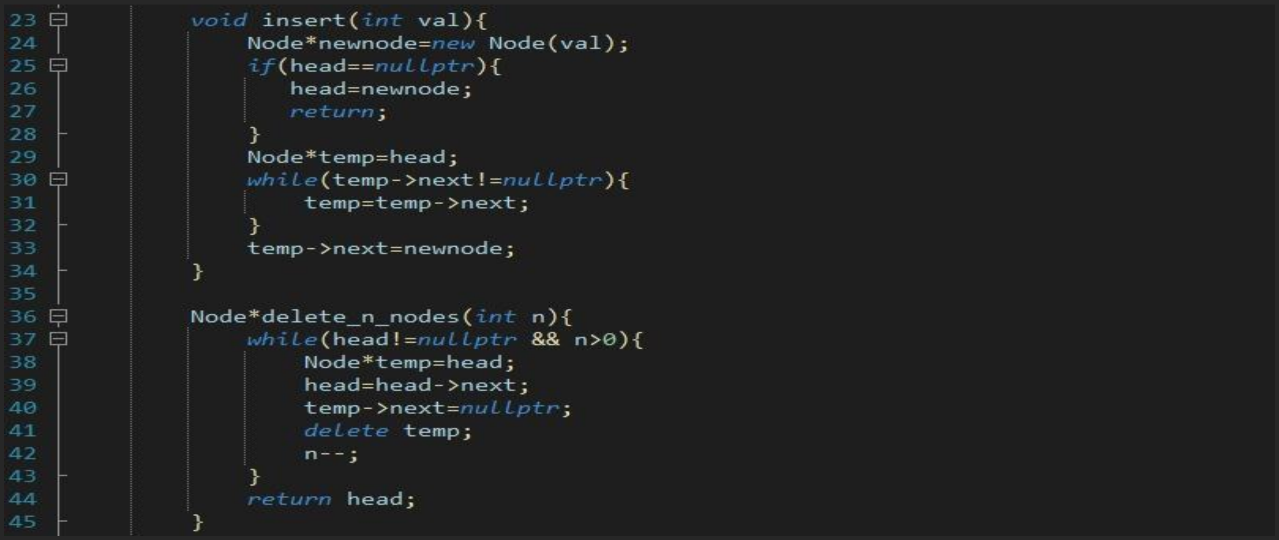
Question 6:

Given the head of a linked list, remove the n nodes from the list and return its head.

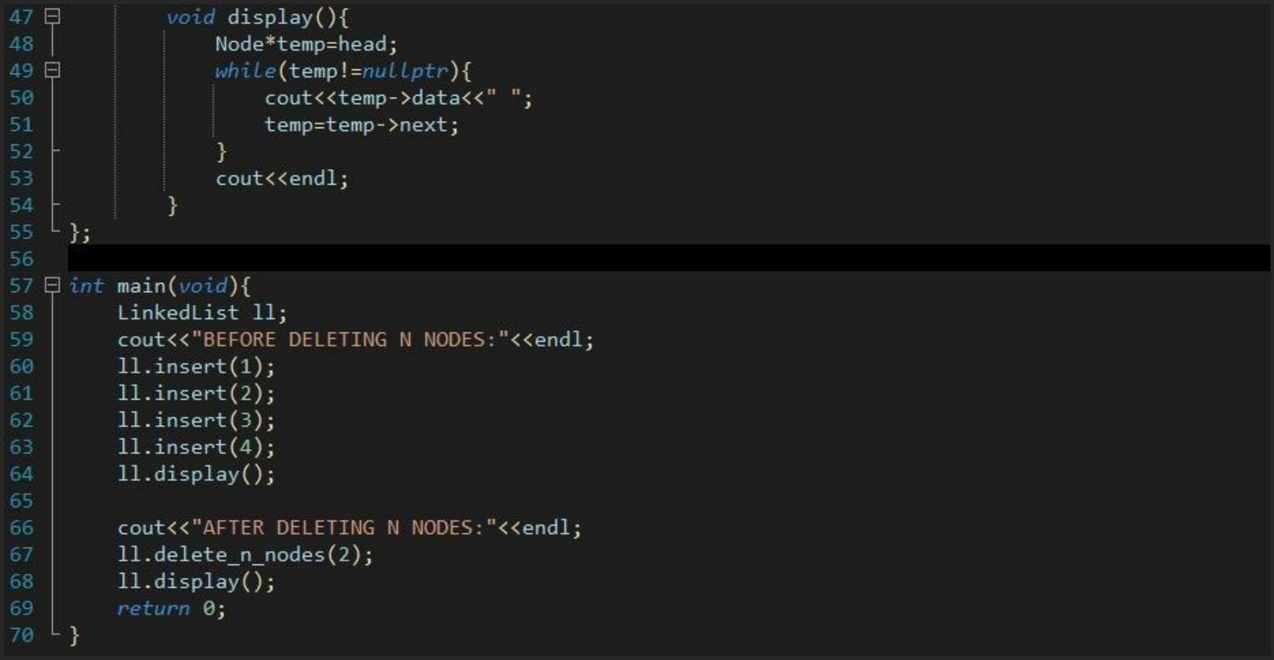
SOURCE CODE:



```cpp
#include<bits/stdc++.h>
using namespace std;

class Node{
    public:
        int data;
        Node*next;

         Node(int val){
           data=val;
           next=nullptr;
         }
};

class LinkedList{
    private:
         Node*head;
    public:
         LinkedList(){
             head=nullptr;
         }
```

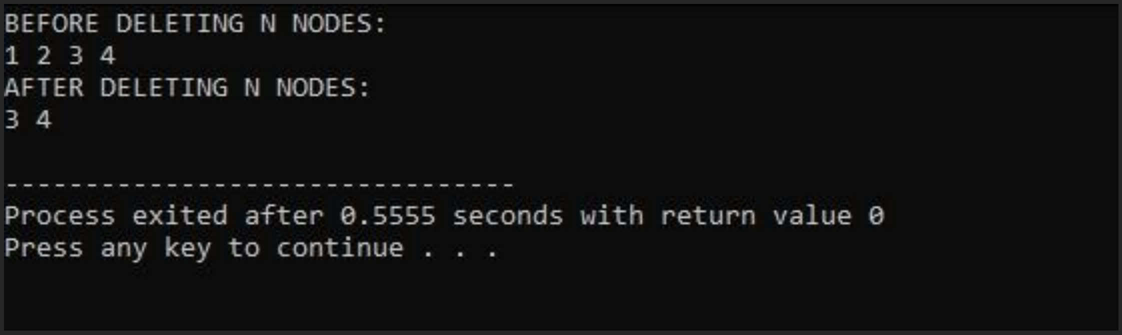


```cpp
        void insert(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                return;
            }
            Node*temp=head;
            while(temp->next!=nullptr){
                temp=temp->next;
            }
            temp->next=newnode;
        }

        Node*delete_n_nodes(int n){
            while(head!=nullptr && n>0){
                Node*temp=head;
                head=head->next;
                temp->next=nullptr;
                delete temp;
                n--;
            }
            return head;
        }
```



```cpp
        void display(){
            Node*temp=head;
            while(temp!=nullptr){
                cout<<temp->data<<" ";
                temp=temp->next;
            }
            cout<<endl;
        }
};

int main(void){
    LinkedList ll;
    cout<<"BEFORE DELETING N NODES:"<<endl;
    ll.insert(1);
    ll.insert(2);
    ll.insert(3);
    ll.insert(4);
    ll.display();

    cout<<"AFTER DELETING N NODES:"<<endl;
    ll.delete_n_nodes(2);
    ll.display();
    return 0;
}
```

OUTPUT:

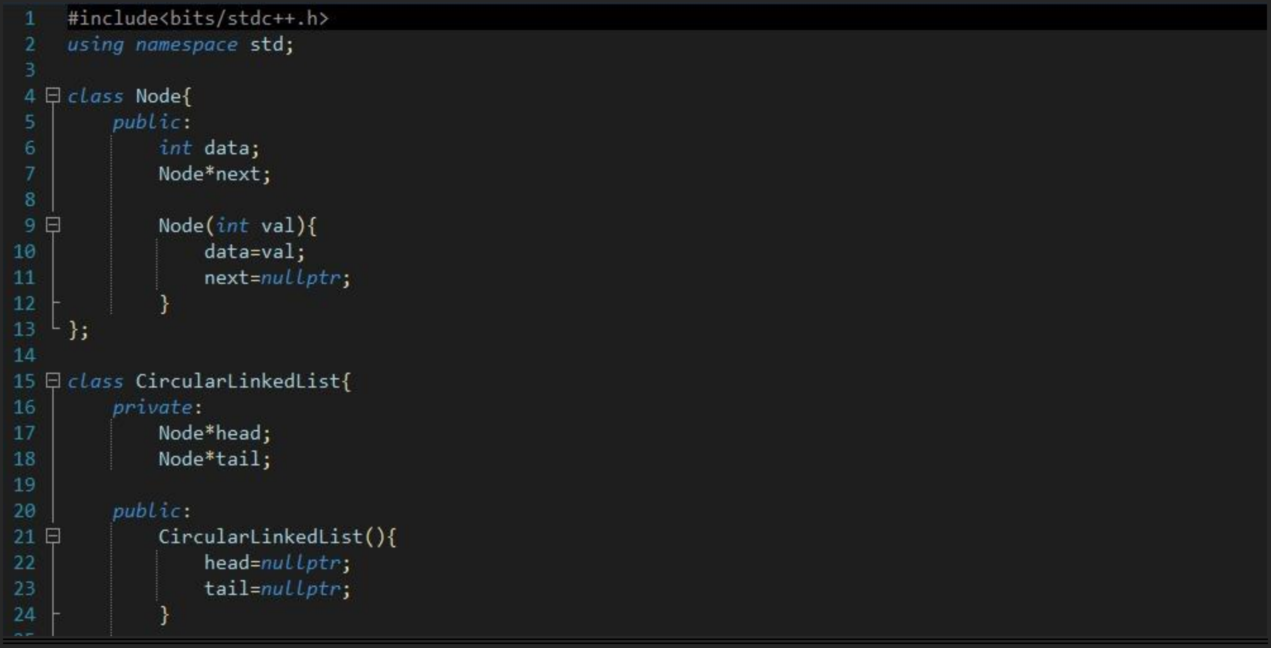


```
BEFORE DELETING N NODES:
1 2 3 4
AFTER DELETING N NODES:
3 4

--------------------------------
Process exited after 0.5555 seconds with return value 0
Press any key to continue . . .
```

