**TASK 01:**

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**TASK 02:**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

A screenshot of a computer program

Description automatically generated

**A screenshot of a computer program

Description automatically generated**

**TASK 03:**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer code

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A computer screen shot of a program

Description automatically generated**

**TASK 04:**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screenshot of a computer program

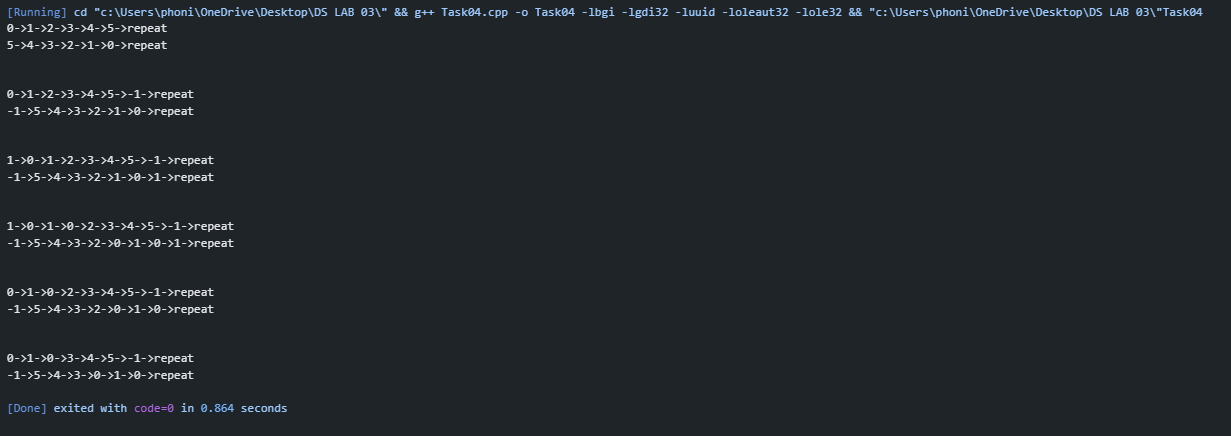
Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

****

**TASK 05:**

**A screen shot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

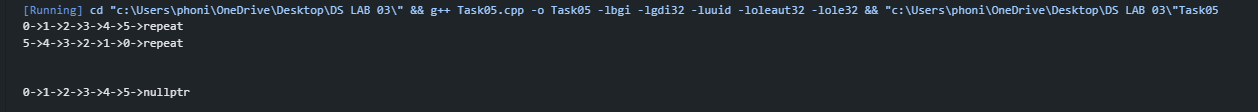
Description automatically generated**

**A computer screen shot of a program code

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

****

**TASK 06:**

#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;

        }

        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;

        }

    }

};

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;

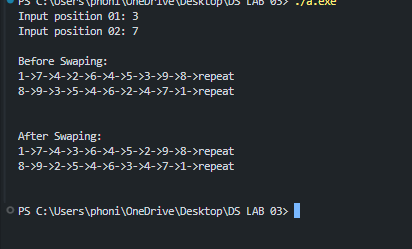
    list.display();

    list.display\_reverse();

    cout<<endl<<endl;

    return 0;

}

****

**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;

        }

        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;

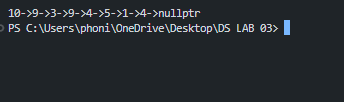
    list.get\_user\_input();

    list.aternate\_list();

    list.display();

    return 0;

}

****