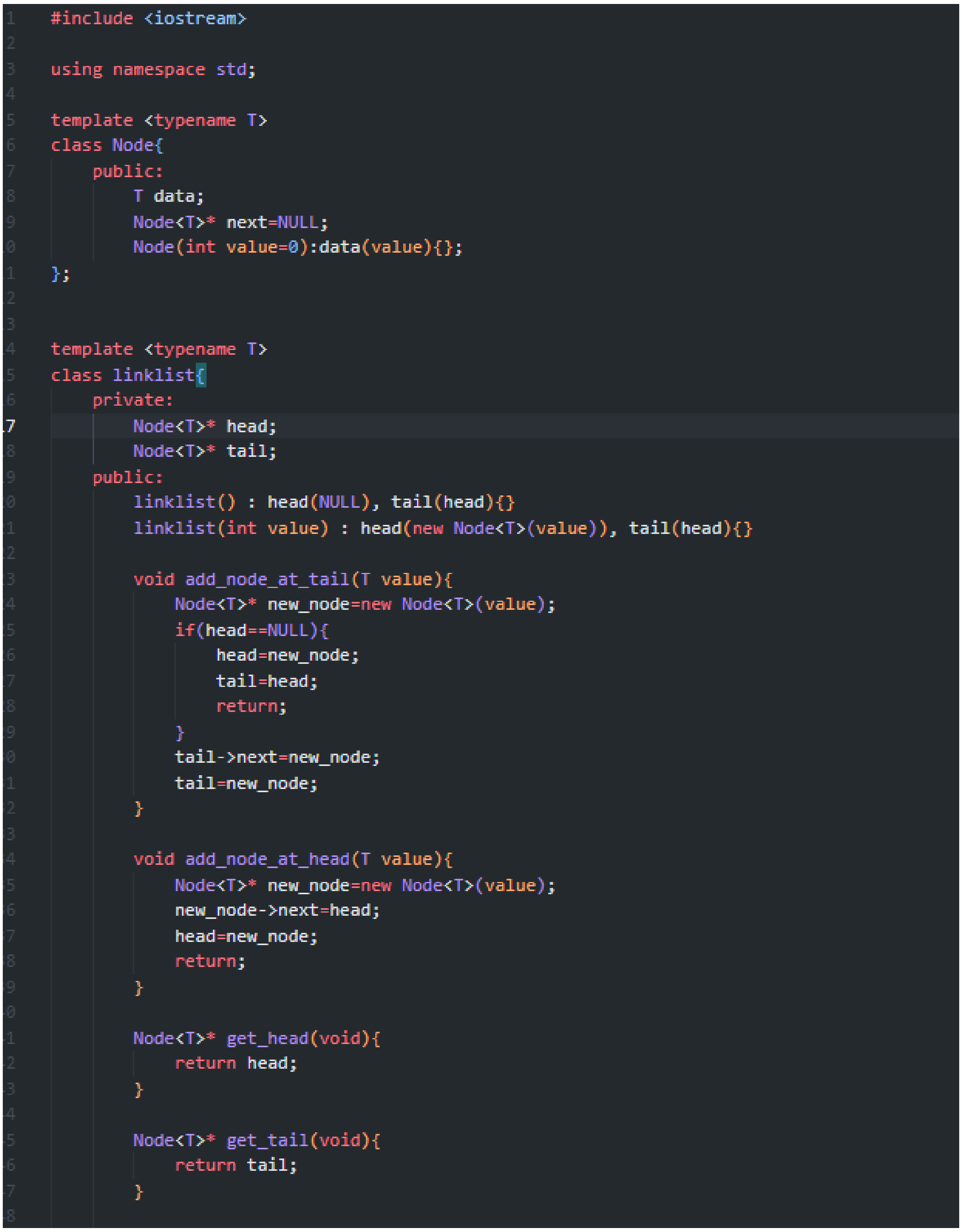
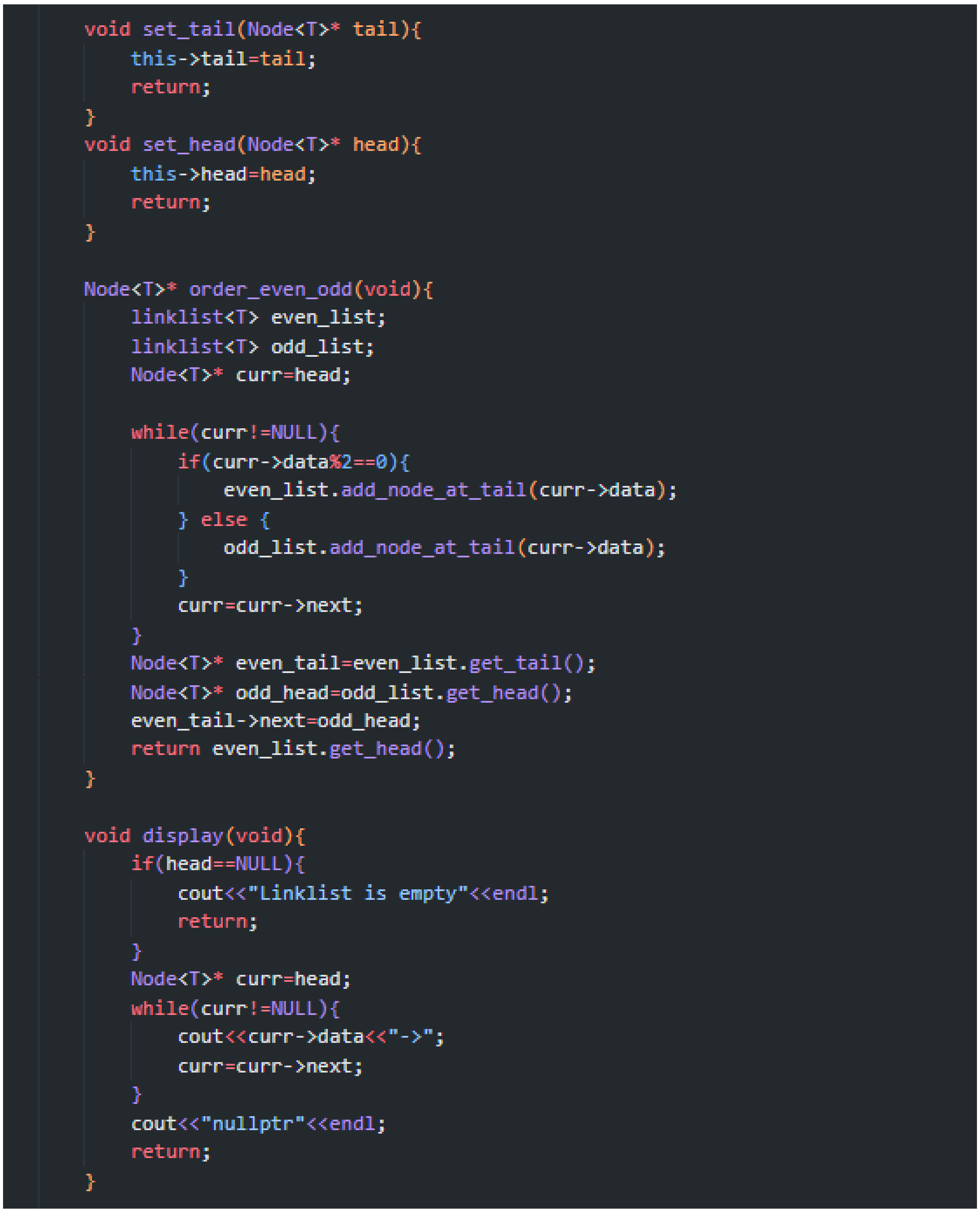
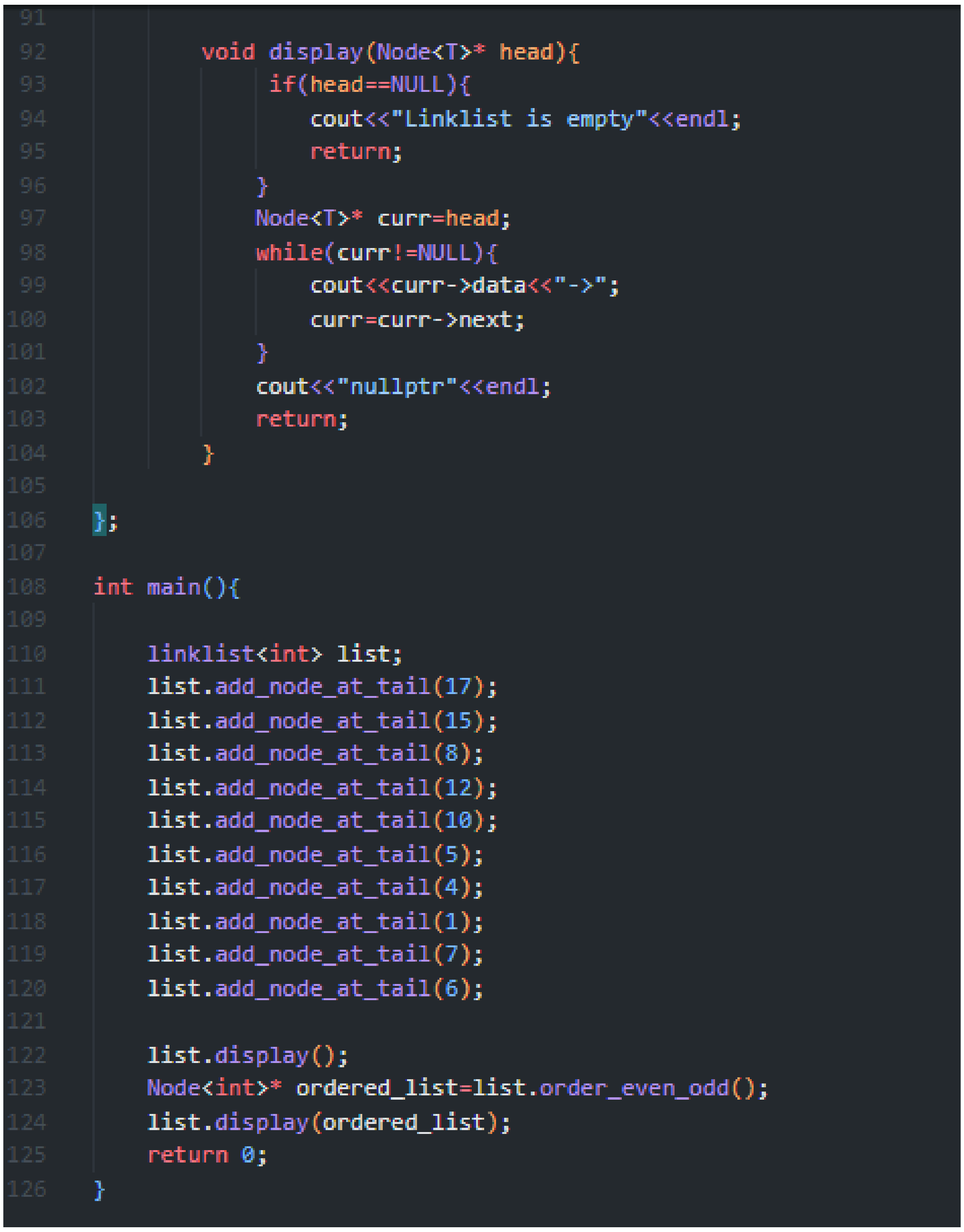
TASK 01:



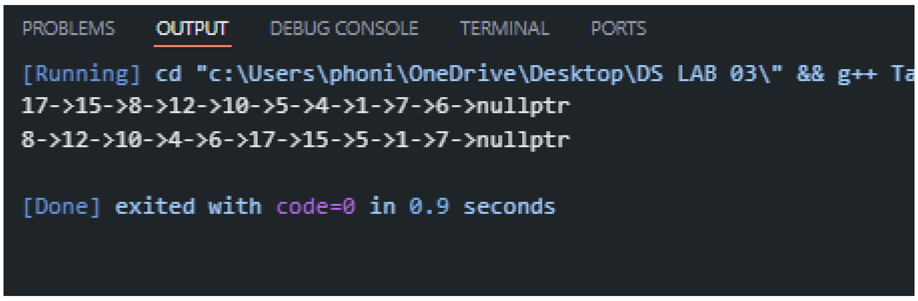
```cpp
#include <iostream>

using namespace std;

template <typename T>
class Node{
    public:
        T data;
        Node<T>* next=NULL;
        Node(int value=0):data(value){};
};


template <typename T>
class linklist{
    private:
        Node<T>* head;
        Node<T>* tail;
    public:
        linklist() : head(NULL), tail(head){}
        linklist(int value) : head(new Node<T>(value)), tail(head){}

        void add_node_at_tail(T value){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            tail->next=new_node;
            tail=new_node;
        }

        void add_node_at_head(T value){
            Node<T>* new_node=new Node<T>(value);
            new_node->next=head;
            head=new_node;
            return;
        }

        Node<T>* get_head(void){
            return head;
        }

        Node<T>* get_tail(void){
            return tail;
        }

```



```cpp
    void set_tail(Node<T>* tail){
        this->tail=tail;
        return;
    }
    void set_head(Node<T>* head){
        this->head=head;
        return;
    }

    Node<T>* order_even_odd(void){
        linklist<T> even_list;
        linklist<T> odd_list;
        Node<T>* curr=head;

        while(curr!=NULL){
            if(curr->data%2==0){
                even_list.add_node_at_tail(curr->data);
            } else {
                odd_list.add_node_at_tail(curr->data);
            }
            curr=curr->next;
        }
        Node<T>* even_tail=even_list.get_tail();
        Node<T>* odd_head=odd_list.get_head();
        even_tail->next=odd_head;
        return even_list.get_head();
    }

    void display(void){
        if(head==NULL){
            cout<<"Linklist is empty"<<endl;
            return;
        }
        Node<T>* curr=head;
        while(curr!=NULL){
            cout<<curr->data<<"->";
            curr=curr->next;
        }
        cout<<"nullptr"<<endl;
        return;
    }
```

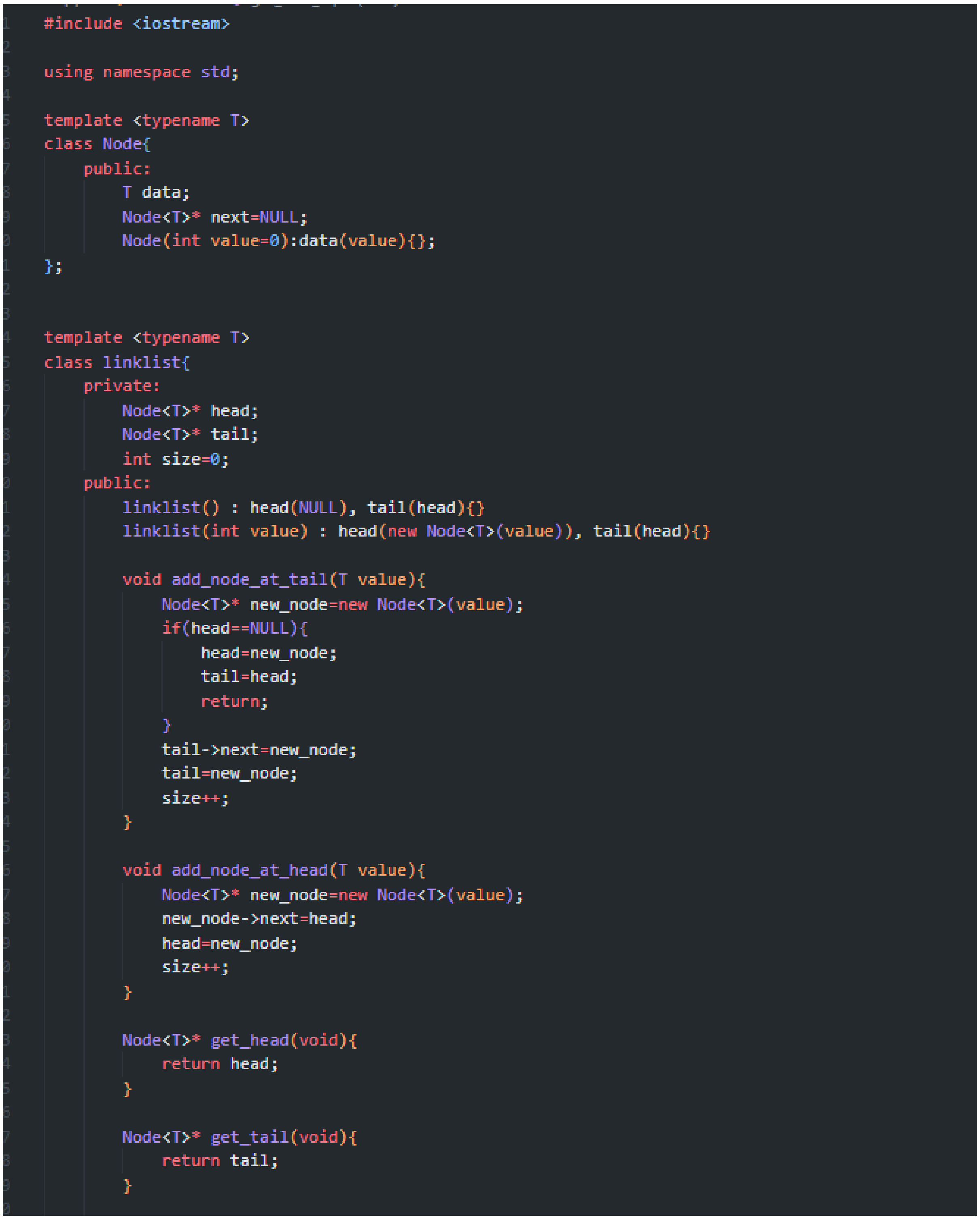


```cpp

        void display(Node<T>* head){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=NULL){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<"nullptr"<<endl;
            return;
        }

};

int main(){

    linklist<int> list;
    list.add_node_at_tail(17);
    list.add_node_at_tail(15);
    list.add_node_at_tail(8);
    list.add_node_at_tail(12);
    list.add_node_at_tail(10);
    list.add_node_at_tail(5);
    list.add_node_at_tail(4);
    list.add_node_at_tail(1);
    list.add_node_at_tail(7);
    list.add_node_at_tail(6);

    list.display();
    Node<int>* ordered_list=list.order_even_odd();
    list.display(ordered_list);
    return 0;
}
```