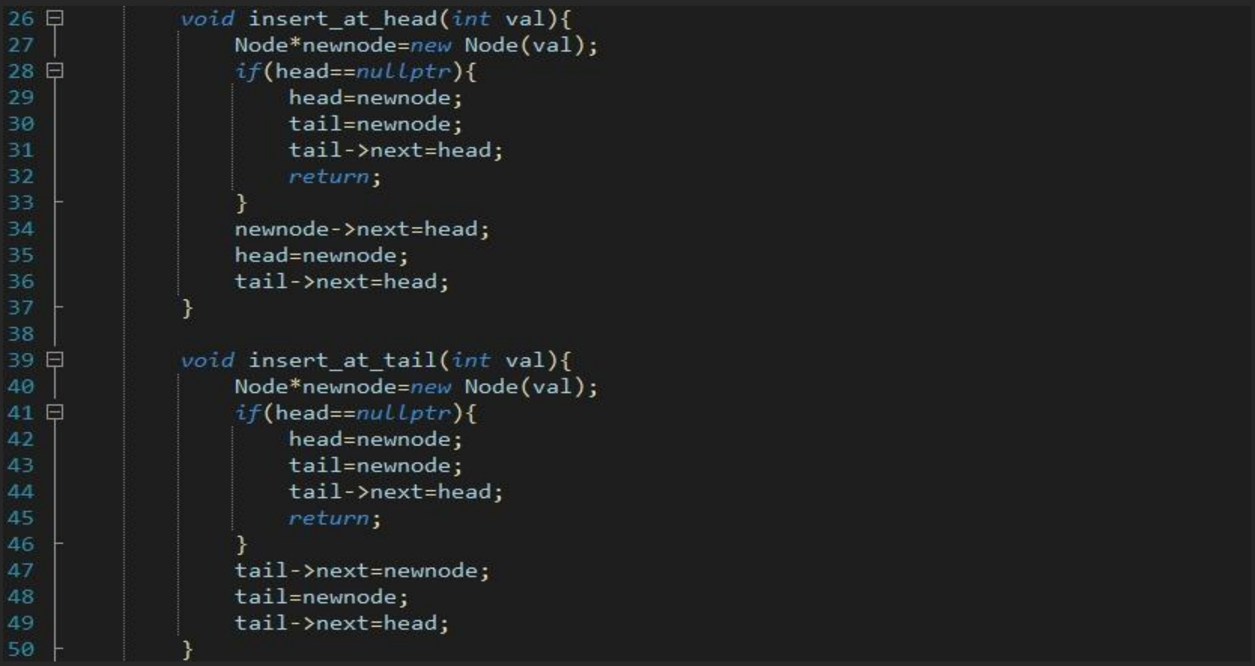
Question 7:

Write a function to insert a new node at a specific position (e.g., at the beginning, at the end, or after a given node) in a circular singly linked list.

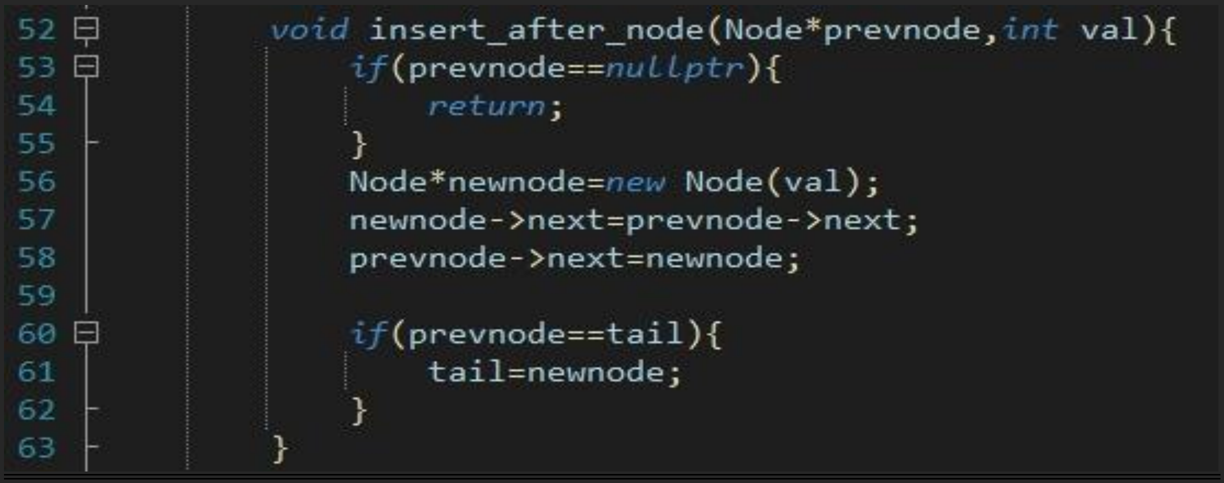
SOURCE CODE:



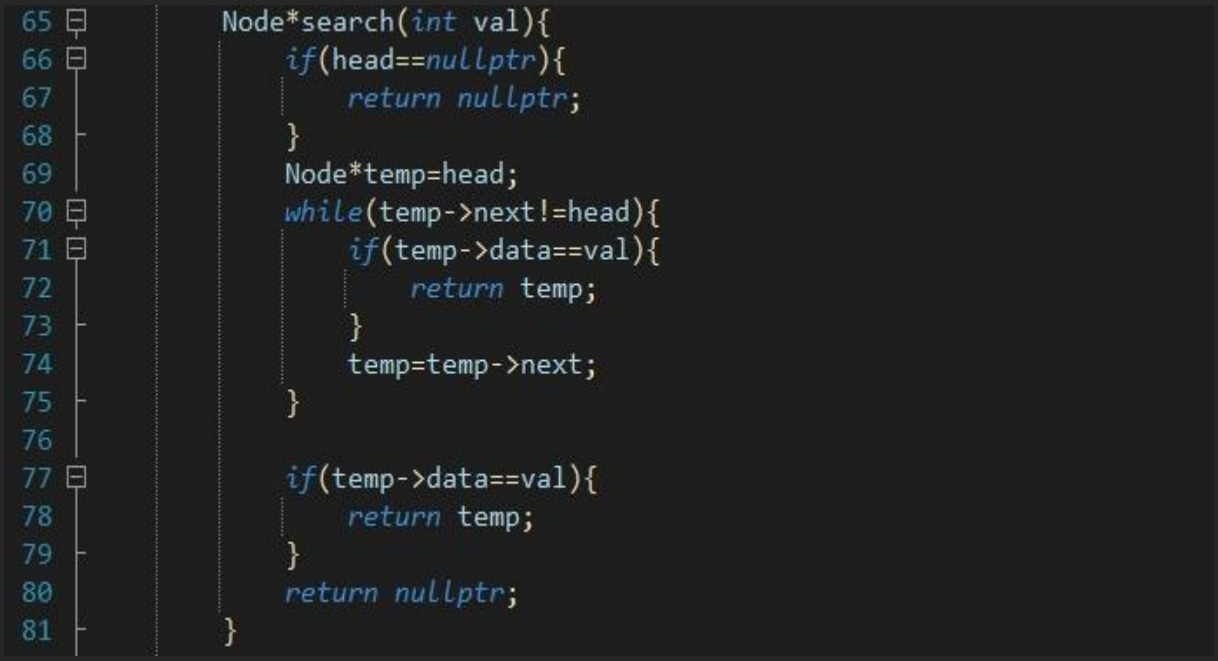
```cpp
#include<bits/stdc++.h>
using namespace std;

class Node{
    public:
        int data;
        Node*next;

        Node(int val){
            data=val;
            next=nullptr;
        }
};

class CircularLinkedList{
    private:
        Node*head;
        Node*tail;

    public:
        CircularLinkedList(){
            head=nullptr;
            tail=nullptr;
        }
```



