# **PHẦN 1: DANH SÁCH LIÊN KẾT**

## **1. Biên dịch đoạn chương trình nêu trên.**

- Element để đại diện cho các phần tử trong danh sách liên kết và linkedlist để quản lý danh sách liên kết. Chương trình thêm ba phần tử vào danh sách liên kết và sau đó in ra các phần tử này

## **2. Cho biết kết quả in ra màn hình:**

9 10 8

10 8

## **3. Nêu nhận xét ngắn gọn mối liên hệ giữa thứ tự nhập dữ liệu vào với thứ tự in dữ liệu ra màn hình.**

- Dữ liệu được in ra theo đúng thứ tự thêm vào, các phần tử được chèn đuôi và duyệt từ đầu danh sách đến cuối.

## **4. Vẽ hình danh sách liên kết theo dữ liệu được nhập ở câu 2:**

[9] -> [10] -> [8]

## **5. Viết hàm RemoveAll xóa tất cả các phần tử trong DSLK:**

linkedlist.h

void RemoveAll();

linkedlist.cpp

void linkedlist::RemoveAll()

{

    while (this->head != nullptr)

    {

        element\* temp = this->head;

        this->head = this->head->Getpointer();

        delete temp;

    }

    this->tail = nullptr;

    this->nNum = 0;

}

# **Áp dụng – Nâng cao**

## **1. Bổ sung chương trình mẫu cho phép tính tổng giá trị các phần tử trên danh sách liên kết đơn gồm các giá trị nguyên. Gợi ý: tham khảo hàm Travel để viết hàm SumList.**

### **linkedlist.h**

int SumList(); // Bổ sung phương thức tính tổng

### **linkedlist.cpp**

int linkedlist::SumList()

{

    int sum = 0;

    element \*p = this->head;

    while (p != nullptr)

    {

        sum += p->Getdata();

        p = p->Getpointer();

    }

    return sum;

}

### **main.cpp**

cout << "Tong gia tri cac phan tu tren danh sach la: " << list\_->SumList() << endl;

**2. Bổ sung chương trình mẫu cho phép tìm giá trị nguyên lớn nhất trong số các phần tử nguyên trên danh sách liên kết đơn gồm các giá trị nguyên.**

Gợi ý: tham khảo hàm Travel để viết hàm MaxList.

### **linkedlist.h**

int MaxList(); // Bổ sung phương thức tìm giá trị lớn nhất

### **linkedlist.cpp**

int linkedlist::MaxList()

{

int max\_val = INT\_MIN; // Giả sử giá trị lớn nhất ban đầu là INT\_MIN

element\* p = this->head;

while (p != nullptr) {

int current\_val = p->Getdata();

if (current\_val > max\_val) {

max\_val = current\_val;

}

p = p->Getpointer();

}

return max\_val;

}

### **main.cpp**

cout << "Gia tri lon nhat trong danh sach la: " << list\_->MaxList() << endl;

## **3. Bổ sung chương trình mẫu cho phép tính số lượng các phần tử là số nguyên tố của danh sách liên kết đơn gồm các giá trị nguyên. Gợi ý: tham khảo hàm Travel để viết hàm CountPrime.**

### **linkedlist.h**

int CountPrime();

### **linkedlist.cpp**

bool isPrime(int x)

{

    if (x < 2) return false;

    for (int i = 2; i \* i <= x; i++)

        if (x % i == 0) return false;

    return true;

}

int linkedlist::CountPrime()

{

    int count = 0;

    element\* p = this->head;

    while (p != nullptr)

    {

        if (isPrime(p->Getdata()))

            count++;

        p = p->Getpointer();

    }

    return count;

}

### **main.cpp**

    cout << "Number of prime elements: " << list\_->CountPrime() << endl;

## **4. Bổ sung chương trình mẫu cho phép thêm vào cuối danh sách liên kết đơn một giá trị nguyên. Gợi ý: tham khảo hàm InsertFirst để viết hàm InsertTail.**

### **linkedlist.h**

void InsertTail(int val);

### **linkedlist.cpp**

void linkedlist::InsertTail(int val)

{

    element\* e = **new** element(val);

    if (this->head == nullptr)

    {

        this->head = e;

        this->tail = e;

    }

    else

    {

        this->tail->Setpointer(e);

        this->tail = e;

    }

    this->nNum++;

}

### **main.cpp**

list\_->InsertTail(15);

list\_->Travel();

## **5. Bổ sung chương trình mẫu cho phép thêm phần tử vào sau p(tham số truyền vào là 1 con trỏ) danh sách liên kết đơn một giá trị nguyên.**

### **linkedlist.h**

void InsertAfter(element\* p, int val);

### **linkedlist.cpp**

void linkedlist::InsertAfter(element\* p, int val)

{

    if (p == nullptr) return;

    element\* e = **new** element(val);

    e->Setpointer(p->Getpointer());

    p->Setpointer(e);

    if (p == this->tail)

        this->tail = e;

    this->nNum++;

}

### **main.cpp**

element\* secondElement = list\_->Gethead()->Getpointer();

list\_->InsertAfter(secondElement, 20);

list\_->Travel();

## **6. Bổ sung chương trình mẫu cho phép xóa phần tử đầu danh sách liên kết đơn.**

### **linkedlist.h**

bool DeleteFirst();

### **linkedlist.cpp**

bool linkedlist::DeleteFirst()

{

    if (this->head == nullptr) return false;

    else

    {

        element \*p = this->head;

        this->head = this->head->Getpointer();

**delete** p;

        return true;

    }

}

### **main.cpp**

list\_->DeleteFirst();

## **7. Bổ sung chương trình mẫu cho phép xóa phần tử cuối danh sách liên kết đơn.**

### **linkedlist.h**

bool DeleteTail();

### **linkedlist.cpp**

bool linkedlist::DeleteTail()

{

    if (this->head == nullptr) return false;

    if (this->head == this->tail)

    {

**delete** this->head;

        this->head = this->tail = nullptr;

        return true;

    }

    element\* p = this->head;

    while (p->Getpointer() != this->tail)

        p = p->Getpointer();

**delete** this->tail;

    this->tail = p;

    this->tail->Setpointer(nullptr);

    return true;

}

### **main.cpp**

    list\_->DeleteTail();

    list\_->Travel();