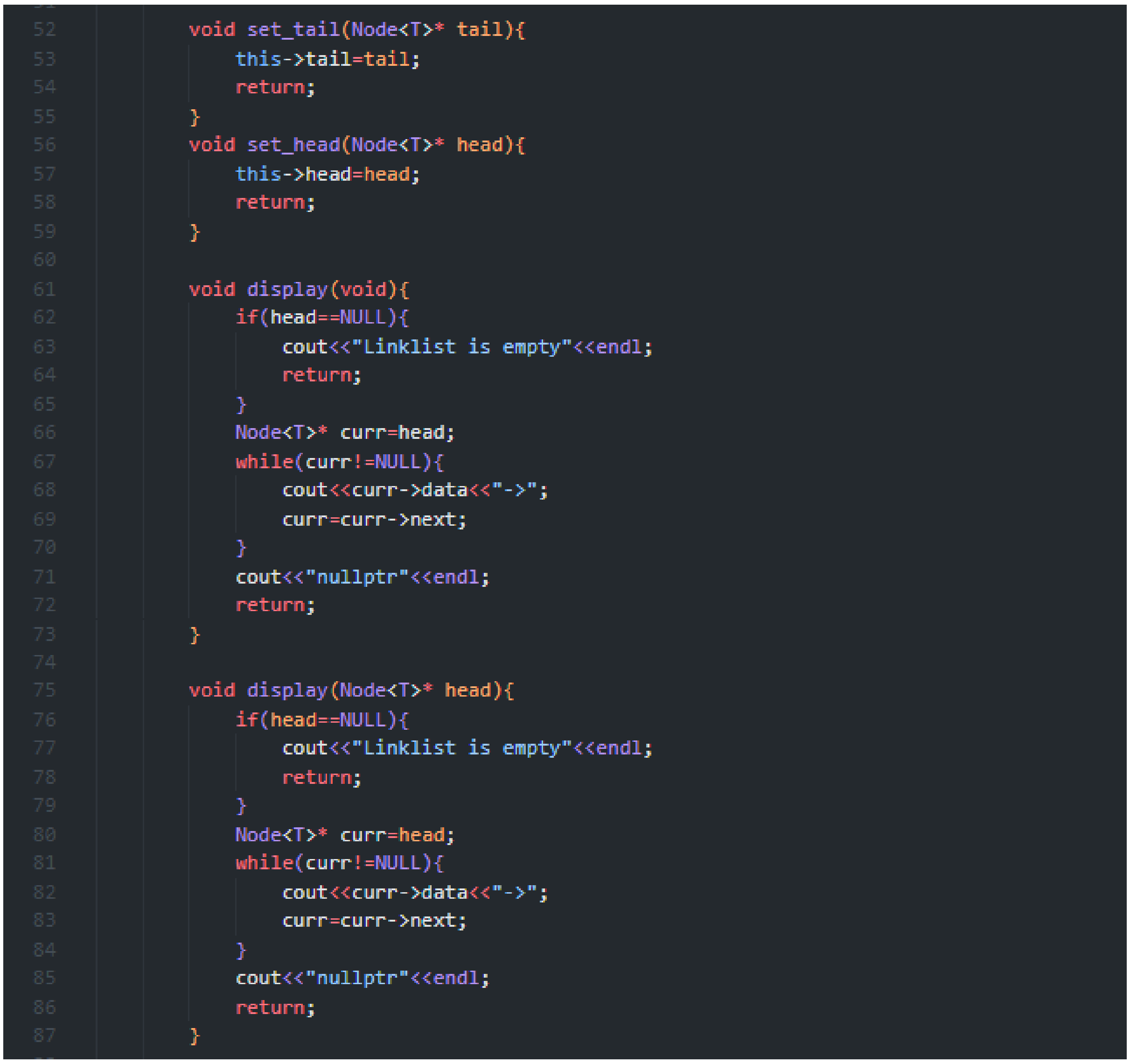


```
[Running] cd "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\" && g++ Ta
17->15->8->12->10->5->4->1->7->6->nullptr
8->12->10->4->6->17->15->5->1->7->nullptr

[Done] exited with code=0 in 0.9 seconds
```

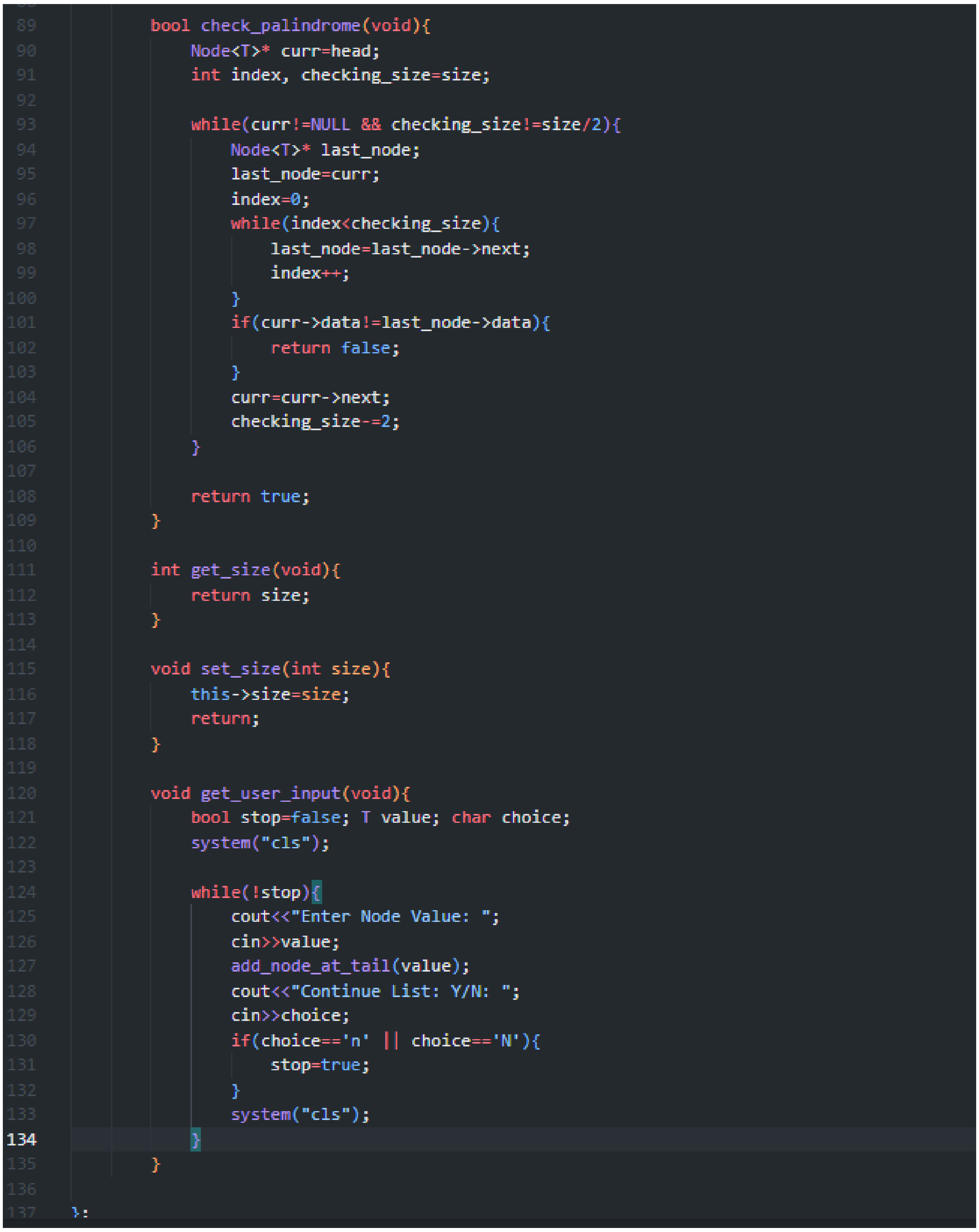
TASK 02:



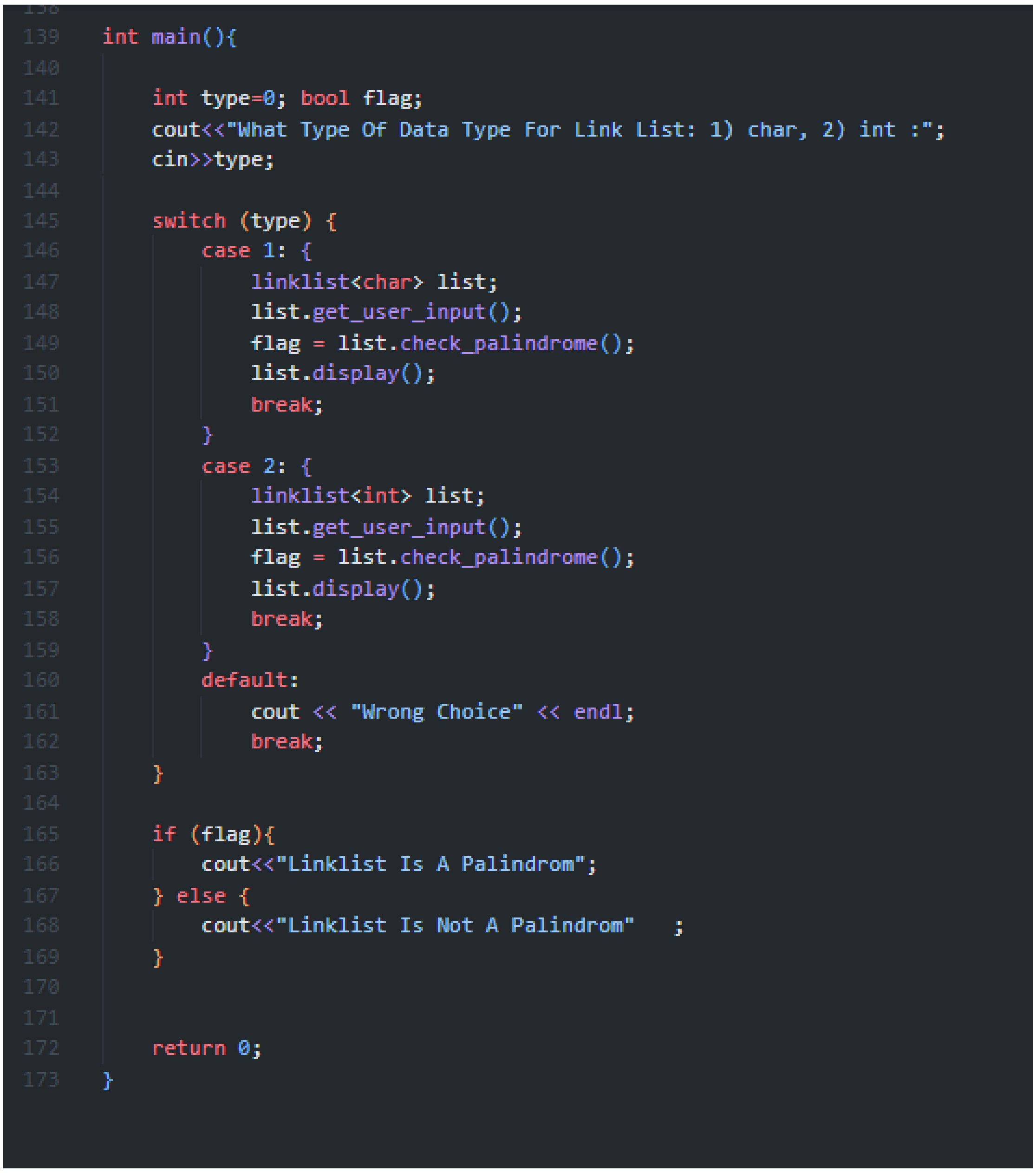
```cpp
#include <iostream>

using namespace std;

template <typename T>
class Node{
    public:
        T data;
        Node<T>* next=NULL;
        Node(int value=0):data(value){};
};


template <typename T>
class linklist{
    private:
        Node<T>* head;
        Node<T>* tail;
        int size=0;
    public:
        linklist() : head(NULL), tail(head){}
        linklist(int value) : head(new Node<T>(value)), tail(head){}

        void add_node_at_tail(T value){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            tail->next=new_node;
            tail=new_node;
            size++;
        }

        void add_node_at_head(T value){
            Node<T>* new_node=new Node<T>(value);
            new_node->next=head;
            head=new_node;
            size++;
        }

        Node<T>* get_head(void){
            return head;
        }

        Node<T>* get_tail(void){
            return tail;
        }

```



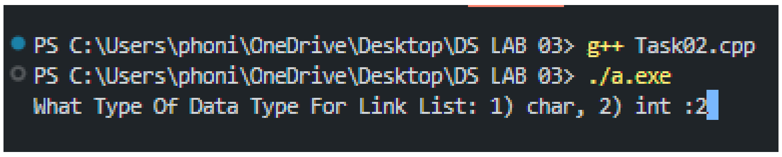
```cpp
        void set_tail(Node<T>* tail){
            this->tail=tail;
            return;
        }
        void set_head(Node<T>* head){
            this->head=head;
            return;
        }

        void display(void){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=NULL){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<"nullptr"<<endl;
            return;
        }

        void display(Node<T>* head){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=NULL){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<"nullptr"<<endl;
            return;
        }
```



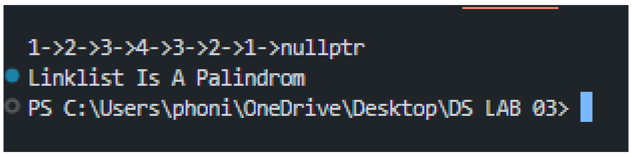
```cpp
        bool check_palindrome(void){
            Node<T>* curr=head;
            int index, checking_size=size;

            while(curr!=NULL && checking_size!=size/2){
                Node<T>* last_node;
                last_node=curr;
                index=0;
                while(index<checking_size){
                    last_node=last_node->next;
                    index++;
                }
                if(curr->data!=last_node->data){
                    return false;
                }
                curr=curr->next;
                checking_size-=2;
            }

            return true;
        }

        int get_size(void){
            return size;
        }

        void set_size(int size){
            this->size=size;
            return;
        }

        void get_user_input(void){
            bool stop=false; T value; char choice;
            system("cls");

            while(!stop){
                cout<<"Enter Node Value: ";
                cin>>value;
                add_node_at_tail(value);
                cout<<"Continue List: Y/N: ";
                cin>>choice;
                if(choice=='n' || choice=='N'){
                    stop=true;
                }
                system("cls");
            }
        }

};
```



```cpp
int main(){

    int type=0; bool flag;
    cout<<"What Type Of Data Type For Link List: 1) char, 2) int :";
    cin>>type;

    switch (type) {
        case 1: {
            linklist<char> list;
            list.get_user_input();
            flag = list.check_palindrome();
            list.display();
            break;
        }
        case 2: {
            linklist<int> list;
            list.get_user_input();
            flag = list.check_palindrome();
            list.display();
            break;
        }
        default:
            cout << "Wrong Choice" << endl;
            break;
    }

    if (flag){
        cout<<"Linklist Is A Palindrom";
    } else {
        cout<<"Linklist Is Not A Palindrom"    ;
    }


    return 0;
}
```

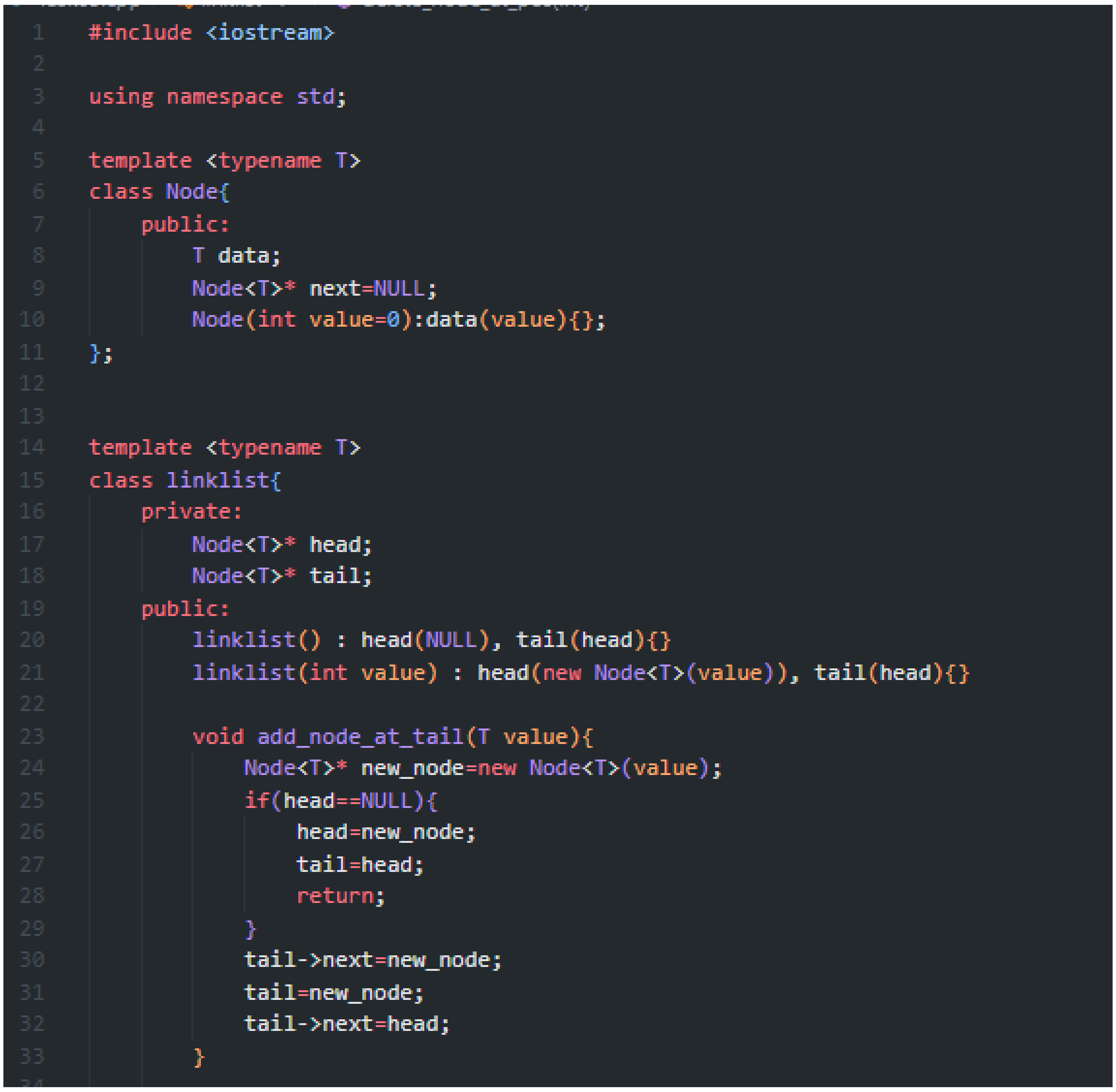


```
PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> g++ Task02.cpp
PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> ./a.exe
What Type Of Data Type For Link List: 1) char, 2) int :2
```