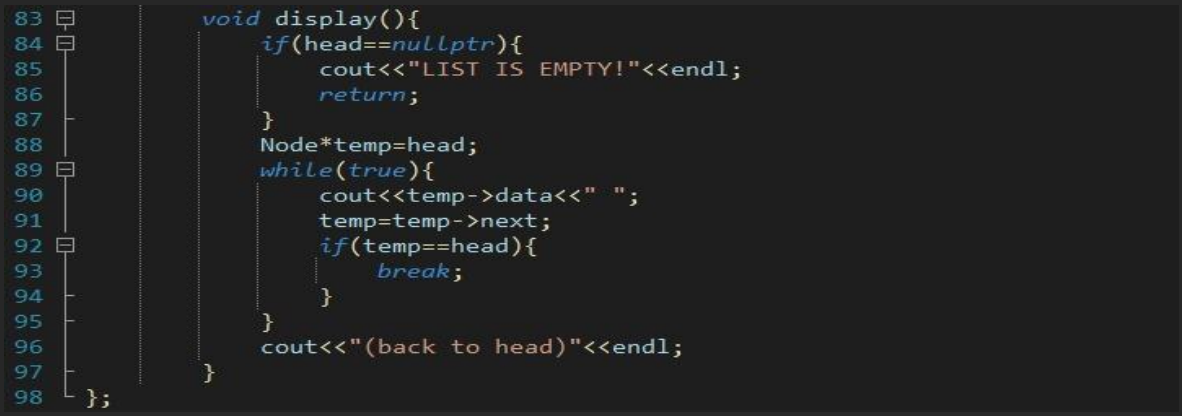
```cpp
        void insert_at_head(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                tail=newnode;
                tail->next=head;
                return;
            }
            newnode->next=head;
            head=newnode;
            tail->next=head;
        }

        void insert_at_tail(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                tail=newnode;
                tail->next=head;
                return;
            }
            tail->next=newnode;
            tail=newnode;
            tail->next=head;
        }
```



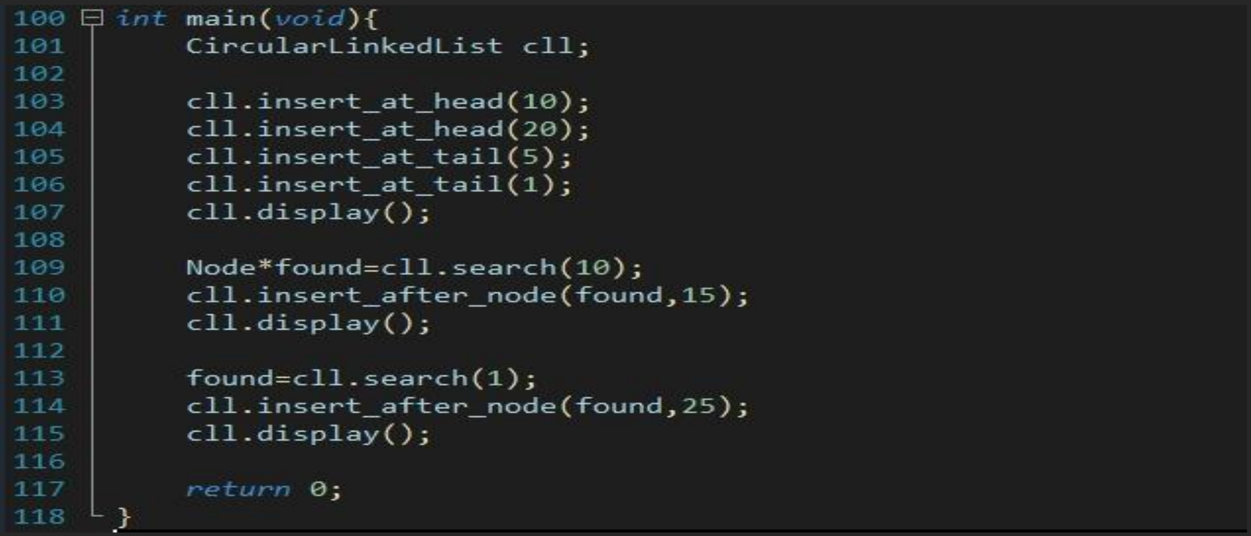
```cpp
    void insert_after_node(Node*prevnode,int val){
        if(prevnode==nullptr){
            return;
        }
        Node*newnode=new Node(val);
        newnode->next=prevnode->next;
        prevnode->next=newnode;

        if(prevnode==tail){
            tail=newnode;
        }
    }
```



```cpp
    Node*search(int val){
        if(head==nullptr){
            return nullptr;
        }
        Node*temp=head;
        while(temp->next!=head){
            if(temp->data==val){
                return temp;
            }
            temp=temp->next;
        }

        if(temp->data==val){
            return temp;
        }
        return nullptr;
    }
```

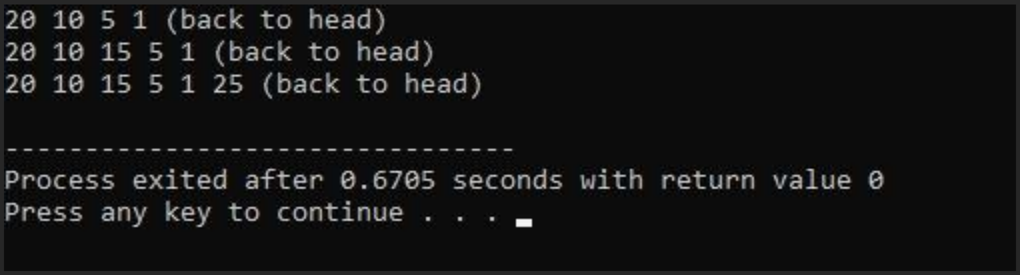


```cpp
    void display(){
        if(head==nullptr){
            cout<<"LIST IS EMPTY!"<<endl;
            return;
        }
        Node*temp=head;
        while(true){
            cout<<temp->data<<" ";
            temp=temp->next;
            if(temp==head){
                break;
            }
        }
        cout<<"(back to head)"<<endl;
    }
};
```



```cpp
int main(void){
    CircularLinkedList cll;

    cll.insert_at_head(10);
    cll.insert_at_head(20);
    cll.insert_at_tail(5);
    cll.insert_at_tail(1);
    cll.display();

    Node*found=cll.search(10);
    cll.insert_after_node(found,15);
    cll.display();

    found=cll.search(1);
    cll.insert_after_node(found,25);
    cll.display();

    return 0;
}
```

OUTPUT:

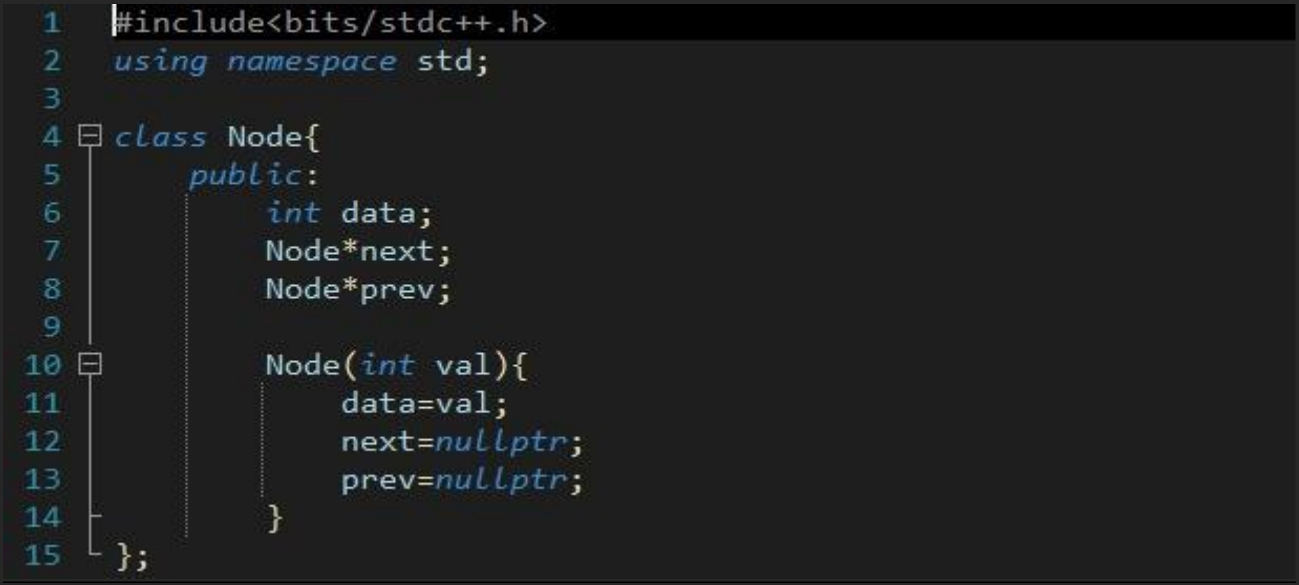


```
20 10 5 1 (back to head)
20 10 15 5 1 (back to head)
20 10 15 5 1 25 (back to head)

--------------------------------
Process exited after 0.6705 seconds with return value 0
Press any key to continue . . .
```