## **8. Bổ sung chương trình mẫu cho phép xóa phần tử p(tham số truyền vào là 1 con trỏ) ở vị trí bất kỳ danh sách liên kết đơn.**

### **linkedlist.h**

bool DeleteAt(element\* p);

### **linkedlist.cpp**

bool linkedlist::DeleteAt(element\* p)

{

    if (p == nullptr) return false;

    if (p == this->head) return DeleteFirst();

    if (p == this->tail) return DeleteTail();

    element\* q = this->head;

    while (q->Getpointer() != p)

        q = q->Getpointer();

    q->Setpointer(p->Getpointer());

**delete** p;

    return true;

}

### **main.cpp**

    list\_->DeleteAt(secondElement);

    list\_->Travel();

## **9. Bổ sung chương trình mẫu cho biết số lượng các phần tử trên danh sách liên kết đơn có giá trị trùng với giá trị x được cho trước. Gợi ý: tham khảo thao tác duyệt danh sách liên kết trong hàm Travel.**

### **linkedlist.h**

int CountValue(int x);

### **linkedlist.cpp**

int linkedlist::CountValue(int x)

{

    int count = 0;

    element\* p = this->head;

    while (p != nullptr)

    {

        if (p->Getdata() == x)

            count++;

        p = p->Getpointer();

    }

    return count;

}

### **main.cpp**

int x = 20;

cout << "\n số lượng các phần tử : " << list\_->CountValue(x) << endl;

## **10. Bổ sung chương trình mẫu cho phép tạo một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên trong đó không có cặp phần tử nào mang giá trị giống nhau. Gợi ý: sử dụng hàm InsertFirst hoặc InsertTail có bổ sung thao tác kiểm tra phần tử giống nhau.**

### **linkedlist.h**

bool InsertUnique(int val);

### **linkedlist.cpp**

bool linkedlist::InsertUnique(int val)

{

element\* p = this->head;

while (p != nullptr)

{

if (p->Getdata() == val)

return false;

p = p->Getpointer();

}

InsertTail(val);

return true;

}

### **main.cpp**

linkedlist\* uniqueList = new linkedlist();

uniqueList->InsertUnique(10);

uniqueList->InsertUnique(20);

uniqueList->InsertUnique(10);

uniqueList->InsertUnique(30);

cout << "DSLK don ko phan tu giong nhau: ";

uniqueList->Travel();

delete uniqueList;

## **11. Cho sẵn một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên và một giá trị nguyên x. Hãy tách danh sách liên kết đã cho thành 2 danh sách liên kết: một danh sách gồm các phần tử có giá trị nhỏ hơn giá trị x và một danh sách gồm các phần tử có giá trị lớn hơn giá trị x.**

### **linkedlist.h**

void SplitList(int x, linkedlist\* lessThanX, linkedlist\* greaterThanX);

### **linkedlist.cpp**

void linkedlist::SplitList(int x, linkedlist\* lessThanX, linkedlist\* greaterThanX)

{

element\* p = this->head;

while (p != nullptr)

{

if (p->Getdata() < x)

lessThanX->InsertTail(p->Getdata());

else

greaterThanX->InsertTail(p->Getdata());

p = p->Getpointer();

}

}

### **main.cpp**

linkedlist\* lessThanX = **new** linkedlist();

linkedlist\* greaterThanX = **new** linkedlist();

int x = 15;

list\_->SplitList(x, lessThanX, greaterThanX);

cout << "Elements less than " << x << ": ";

lessThanX->Travel();

cout << "\nElements greater than " << x << ": ";

greaterThanX->Travel();

delete lessThanX;

delete greaterThanX;

# **BÀI TẬP ỨNG DỤNG**

## **1. Đề xuất cấu trúc dữ liệu thích hợp để biểu diễn đa thức (anx n + an-1x n-1+..+ a1x + a0) bằng danh sách liên kết (đơn và kép). Cài đặt các thao tác trên danh sách liên kết đơn biểu diễn đa thức:**

a. In đa thức

b. Tính giá trị đa thức (với giá trị x nhập vào)

c. Cộng hai đa thức

d. Nhân hai đa thức

### **Node.h**

#pragma once

class Node

{

    private:

        int coefficient;

        int exponent;

        Node\* pointer;

    public:

        Node();

        Node(int, int);

        virtual ~Node();

        int GetCoefficient()

        {

            return coefficient;

        }

        void SetCoefficient(int val)

        {

            coefficient = val;

        }

        int GetExponent() const;

        void SetExponent(int val);

        Node\* Getpointer() const;

        void Setpointer(Node\* ptr);

};

### **Node.cpp**

#include "Node.h"

Node::Node() : coefficient(0), exponent(0), pointer(nullptr) {}

Node::Node(int coef, int exp) : coefficient(coef), exponent(exp), pointer(nullptr) {}

Node::~Node() {}

int Node::GetExponent() const {

    return exponent;

}

void Node::SetExponent(int val) {

    exponent = val;

}

Node\* Node::Getpointer() const {

    return pointer;

}

void Node::Setpointer(Node\* ptr) {

    pointer = ptr;

}

### **Polynomial.h**

#pragma once

#include "Node.h"

class Polynomial {

private:

    Node\* head;

    Node\* tail;

    int nNum;

public:

    Polynomial();

    virtual ~Polynomial();

    Node\* GetHead() { return head; }

    void SetHead(Node\* val) { head = val; }

    Node\* GetTail() { return tail; }

    void SetTail(Node\* val) { tail = val; }

    void InsertFirst(Node\* );

    void InsertTail(Node\* );

    bool DeleteFirst();

    void InsertPoly(int coefficient, int exponent);

    void printPolynomial();

    int EvaluatePolynomial(int x);

    Polynomial add2Polynomial(const Polynomial& other);

    Polynomial multiply2Poly(const Polynomial& other);

    Polynomial multiply(const Polynomial& other);

    Polynomial operator+(const Polynomial& other) const;

    Polynomial operator\*(const Polynomial& other) const;

    void AddTerm(int coef, int exp);

};

### **Polynomial.cpp**

#include "Polynomial.h"

#include <iostream>

#include <cmath>

using namespace std;

Polynomial::Polynomial()