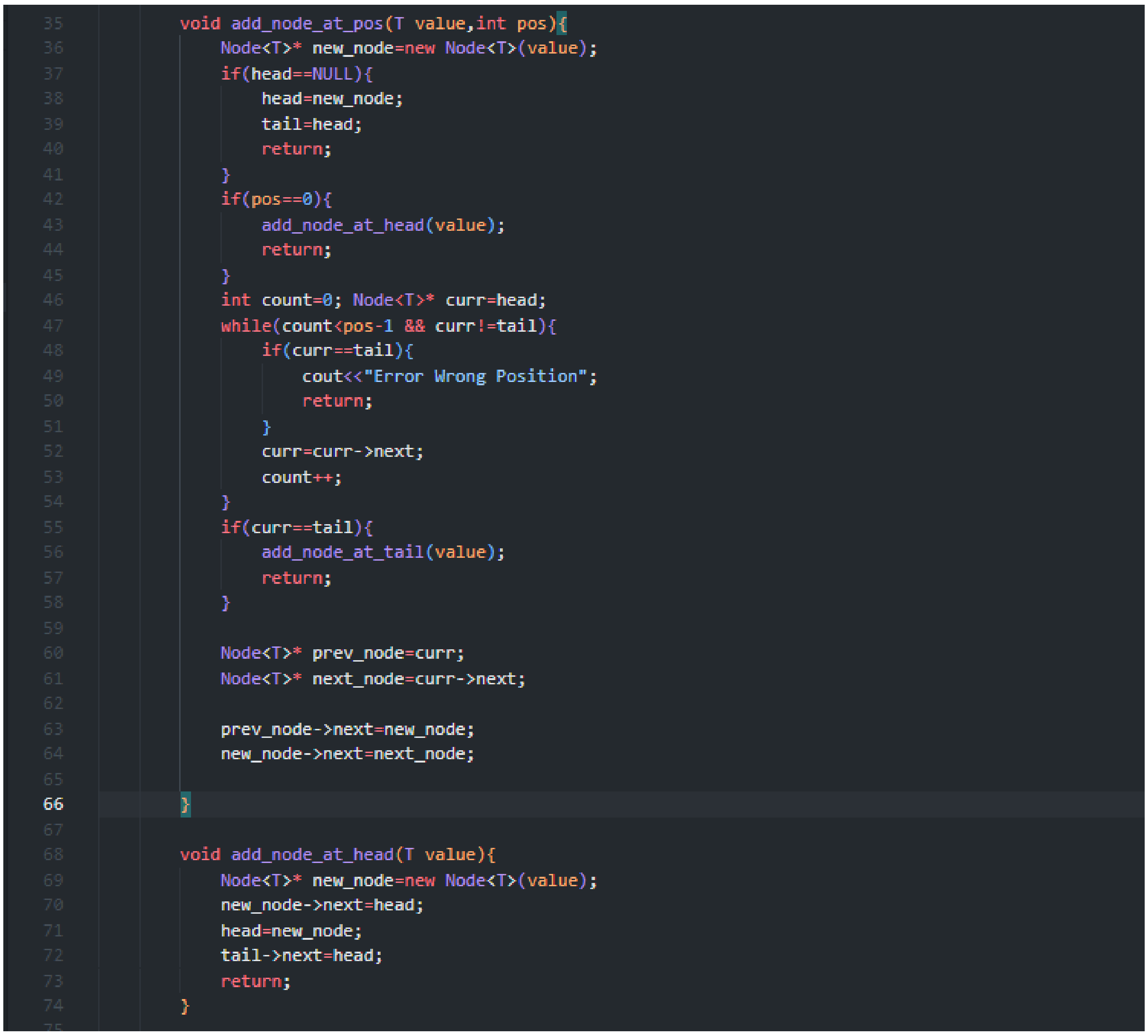


```
1->2->3->4->3->2->1->nullptr
Linklist Is A Palindrom
PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03>
```

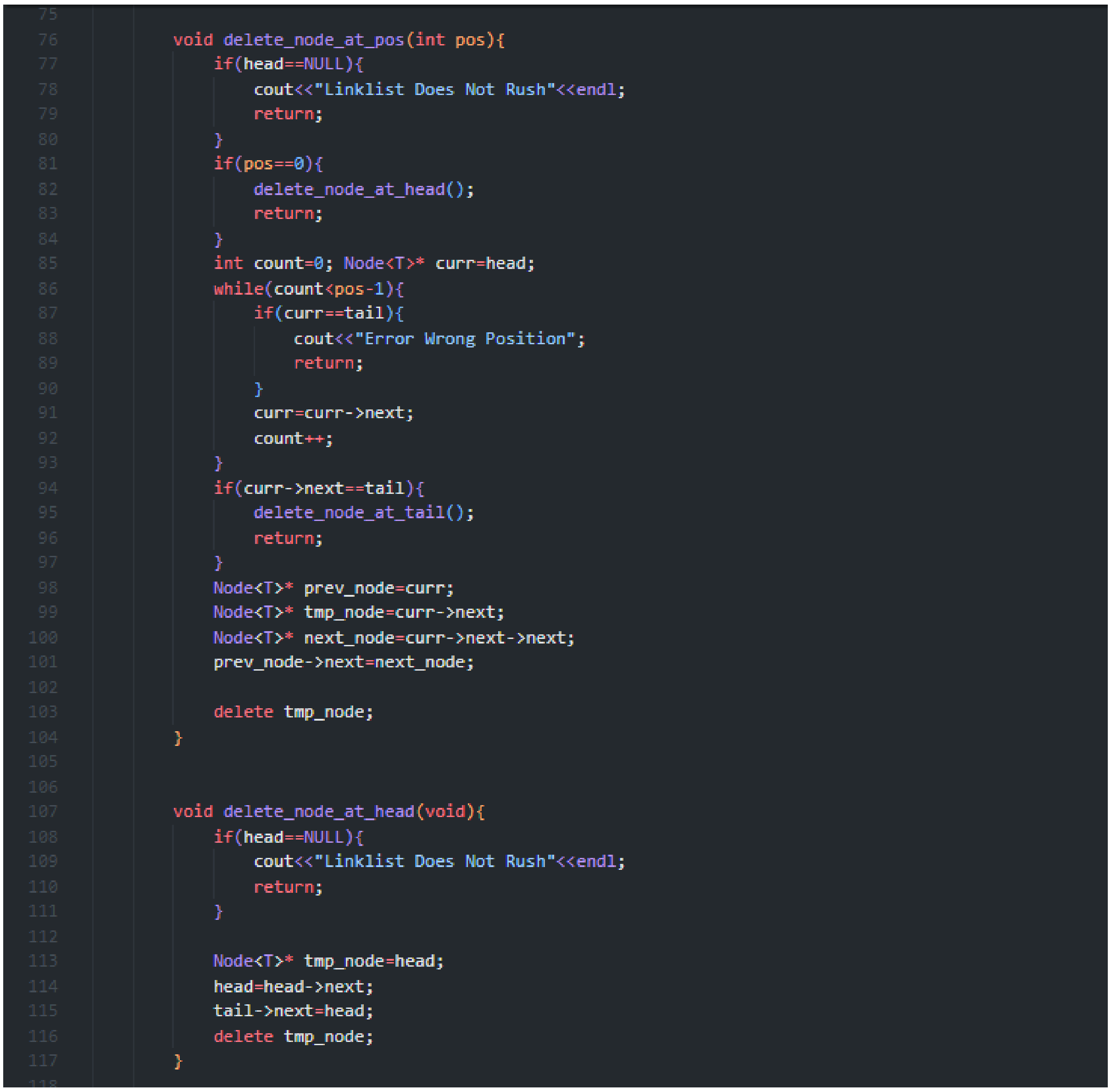
TASK 03:



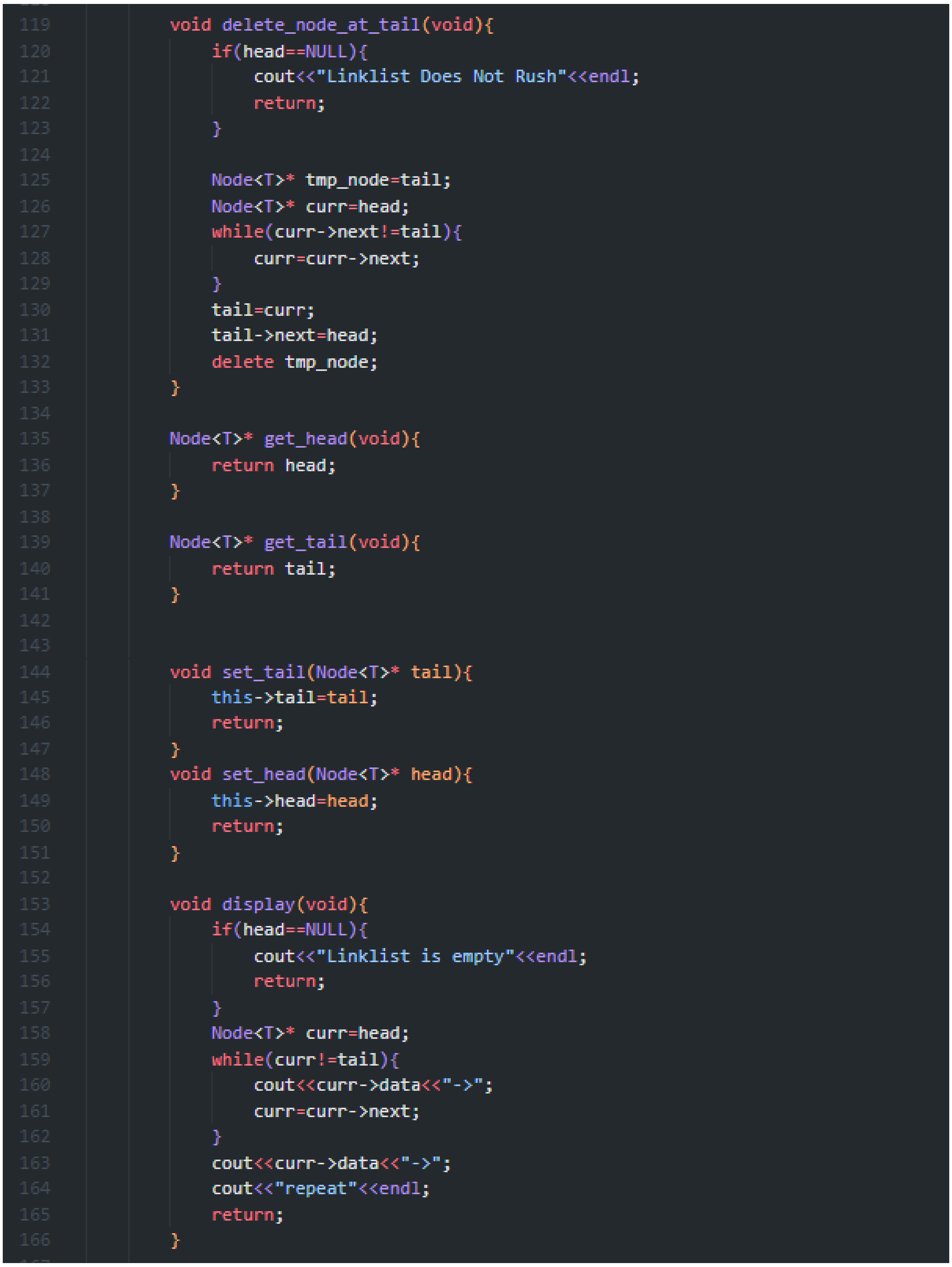
```cpp
#include <iostream>

using namespace std;

template <typename T>
class Node{
    public:
        T data;
        Node<T>* next=NULL;
        Node(int value=0):data(value){};
};


template <typename T>
class linklist{
    private:
        Node<T>* head;
        Node<T>* tail;
    public:
        linklist() : head(NULL), tail(head){}
        linklist(int value) : head(new Node<T>(value)), tail(head){}

        void add_node_at_tail(T value){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            tail->next=new_node;
            tail=new_node;
            tail->next=head;
        }
```

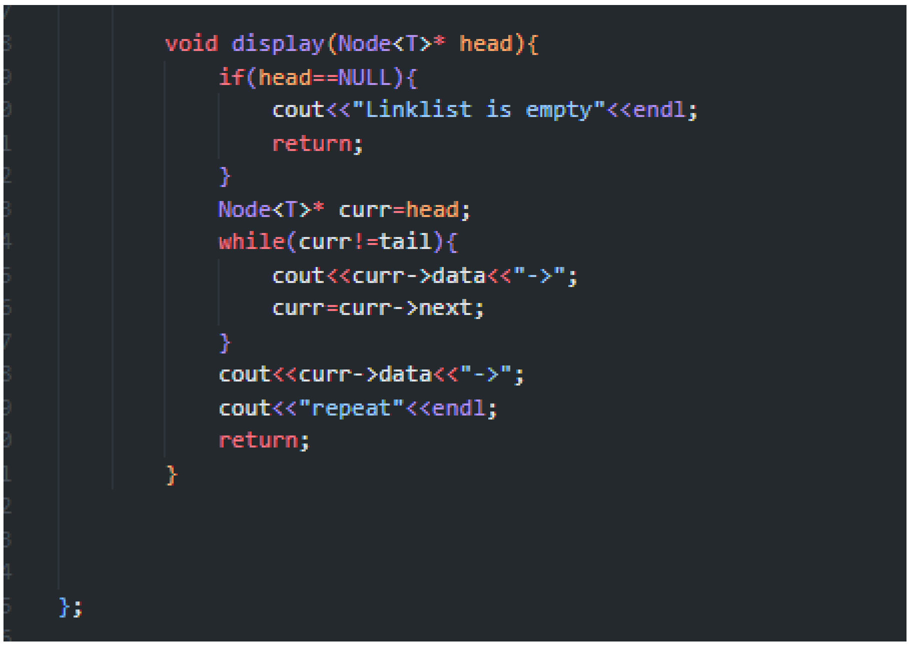


```
void add_node_at_pos(T value,int pos){
    Node<T>* new_node=new Node<T>(value);
    if(head==NULL){
        head=new_node;
        tail=head;
        return;
    }
    if(pos==0){
        add_node_at_head(value);
        return;
    }
    int count=0; Node<T>* curr=head;
    while(count<pos-1 && curr!=tail){
        if(curr==tail){
            cout<<"Error Wrong Position";
            return;
        }
        curr=curr->next;
        count++;
    }
    if(curr==tail){
        add_node_at_tail(value);
        return;
    }

    Node<T>* prev_node=curr;
    Node<T>* next_node=curr->next;

    prev_node->next=new_node;
    new_node->next=next_node;

}

void add_node_at_head(T value){
    Node<T>* new_node=new Node<T>(value);
    new_node->next=head;
    head=new_node;
    tail->next=head;
    return;
}
```

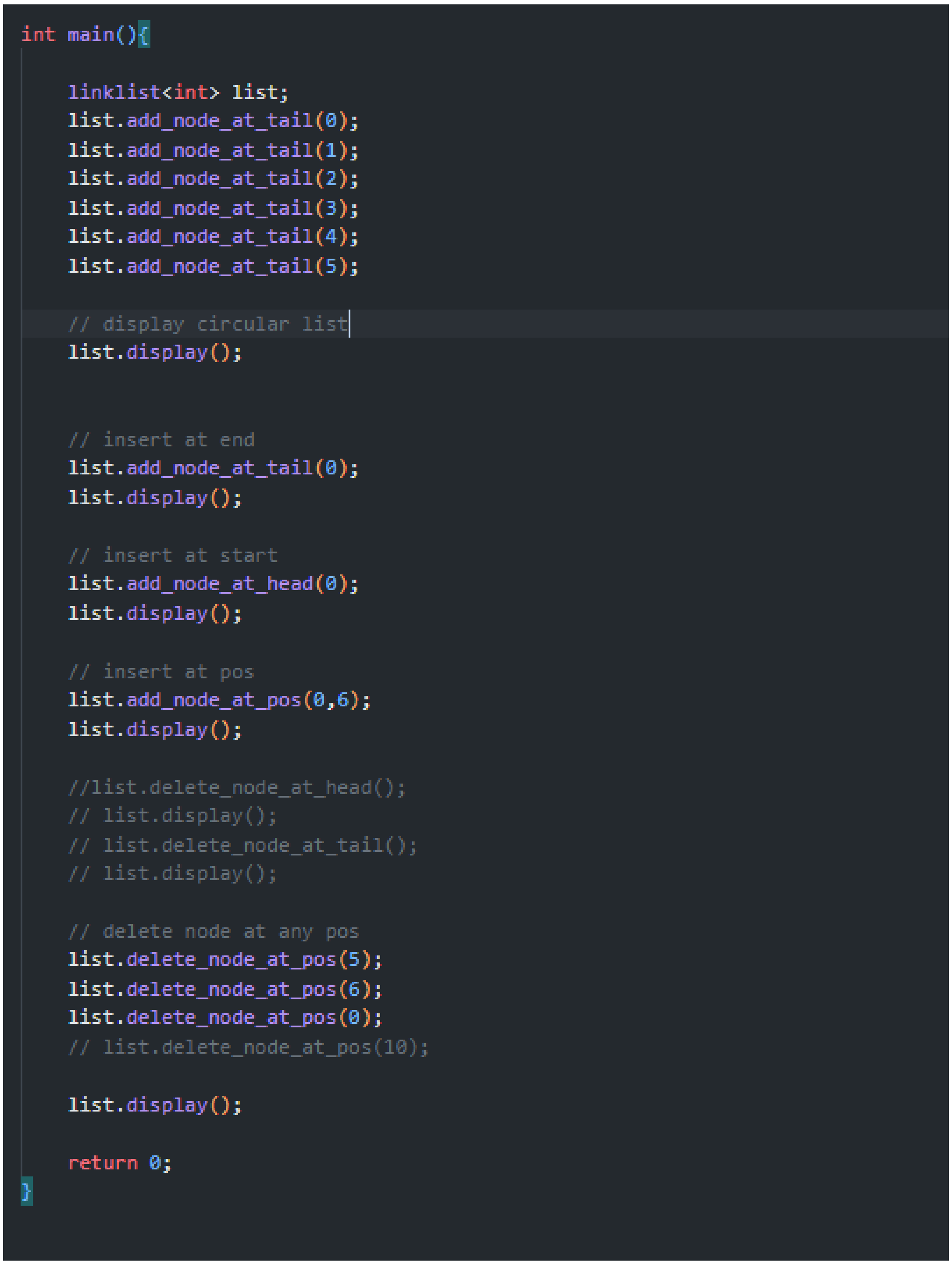


```
        void delete_node_at_pos(int pos){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }
            if(pos==0){
                delete_node_at_head();
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr->next==tail){
                delete_node_at_tail();
                return;
            }
            Node<T>* prev_node=curr;
            Node<T>* tmp_node=curr->next;
            Node<T>* next_node=curr->next->next;
            prev_node->next=next_node;

            delete tmp_node;
        }


        void delete_node_at_head(void){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }

            Node<T>* tmp_node=head;
            head=head->next;
            tail->next=head;
            delete tmp_node;
        }
```