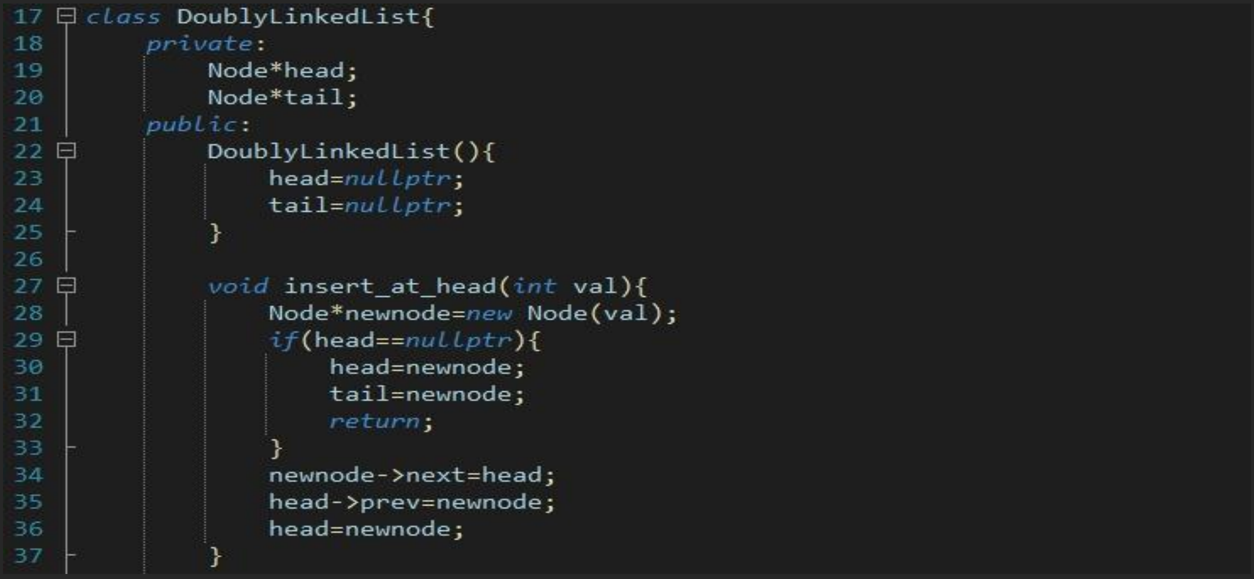
Question 8:

Implement a function to delete a node from a doubly linked list, given a pointer to the node to be deleted.

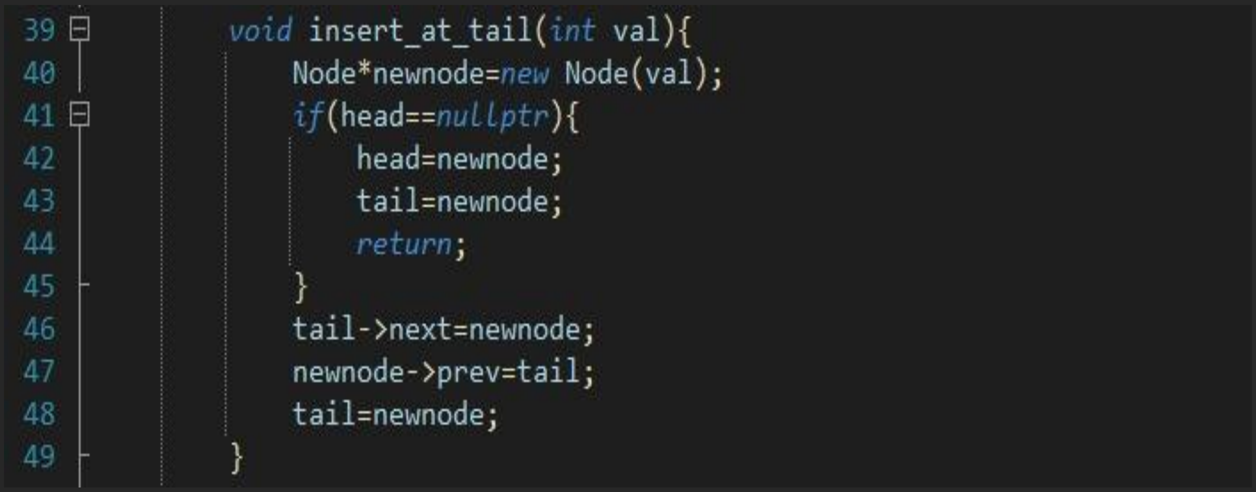
SOURCE CODE:



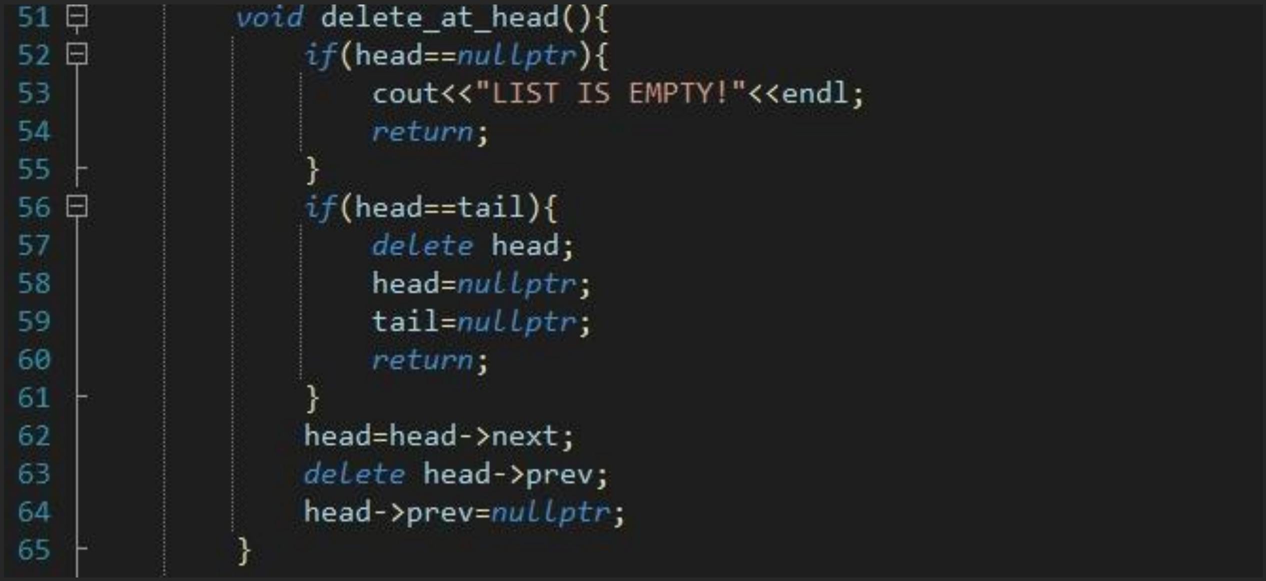
```
#include<bits/stdc++.h>
using namespace std;

class Node{
    public:
        int data;
        Node*next;
        Node*prev;

        Node(int val){
            data=val;
            next=nullptr;
            prev=nullptr;
        }
};
```



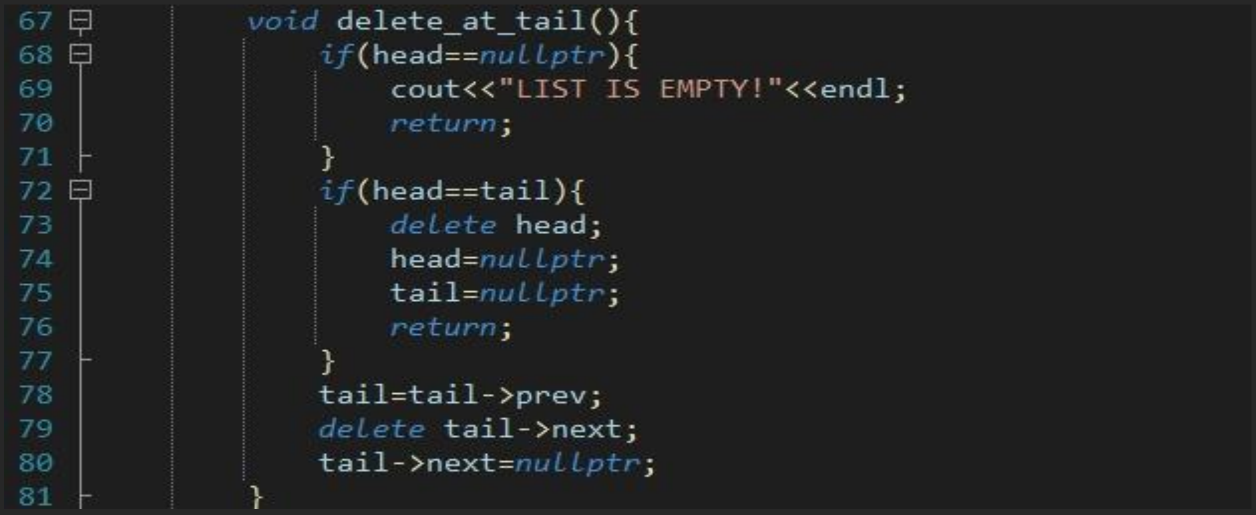
```
class DoublyLinkedList{
    private:
        Node*head;
        Node*tail;
    public:
        DoublyLinkedList(){
            head=nullptr;
            tail=nullptr;
        }

        void insert_at_head(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                tail=newnode;
                return;
            }
            newnode->next=head;
            head->prev=newnode;
            head=newnode;
        }
```