{

    //ctor

    this->head = nullptr;

    this->tail = nullptr;

    this->nNum = 0;

}

Polynomial::~Polynomial()

{

}

void Polynomial::InsertFirst(Node\* e)

{

    if (this->head == nullptr)

        this->head = this->tail = e;

    else

    {

        e->Setpointer(this->head);

        this->head = e;

    }

    this->nNum++;

}

void Polynomial::InsertTail(Node\* e)

{

    if (this->head == nullptr)

        this->head = this->tail = e;

    else

    {

        this->tail->Setpointer(e);

        this->tail = e;

    }

    this->nNum++;

}

void Polynomial::InsertPoly(int coefficient, int exponent) {

    Node\* newNode = **new** Node(coefficient, exponent);

    if (!head || head->GetExponent() < exponent)

    {

        newNode->Setpointer(this->head);

        this->head = newNode;

    }

    else

    {

        Node\* currNode = this->head;

        while (currNode->Getpointer()!=nullptr && currNode->Getpointer()->GetExponent() > exponent)

        {

            currNode = currNode->Getpointer();

        }

        if (currNode->GetExponent() == exponent)

        {

            currNode->SetCoefficient(currNode->GetCoefficient() + coefficient);

        }

        else

        {

            newNode->Setpointer(currNode->Getpointer());

            currNode->Setpointer(newNode);

        }

    }

}

void Polynomial::printPolynomial()

{

    Node\* p = this->head;

    while (p != nullptr)

    {

        cout << p->GetCoefficient();

        if (p->GetExponent() > 1)

        {

            cout<< "x^" << p->GetExponent();

        }

        else if (p->GetExponent() == 1)

        {

            cout << "x";

        }

        p = p->Getpointer();

        if (p != nullptr) cout << " + ";

    }

}

int Polynomial::EvaluatePolynomial(int x)

{

    int resultOfEvaPoly = 0;

    Node\* p = this->head;

    while (p != nullptr)

    {

        resultOfEvaPoly += p->GetCoefficient() \* pow(x, p->GetExponent());

        p = p->Getpointer();

    }

    return resultOfEvaPoly;

}

Polynomial Polynomial::add2Polynomial(const Polynomial& other)

{

    Polynomial resultAdd;

    Node\* p1 = this->head;

    Node\* p2 = other.head;

    while (p1 != nullptr && p2 != nullptr)

    {

        if (p1->GetExponent() > p2->GetExponent())

        {

            Node\* p3 = **new** Node(p1->GetCoefficient(), p1->GetExponent());

            resultAdd.InsertTail(p3);

            p1 = p1->Getpointer();

        }

        else if (p1->GetExponent() < p2->GetExponent())

        {

            Node\* p3 = **new** Node(p2->GetCoefficient(), p2->GetExponent());

            resultAdd.InsertTail(p3);

            p2 = p2->Getpointer();

        }

        else

        {

            Node\* p3 = **new** Node(p1->GetCoefficient() + p2->GetCoefficient(), p1->GetExponent());

            resultAdd.InsertTail(p3);

            p1 = p1->Getpointer();

            p2 = p2->Getpointer();

        }

    }

    while (p1 != nullptr)

    {

        Node\* p3 = **new** Node(p1->GetCoefficient(), p1->GetExponent());

        resultAdd.InsertTail(p3);

        p1 = p1->Getpointer();

    }

    while (p2 != nullptr)

    {

        Node\* p3 = **new** Node(p2->GetCoefficient(), p2->GetExponent());

        resultAdd.InsertTail(p3);

        p2 = p2->Getpointer();

    }

    return resultAdd;

}

Polynomial Polynomial::operator+(const Polynomial& other) const

{

    Polynomial resultAdd;

    Node\* p1 = this->head;

    Node\* p2 = other.head;

    while (p1 != nullptr && p2 != nullptr)

    {

        if (p1->GetExponent() > p2->GetExponent())

        {

            Node\* p3 = **new** Node(p1->GetCoefficient(), p1->GetExponent());

            resultAdd.InsertTail(p3);

            p1 = p1->Getpointer();

        }

        else if (p1->GetExponent() < p2->GetExponent())

        {

            Node\* p3 = **new** Node(p2->GetCoefficient(), p2->GetExponent());

            resultAdd.InsertTail(p3);

            p2 = p2->Getpointer();

        }

        else

        {

            Node\* p3 = **new** Node(p1->GetCoefficient() + p2->GetCoefficient(), p1->GetExponent());

            resultAdd.InsertTail(p3);

            p1 = p1->Getpointer();

            p2 = p2->Getpointer();

        }

    }

    while (p1 != nullptr)

    {

        Node\* p3 = **new** Node(p1->GetCoefficient(), p1->GetExponent());

        resultAdd.InsertTail(p3);

        p1 = p1->Getpointer();

    }

    while (p2 != nullptr)

    {

        Node\* p3 = **new** Node(p2->GetCoefficient(), p2->GetExponent());

        resultAdd.InsertTail(p3);

        p2 = p2->Getpointer();

    }

    return resultAdd;

}

Polynomial Polynomial::multiply2Poly(const Polynomial& other)

{

    Polynomial result;

    Node\* p1 = this->head;

    if (p1 == nullptr)

    {

        result = other;

    }

    while (p1!=nullptr)

    {

        Node\* p2 = other.head;

        Polynomial tempPoly;

        while (p2!=nullptr)

        {

            Node\* p3 = **new** Node(p1->GetCoefficient() \* p2->GetCoefficient(), p1->GetExponent() + p2->GetExponent());

            tempPoly.InsertTail(p3);

            p2 = p2->Getpointer();

        }

        result = result.add2Polynomial(tempPoly);

        p1 = p1->Getpointer();

    }

    return result;

}

Polynomial Polynomial::operator\*(const Polynomial& other) const

{

    Polynomial result;

    Node\* p1 = this->head;

    if (p1 == nullptr)

    {

        result = other;

    }

    while (p1 != nullptr)

    {

        Node\* p2 = other.head;

        Polynomial tempPoly;

        while (p2 != nullptr)

        {

            Node\* p3 = **new** Node(p1->GetCoefficient() \* p2->GetCoefficient(), p1->GetExponent() + p2->GetExponent());

            tempPoly.InsertTail(p3);

            p2 = p2->Getpointer();