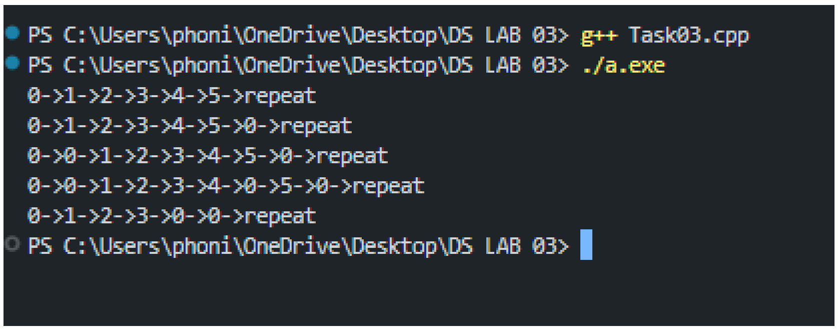

```
        void delete_node_at_tail(void){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }

            Node<T>* tmp_node=tail;
            Node<T>* curr=head;
            while(curr->next!=tail){
                curr=curr->next;
            }
            tail=curr;
            tail->next=head;
            delete tmp_node;
        }

        Node<T>* get_head(void){
            return head;
        }

        Node<T>* get_tail(void){
            return tail;
        }


        void set_tail(Node<T>* tail){
            this->tail=tail;
            return;
        }
        void set_head(Node<T>* head){
            this->head=head;
            return;
        }

        void display(void){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=tail){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }
```



```
        void display(Node<T>* head){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=tail){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }



};
```

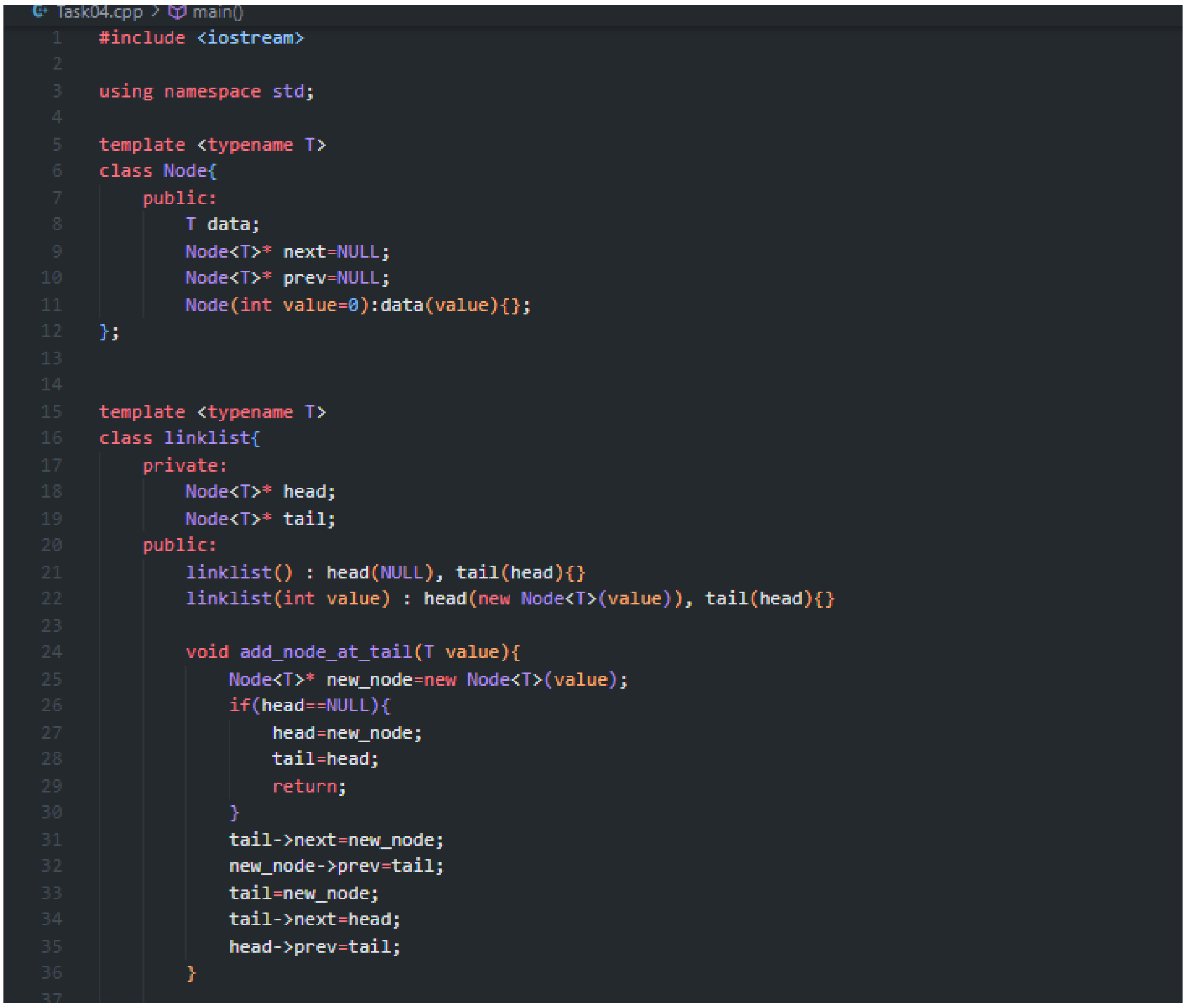


```cpp
int main(){

    linklist<int> list;
    list.add_node_at_tail(0);
    list.add_node_at_tail(1);
    list.add_node_at_tail(2);
    list.add_node_at_tail(3);
    list.add_node_at_tail(4);
    list.add_node_at_tail(5);

    // display circular list
    list.display();


    // insert at end
    list.add_node_at_tail(0);
    list.display();

    // insert at start
    list.add_node_at_head(0);
    list.display();

    // insert at pos
    list.add_node_at_pos(0,6);
    list.display();

    //list.delete_node_at_head();
    // list.display();
    // list.delete_node_at_tail();
    // list.display();

    // delete node at any pos
    list.delete_node_at_pos(5);
    list.delete_node_at_pos(6);
    list.delete_node_at_pos(0);
    // list.delete_node_at_pos(10);

    list.display();

    return 0;
}
```



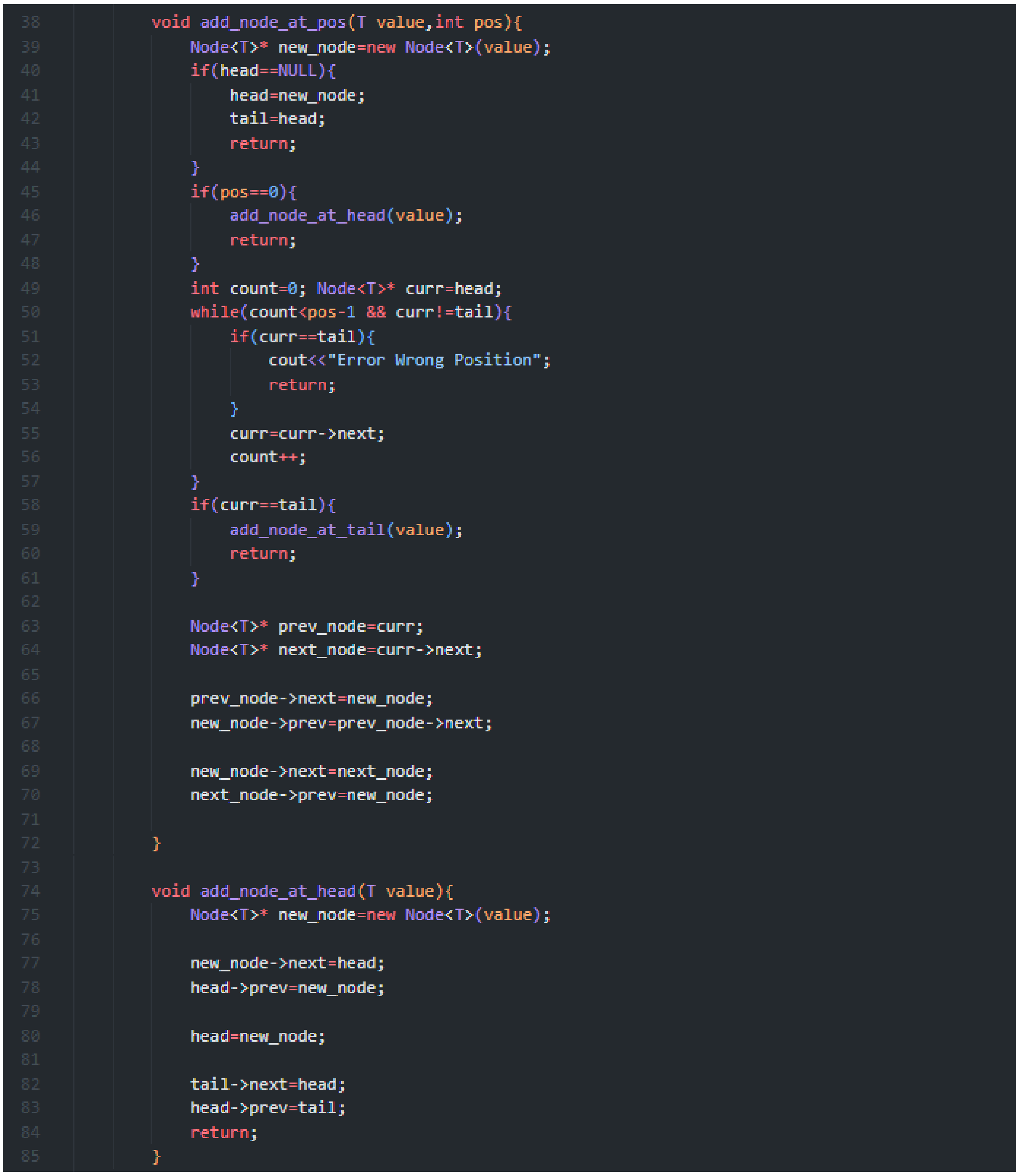
```
PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> g++ Task03.cpp
PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> ./a.exe
0->1->2->3->4->5->repeat
0->1->2->3->4->5->0->repeat
0->0->1->2->3->4->5->0->repeat
0->0->1->2->3->4->0->5->0->repeat
0->1->2->3->0->0->repeat
PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03>
```

TASK 04:

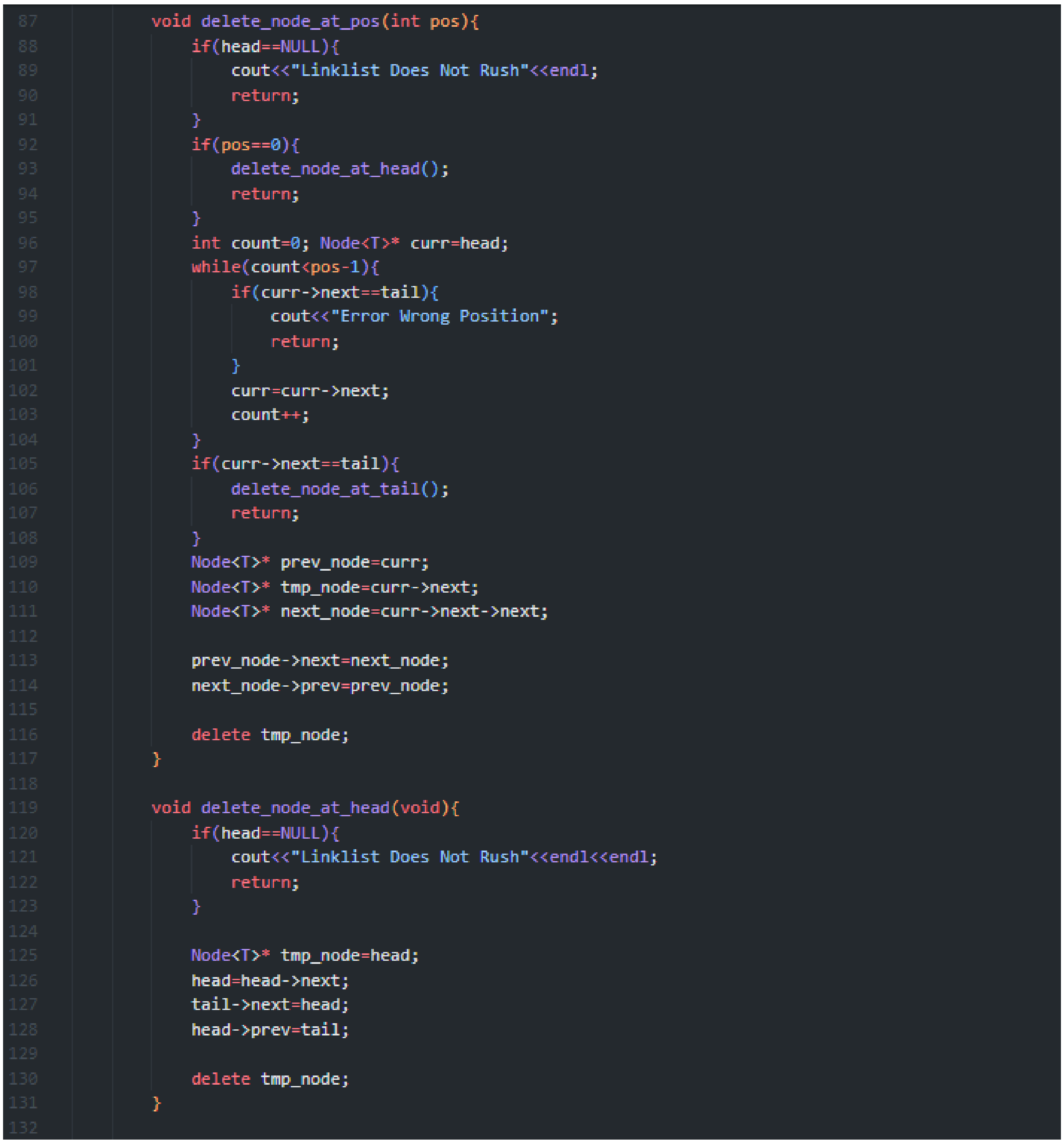


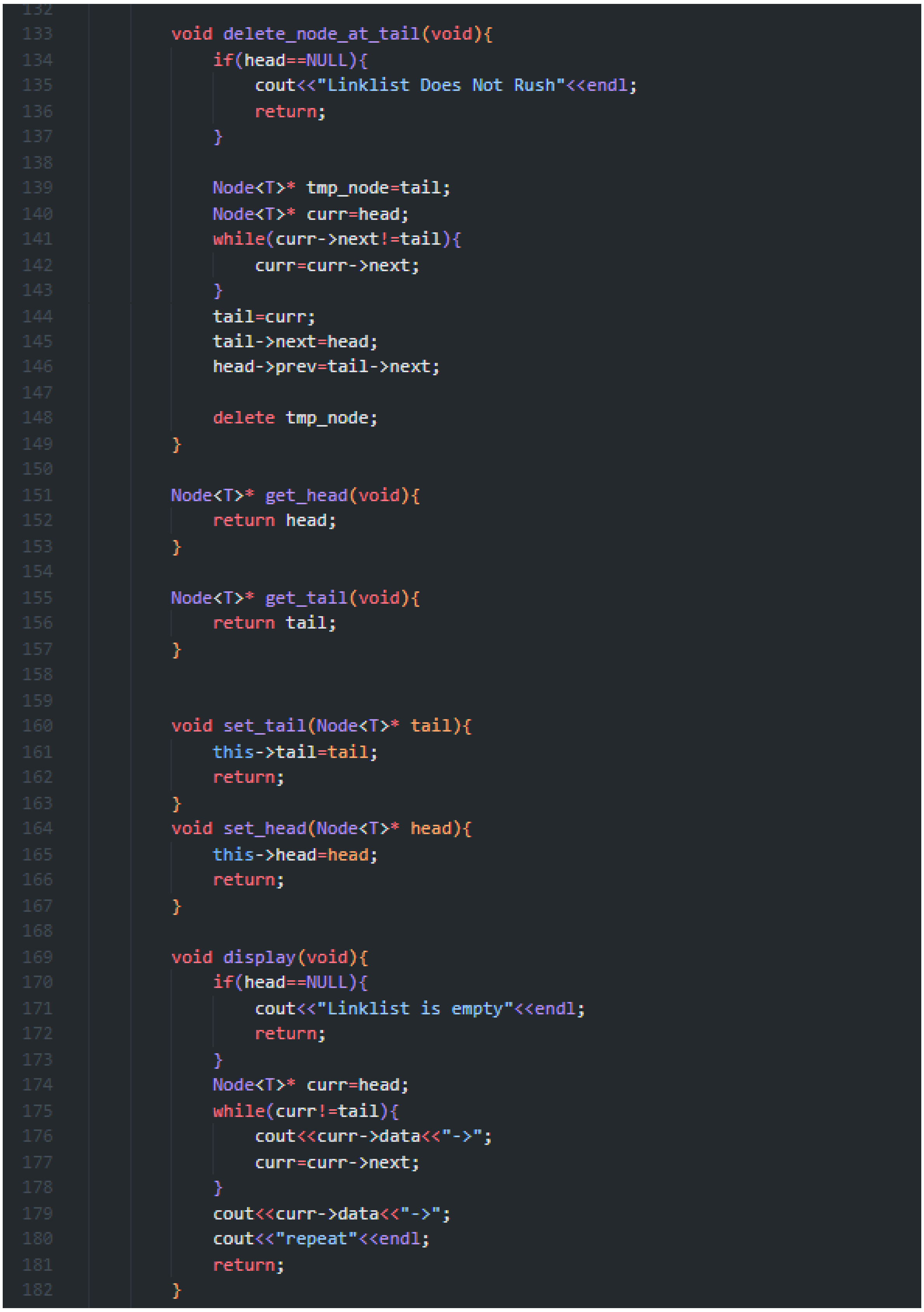
Task04.cpp > main()

```cpp
#include <iostream>

using namespace std;

template <typename T>
class Node{
    public:
        T data;
        Node<T>* next=NULL;
        Node<T>* prev=NULL;
        Node(int value=0):data(value){};
};


template <typename T>
class linklist{
    private:
        Node<T>* head;
        Node<T>* tail;
    public:
        linklist() : head(NULL), tail(head){}
        linklist(int value) : head(new Node<T>(value)), tail(head){}

        void add_node_at_tail(T value){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            tail->next=new_node;
            new_node->prev=tail;
            tail=new_node;
            tail->next=head;
            head->prev=tail;
        }
```