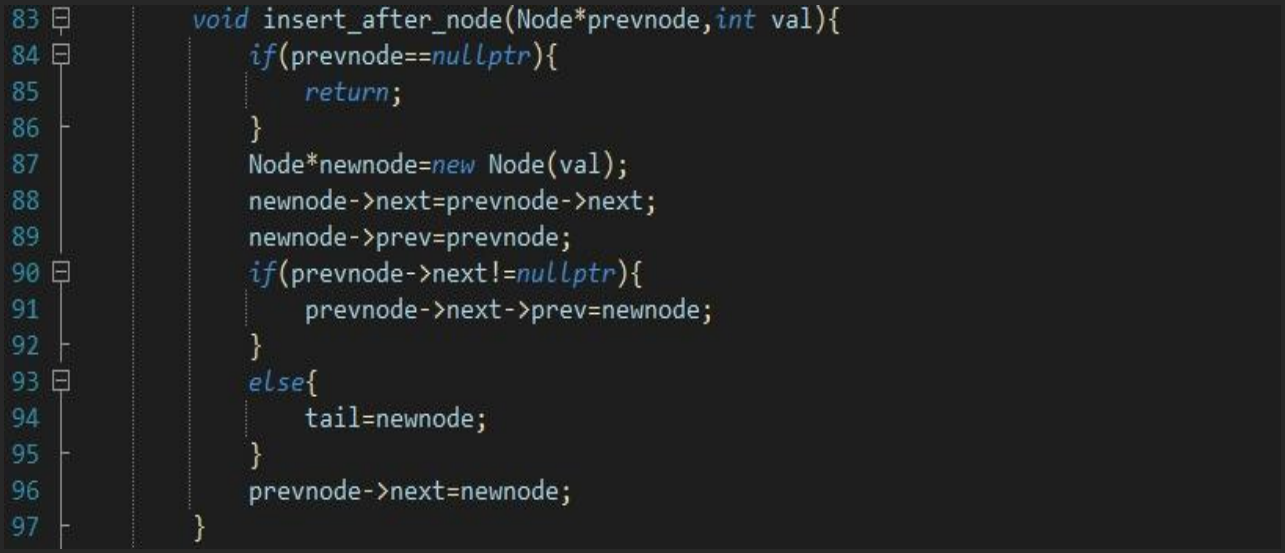
```cpp
void insert_at_tail(int val){
    Node*newnode=new Node(val);
    if(head==nullptr){
        head=newnode;
        tail=newnode;
        return;
    }
    tail->next=newnode;
    newnode->prev=tail;
    tail=newnode;
}
```



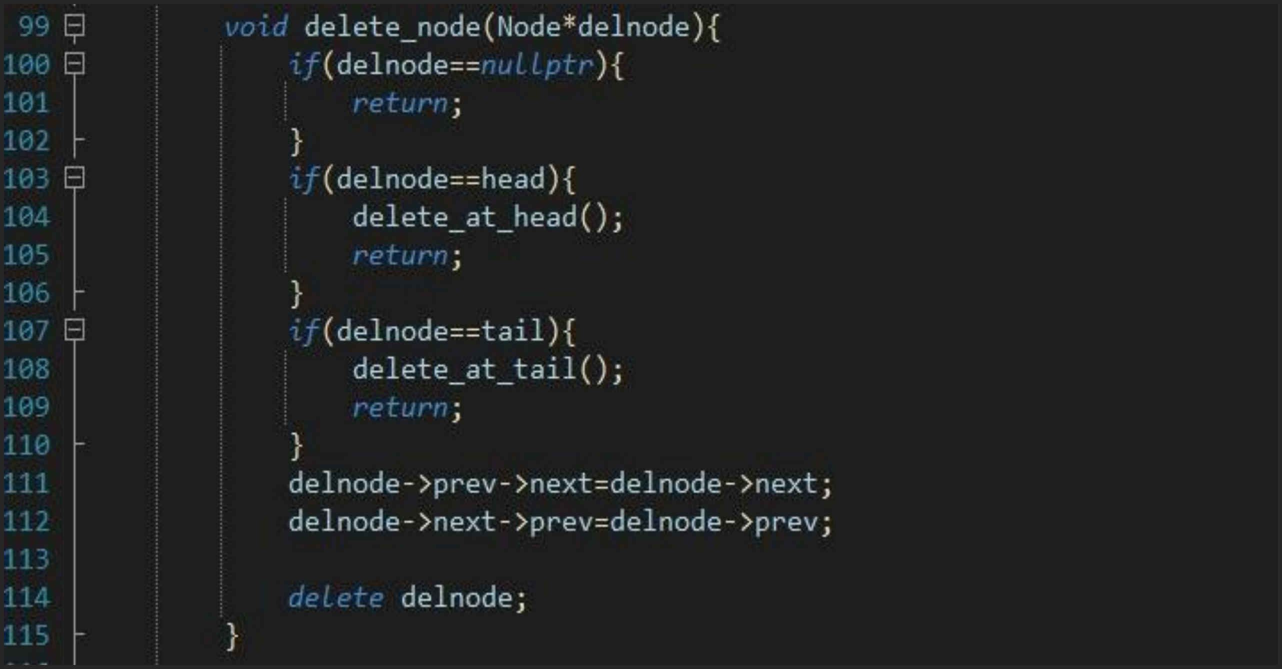
```cpp
void delete_at_head(){
    if(head==nullptr){
        cout<<"LIST IS EMPTY!"<<endl;
        return;
    }
    if(head==tail){
        delete head;
        head=nullptr;
        tail=nullptr;
        return;
    }
    head=head->next;
    delete head->prev;
    head->prev=nullptr;
}
```



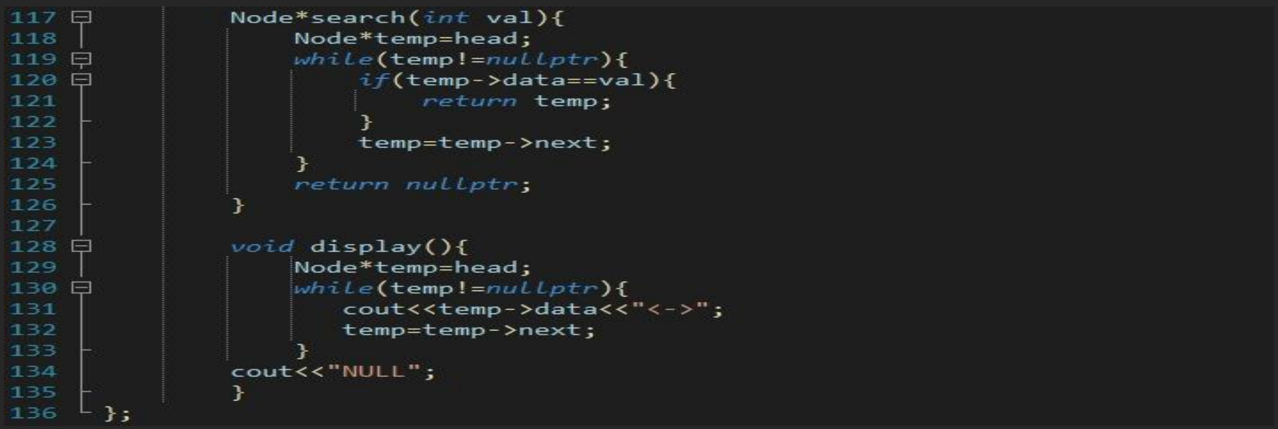
```cpp
void delete_at_tail(){
    if(head==nullptr){
        cout<<"LIST IS EMPTY!"<<endl;
        return;
    }
    if(head==tail){
        delete head;
        head=nullptr;
        tail=nullptr;
        return;
    }
    tail=tail->prev;
    delete tail->next;
    tail->next=nullptr;
}
```



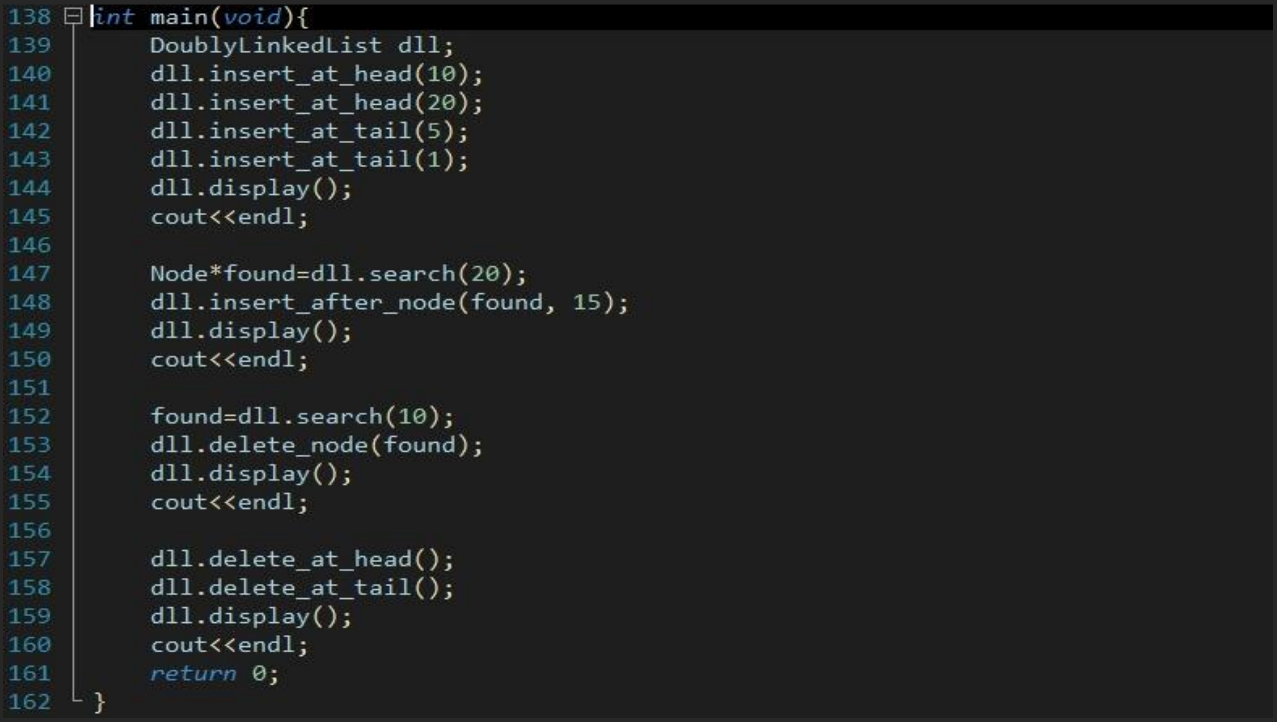
```
    void insert_after_node(Node*prevnode,int val){
        if(prevnode==nullptr){
            return;
        }
        Node*newnode=new Node(val);
        newnode->next=prevnode->next;
        newnode->prev=prevnode;
        if(prevnode->next!=nullptr){
            prevnode->next->prev=newnode;
        }
        else{
            tail=newnode;
        }
        prevnode->next=newnode;
    }
```



```
    void delete_node(Node*delnode){
        if(delnode==nullptr){
            return;
        }
        if(delnode==head){
            delete_at_head();
            return;
        }
        if(delnode==tail){
            delete_at_tail();
            return;
        }
        delnode->prev->next=delnode->next;
        delnode->next->prev=delnode->prev;

        delete delnode;
    }
```

