**СТРУКТУРА**

**звіту з лабораторної роботи**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО”**

**ЗВІТ**

**з лабораторної роботи №5**

**з навчальної дисципліни “Програмування складних алгоритмів”**

**Тема: Одно- та Дво- зв’язний список**

**Варіант №1**

**Виконав студент групи ТР–15**

Руденко Владислав Ігорович

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

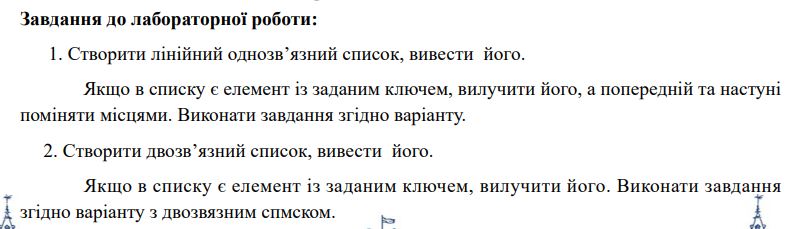
**Перевірив доцент кафедри**

Андрій ОНИСЬКО\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ року

**Київ 2022**

**І. Завдання + Мета:**

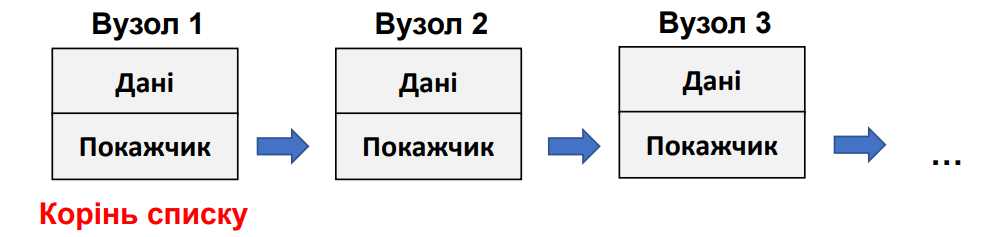
**Завдання:**



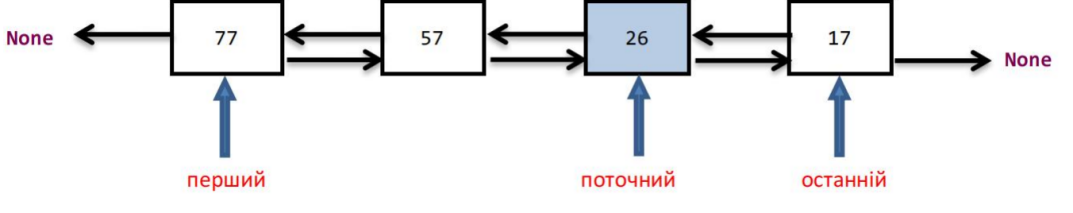
Додаток В- 1 (Завдання 2)

**Теоретична частина.**Зв’язний список є найпростішим типом даних динамічної структури, що складається з елементів (вузлів). Кожен вузол включає в себе в класичному варіанті два поля:

• дані (в якості даних може виступати змінна, об'єкт класу або структури і т. д.)

• покажчик на наступний вузол в списку  


*Лiнiйний список –* це набiр однотипних компонентiв, якi послiдовно пов’язанi мiж собою за допомогою покажчикiв. Кожен компонент списку може складатися iз кiлькох iнформацiйних полiв та покажчика на наступний елемент. Основні операції: ü пошук елемента з заданими властивостями; ü визначення i-того елемента; ü додавання елемента до, або після вказаного; ü вилучення певного елемента зі списку; ü впорядкування *Двобічно зв’язаний (двозв’язний) список* – динамічна структура даних, що складається з елементів одного типу, зв'язаних між собою у строго визначеному порядку. При цьому визначено перший та останній елементи у списку, а кожен елемент списку вказує на наступний і попередній елементи у списку. Перший лемент має попереднім елементом по силання на невизначений елемент None. Аналогічно, None буде наступним елементом для останнього елементу списк