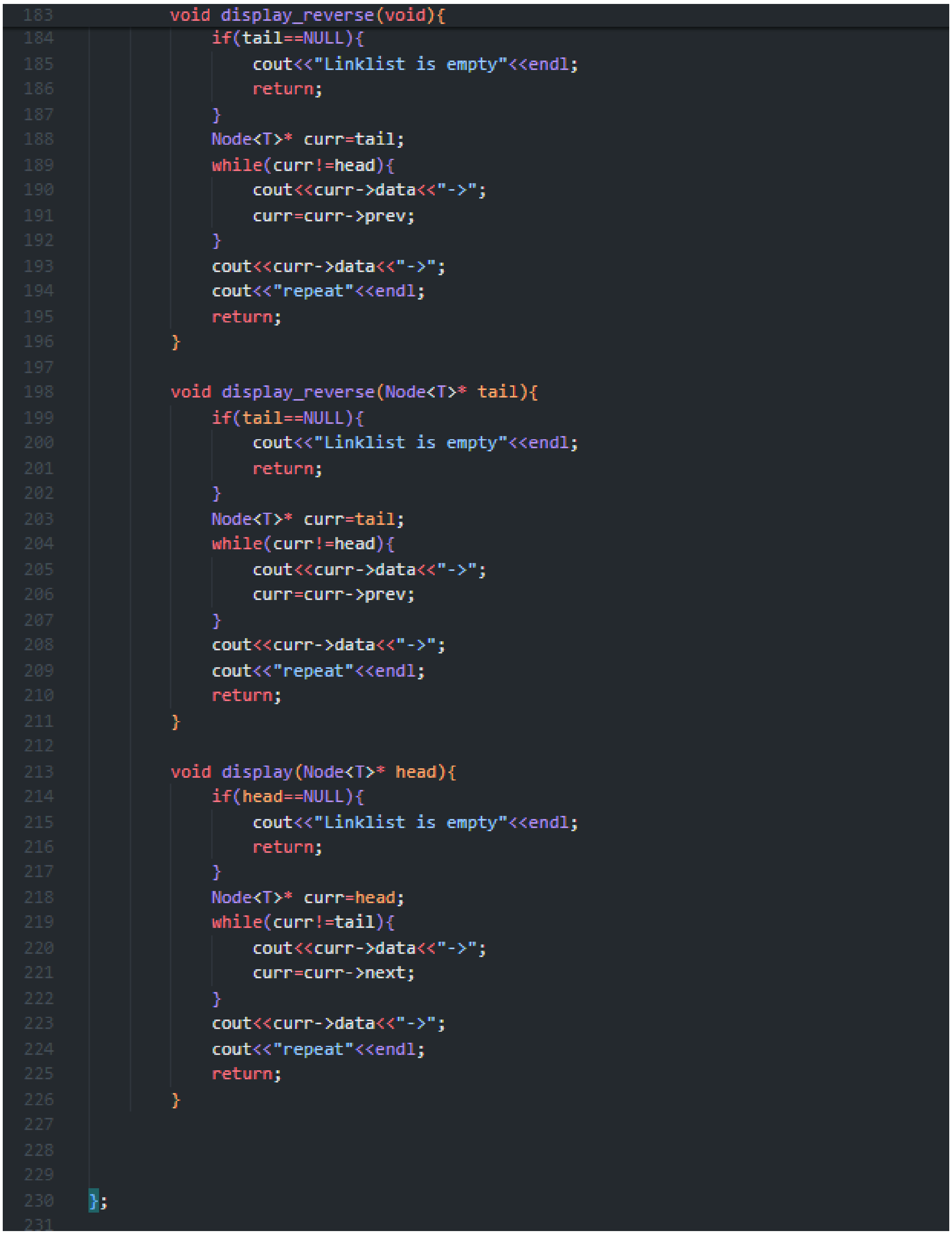


```cpp
        void add_node_at_pos(T value,int pos){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            if(pos==0){
                add_node_at_head(value);
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1 && curr!=tail){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr==tail){
                add_node_at_tail(value);
                return;
            }

            Node<T>* prev_node=curr;
            Node<T>* next_node=curr->next;

            prev_node->next=new_node;
            new_node->prev=prev_node->next;

            new_node->next=next_node;
            next_node->prev=new_node;

        }

        void add_node_at_head(T value){
            Node<T>* new_node=new Node<T>(value);

            new_node->next=head;
            head->prev=new_node;

            head=new_node;

            tail->next=head;
            head->prev=tail;
            return;
        }
```

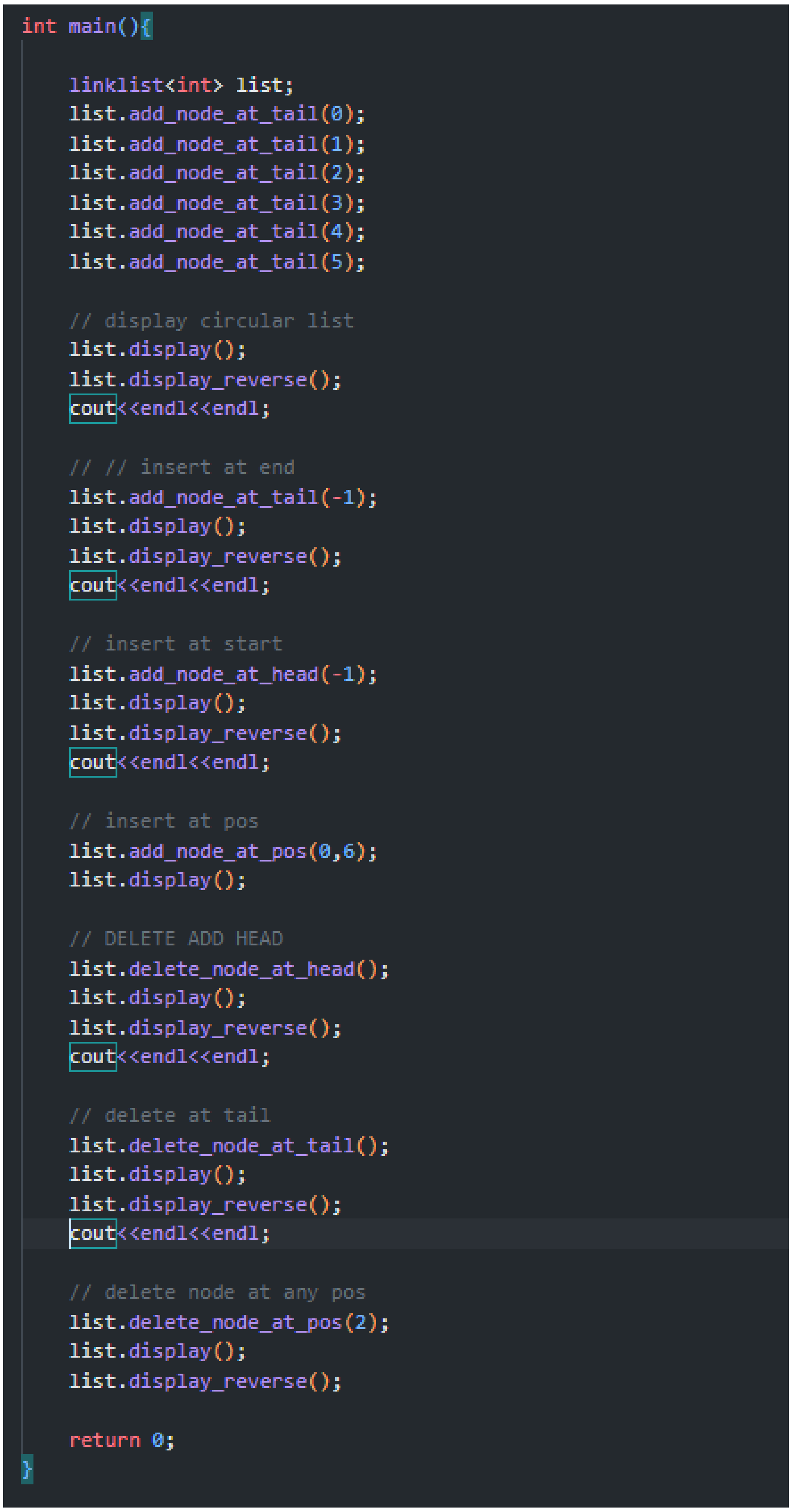

```
        void delete_node_at_pos(int pos){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }
            if(pos==0){
                delete_node_at_head();
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1){
                if(curr->next==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr->next==tail){
                delete_node_at_tail();
                return;
            }
            Node<T>* prev_node=curr;
            Node<T>* tmp_node=curr->next;
            Node<T>* next_node=curr->next->next;

            prev_node->next=next_node;
            next_node->prev=prev_node;

            delete tmp_node;
        }

        void delete_node_at_head(void){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl<<endl;
                return;
            }

            Node<T>* tmp_node=head;
            head=head->next;
            tail->next=head;
            head->prev=tail;

            delete tmp_node;
        }

```



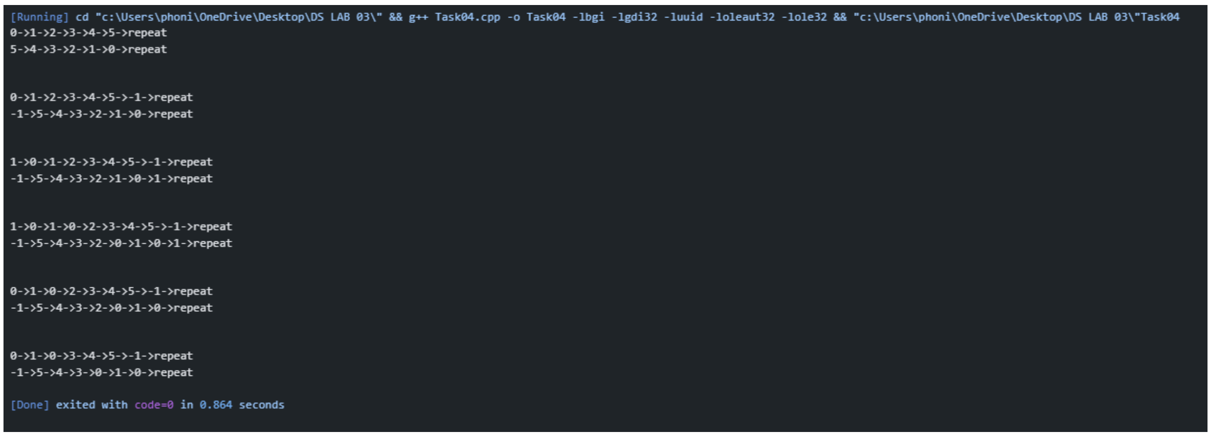
```cpp
        void delete_node_at_tail(void){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }

            Node<T>* tmp_node=tail;
            Node<T>* curr=head;
            while(curr->next!=tail){
                curr=curr->next;
            }
            tail=curr;
            tail->next=head;
            head->prev=tail->next;

            delete tmp_node;
        }

        Node<T>* get_head(void){
            return head;
        }

        Node<T>* get_tail(void){
            return tail;
        }


        void set_tail(Node<T>* tail){
            this->tail=tail;
            return;
        }
        void set_head(Node<T>* head){
            this->head=head;
            return;
        }

        void display(void){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=tail){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }
```



```
        void display_reverse(void){
            if(tail==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=tail;
            while(curr!=head){
                cout<<curr->data<<"->";
                curr=curr->prev;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }

        void display_reverse(Node<T>* tail){
            if(tail==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=tail;
            while(curr!=head){
                cout<<curr->data<<"->";
                curr=curr->prev;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }

        void display(Node<T>* head){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=tail){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }



};
```

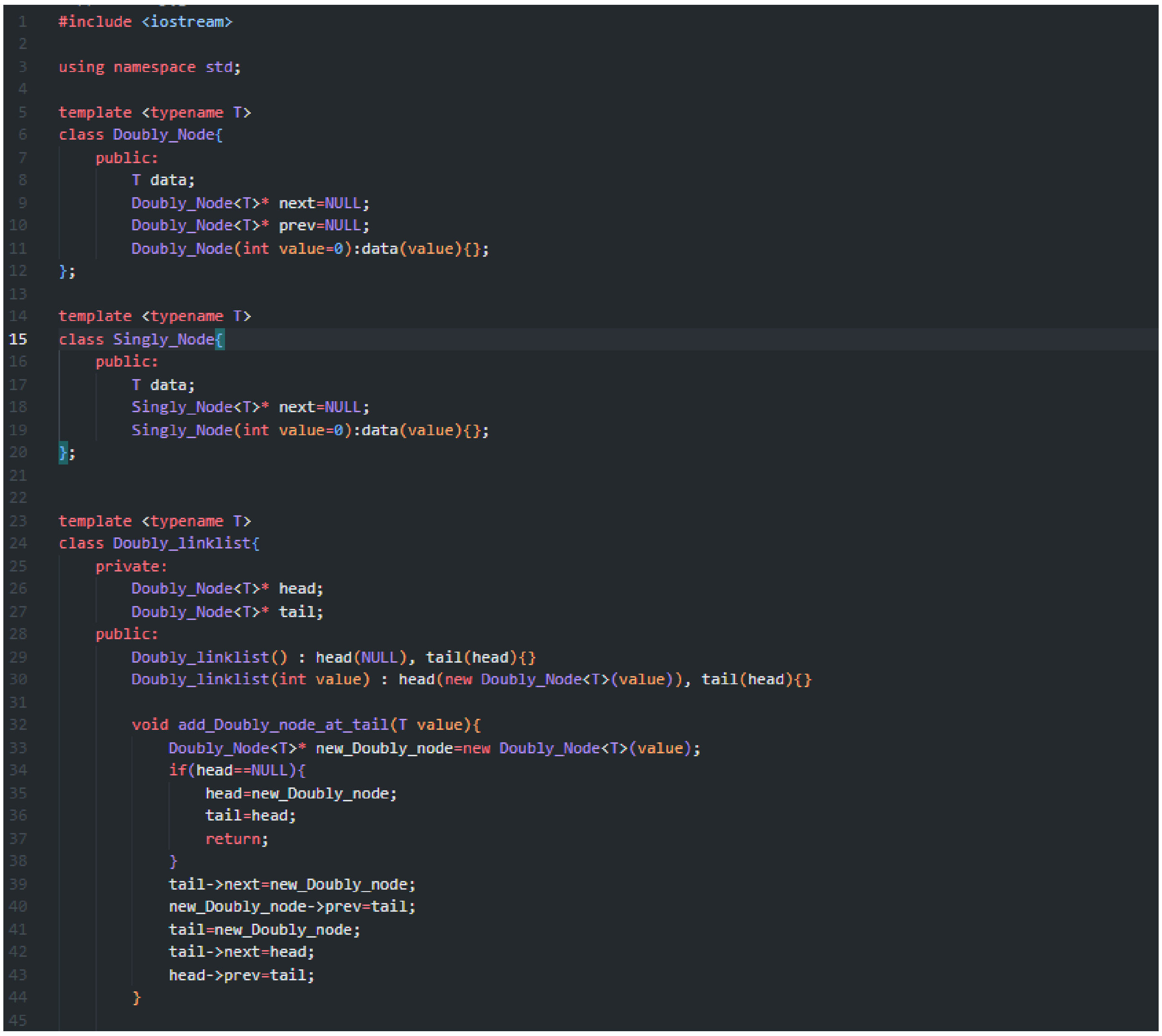


```cpp
int main(){

    linklist<int> list;
    list.add_node_at_tail(0);
    list.add_node_at_tail(1);
    list.add_node_at_tail(2);
    list.add_node_at_tail(3);
    list.add_node_at_tail(4);
    list.add_node_at_tail(5);

    // display circular list
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // // insert at end
    list.add_node_at_tail(-1);
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // insert at start
    list.add_node_at_head(-1);
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // insert at pos
    list.add_node_at_pos(0,6);
    list.display();

    // DELETE ADD HEAD
    list.delete_node_at_head();
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // delete at tail
    list.delete_node_at_tail();
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // delete node at any pos
    list.delete_node_at_pos(2);
    list.display();
    list.display_reverse();

    return 0;
}
```