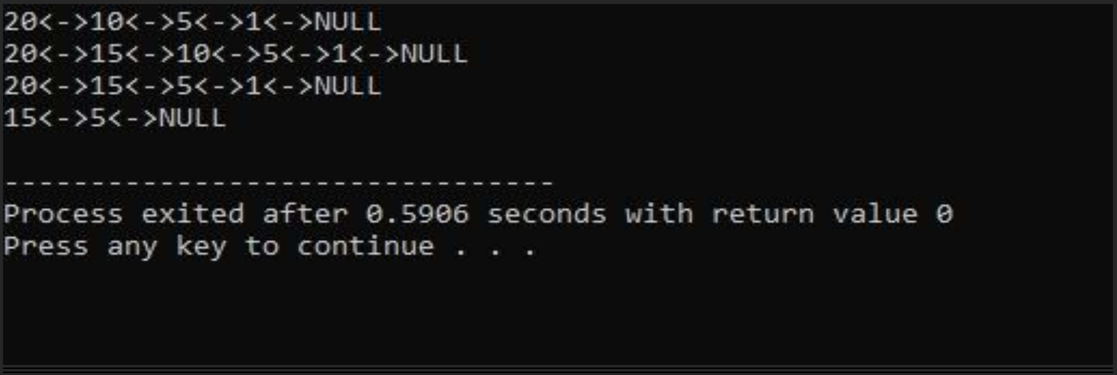


```
    Node*search(int val){
        Node*temp=head;
        while(temp!=nullptr){
            if(temp->data==val){
                return temp;
            }
            temp=temp->next;
        }
        return nullptr;
    }

    void display(){
        Node*temp=head;
        while(temp!=nullptr){
            cout<<temp->data<<"<->";
            temp=temp->next;
        }
    cout<<"NULL";
    }
};
```



```cpp
int main(void){
    DoublyLinkedList dll;
    dll.insert_at_head(10);
    dll.insert_at_head(20);
    dll.insert_at_tail(5);
    dll.insert_at_tail(1);
    dll.display();
    cout<<endl;

    Node*found=dll.search(20);
    dll.insert_after_node(found, 15);
    dll.display();
    cout<<endl;

    found=dll.search(10);
    dll.delete_node(found);
    dll.display();
    cout<<endl;

    dll.delete_at_head();
    dll.delete_at_tail();
    dll.display();
    cout<<endl;
    return 0;
}
```

OUTPUT:

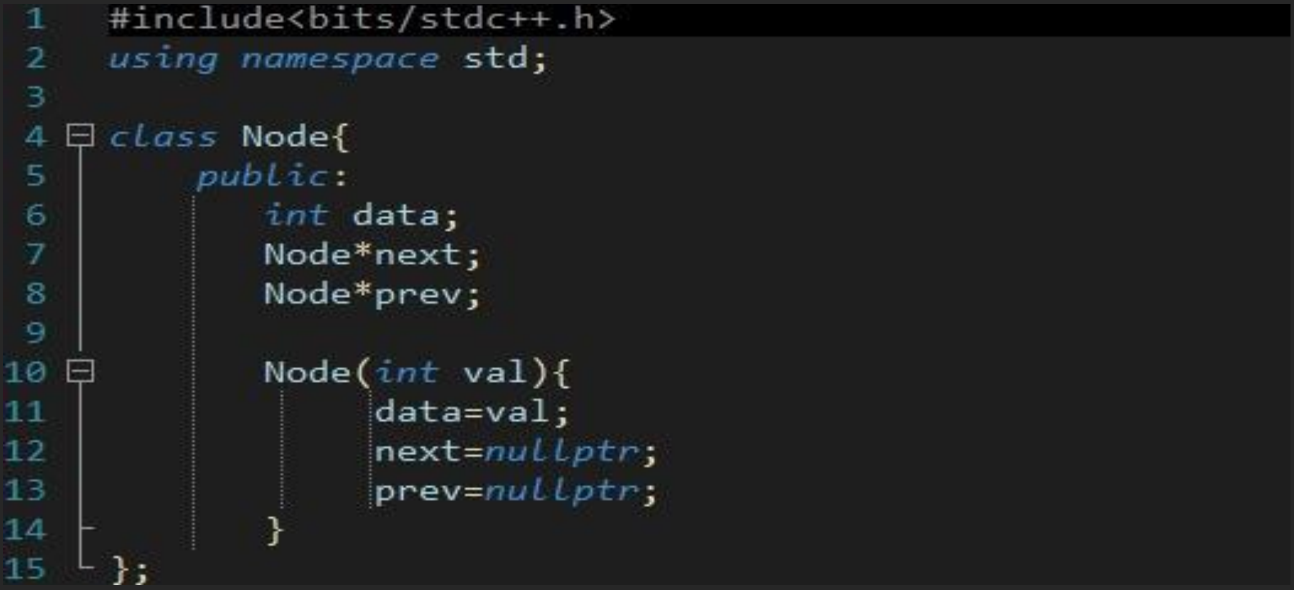


```
20<->10<->5<->1<->NULL
20<->15<->10<->5<->1<->NULL
20<->15<->5<->1<->NULL
15<->5<->NULL

--------------------------------
Process exited after 0.5906 seconds with return value 0
Press any key to continue . . .
```

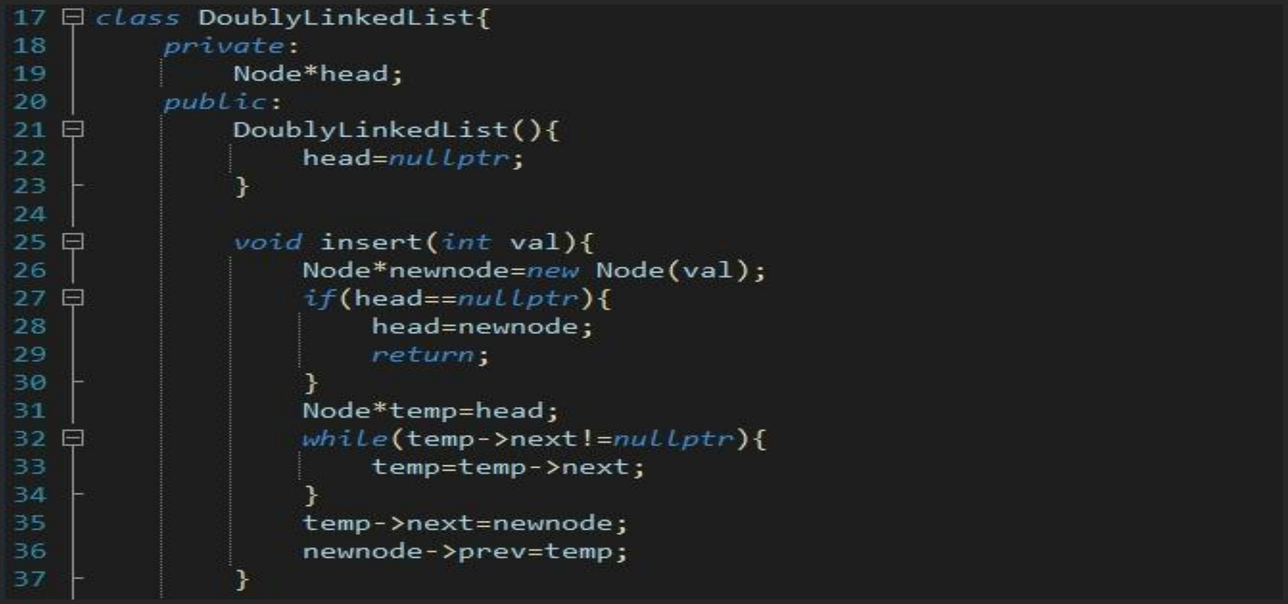
__Question 9:__

Write a function to reverse a doubly linked list. Your solution should correctly update the next and prev pointers for each node.