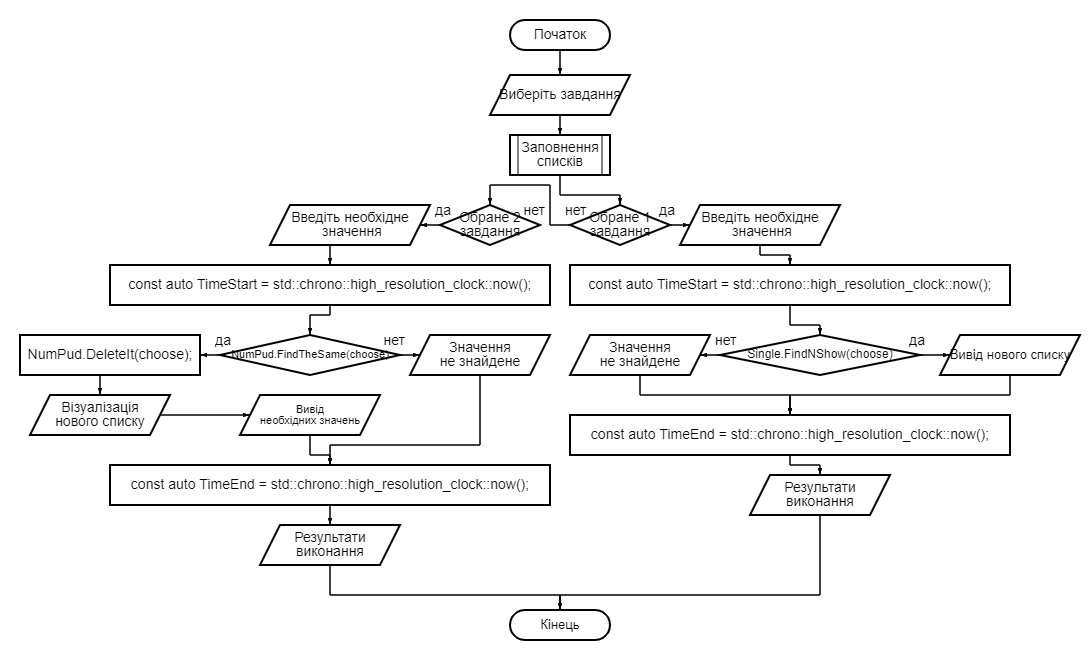


Базовий набір дій над двозв’язними списками:

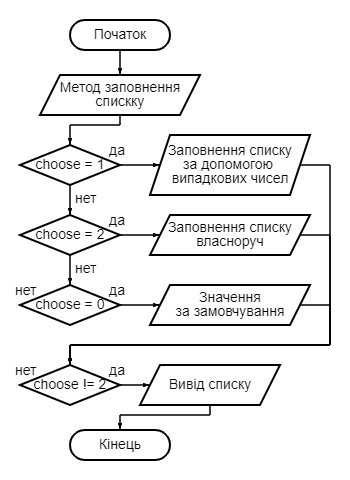
1. Створити список.
2. Операція визначення, чи порожній список. Повертає булеве значення.
3. Зробити поточними перший елемент списку.
4. Зробити поточними останній елемент списку.
5. Перейти до наступного елемента. 6. Перейти до попереднього елемента.
6. Отримати поточний елемент.
7. Вставити новий елемент перед поточним.
8. Вставити новий елемент після поточного.
9. Видалити

**ІІ. Результати виконання лабораторної роботи.**

**Блок Схеми:**

****

*Блок-схема 1 (main.cpp|main)*



*Блок-схема 2 (main.cpp | Заповнення списків)*

**Опис програми (написана на мові Сі++):**

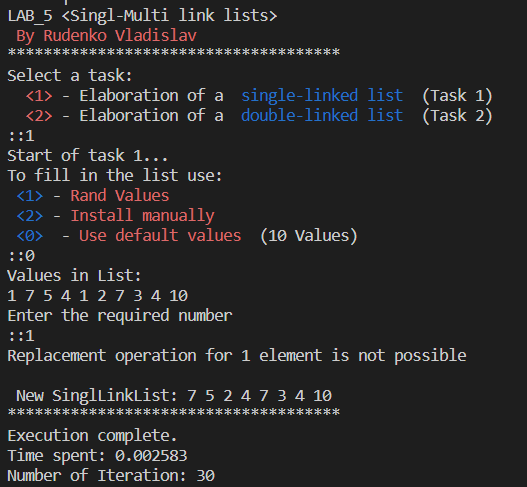
На початку виконання відбувається запит користувача про варіант завдання який буде виконуватись. За для зменшення об’єму коду відбувається заповнення одночасно двох списків (одно- зв’язного та дво-зв’язного), кожен із цих списків має індивідуальний код в окремому виконавчому файлі, які приєднанні до main .  
-При виборі першого варіанту програма запитує шуканий елемент після чого при проході по списку вона запам’ятовує 2 положення (попередне місце, та поточне) при знаходженні содства в елементах запам’ятовується трете положення (наступне місце). Попередне та наступне місце міняють дані місцями, а поточне видаляється, після чого пошук продовжується. Візуалізація видалених та змінених елементів відсутня, адже це суттєво сповільнить код.