        }

        result = result.add2Polynomial(tempPoly);

        p1 = p1->Getpointer();

    }

    return result;

}

Polynomial Polynomial::multiply(const Polynomial& other)

{

    Polynomial result;

    for (Node\* p1 = this->head; p1; p1 = p1->Getpointer())

    {

        Polynomial temp;

        for (Node\* p2 = other.head; p2; p2 = p2->Getpointer()) {

            temp.InsertPoly(p1->GetCoefficient() \* p2->GetCoefficient(), p1->GetExponent() + p2->GetExponent());

        }

        result = result.add2Polynomial(temp);

    }

    return result;

}

### **main.cpp**

#include <bits/stdc++.h>

#include "Polynomial.h"

#include "Node.h"

using namespace std;

int main() {

    Polynomial poly1;

    Polynomial poly2;

    // Đa thức 1: 3x^4 + 2x^3 + x

    poly1.InsertPoly(3, 4);

    poly1.InsertPoly(2, 3);

    poly1.InsertPoly(1, 1);

    // Đa thức 2: 5x^3 + 4x + 2

    poly2.InsertPoly(5, 3);

    poly2.InsertPoly(4, 1);

    poly2.InsertPoly(2, 0);

    cout << "Da thuc 1: ";

    poly1.printPolynomial();

    cout << "\n";

    cout << "Da thuc 2: ";

    poly2.printPolynomial();

    cout << "\n";

    // Tính giá trị đa thức tại x = 2

    int x = 2;

    cout << "Gia tri Da thuc 1 tai x = " << x << ": " << poly1.EvaluatePolynomial(x) << endl;

    cout << "Gia tri Da thuc 2 tai x = " << x << ": " << poly2.EvaluatePolynomial(x) << endl;

    // Cộng hai đa thức

    Polynomial sum = poly1 + poly2;

    cout << "Tong: ";

    sum.printPolynomial();

    cout << "\n";

    // Nhân hai đa thức

    Polynomial product = poly1 \* poly2;

    cout << "Tich: ";

    product.printPolynomial();

    return 0;

}

**2. Thông tin của một quyển sách trong thư viện gồm các thông tin:**

**• Tên sách (chuỗi)**

**• Tác giả (chuỗi, tối đa 5 tác giả)**

**• Nhà xuất bản (chuỗi)**

**• Năm xuất bản (số nguyên)**

a. Hãy tạo danh sách liên kết (đơn hoặc kép) chứa thông tin các quyển sách có trong thư viện (được nhập từ bàn phím).

b. Cho biết số lượng các quyển sách của một tác giả bất kỳ (nhập từ bàn phím).

c. Trong năm YYYY (nhập từ bàn phím), nhà xuất bản ABC (nhập từ bàn phím) đã phát hành những quyển sách nào.

### **Book.h**

#pragma once

#include <iostream>

#include <cstring>

#include <vector>

using namespace std;

class Book

{

private:

    string title;

    vector<string> author;

    string publisher;

    int year;

    Book \*next;

public:

    Book(string, vector<string>, string, int);

    string GetTitle()

    {

        return title;

    }

    void SetTitle(int value)

    {

        title = value;

    }

    vector<string> GetAuthors();

    string GetPublisher();

    int GetYear();

    Book\* GetNextPointer();

    void SetNextPointer(Book\* nextBook);

};

### **Book.cpp**

#include "Book.h"

#include <string>

#include <bits/stdc++.h>

Book::Book(string t, vector<string> a, string p, int y)

{

    title = t;

    author = a;

    publisher = p;

    year = y;

    next = nullptr;

}

vector<string> Book::GetAuthors()

{

    return author;

}

string Book::GetPublisher()

{

    return publisher;

}

int Book::GetYear()

{

    return year;

}

Book\* Book::GetNextPointer()

{

    return next;

}

void Book::SetNextPointer(Book\* ptr)

{

    next = ptr;

}

### **Library.h**

#pragma once

#include "Book.h"

#include <string>

using namespace std;

class Library

{

private:

    Book\* head;

    Book\* tail;

    int nNum;

public:

    Library();

    ~Library();

    void InsertFirst(Book\* e);

    void InsertTail(Book\* e);

    void printBooks();

    int countBooksByAuthor(string author);

    void printBooksByPublisherAndYear(string publisher, int year);

};

### **Library.cpp**

#include "Library.h"

#include <iostream>

using namespace std;

Library::Library()

{

    //ctor

    this->head = nullptr;

    this->tail = nullptr;

    this->nNum = 0;

}

Library::~Library()

{

}

void Library::InsertFirst(Book\* e)

{

    if (this->head == nullptr)

        this->head = this->tail = e;

    else

    {

        e->SetNextPointer(this->head);

        this->head = e;

    }

    this->nNum++;

}

void Library::InsertTail(Book\* e)

{

    if (this->head == nullptr)

        this->head = this->tail = e;

    else

    {

        this->tail->SetNextPointer(e);

        this->tail = e;

    }

    this->nNum++;

}

void Library::printBooks()

{

    Book\* currBook = this->head;

    while (currBook != nullptr)

    {

        cout << "Title: " << currBook->GetTitle() << endl;

        cout << "Authors: ";

        for (const auto& author : currBook->GetAuthors())

        {

            cout << author << " ";

        }

        cout << endl;

        cout << "Publisher: " << currBook->GetPublisher() << endl;

        cout << "Year: " << currBook->GetYear() << endl;

        cout << endl;

        currBook = currBook->GetNextPointer();

    }

}

int Library::countBooksByAuthor(string author)

{

    int countBooks = 0;

    Book\* currBook = this->head;

    while (currBook!=nullptr)

    {

        for (const auto& a : currBook->GetAuthors())

        {

            if (a == author)

            {

                countBooks++;

                break;

            }

        }

        currBook = currBook->GetNextPointer();

    }

    return countBooks;

}

void Library::printBooksByPublisherAndYear(string publisher, int year)

{

    Book\* currBook = this->head;

    while (currBook!=nullptr)

    {

        if (currBook->GetPublisher() == publisher && currBook->GetYear() == year)

        {

            cout << "Title: " << currBook->GetTitle() << endl;

            cout << "Authors: ";

            for (const auto& author : currBook->GetAuthors())