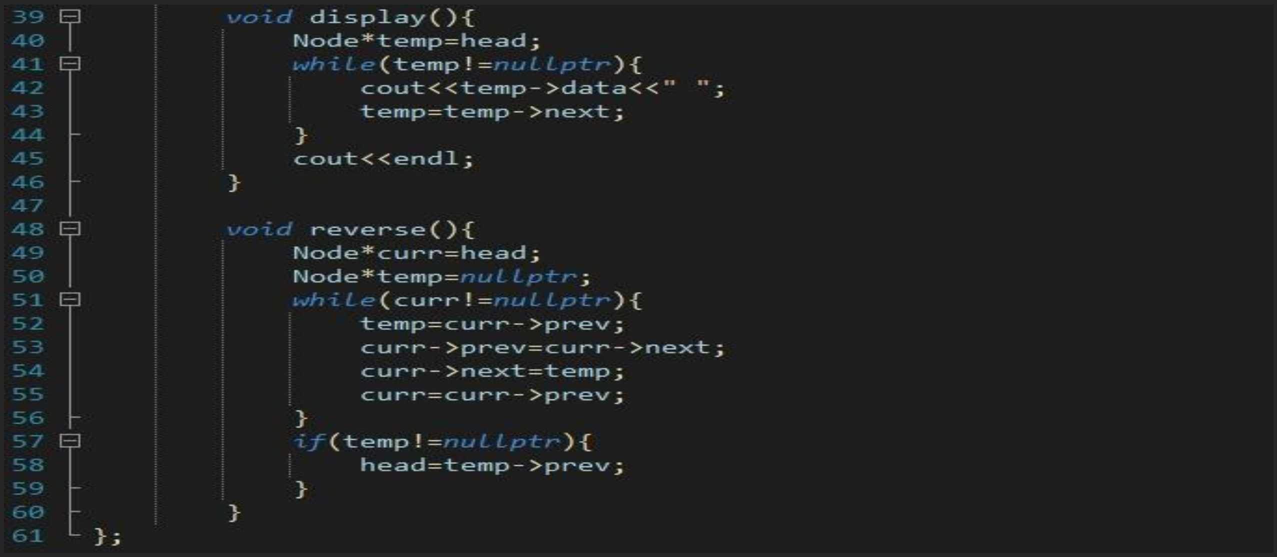
__SOURCE CODE:__



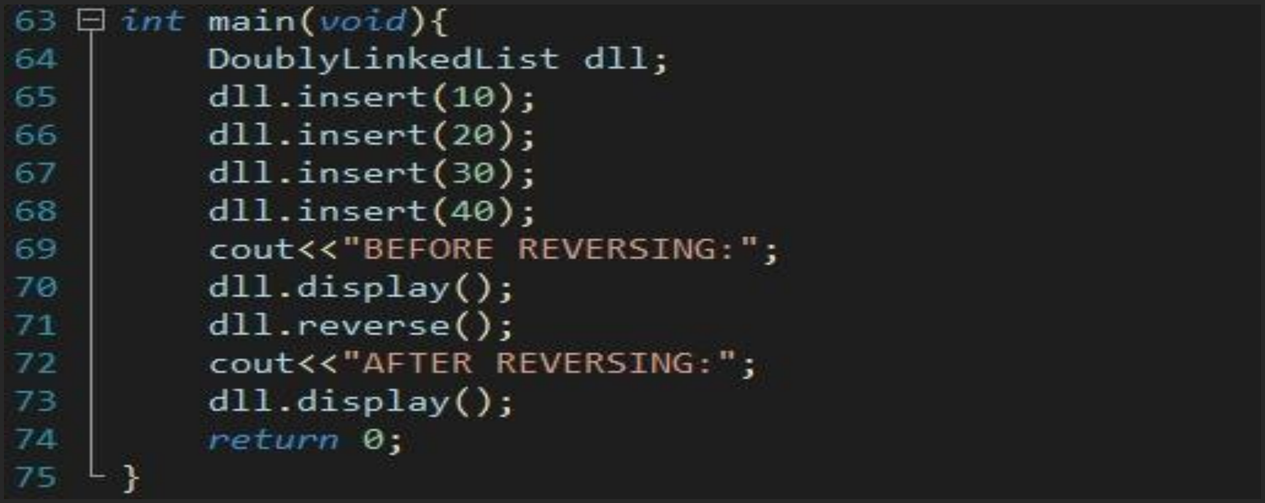
```cpp
#include<bits/stdc++.h>
using namespace std;

class Node{
    public:
        int data;
        Node*next;
        Node*prev;

        Node(int val){
            data=val;
            next=nullptr;
            prev=nullptr;
        }
};
```



```cpp
class DoublyLinkedList{
    private:
        Node*head;
    public:
        DoublyLinkedList(){
            head=nullptr;
        }

        void insert(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                return;
            }
            Node*temp=head;
            while(temp->next!=nullptr){
                temp=temp->next;
            }
            temp->next=newnode;
            newnode->prev=temp;
        }
```

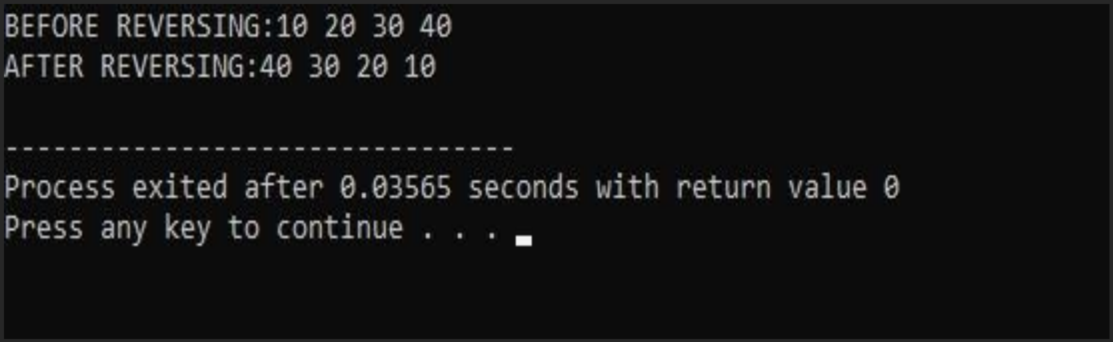


```cpp
        void display(){
            Node*temp=head;
            while(temp!=nullptr){
                cout<<temp->data<<" ";
                temp=temp->next;
            }
            cout<<endl;
        }

        void reverse(){
            Node*curr=head;
            Node*temp=nullptr;
            while(curr!=nullptr){
                temp=curr->prev;
                curr->prev=curr->next;
                curr->next=temp;
                curr=curr->prev;
            }
            if(temp!=nullptr){
                head=temp->prev;
            }
        }
};
```



```cpp
int main(void){
    DoublyLinkedList dll;
    dll.insert(10);
    dll.insert(20);
    dll.insert(30);
    dll.insert(40);
    cout<<"BEFORE REVERSING:";
    dll.display();
    dll.reverse();
    cout<<"AFTER REVERSING:";
    dll.display();
    return 0;
}
```

OUTPUT:

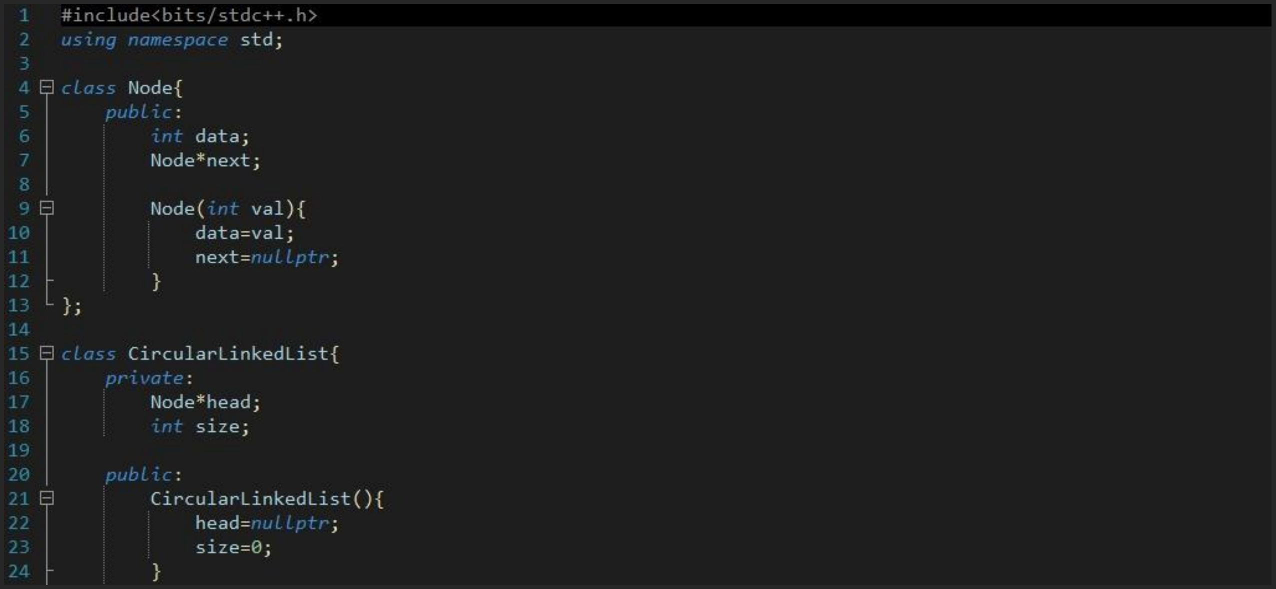


```
BEFORE REVERSING:10 20 30 40
AFTER REVERSING:40 30 20 10


--------------------------------
Process exited after 0.03565 seconds with return value 0
Press any key to continue . . .
```