-При виборі другого завдання програма також запитує шуканий елемент та шукає його в списку двонаправленим шляхом. Тобто програма проходить від початку до першого шуканого елементу списку, та від кінця до першого знайденого елменту (так програма ставить границі роботи) після чого проходить по всій цій границі та видаляє необхідний елемент. Програма візуалізує усі проведені зміни, так що корисутвач може зрозуміти що відбулося без особих зусиль.

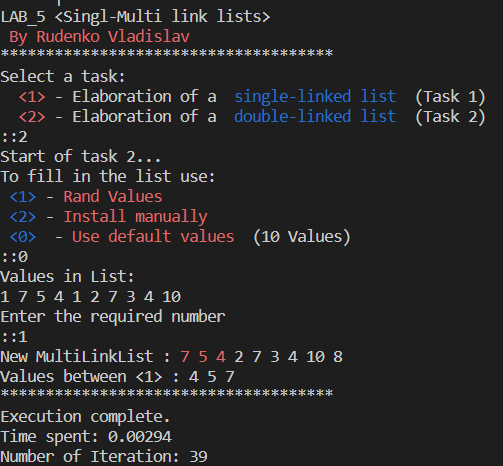
**Результати роботи:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поріврнянн виконання двох завдань | | | | |
| Розмір  списку | Кількість ітерацій  Singl link list | Кількість ітерацій  Multi link list | Час виконання Singl link list  *ms* | Час виконання Multi link list  *ms* |
| **10** | **30** | **0.001846** | **39** | **0.003013** |
| **50** | **209** | **0.006388** | **58** | **0.01897** |
| **100** | **484** | **0.012395** | **230** | **0.035262** |
| **500** | **5759** | **0.058319** | **1099** | **0.197796** |

**Результати роботи у вигляді скріншотів:**



*Результати роботи - 1 (Виконання 1го завдання)*



*Результати роботи - 2 (Виконання завдання 2)*

**Посилання на repl.it:  
<https://replit.com/join/srglqfqraw-hetik>**

**Ш. Висновки.**

В ході виконання лабораторної роботи №5 було ознайомлено та використано основи роботи з одно- та дво- зв’язним списком. В якості завдання розроблено програму, що виконує два завдання, кожне з яких передбачає роботу з одно- або -дво зв’язним списком. Виявлено що за допомогою списку можна бистріше та легше опрацьовувати дані, порівнюючи з масивом. Було виявлено та виправлено деяку кількість помилок.

**Програмний код:  
main.cpp:**

#include <iostream>

#include "ListM.cpp"

#include "ListSingle.cpp"

#include <chrono>

#include <vector>

#define RED     "\x1b[31m"

#define BLUE    "\x1b[34m"

#define PINK  "\033[1;35m"

#define RESET   "\x1b[0m"

using namespace std;

void Collapse(ListM<int>& NumPud,ListS<int>& Single,int choose);

int main()

{

    srand(time(NULL));

    ListM<int> NumPud;

    ListS<int> Single;

    int choose;

    cout << "LAB\_5 <Singl-Multi link lists>\n" RED " By Rudenko Vladislav " RESET " \n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

    cout << "Select a task: \n " RED " <1> " RESET "- Elaboration of a " BLUE " single-linked list " RESET " (Task 1)\n " RED " <2> " RESET "- Elaboration of a " BLUE " double-linked list " RESET " (Task 2) \n::";

    cin >> choose;

    cout << "Start of task "<< choose <<"...\n";

    Collapse(NumPud, Single, choose);

    switch(choose)

    {

        case 1:

        {

        int WasDeleted;

        cout << "\nEnter the required number\n::";

        cin >> choose;

        WasDeleted = choose;

        //////////////////////////////////////////////////////////////

        const auto TimeStart1 = std::chrono::high\_resolution\_clock::now();

        if(Single.FindNShow(choose) == true)

        {

            cout << "\n New SinglLinkList: ";

            for(int i=0;i<Single.GetSize();i++ )