            {

                cout << author << " ";

            }

            cout << endl;

            cout << "Publisher: " << currBook->GetPublisher() << endl;

            cout << "Year: " << currBook->GetYear() << endl;

            cout << endl;

        }

        currBook = currBook->GetNextPointer();

    }

}

### **Sourcecode.cpp**

#include <iostream>

#include "Library.h"

#include "Book.h"

using namespace std;

int main()

{

    Library lib;

    Book\* book1;

    Book\* book2;

    Book\* book3;

    Book\* book4;

    book1 = **new** Book("Book1", {"Author1", "Author2"}, "Publisher1", 2020);

    book2 = **new** Book("Book2", {"Author3", "Author4"}, "Publisher2", 2021);

    book3 = **new** Book("Book3", {"Author5", "Author6"}, "Publisher2", 2022);

    book4 = **new** Book("Book4", {"Author7", "Author8"}, "Publisher1", 2020);

    lib.InsertTail(book1);

    lib.InsertTail(book2);

    lib.InsertTail(book3);

    lib.InsertTail(book4);

    // Cau b

    string searchAuthor;

    cout << "Nhap ten tac gia de dem: ";

    getline(cin, searchAuthor);

    cout << "Number of books by " << searchAuthor << ": " << lib.countBooksByAuthor(searchAuthor) << "\n";

    // Cau c

    string searchPublisher;

    int searchYear;

    cout << "Nhap ten nha phat hanh: ";

    getline(cin, searchPublisher);

    cout << "Nhap nam: ";

    cin >> searchYear;

    cin.ignore();

    lib.printBooksByPublisherAndYear(searchPublisher, searchYear);

    return 0;

}

# **PHẦN 2: STACK - QUEUE**

**1. Sử dụng danh sách liên kết để cài đặt cấu trúc Stack, Queue.**

### **element.h**

#ifndef ELEMENT\_H

#define ELEMENT\_H

class element {

private:

    int data;

    element\* pointer;

public:

    element(int value);

    virtual ~element();

    int Getdata() { return data; }

    void Setdata(int val) { data = val; }

    element\* Getpointer() { return pointer; }

    void Setpointer(element\* p) { pointer = p; }

};

#endif

### **element.cpp**

#include "element.h"

element::element(int value)

{

    this->data = value;

    this->pointer = nullptr;

}

element::~element()

{

}

### **linkedlist.h**

#ifndef LINKEDLIST\_H

#define LINKEDLIST\_H

#include "element.h"

class linkedList {

private:

    element\* head;

    element\* tail;

    int nNum;

public:

    linkedList();

    virtual ~linkedList();

    element\* getHead() { return head; }

    void setHead(element\* val) { head = val; }

    void InsertFirst(element\* e);

    void InsertFirstNotDuplicate(element\* e);

    void InsertTail(element\* e);

    bool CheckContains(int x);

    void InsertTailNotDulicate(element\* e);

    void Travel();

    int SumList();

    bool DeleteFirst();

    bool DeleteTail();

    bool DeleteP(element\* p);

    bool removeAll();

    int MaxList();

    int CountPrime();

    void InsertAfterP(element\* Data, element\* p);

    int CountDuplicate(int x);

    void Split2List(int x, linkedList\* lessThanX, linkedList\* greaterOrEqualX);

};

#endif

### **linkedlist.cpp**

#include "linkedList.h"

#include <iostream>

#include <cmath>

using namespace std;

linkedList::linkedList()

{

    //ctor

    this->head = nullptr;

    this->tail = nullptr;

    this->nNum = 0;

}

linkedList::~linkedList()

{

    //dtor

}

void linkedList::InsertFirst(element\* e)

{

    if (this->head == nullptr)

        this->head = this->tail = e;

    else

    {

        e->Setpointer(this->head); //step 1

        //e->pointer = this->head;

        this->head = e;  // step 2

    }

    this->nNum++;

}

void linkedList::InsertFirstNotDuplicate(element\* e) {

    if (this->head == nullptr)

    {

        this->head = this->tail = e;

        this->nNum++;

    }

    else

    {

        if (CheckContains(e->Getdata()) == false)

        {

            e->Setpointer(this->head);

            this->head = e;

            this->nNum++;

        }

    }

}

void linkedList::InsertTail(element\* e)

{

    if (this->head == nullptr)

        this->head = this->tail = e;

    else

    {

        this->tail->Setpointer(e);// step 1

        this->tail = e;    // step 2

    }

    this->nNum++;

}

bool linkedList::CheckContains(int x)

{

    element\* p = this->head;

    while (p != nullptr)

    {

        if (p->Getdata() == x) return true;

        p = p->Getpointer();

    }

    return false;

}

void linkedList::InsertTailNotDulicate(element\* e) {

    if (this->head == nullptr)

    {

        this->head = this->tail = e;

        this->nNum++;

    }

    else

    {

        if (CheckContains(e->Getdata()) == false)

        {

            this->tail->Setpointer(e);

                this->tail = e;

            this->nNum++;

        }

    }

}

void linkedList::Travel()

{

    element\* p = this->head;

    while (p != nullptr)

    {

        cout << p->Getdata() << "\t"; // p -> data

        p = p->Getpointer(); // p = p -> pointer

    }

}

int linkedList::SumList()

{

    int sumNum = 0;

    element\* p = this->head;

    while (p != nullptr)

    {

        sumNum += p->Getdata();

        p = p->Getpointer();

    }

    return sumNum;

}

bool linkedList::DeleteFirst()

{

    if (this->head == nullptr)

        return false;

    else

    {

        element\* p = this->head;

        this->head = this->head->Getpointer();

**delete** p;

        return true;

    }

    return true;

}

bool linkedList::DeleteTail()

{

    if (this->head == nullptr)

        return false;

    else

    {

        element\* p = this->head;

        while (p->Getpointer() != this->tail)

            p = p->Getpointer();

**delete** this->tail;

        this->tail = p;

        this->tail->Setpointer(nullptr);  // this->tail->pointer = nullptr;

        return true;

    }

    return true;

}

bool linkedList::DeleteP(element\* p)

{

    element\* q;

    q = this->head;

    if (this->head == p)

    {

        this->head = p->Getpointer();