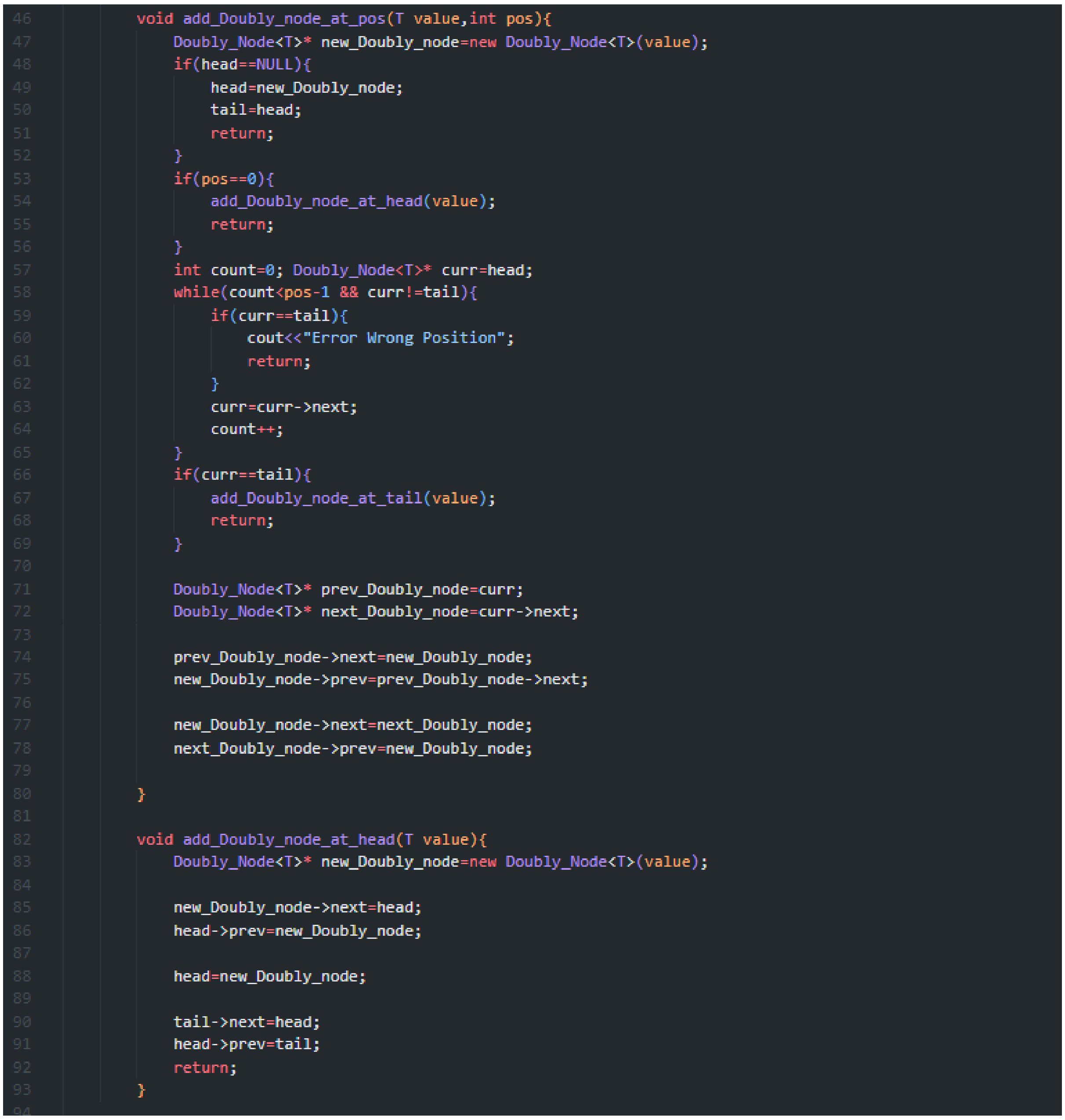


```
[Running] cd "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\" && g++ Task04.cpp -o Task04 -lbgi -lgdi32 -luuid -loleaut32 -lole32 && "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\"Task04
0->1->2->3->4->5->repeat
5->4->3->2->1->0->repeat


0->1->2->3->4->5->-1->repeat
-1->5->4->3->2->1->0->repeat


1->0->1->2->3->4->5->-1->repeat
-1->5->4->3->2->1->0->1->repeat


1->0->1->0->2->3->4->5->-1->repeat
-1->5->4->3->2->0->1->0->1->repeat


0->1->0->2->3->4->5->-1->repeat
-1->5->4->3->2->0->1->0->repeat


0->1->0->3->4->5->-1->repeat
-1->5->4->3->0->1->0->repeat

[Done] exited with code=0 in 0.864 seconds
```

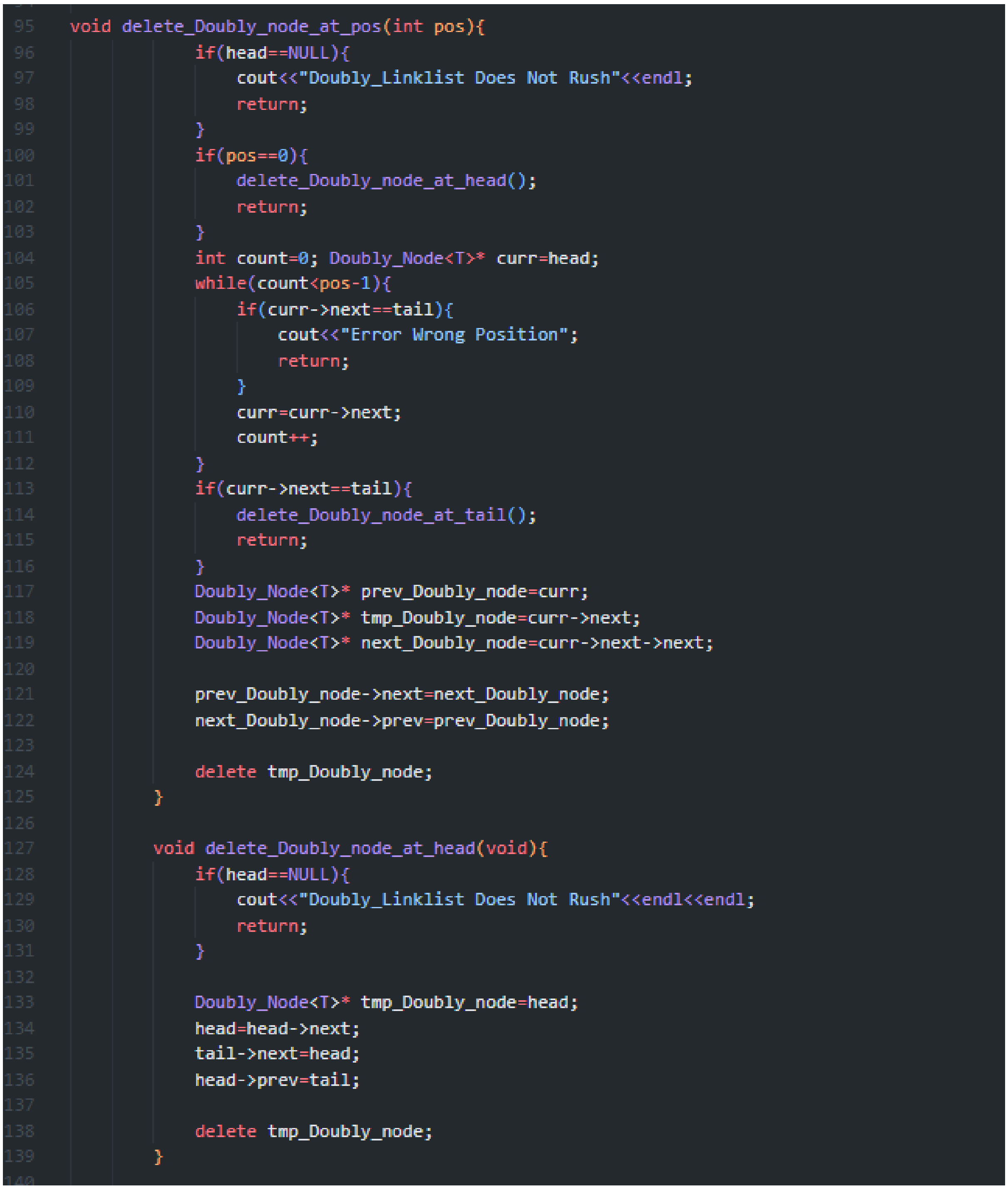
TASK 05:



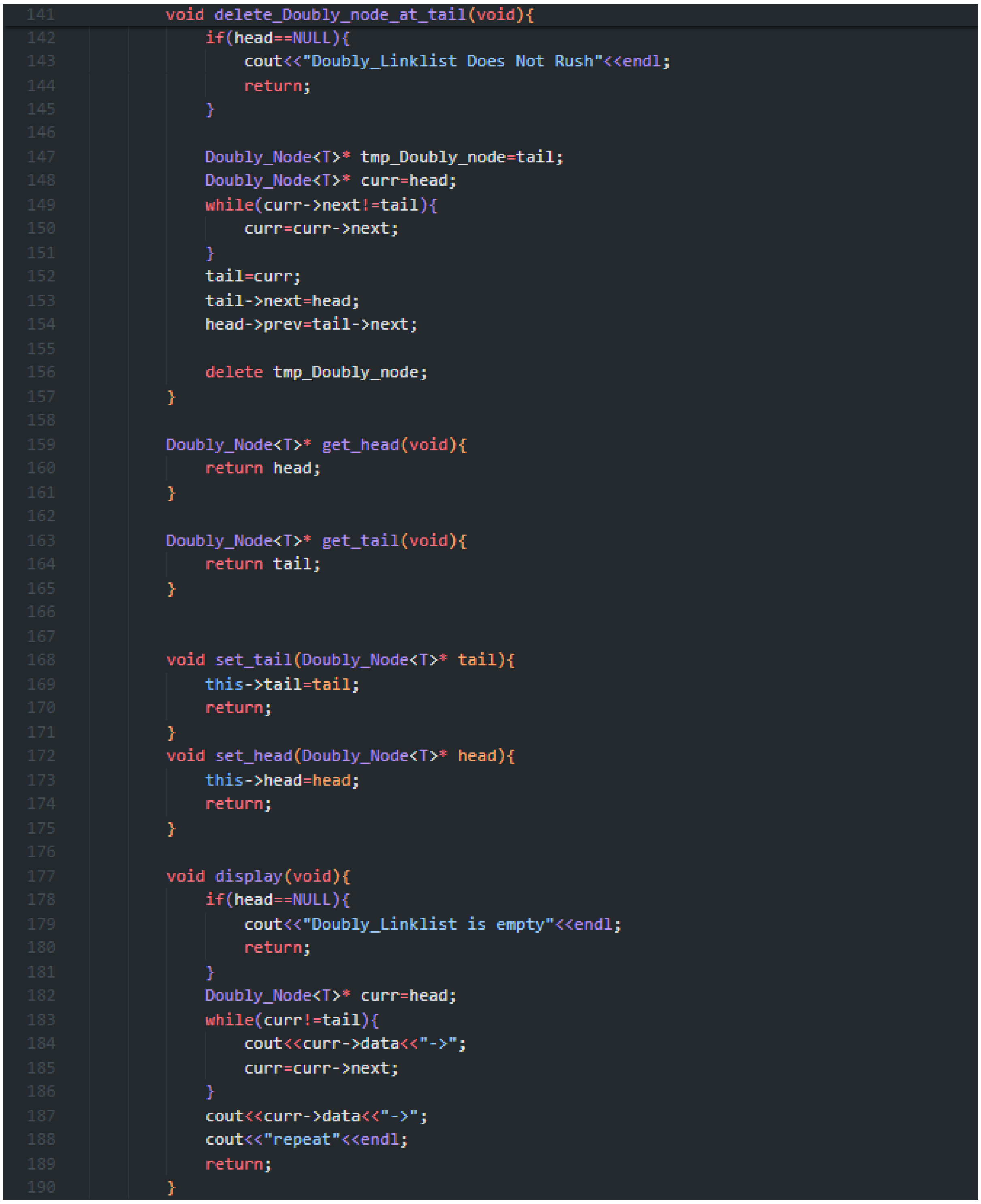
```cpp
#include <iostream>

using namespace std;

template <typename T>
class Doubly_Node{
    public:
        T data;
        Doubly_Node<T>* next=NULL;
        Doubly_Node<T>* prev=NULL;
        Doubly_Node(int value=0):data(value){};
};

template <typename T>
class Singly_Node{
    public:
        T data;
        Singly_Node<T>* next=NULL;
        Singly_Node(int value=0):data(value){};
};


template <typename T>
class Doubly_linklist{
    private:
        Doubly_Node<T>* head;
        Doubly_Node<T>* tail;
    public:
        Doubly_linklist() : head(NULL), tail(head){}
        Doubly_linklist(int value) : head(new Doubly_Node<T>(value)), tail(head){}

        void add_Doubly_node_at_tail(T value){
            Doubly_Node<T>* new_Doubly_node=new Doubly_Node<T>(value);
            if(head==NULL){
                head=new_Doubly_node;
                tail=head;
                return;
            }
            tail->next=new_Doubly_node;
            new_Doubly_node->prev=tail;
            tail=new_Doubly_node;
            tail->next=head;
            head->prev=tail;
        }

```



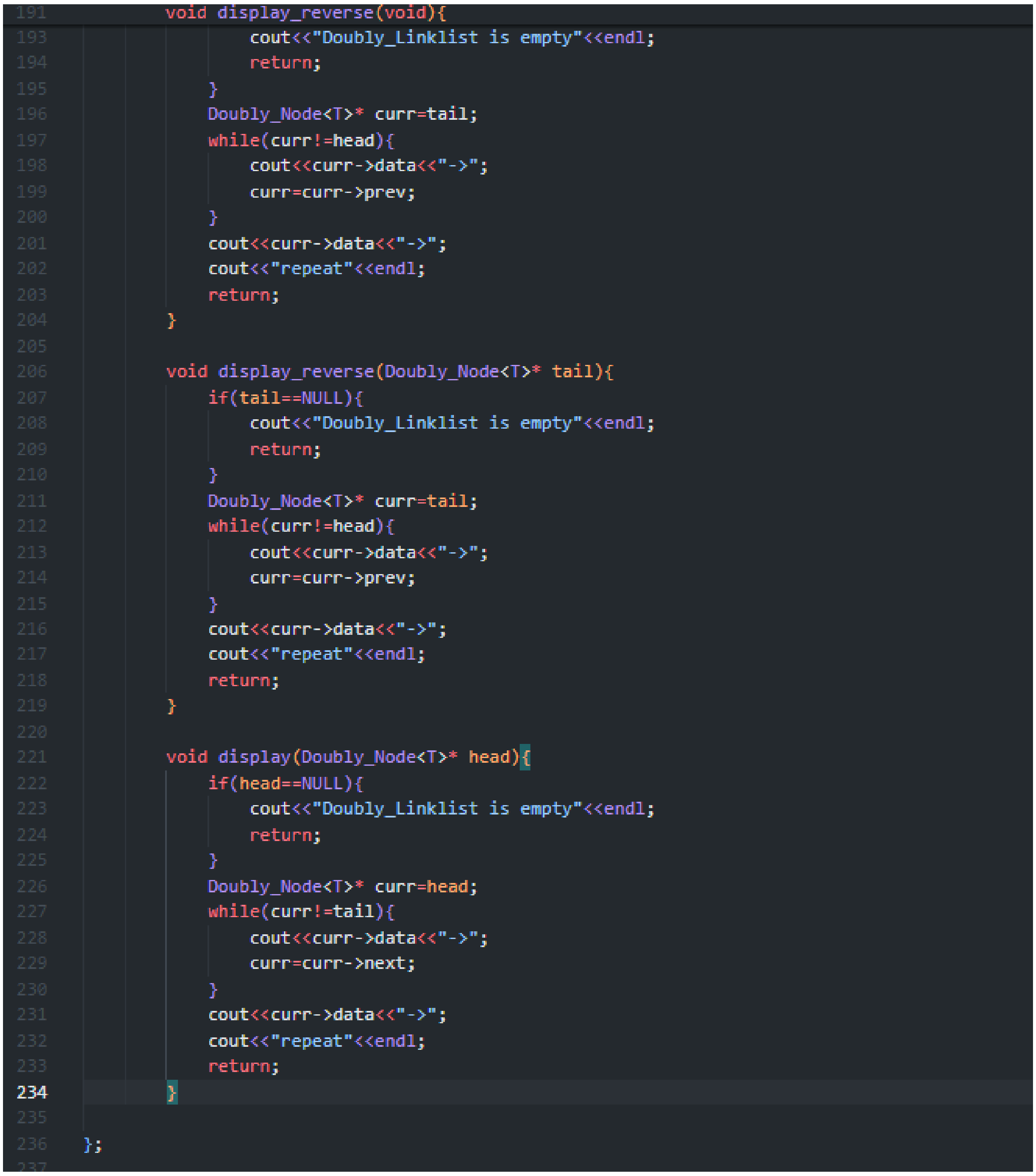
```
        void add_Doubly_node_at_pos(T value,int pos){
            Doubly_Node<T>* new_Doubly_node=new Doubly_Node<T>(value);
            if(head==NULL){
                head=new_Doubly_node;
                tail=head;
                return;
            }
            if(pos==0){
                add_Doubly_node_at_head(value);
                return;
            }
            int count=0; Doubly_Node<T>* curr=head;
            while(count<pos-1 && curr!=tail){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr==tail){
                add_Doubly_node_at_tail(value);
                return;
            }

            Doubly_Node<T>* prev_Doubly_node=curr;
            Doubly_Node<T>* next_Doubly_node=curr->next;

            prev_Doubly_node->next=new_Doubly_node;
            new_Doubly_node->prev=prev_Doubly_node->next;

            new_Doubly_node->next=next_Doubly_node;
            next_Doubly_node->prev=new_Doubly_node;

        }

        void add_Doubly_node_at_head(T value){
            Doubly_Node<T>* new_Doubly_node=new Doubly_Node<T>(value);

            new_Doubly_node->next=head;
            head->prev=new_Doubly_node;

            head=new_Doubly_node;

            tail->next=head;
            head->prev=tail;
            return;
        }
```