            {

                cout << Single.operator[](i) << " ";

            }

        }

        else

            cout << "Value is not found.";

        const auto TimeEnd1 = std::chrono::high\_resolution\_clock::now();

        ///////////////////////////////////////////////////////////////

        cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nExecution complete." << endl;

        cout << "Time spent: " << chrono::duration<float>(TimeEnd1 - TimeStart1).count() << endl;

        cout << "Number of Iteration: " << Single.GetINTER() << endl;

        }

        break;

        case 2:

        {

        cout << "\nEnter the required number\n::";

        cin >> choose;

        ////////////////////////////////////////////////////////////////

        const auto TimeStart2 = std::chrono::high\_resolution\_clock::now();

        if(NumPud.FindTheSame(choose) == true)

        {

                NumPud.DeleteIt(choose);

            cout << "New MultiLinkList : ";

                for(int i=0;i<NumPud.GetSize();i++ )

                    {

                        if(i>=NumPud.GetFirst() && i<=NumPud.GetLast())

                            cout << RED << NumPud.operator[](i)<< RESET << " ";

                        else

                            cout << NumPud.operator[](i) << " ";

                    }

            cout << NumPud.GetSize();

            cout << "\nValues between <" << choose << "> : ";

            NumPud.FindNShow();

        }

        else

            cout << "Coincidence is not found.\n";

        const auto TimeEnd2 = std::chrono::high\_resolution\_clock::now();

        ///////////////////////////////////////////////////////////////

        cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nExecution complete." << endl;

        cout << "Time spent: " << chrono::duration<float>(TimeEnd2 - TimeStart2).count() << endl;

        cout << "Number of Iteration: " << NumPud.GetINTER() << endl;

        NumPud.clear();

        }

        break;

    default:

    cout << "Incorrect value";

    exit(0);

    break;

}

}

void Collapse(ListM<int>& NumPud,ListS<int>& Single,int EX)

{

    int DValues[10] = {1,7,5,4,1,2,7,3,4,10};

    int Range = 10, choose;

    cout << "To fill in the list use:\n " BLUE "<1>" RESET " - " RED "Rand Values" RESET "\n " BLUE "<2>" RESET " - " RED "Install manually" RESET "\n " BLUE "<0> " RESET " - " RED "Use default values " RESET " (10 Values)\n::";

        cin >> choose;

        switch(choose)

        {

            case 1:

                cout << "Enter the number of values: ";

                cin >> Range;

                for(int i=0;i<Range;i++)

                    {NumPud.push\_back(-10 + rand()%20); Single.push\_back(-10 + rand()%20);}

            break;

            case 2:

                cout << "Enter the number of values: ";

                cin >> Range;

                cout << "Enter " << Range << " values\n:: ";

                for(int i=0;i<Range;i++)

                    {

                        cin >> choose;

                        NumPud.push\_back(choose);

                        Single.push\_back(choose);

                    }

            break;

            case 0:

                for(int i=0;i<10;i++)

                    {NumPud.push\_back(DValues[i]); Single.push\_back(DValues[i]);}

            break;

            default:

            cout << "Error...Unknown choose";

            exit(1);

            break;

        }

        if(choose != 2)

        {

            cout << "Values in List:" << endl;

            for(int i=0;i<Range;i++ )

            {

                if(EX == 2)

                    cout << NumPud.operator[](i) << " ";

                else

                    cout << Single.operator[](i) << " ";

            }

        }

}

**ListM.cpp**

#include <iostream>

    using namespace std;

template<typename T>

class Node

{

    public:

    Node\* pNext;

    Node\* pPerv;

    T Data;

    Node(T Data = T(), Node \*pNext = nullptr, Node \*pPerv = nullptr)

    {

        this->Data = Data;

        this->pNext = pNext;

        this->pPerv = pPerv;

    }

};

template<typename T>

class ListM

{

public:

    ListM();

    ~ListM();

    void push\_back(T Data);

    T& operator[](const int index);

    int GetSize() { return Size;};

    void pop\_front();

    void clear();

    void removeAt(int index);

    int GetINTER(){return INTER;};

    bool FindTheSame(int Value);

    void FindNShow();

    int GetFirst(){return First;};

    int GetLast(){return Last;};

    void DeleteIt(int Value);

private:

    int Size, First, Last, INTER;

    Node<T> \*head;

    Node<T> \*tail;

};