**delete** p;

        return true;//Xoa thanh cong

    }

    while ((q != nullptr) && (q->Getpointer() != p)) //Tim q lien truoc p

        q = q->Getpointer();

    if (q == nullptr)

        return false; //khong tim thay phan tu co khoa bang p

    else

    {

        q->Setpointer(p->Getpointer());

        //q->pointer = p->pointer;

**delete** p;

        return true; //Xoa thanh cong

    }

    return true;

}

bool linkedList::removeAll()

{

    while (this->head != nullptr)

    {

        DeleteFirst();

    }

    return true;

}

int linkedList::MaxList()

{

    element\* p = this->head;

    int maxNum = p->Getdata();

    while (p != nullptr)

    {

        if (maxNum < p->Getdata())

            maxNum = p->Getdata();

        p = p->Getpointer();

    }

    return maxNum;

}

bool isPrime(int num)

{

    if (num < 2)

        return false;

    for (int i = 2; i <= sqrt(num); i++)

    {

        if (num % i == 0)

        {

            return false;

        }

    }

    return true;

}

int linkedList::CountPrime()

{

    int coutPrimeNum = 0;

    element\* p = this->head;

    while (p != nullptr)

    {

        if (isPrime(p->Getdata()) == true)

            coutPrimeNum++;

        p = p->Getpointer();

    }

    return coutPrimeNum;

}

void linkedList::InsertAfterP(element\* Data, element\* p)

{

    if (p != NULL)

    {

        Data->Setpointer(p->Getpointer());//Data->pointer = p->pointer;

        p->Setpointer(Data); // p->pointer = Data;

        if (this->tail == p)

            this->tail = Data;

    }

    else

        InsertFirst(Data);

}

int linkedList::CountDuplicate(int x)

{

    int countDuplicateValue = 0;

    element\* p = this->head;

    while (p != nullptr)

    {

        if (p->Getdata() == x) countDuplicateValue++;

        p = p->Getpointer();

    }

    return countDuplicateValue;

}

void linkedList::Split2List(int x, linkedList\* lessThanX, linkedList\* greaterOrEqualX)

{

    element\* currElement = this->head;

    while (currElement != nullptr)

    {

        element\* e = **new** element(currElement->Getdata());

        if (currElement->Getdata() < x)

        {

            lessThanX->InsertTail(e);

        }

        else

        {

            greaterOrEqualX->InsertTail(e);

        }

        currElement = currElement->Getpointer();

    }

}

**2. Sử dụng các hàm PushStack, PopStack, EnQueue, DeQueue để cài đặt:….**

### **a. Về Stack**

#### **Stack.h**

#ifndef STACK\_H

#define STACK\_H

#include "linkedlist.h"

#include <iostream>

using namespace std;

class Stack {

private:

    int nNum;

    linkedList\* linkedlist;

public:

    Stack();

    virtual ~Stack();

    void push(int);

    int pop();

    void printStack();

    bool isEmpty();

    int top();

};

#endif // STACK\_H

#### **Stack.cpp**

#include "Stack.h"

#include "element.h"

void Stack::push(int x)

{

    this->linkedlist->InsertFirst(**new** element(x));

    this->nNum++;

}

int Stack::pop()

{

    if (nNum == 0)

    {

        cout << "STACK RONG, KHONG LAY DUOC PHAN TU" << endl;

        return -1;

    }

    int p = this->linkedlist->getHead()->Getdata();

    this->linkedlist->DeleteFirst();

    this->nNum--;

    return p;

}

void Stack::printStack()

{

    if (nNum == 0)

    {

        cout << "STACK RONG" << endl;

        return;

    }

    this->linkedlist->Travel();

}

bool Stack::isEmpty()

{

    return this->linkedlist->getHead() == nullptr;

}

int Stack::top()

{

    if (this->nNum == 0)

    {

        cout << "STACK RONG" << endl;

        return -1;

    }

    return this->linkedlist->getHead()->Getdata();

}

Stack::Stack()

{

    this->nNum = 0;

    this->linkedlist = **new** linkedList();

}

Stack::~Stack()

{

**delete** linkedlist;

}

#### **main.cpp**

int main()

{

    cout << "=== Stack  ===" << endl;

    Stack\* s = **new** Stack();

    int stackInput;

    for(int i = 0; i < 3; ++i) {

        cout << "Nhap gia tri vao stack(nhap 1 gia tri trong 1 cau lenh): ";

        cin >> stackInput;

        s->push(stackInput);

    }

    s->push(20);

    cout << "Stack hien tai: ";

    s->printStack();

    cout << "\n";

    for(int i = 0; i < 4; ++i) {

        int poppedValue = s->pop();

        if(poppedValue != -1) {

            cout << "Lay ra tu stack: " << poppedValue << endl;

        }

    }

    int extraPop = s->pop();

    if(extraPop != -1) {

        cout << "Lay ra tu stack: " << extraPop << endl;

    }

**delete** s;

### **b.** **Về Queue**

#### **Queue.h**

#ifndef QUEUE\_H

#define QUEUE\_H

#include "linkedlist.h"

#include <iostream>

using namespace std;

class Queue {

private:

    int nNum;

    linkedList\* linkedlist;

public:

    Queue();

    virtual ~Queue();

    void enqueue(int);

    int dequeue();

    bool isEmpty();

    void printQueue();

    int front();

    int rear();

};

#endif // QUEUE\_H

#### **Queue.cpp**

#include "Queue.h"

#include "element.h"

using namespace std;

void Queue::enqueue(int x)

{

    element\* p = **new** element(x);

    this->linkedlist->InsertTail(p);

    nNum++;

}

int Queue::dequeue()

{

    if (nNum == 0)

    {

        cout << "QUEUE RONG, KHONG LAY DUOC PHAN TU" << endl;

        return -1;

    }

    int p = this->linkedlist->getHead()->Getdata();

    this->linkedlist->DeleteFirst();

    nNum--;

    return p;

}

bool Queue::isEmpty()

{

    return this->linkedlist->getHead() == nullptr;

}

void Queue::printQueue()

{

    if (nNum == 0)

    {

        cout << "QUEUE RONG" << endl;

        return;

    }

    this->linkedlist->Travel();

}

int Queue::front()

{

    if (this->nNum == 0)

    {

        cout << "QUEUE RONG";