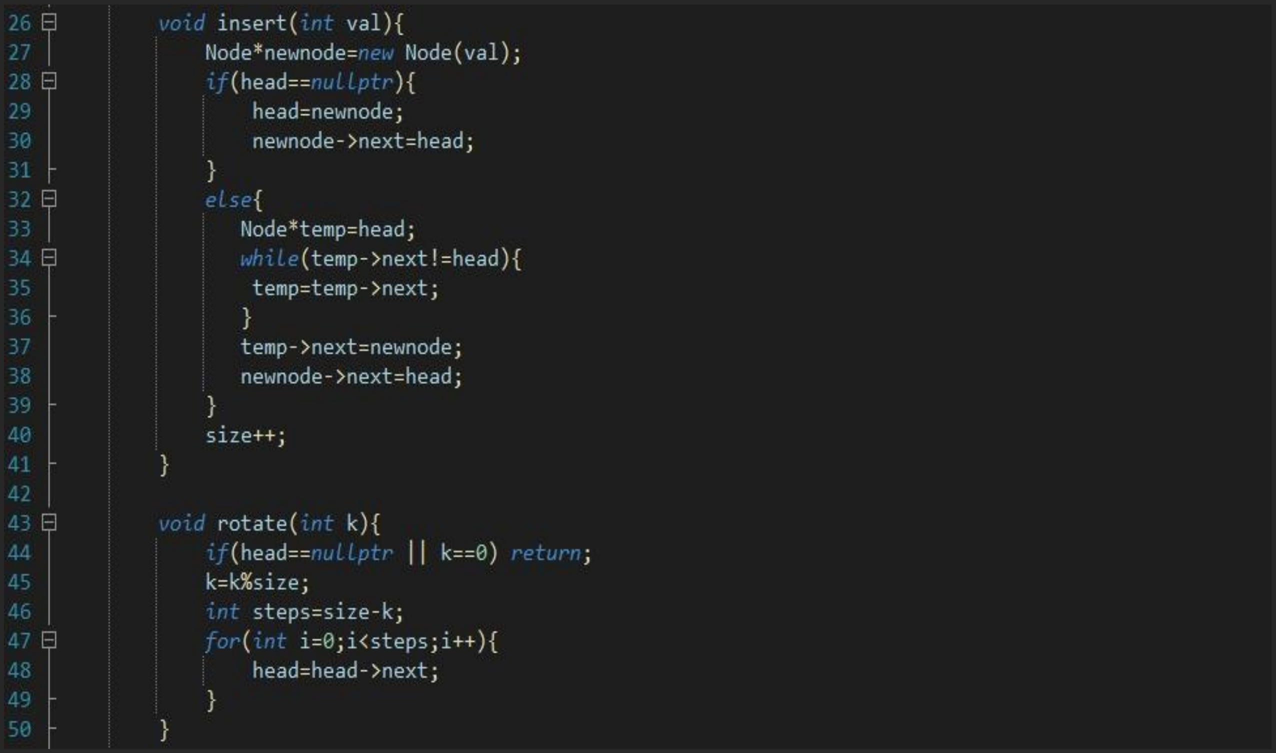
Question 10:

Given a circular singly linked list, rotate it by k positions. For example, with n=7 and k=3, the list 1, 2, 3, 4, 5, 6, 7 becomes 5, 6, 7, 1, 2, 3, 4.

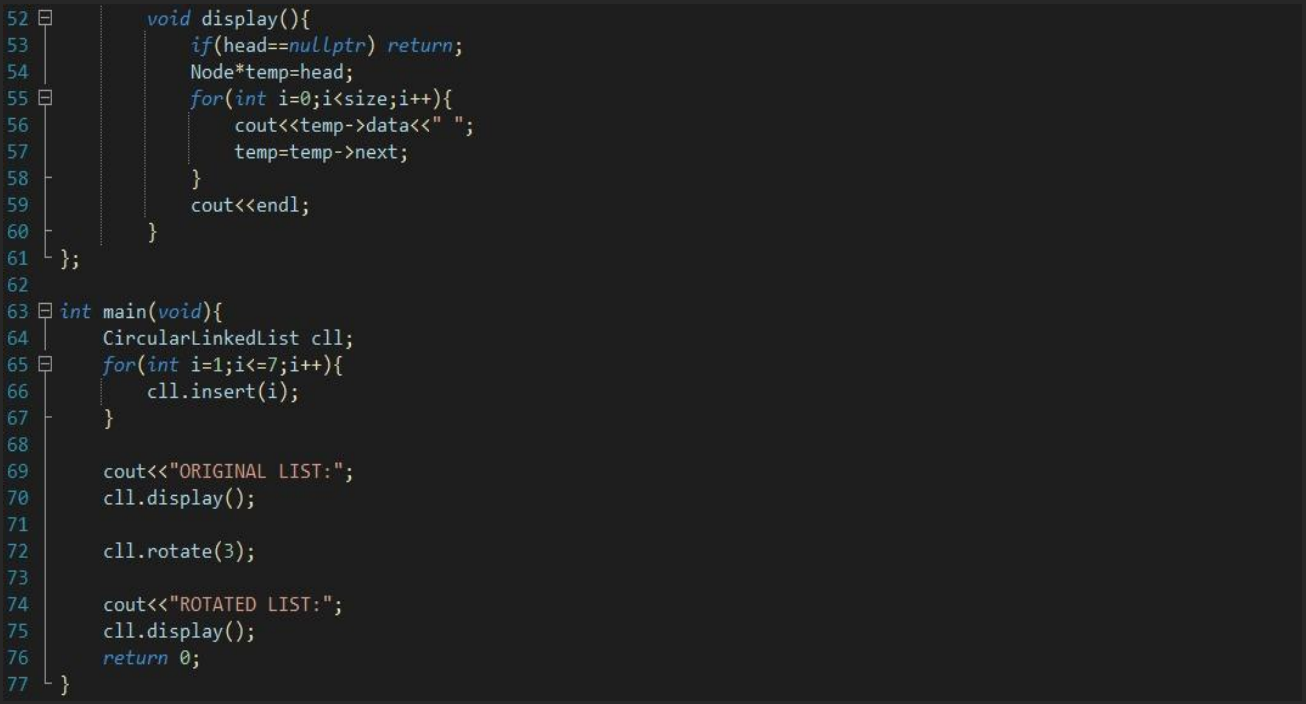
SOURCE CODE:



```cpp
#include<bits/stdc++.h>
using namespace std;

class Node{
    public:
        int data;
        Node*next;

        Node(int val){
            data=val;
            next=nullptr;
        }
};

class CircularLinkedList{
    private:
        Node*head;
        int size;

    public:
        CircularLinkedList(){
            head=nullptr;
            size=0;
        }
```

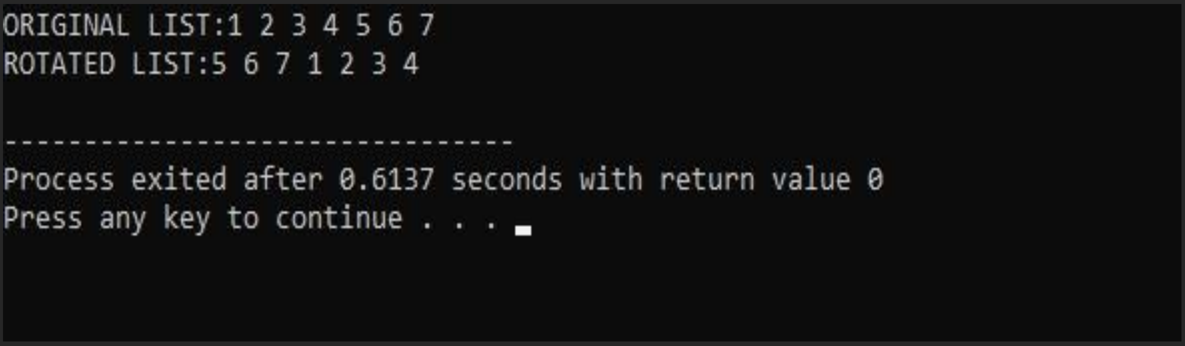


```cpp
        void insert(int val){
            Node*newnode=new Node(val);
            if(head==nullptr){
                head=newnode;
                newnode->next=head;
            }
            else{
               Node*temp=head;
               while(temp->next!=head){
                temp=temp->next;
               }
               temp->next=newnode;
               newnode->next=head;
            }
            size++;
        }

        void rotate(int k){
            if(head==nullptr || k==0) return;
            k=k%size;
            int steps=size-k;
            for(int i=0;i<steps;i++){
                head=head->next;
            }
        }
```



```cpp
        void display(){
            if(head==nullptr) return;
            Node*temp=head;
            for(int i=0;i<size;i++){
                cout<<temp->data<<" ";
                temp=temp->next;
            }
            cout<<endl;
        }
};

int main(void){
    CircularLinkedList cll;
    for(int i=1;i<=7;i++){
        cll.insert(i);
    }

    cout<<"ORIGINAL LIST:";
    cll.display();

    cll.rotate(3);

    cout<<"ROTATED LIST:";
    cll.display();
    return 0;
}
```

OUTPUT:



```
ORIGINAL LIST:1 2 3 4 5 6 7
ROTATED LIST:5 6 7 1 2 3 4


--------------------------------
Process exited after 0.6137 seconds with return value 0
Press any key to continue . . .
```