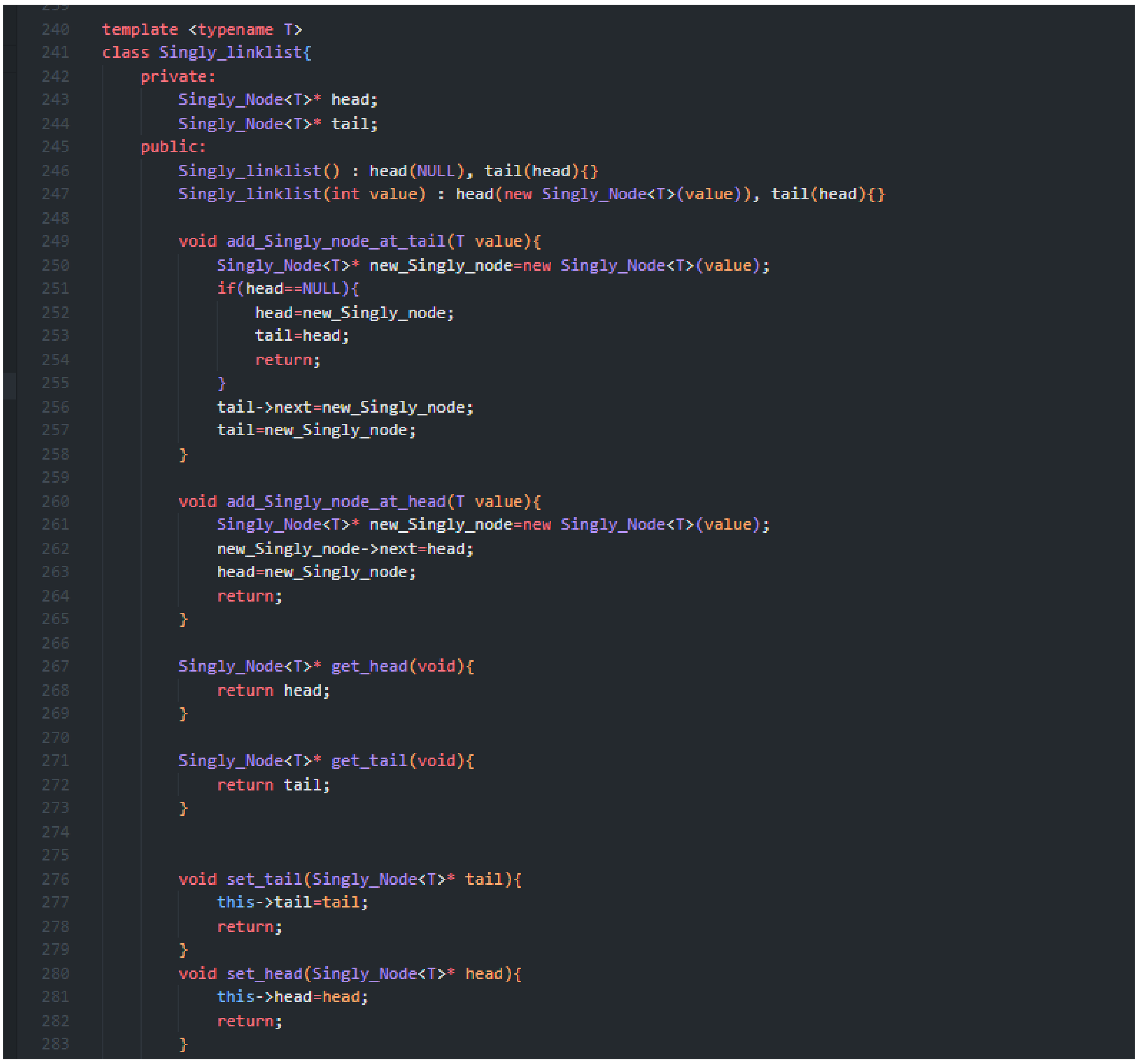
```cpp
void delete_Doubly_node_at_pos(int pos){
            if(head==NULL){
                cout<<"Doubly_Linklist Does Not Rush"<<endl;
                return;
            }
            if(pos==0){
                delete_Doubly_node_at_head();
                return;
            }
            int count=0; Doubly_Node<T>* curr=head;
            while(count<pos-1){
                if(curr->next==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr->next==tail){
                delete_Doubly_node_at_tail();
                return;
            }
            Doubly_Node<T>* prev_Doubly_node=curr;
            Doubly_Node<T>* tmp_Doubly_node=curr->next;
            Doubly_Node<T>* next_Doubly_node=curr->next->next;

            prev_Doubly_node->next=next_Doubly_node;
            next_Doubly_node->prev=prev_Doubly_node;

            delete tmp_Doubly_node;
        }

        void delete_Doubly_node_at_head(void){
            if(head==NULL){
                cout<<"Doubly_Linklist Does Not Rush"<<endl<<endl;
                return;
            }

            Doubly_Node<T>* tmp_Doubly_node=head;
            head=head->next;
            tail->next=head;
            head->prev=tail;

            delete tmp_Doubly_node;
        }
```



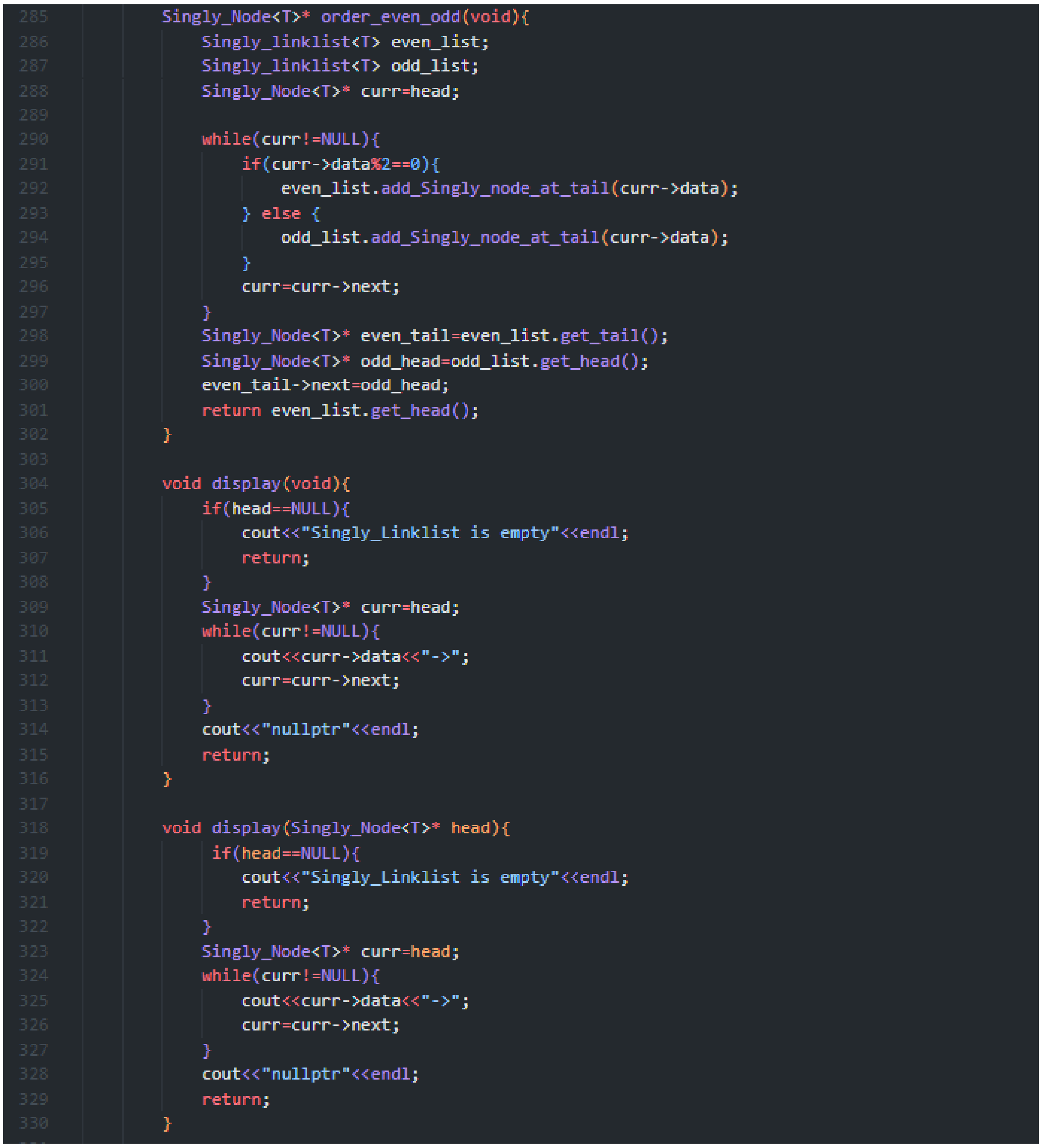
```cpp
        void delete_Doubly_node_at_tail(void){
            if(head==NULL){
                cout<<"Doubly_Linklist Does Not Rush"<<endl;
                return;
            }

            Doubly_Node<T>* tmp_Doubly_node=tail;
            Doubly_Node<T>* curr=head;
            while(curr->next!=tail){
                curr=curr->next;
            }
            tail=curr;
            tail->next=head;
            head->prev=tail->next;

            delete tmp_Doubly_node;
        }

        Doubly_Node<T>* get_head(void){
            return head;
        }

        Doubly_Node<T>* get_tail(void){
            return tail;
        }


        void set_tail(Doubly_Node<T>* tail){
            this->tail=tail;
            return;
        }
        void set_head(Doubly_Node<T>* head){
            this->head=head;
            return;
        }

        void display(void){
            if(head==NULL){
                cout<<"Doubly_Linklist is empty"<<endl;
                return;
            }
            Doubly_Node<T>* curr=head;
            while(curr!=tail){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }
```



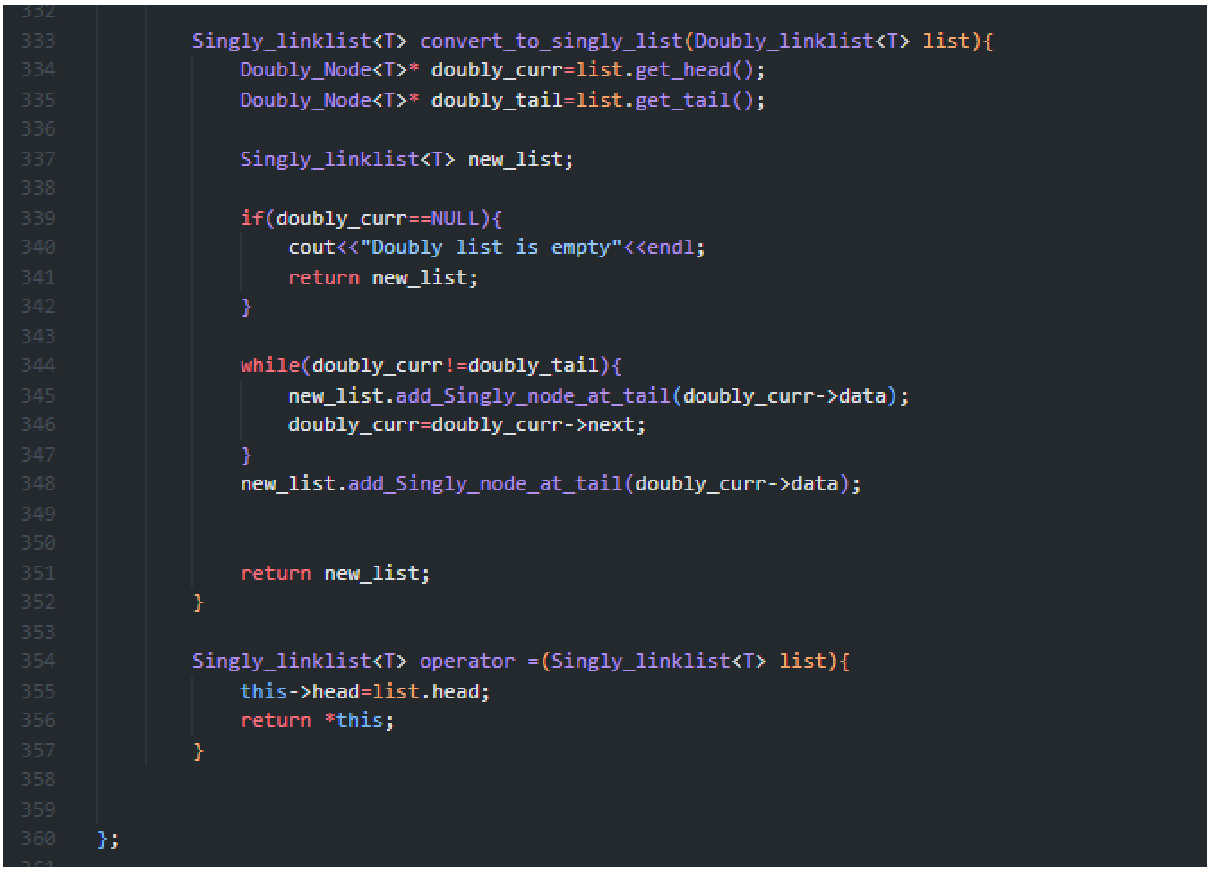
```
        void display_reverse(void){
                cout<<"Doubly_Linklist is empty"<<endl;
                return;
            }
            Doubly_Node<T>* curr=tail;
            while(curr!=head){
                cout<<curr->data<<"->";
                curr=curr->prev;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }

        void display_reverse(Doubly_Node<T>* tail){
            if(tail==NULL){
                cout<<"Doubly_Linklist is empty"<<endl;
                return;
            }
            Doubly_Node<T>* curr=tail;
            while(curr!=head){
                cout<<curr->data<<"->";
                curr=curr->prev;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }

        void display(Doubly_Node<T>* head){
            if(head==NULL){
                cout<<"Doubly_Linklist is empty"<<endl;
                return;
            }
            Doubly_Node<T>* curr=head;
            while(curr!=tail){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }

};
```



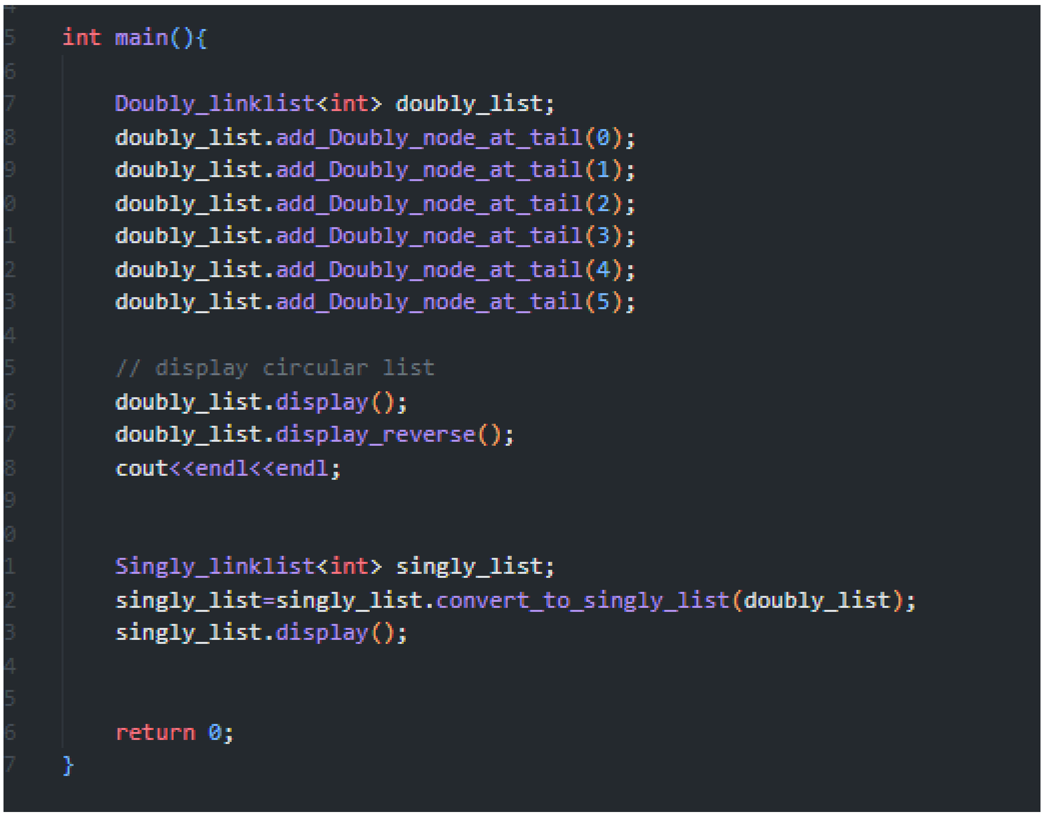
```cpp
template <typename T>
class Singly_linklist{
    private:
        Singly_Node<T>* head;
        Singly_Node<T>* tail;
    public:
        Singly_linklist() : head(NULL), tail(head){}
        Singly_linklist(int value) : head(new Singly_Node<T>(value)), tail(head){}

        void add_Singly_node_at_tail(T value){
            Singly_Node<T>* new_Singly_node=new Singly_Node<T>(value);
            if(head==NULL){
                head=new_Singly_node;
                tail=head;
                return;
            }
            tail->next=new_Singly_node;
            tail=new_Singly_node;
        }

        void add_Singly_node_at_head(T value){
            Singly_Node<T>* new_Singly_node=new Singly_Node<T>(value);
            new_Singly_node->next=head;
            head=new_Singly_node;
            return;
        }

        Singly_Node<T>* get_head(void){
            return head;
        }

        Singly_Node<T>* get_tail(void){
            return tail;
        }


        void set_tail(Singly_Node<T>* tail){
            this->tail=tail;
            return;
        }
        void set_head(Singly_Node<T>* head){
            this->head=head;
            return;
        }
```