    }

    return this->linkedlist->getHead()->Getdata();

}

int Queue::rear()

{

    if (this->nNum == 0) {

        cout << "QUEUE RONG";

    }

    element\* current = this->linkedlist->getHead();

    while (current->Getpointer() != nullptr)

    {

        current = current->Getpointer();

    }

    return current->Getdata();

}

Queue::Queue()

{

    this->nNum = 0;

    this->linkedlist = **new** linkedList();

}

Queue::~Queue()

{

**delete** linkedlist;

}

#### **main.cpp**

    cout << "\n=== Queue ===" << endl;

    Queue\* q = **new** Queue();

    int queueInput;

    for(int i = 0; i < 3; ++i) {

        cout << "Nhap gia tri vao queue(nhap 1 gia tri trong 1 cau lenh): ";

        cin >> queueInput;

        q->enqueue(queueInput);

    }

    q->enqueue(25);

    cout << "Queue hien tai: ";

    q->printQueue();

    cout << "\n";

    for(int i = 0; i < 4; ++i) {

        int dequeuedValue = q->dequeue();

        if(dequeuedValue != -1) {

            cout << "Lay ra tu queue: " << dequeuedValue << endl;

        }

    }

    int extraDequeue = q->dequeue();

    if(extraDequeue != -1) {

        cout << "Lay ra tu queue: " << extraDequeue << endl;

    }

**delete** q;

    return 0;

}

# **Áp dụng – Nâng cao**

## **1. Sử dụng hàm InsertTail và DeleteTail trong LinkedList để có phiên bản cài đặt Stack (thêm phần tử vào cuối danh sách và lấy phần tử ở cuối danh sách liên kết) cũng như áp dụng 1 phiên bản khác khi cài đặt Queue (thêm phần tử vào cuối danh sách liên kết và lấy phần tử ở đầu danh sách liên kết).**

### **Stack.cpp**

#include "Stack.h"

#include "element.h"

using namespace std;

Stack::Stack()

{

    this->nNum = 0;

    this->linkedlist = **new** linkedList();

}

Stack::~Stack()

{

**delete** linkedlist;

}

void Stack::push(int x)

{

    // Thay đổi từ InsertFirst sang InsertTail

    this->linkedlist->InsertTail(**new** element(x));

    this->nNum++;

}

int Stack::pop()

{

    if (nNum == 0)

    {

        cout << "STACK RONG, KHONG LAY DUOC PHAN TU" << endl;

        return -1;

    }

    // Lấy giá trị của phần tử cuối

    element\* current = this->linkedlist->getHead();

    while (current->Getpointer() != nullptr) {

        current = current->Getpointer();

    }

    int value = current->Getdata();

    // Xóa phần tử cuối

    this->linkedlist->DeleteTail();

    this->nNum--;

    return value;

}

int Stack::top()

{

    if (this->nNum == 0)

    {

        cout << "Stack empty" << endl;

        return -1;

    }

    // Lấy giá trị của phần tử cuối

    element\* current = this->linkedlist->getHead();

    while (current->Getpointer() != nullptr) {

        current = current->Getpointer();

    }

    return current->Getdata();

}

void Stack::printStack()

{

    if (nNum == 0)

    {

        cout << "STACK RONG" << endl;

        return;

    }

    this->linkedlist->Travel();

}

bool Stack::isEmpty()

{

    return this->linkedlist->getHead() == nullptr;

}

### **Queue.cpp**

#include "Queue.h"

#include "element.h"

using namespace std;

Queue::Queue()

{

    this->nNum = 0;

    this->linkedlist = **new** linkedList();

}

Queue::~Queue()

{

**delete** linkedlist;

}

void Queue::enqueue(int x)

{

    element\* p = **new** element(x);

    this->linkedlist->InsertTail(p);

    nNum++;

}

int Queue::dequeue()

{

    if (nNum == 0)

    {

        cout << "QUEUE RONG, KHONG LAY DUOC PHAN TU" << endl;

        return -1;

    }

    int p = this->linkedlist->getHead()->Getdata();

    this->linkedlist->DeleteFirst();

    nNum--;

    return p;

}

bool Queue::isEmpty()

{

    return this->linkedlist->getHead() == nullptr;

}

void Queue::printQueue()

{

    if (nNum == 0)

    {

        cout << "QUEUE RONG" << endl;

        return;

    }

    this->linkedlist->Travel();

}

int Queue::front()

{

    if (this->nNum == 0)

    {

        cout << "QUEUE RONG";

    }

    return this->linkedlist->getHead()->Getdata();

}

int Queue::rear()

{

    if (this->nNum == 0) {

        cout << "QUEUE RONG" << endl;

        return -1;

    }

    element\* current = this->linkedlist->getHead();

    while (current->Getpointer() != nullptr)

    {

        current = current->Getpointer();

    }

    return current->Getdata();

}

**2. Nhận xét cách cài đặt trên ở phần 1 (áp dụng – nâng cao) so với chương trình mẫu đối với trường hợp stack cũng như queue.**

- Chương trình mẫu sử dụng InsertFirst và DeleteFirst thì Stack được cài đặt bằng Push: Thêm phần tử vào **đầu** danh sách liên kết (InsertFirst) với Pop: Xóa và lấy phần tử từ **đầu** danh sách liên kết (DeleteFirst), ưu điểm là nó dễ hơn không cần duyệt danh sách liên kết để tìm phần tử cuối cùng, còn nhược điểm thì theo google thì nó không phù hợp để mô phỏng Stack dựa trên các thao tác với phần tử cuối danh sách.

- Còn về cách cài đặt ở phần 1 là sử dụng InsertTail và DeleteTail, Stack được cài đặt bằng Push: Thêm phần tử vào **cuối** danh sách liên kết (InsertTail) với Pop: Xóa và lấy phần tử từ **cuối** danh sách liên kết (DeleteTail), ưu điểm là Tuân Thủ Nguyên Tắc Stack, còn nhược điểm là nó lâu mất thời gian, ko hiểu quả đối với cái chương trình mẫu.