////////////////////////////////////

template<typename T>

void ListM<T>::push\_back(T Data)

{

    if(head == nullptr)

    {

        head = new Node<T>(Data);

        tail = head;

    }

    else

    {

      Node<T> \*current = this->head;

      while(current->pNext != nullptr)

      {

          current = current->pNext;

      }

      current ->pNext = new Node<T>(Data, nullptr, current);

      tail = current->pNext;

    }

    Size++;

}

template<typename T>

T & ListM<T>::operator[](const int index)

{

    int counter = 0;

    Node<T> \*current = this->head;

    while (current != nullptr)

    {

        if (counter == index)

        {

            return current->Data;

        }

        current = current->pNext;

        counter++;

    }

}

////////////////////////////////////

////////////////////////////////////

template<typename T>

void ListM<T>::pop\_front()

{

    Node<T> \*temp = head;

    if(temp->pNext !=nullptr)

    {

        head = head->pNext;

        head->pPerv=nullptr;

    }

    delete temp;

    Size--;

}

template<typename T>

void ListM<T>::clear()

{

    while (Size)

    {

        pop\_front();

    }

}

////////////////////////////////////

////////////////////////////////////

template<typename T>

void ListM<T>::removeAt(int index)

{

    if (index == 0)

    {

        pop\_front();

    }

    else

    {

        Node<T> \*elem = this->head;

        Node<T> \*elem\_prev;

        Node<T> \*elem\_next;

        for (int i = 0; i < index; i++)

        {

            elem = elem->pNext;

        }

        elem\_prev=elem->pPerv;

        elem\_next=elem->pNext;

        elem\_next->pPerv=elem\_prev;

        elem\_prev->pNext=elem\_next;

        delete elem;

        Size--;

    }

}

////////////////////////////////////

////////////////////////////////////

template<typename T>

bool ListM<T>::FindTheSame(int Value)

{

    Node<T> \*current\_head = this->head;

    Node<T> \*current\_tail = this->tail;

    int TheSame=0, lock = 0;

    for(int i=0;i<Size;i++)

    {

        if(TheSame==2)

        {

            return true;

        }

        if(current\_head->Data == Value)

        {

            if(First == 0 && lock == 0)

            {

                First = i;

                lock = 1;

                TheSame++;

                INTER+=2;

            }

            else

                INTER++;

        }

        else if(lock == 0)

        {

            current\_head=current\_head->pNext;

            INTER++;

        }

        if(current\_tail->Data == Value)

        {

            if(Last == 0)

                {

                    Last = Size-i-1;

                    TheSame++;

                    INTER+=2;

                }

            else

                INTER++;

        }

        else if(Last == 0)

        {

            current\_tail=current\_tail->pPerv;

            INTER++;

        }

    }

    return false;

}

template<typename T>

void ListM<T>::FindNShow()

{

    Node<T> \*current\_head = this->head;

    Node<T> \*current\_tail = this->tail;

    int i=0;

    if(Last <= Size/2)

    {

        for(i = 0;i<=Size/2;i++)

        {

            if(i==Last)

            {

                current\_tail=current\_head;

                INTER+=2;

                for(int j=0;j<=Last-First;j++)

                {

                    cout << current\_tail->Data << " ";

                    current\_tail=current\_tail->pPerv;

                    INTER++;

                }

                break;

            }

            else

                {

                    current\_head=current\_head->pNext;

                    INTER++;

                }

        }

    }

    else

    {

        for(i = Size-1;i>=Size/2;i--)

        {

            if(i==Last)

            {

                INTER++;

                for(int j=0;j<=Last-First;j++)

                {

                    cout << current\_tail->Data << " ";

                    current\_tail=current\_tail->pPerv;

                    INTER++;

                }

                break;

            }

            else

                {

                    current\_tail=current\_tail->pPerv;

                    INTER++;

                }

        }

    }

}

template<typename T>

void ListM<T>::DeleteIt(int Value)

{

    Node<T> \*current\_tail = this->tail;

    Node<T> \*temp;

    for(int i=Size-1;i>=0;i--)

    {

        if(current\_tail->Data==Value)

        {

            temp=current\_tail->pPerv;

            removeAt(i);

            Last--;

            current\_tail=temp;

            INTER+=4;

        }

        else

            current\_tail=current\_tail->pPerv;

        INTER++;

    }

}

////////////////////////////////////

////////////////////////////////////

template<typename T>

ListM<T>::ListM()