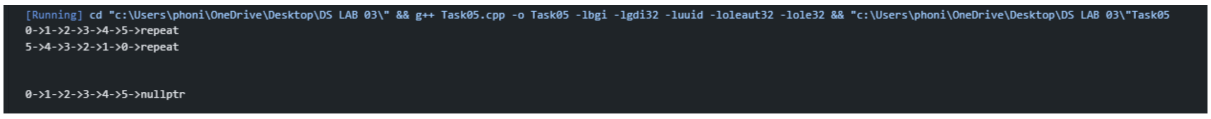
```
        Singly_Node<T>* order_even_odd(void){
            Singly_linklist<T> even_list;
            Singly_linklist<T> odd_list;
            Singly_Node<T>* curr=head;

            while(curr!=NULL){
                if(curr->data%2==0){
                    even_list.add_Singly_node_at_tail(curr->data);
                } else {
                    odd_list.add_Singly_node_at_tail(curr->data);
                }
                curr=curr->next;
            }
            Singly_Node<T>* even_tail=even_list.get_tail();
            Singly_Node<T>* odd_head=odd_list.get_head();
            even_tail->next=odd_head;
            return even_list.get_head();
        }

        void display(void){
            if(head==NULL){
                cout<<"Singly_Linklist is empty"<<endl;
                return;
            }
            Singly_Node<T>* curr=head;
            while(curr!=NULL){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<"nullptr"<<endl;
            return;
        }

        void display(Singly_Node<T>* head){
             if(head==NULL){
                cout<<"Singly_Linklist is empty"<<endl;
                return;
            }
            Singly_Node<T>* curr=head;
            while(curr!=NULL){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<"nullptr"<<endl;
            return;
        }
```



```cpp
        Singly_linklist<T> convert_to_singly_list(Doubly_linklist<T> list){
            Doubly_Node<T>* doubly_curr=list.get_head();
            Doubly_Node<T>* doubly_tail=list.get_tail();

            Singly_linklist<T> new_list;

            if(doubly_curr==NULL){
                cout<<"Doubly list is empty"<<endl;
                return new_list;
            }

            while(doubly_curr!=doubly_tail){
                new_list.add_Singly_node_at_tail(doubly_curr->data);
                doubly_curr=doubly_curr->next;
            }
            new_list.add_Singly_node_at_tail(doubly_curr->data);


            return new_list;
        }

        Singly_linklist<T> operator =(Singly_linklist<T> list){
            this->head=list.head;
            return *this;
        }


};
```



```cpp
int main(){

    Doubly_linklist<int> doubly_list;
    doubly_list.add_Doubly_node_at_tail(0);
    doubly_list.add_Doubly_node_at_tail(1);
    doubly_list.add_Doubly_node_at_tail(2);
    doubly_list.add_Doubly_node_at_tail(3);
    doubly_list.add_Doubly_node_at_tail(4);
    doubly_list.add_Doubly_node_at_tail(5);

    // display circular list
    doubly_list.display();
    doubly_list.display_reverse();
    cout<<endl<<endl;


    Singly_linklist<int> singly_list;
    singly_list=singly_list.convert_to_singly_list(doubly_list);
    singly_list.display();


    return 0;
}
```



```
[Running] cd "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\" && g++ Task05.cpp -o Task05 -lbgi -lgdi32 -luuid -loleaut32 -lole32 && "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\"Task05
0->1->2->3->4->5->repeat
5->4->3->2->1->0->repeat


0->1->2->3->4->5->nullptr
```

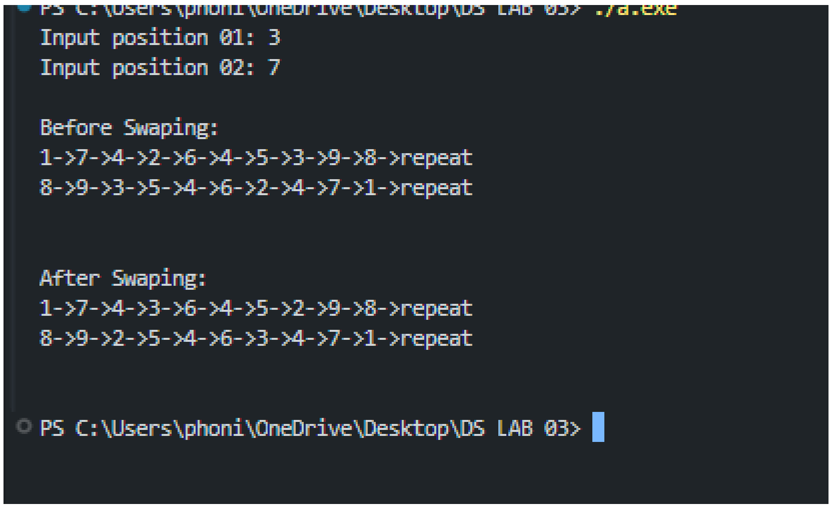
TASK 06:

```cpp
#include <iostream>

using namespace std;

template <typename T>
class Node{
    public:
        T data;
        Node<T>* next=NULL;
        Node<T>* prev=NULL;
        Node(int value=0):data(value){};
};


template <typename T>
class linklist{
    private:
        Node<T>* head;
        Node<T>* tail;
    public:
        linklist() : head(NULL), tail(head){}
        linklist(int value) : head(new Node<T>(value)), tail(head){}

        void add_node_at_tail(T value){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            tail->next=new_node;
            new_node->prev=tail;
            tail=new_node;
            tail->next=head;
            head->prev=tail;
        }

        void add_node_at_pos(T value,int pos){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            if(pos==0){
                add_node_at_head(value);
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1 && curr!=tail){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr==tail){
                add_node_at_tail(value);
                return;
            }

            Node<T>* prev_node=curr;
            Node<T>* next_node=curr->next;

            prev_node->next=new_node;
            new_node->prev=prev_node->next;

            new_node->next=next_node;
            next_node->prev=new_node;

        }

        void add_node_at_head(T value){
            Node<T>* new_node=new Node<T>(value);

            new_node->next=head;
            head->prev=new_node;

            head=new_node;

            tail->next=head;
            head->prev=tail;
            return;
        }

        Node<T>* get_head(void){
            return head;
        }

        Node<T>* get_tail(void){
            return tail;
```

```cpp
        }


        void set_tail(Node<T>* tail){
            this->tail=tail;
            return;
        }
        void set_head(Node<T>* head){
            this->head=head;
            return;
        }

        void display(void){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=tail){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }
        void display_reverse(void){
            if(tail==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=tail;
            while(curr!=head){
                cout<<curr->data<<"->";
                curr=curr->prev;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }

        void display_reverse(Node<T>* tail){
            if(tail==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=tail;
            while(curr!=head){
                cout<<curr->data<<"->";
                curr=curr->prev;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }

        void display(Node<T>* head){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr!=tail){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"repeat"<<endl;
            return;
        }


    void swap_nodes(int pos01,int pos02){
        if(head==tail){
            cout<<"List Too Short"<<endl;
            return;
        }


        int cnt01=0;
        int cnt02=0;
        Node<T>* node01=head;
        Node<T>* node02=head;

        while(cnt01<pos01 || cnt02<pos02){
            if(cnt01<pos01){
                node01=node01->next;
                cnt01++;
            }
            if(cnt02<pos02){
                node02=node02->next;
                cnt02++;
            }
        }

        if (node01==head || node01==tail || node02==head || node02==tail) {
            if (node01==head && node02!=tail) {
                swap_head(pos02);
            } else if (node02==head && node01!=tail) {
                swap_head(pos01);
            } else if (node01==tail && node02!=head) {
                swap_tail(pos02);
            } else if (node02==tail && node01!=head) {
                swap_tail(pos01);
            } else if ((node01==head && node02==tail) || (node01==tail &&
node02==head)) {
                swap_head_tail();
            }
            return;
        }

        if(node01->next==node02 && node02->prev==node01){
            swap_adjacent(node01,node02);
        }

        if(node02->next==node01 && node01->prev==node02){
            swap_adjacent(node02,node01);
        }

        Node<T>* prev01=node01->prev;
        Node<T>* next01=node01->next;
        Node<T>* prev02=node02->prev;
        Node<T>* next02=node02->next;

        prev01->next=node02;
        next01->prev=node02;
        node02->next=next01;
        node02->prev=prev01;

        prev02->next=node01;
        next02->prev=node01;
        node01->next=next02;
        node01->prev=prev02;

    }

    void swap_head(int pos) {
        if (head==NULL) {
            cout<<"List is empty"<<endl;
            return;
        }

        if (pos==0) {
            cout<<"Position is the head, no swap needed."<<endl;
            return;
        }

        Node<T>* node01=head;
        Node<T>* node02=head;
        int cnt=0;

        while (cnt<pos) {
            node02=node02->next;
            cnt++;
        }

        if (node02==tail) {
            swap_head_tail();
            return;
        }

        Node<T>* prev01=node01->prev;
        Node<T>* prev02=node02->prev;
        Node<T>* next01=node01->next;
        Node<T>* next02=node02->next;


        prev01->next=node02;
        next01->prev=node02;
        node02->next=next01;
        node02->prev=prev01;

        prev02->next=node01;
        next02->prev=node01;
        node01->next=next02;
        node01->prev=prev02;

        head=node02;
        tail->next=head;
        head->prev=tail;
    }

    void swap_tail(int pos) {
        if (head==NULL) {
            cout<<"List is empty"<<endl;
            return;
        }

        if (pos==0) {
            swap_head_tail();
            return;
        }
        Node<T>* node01=head;
        Node<T>* node02=tail;
        int cnt=0;

        while (cnt<pos) {
            node01=node01->next;
            cnt++;
        }


        if (node01==head) {
            swap_head_tail();
            return;
        }

        if (node01==tail) {
            cout<<"Node is already the head."<<endl;
            return;
        }

        Node<T>* prev01=node01->prev;
        Node<T>* next01=node01->next;
        Node<T>* prev02=tail->prev;
        Node<T>* next02=tail->next;


        tail->next=next01;
        tail->prev=prev01;
        prev01->next=tail;
        next01->prev=tail;
        tail=node01;


        prev02->next=node01;
        next02->prev=node01;
        node01->next=next02;
        node01->prev=prev02;

    }

    void swap_head_tail() {
        if (head==NULL || head==tail) {
            cout<<"List is too short to swap head and tail."<<endl;
            return;
        }

        Node<T>* temp_head=head;
        Node<T>* temp_tail=tail;
```

```
        Node<T>* prev_head=head->prev;
        Node<T>* next_head=head->next;
        Node<T>* prev_tail=tail->prev;
        Node<T>* next_tail=tail->next;

        head=temp_tail;
        tail=temp_head;


        head->next=next_head;
        head->prev=prev_head;
        next_head->prev=head;
        head->prev=tail;

        tail->next=next_tail;
        tail->prev=prev_tail;
        prev_tail->next=tail;
        tail->next=head;
    }

    void swap_adjacent(Node<T>* node1, Node<T>* node2) {
        if(node1==NULL || node2==NULL) {
            cout<<"Invalid nodes."<<endl;
            return;
        }

        if((node1==tail && node2==head) || (node2==tail && node1==head)) {
            swap_head_tail();
            return;
        }


        Node<T>* prev1=node1->prev;
        Node<T>* next2=node2->next;

        prev1->next=node2;
        next2->prev=node1;

        node2->prev=prev1;
        node2->next=node1;

        node1->prev=node2;
        node1->next=next2;

        if (node1==head) {
            head=node2;
        } else if (node2==tail) {
            tail=node1;
        }
```

```cpp
    }

};

int main(){

    linklist<int> list;
    list.add_node_at_tail(1);
    list.add_node_at_tail(7);
    list.add_node_at_tail(4);
    list.add_node_at_tail(2);
    list.add_node_at_tail(6);
    list.add_node_at_tail(4);
    list.add_node_at_tail(5);
    list.add_node_at_tail(3);
    list.add_node_at_tail(9);
    list.add_node_at_tail(8);

    int pos01=0,pos02=0;
    cout<<"Input position 01: ";
    cin>>pos01;
    cout<<"Input position 02: ";
    cin>>pos02;

    cout<<endl<<"Before Swaping: "<<endl;
    list.display();
    list.display_reverse();
    cout<<endl;
    list.swap_nodes(pos01,pos02);
    cout<<endl<<"After Swaping: "<<endl;
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    return 0;
}
```

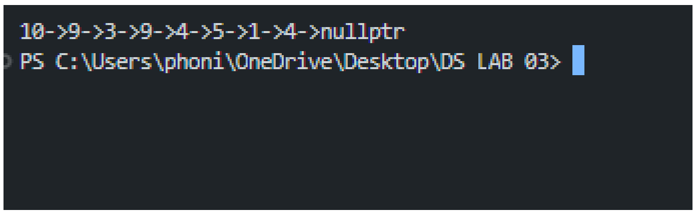


```
PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> ./a.exe
Input position 01: 3
Input position 02: 7

Before Swaping:
1->7->4->2->6->4->5->3->9->8->repeat
8->9->3->5->4->6->2->4->7->1->repeat


After Swaping:
1->7->4->3->6->4->5->2->9->8->repeat
8->9->2->5->4->6->3->4->7->1->repeat

PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> 
```

TASK 07:

```
#include <iostream>

using namespace std;

template <typename T>
class Node{
    public:
        T data;
        Node<T>* next=nullptr;
        Node<T>* prev=nullptr;
        Node(int value=0):data(value){};
};


template <typename T>
class linklist{
    private:
        Node<T>* head;
        Node<T>* tail;
    public:
        linklist() : head(nullptr), tail(head){}
        linklist(int value) : head(new Node<T>(value)), tail(head){}

        void add_node_at_pos(T value,int pos){
            Node<T>* new_node=new Node<T>(value);
            if(head==nullptr){
                head=new_node;
                tail=head;
                return;
            }
            if(pos==0){
                add_node_at_head(value);
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1 && curr!=tail){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr==tail){
                add_node_at_tail(value);
                return;
            }

            Node<T>* prev_node=curr;
            Node<T>* next_node=curr->next;

            prev_node->next=new_node;
            new_node->prev=prev_node->next;

            new_node->next=next_node;
            next_node->prev=new_node;

        }

        void add_node_at_head(T value){
            Node<T>* new_node=new Node<T>(value);

            new_node->next=head;
            head->prev=new_node;

            head=new_node;

            tail->next=head;
            head->prev=nullptr;
            return;
        }

        Node<T>* get_head(void){
            return head;
        }

        Node<T>* get_tail(void){
            return tail;
        }
```

```cpp
    void set_tail(Node<T>* tail){
        this->tail=tail;
        return;
    }
    void set_head(Node<T>* head){
        this->head=head;
        return;
    }

    void display(void){
        if(head==nullptr){
            cout<<"Linklist is empty"<<endl;
            return;
        }
        Node<T>* curr=head;
        while(curr){
            cout<<curr->data<<"->";
            curr=curr->next;
        }
        cout<<"nullptr"<<endl;
        return;
    }
    void display_reverse(void){
        if(tail==nullptr){
            cout<<"Linklist is empty"<<endl;
            return;
        }
        Node<T>* curr=tail;
        while(curr){
            cout<<curr->data<<"->";
            curr=curr->prev;
        }
        cout<<"nullptr"<<endl;
        return;
    }

    void display_reverse(Node<T>* tail){
        if(tail==nullptr){
            cout<<"Linklist is empty"<<endl;
            return;
        }
        Node<T>* curr=tail;
        while(curr){
            cout<<curr->data<<"->";
            curr=curr->prev;
        }
        cout<<"nullptr"<<endl;
        return;
    }

    void display(Node<T>* head){
        if(head==nullptr){
            cout<<"Linklist is empty"<<endl;
            return;
        }
        Node<T>* curr=head;
        while(curr){
            cout<<curr->data<<"->";
            curr=curr->next;
        }
        cout<<"nullptr"<<endl;
        return;
    }


    linklist<T> operator =(linklist<T> list){
        this->head=list.head;
        this->tail=list.tail;
        return *this;
    }

    void reverse(void){
        Node<T>* prevptr=nullptr;
        Node<T>* curr=head;
        while(curr->next){
            Node<T>* nextptr=curr->next;
            curr->prev=nextptr;
            curr->next=prevptr;
            prevptr=curr;
            curr=nextptr;
        }

        tail=head;

        head=prevptr->prev;
        head->next=prevptr;

        head->prev=nullptr;
        tail->next=nullptr;
    }


    void add_node_at_tail(T value){
        Node<T>* new_node=new Node<T>(value);
        if(head==nullptr){
            head=new_node;
            tail=head;
            return;
        }
        tail->next=new_node;
        new_node->prev=tail;
        tail=new_node;
        tail->next=nullptr;
    }


    void delete_node_at_head(void){
        if(head==nullptr){
            cout<<"Linklist Does Not Rush"<<endl<<endl;
            return;
        }

        Node<T>* tmp_node=head;
        head=head->next;
        tail->next=head;
        head->prev=nullptr;

        delete tmp_node;
    }

    void delete_node_at_tail(void){
        if(head==nullptr){
            cout<<"Linklist Does Not Rush"<<endl;
            return;
        }

        Node<T>* tmp_node=tail;
        tail=tail->prev;
        tail->next=nullptr;

        delete tmp_node;
    }

    void delete_node(Node<T>* &node){
        if(head==nullptr){
            cout<<"Linklist Does Not Rush"<<endl;
            return;
        }
        if(node==head){
            delete_node_at_head();
            return;
        }
        if(node==tail){
            delete_node_at_tail();
            return;
        }

        Node<T>* tmp=node;
        Node<T>* prev=node->prev;
        Node<T>* next=node->next;

        prev->next=next;
        next->prev=prev;

        node=node->next;
        delete tmp;

    }

    void aternate_list(void){
        if(head==nullptr){
            cout<<"Linklist Does Not Rush"<<endl;
            return;
        }

        Node<T>* curr=head;
        int index=0;
        while(curr!=tail){
            Node<T>* next=curr->next;
            if(index++%2==1){
                add_node_at_tail(curr->data);
                delete_node(curr);
            }
            curr=next;
        }
    }

    void get_user_input(void){
        bool stop=false; T value; char choice;
        system("cls");

        while(!stop){
            cout<<"Enter Node Value: ";
            cin>>value;
            add_node_at_tail(value);
            cout<<"Continue List: Y/N: ";
            cin>>choice;
            if(choice=='n' || choice=='N'){
                stop=true;
            }
            system("cls");
        }
    }

};

int main(){
linklist<int> list;

list.get_user_input();

list.aternate_list();
list.display();

return 0;
}
```



```
10->9->3->9->4->5->1->4->nullptr
PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03>
```