{

    head = nullptr;

    tail = nullptr;

    Size = 0;

    First = 0;

    Last = 0;

    INTER = 0;

}

template<typename T>

ListM<T>::~ListM(){clear();}

**ListSingle.cpp:**

#include <fstream>

#include <iostream>

#include "string"

using namespace std;

template<typename T>

class NodeS

{

    public:

    NodeS\* pNext;

    T Data;

    NodeS(T Data = T(), NodeS \*pNext = nullptr)

    {

        this->Data = Data;

        this->pNext = pNext;

    }

};

template<typename T>

class ListS

{

public:

    ListS();

    ~ListS();

    void push\_back(T Data);

    T& operator[](const int index);

    int GetSize() { return Size; }

    void pop\_front();

    void clear();

    void insert(T data, int index);

    void removeAt(int index);

    bool FindNShow(int Value);

    int GetINTER(){return INTER;};

    void ReadFromFile(std::istream &is);

private:

    int Size, INTER;

    NodeS<T> \*head;

    NodeS<T> \*tail;

};

////////////////////////////////////

template<typename T>

void ListS<T>::push\_back(T Data)

{

    if(head == nullptr)

    {

        head = new NodeS<T>(Data);

        tail = head;

    }

    else

    {

      NodeS<T> \*current = this->head;

      while(current->pNext != nullptr)

      {

          current = current->pNext;

      }

      current ->pNext = new NodeS<T>(Data);

      tail = current->pNext;

    }

    Size++;

}

template<typename T>

T & ListS<T>::operator[](const int index)

{

    int counter = 0;

    NodeS<T> \*current = this->head;

    while (current != nullptr)

    {

        if (counter == index)

        {

            return current->Data;

        }

        current = current->pNext;

        counter++;

    }

}

////////////////////////////////////

////////////////////////////////////

template<typename T>

void ListS<T>::pop\_front()

{

    NodeS<T> \*temp = head;

    head = head->pNext;

    delete temp;

    Size--;

}

template<typename T>

void ListS<T>::clear()

{

    while (Size)

    {

        pop\_front();

    }

}

////////////////////////////////////

////////////////////////////////////

template<typename T>

void ListS<T>::insert(T data, int index)

{

    if (index == 0)

    {

        push\_back(data);

    }

    else

    {

        NodeS<T> \*previous = this->head;

        for (int i = 0; i < index - 1; i++)

        {

            previous = previous->pNext;

        }

        NodeS<T> \*newNode = new NodeS<T>(data, previous->pNext);

        previous->pNext = newNode;

    }

Size++;

}

template<typename T>

void ListS<T>::removeAt(int index)

{

    if (index == 0)

    {

        pop\_front();

    INTER++;

    }

    else

    {

        NodeS<T> \*previous = this->head;

        for (int i = 0; i < index - 1; i++)

        {

            previous = previous->pNext;

      INTER++;

        }

        NodeS<T> \*toDelete = previous->pNext;

        previous->pNext = toDelete->pNext;

        delete toDelete;

    INTER+=4;

        Size--;

    }

}

////////////////////////////////////

template<typename T>

bool ListS<T>::FindNShow(int Value)

{

    NodeS<T> \*current = this->head;

    NodeS<T> \*Next;

    NodeS<T> \*Prev  =  this->head;

    T temp;

    int Found = 0, CD=0;

    for(int i=0;i<Size;i++)

    {

        if(current->Data==Value)

        {

            if(i==0 || i==Size-1)

                {

                    cout << "Replacement operation for "<< i+1 <<" element is not possible\n";

                    Prev = Prev->pNext;

                    current = Prev;

                    removeAt(i);

                    INTER+=3;

                    Found++;

                }

            else

            {

                Next = current->pNext;

                removeAt(i);

                temp = Next->Data;

                Next->Data = Prev->Data;

                Prev->Data = temp;

        INTER+=5;

                Found++;

            }

        }

        if(i != 0)

        {

            Prev=Prev->pNext;

            current=Prev->pNext;

      INTER+=2;

        }

        else

    {

      current=current->pNext;

      INTER++;

    }

    }

    if(Found == 0)

        return false;

    else

        return true;

}

////////////////////////////////////

template<typename T>

ListS<T>::ListS()

{

    head = nullptr;

    tail = nullptr;

    Size = 0;

    INTER=0;

}

template<typename T>

ListS<T>::~ListS(){clear();}