- Phiên Bản Queue Sử Dụng InsertTail và DeleteFirst: Đây là cách cài đặt tiêu chuẩn và hiệu quả cho Queue với các thao tác enqueue và dequeue đều có độ phức tạp thời gian **O(1)**. (source: Google)

**3. Sử dụng cấu trúc Stack để chuyển giá trị từ cơ số 10 sang cơ số 2.**

Gợi ý : thực hiện việc chia liên tiếp giá trị trong cơ số 10 cho 2, lấy phần dư đưa vào stack, cho đến khi giá trị đem đi chia là 0. In giá trị trong stack ra (đó chính là kết quả khi chuyển số từ hệ cơ số 10 sang hệ cơ số 2).

### **main.cpp**

string ConvertDecimalToBinary(Stack\* binaryNo, int decimalNo)

{

    string strBinNo = "";

    if(decimalNo == 0)

    {

        return "0";

    }

    while(decimalNo > 0)

    {

        int remainder = decimalNo % 2;

        binaryNo->push(remainder);

        decimalNo /= 2;

    }

    while(!binaryNo->isEmpty())

    {

        int bit = binaryNo->pop();

        if(bit != -1) { // Kiểm tra xem pop có thành công không

            strBinNo += to\_string(bit);

        }

    }

    return strBinNo;

}

    cout << "\n=== Chuyen Doi Cac So Nhi Phan ===" << endl;

    int decimalNumber;

    cout << "Nhap mot so nguyen duong (co so 10): ";

    cin >> decimalNumber;

    Stack \*binaryStack = **new** Stack();

    string binaryResult = ConvertDecimalToBinary(binaryStack, decimalNumber);

    cout << "Gia tri nhi phan: " << binaryResult << endl;

    // Giải phóng bộ nhớ cho Stack chuyển đổi

**delete** binaryStack;

# **BÀI TẬP ỨNG DỤNG**

## **Bài 1: Tìm đường trong mê cung (thực hiện loang theo chiều rộng <sử dụng queue> hoặc loang theo chiều sâu <sử dụng stack>).**

Bài toán: cho ma trận mxn, mỗi phần tử là số 0 hoặc 1. Giá trị 1 : có thể đi tới và giá trị 0 : không thể đi tới được.

### **Pathfinding.h**

#ifndef PATHFINDING\_H

#define PATHFINDING\_H

#include <vector>

#include <queue>

#include <stack>

#include <utility>

using namespace std;

const vector<pair<int, int>> DIRECTIONS = {

    {-1, 0}, {1, 0}, {0, -1}, {0, 1}

};

bool isValid(int x, int y, int m, int n, const vector<vector<int>>& maze, vector<vector<bool>>& visited);

bool bfs(const vector<vector<int>>& maze, pair<int, int> start, pair<int, int> dest);

bool dfs(const vector<vector<int>>& maze, pair<int, int> start, pair<int, int> dest);

#endif // PATHFINDING\_H

### **Pathfinding.cpp**

#include "Pathfinding.h"

#include <iostream>

bool isValid(int x, int y, int m, int n, const vector<vector<int>>& maze, vector<vector<bool>>& visited) {

    return (x >= 0 && x < m && y >= 0 && y < n && maze[x][y] == 1 && !visited[x][y]);

}

bool bfs(const vector<vector<int>>& maze, pair<int, int> start, pair<int, int> dest) {

    int m = maze.size();

    if(m == 0) return false;

    int n = maze[0].size();

    vector<vector<bool>> visited(m, vector<bool>(n, false));

    queue<pair<int, int>> q;

    q.push(start);

    visited[start.first][start.second] = true;

    while(!q.empty()) {

        pair<int, int> current = q.front();

        q.pop();

        if(current == dest) {

            return true;

        }

        for(auto dir : DIRECTIONS) {

            int newX = current.first + dir.first;

            int newY = current.second + dir.second;

            if(isValid(newX, newY, m, n, maze, visited)) {

                q.push({newX, newY});

                visited[newX][newY] = true;

            }

        }

    }

    return false;

}

bool dfs(const vector<vector<int>>& maze, pair<int, int> start, pair<int, int> dest) {

    int m = maze.size();

    if(m == 0) return false;

    int n = maze[0].size();

    vector<vector<bool>> visited(m, vector<bool>(n, false));

    stack<pair<int, int>> s;

    s.push(start);

    visited[start.first][start.second] = true;

    while(!s.empty()) {

        pair<int, int> current = s.top();

        s.pop();

        if(current == dest) {

            return true;

        }

        for(auto dir : DIRECTIONS) {

            int newX = current.first + dir.first;

            int newY = current.second + dir.second;

            if(isValid(newX, newY, m, n, maze, visited)) {

                s.push({newX, newY});

                visited[newX][newY] = true;

            }

        }

    }

    return false;

}

### **main.cpp**

#include <iostream>

#include <vector>

#include "Pathfinding.h"

using namespace std;

int main() {

    vector<vector<int>> maze = {

        {1, 1, 0, 1, 1},

        {0, 1, 0, 0, 1},

        {1, 1, 1, 0, 0},

        {1, 0, 1, 1, 1},

        {0, 1, 1, 0, 1}

    };

    pair<int, int> start = {0, 0}; // Ô bắt đầu (x1, y1)

    pair<int, int> dest = {4, 4};  // Ô đích (x2, y2)

    if(bfs(maze, start, dest)) {

        cout << "Duong di ton tai tu (" << start.first << ", " << start.second << ") den ("

                  << dest.first << ", " << dest.second << ") bang BFS." << endl;

    } else {

        cout << "Khong co duong di tu (" << start.first << ", " << start.second << ") den ("

                  << dest.first << ", " << dest.second << ") bang BFS." << endl;

    }

    if(dfs(maze, start, dest)) {

        cout << "Duong di ton tai tu (" << start.first << ", " << start.second << ") den ("

                  << dest.first << ", " << dest.second << ") bang DFS." << endl;

    } else {

        cout << "Khong co duong di tu (" << start.first << ", " << start.second << ") den ("

                  << dest.first << ", " << dest.second << ") bang DFS." << endl;

    }

    return 0;

}

**Câu hỏi: Từ ô ban đầu có tọa độ (x1, y1) có thể đi tới ô (x2, y2) không?**

Biết rằng từ 1 ô (x,y) chỉ có thể đi qua ô có chung cạnh với ô đang đứng và mang giá trị là 1, ngược lại không có đường đi.

* Có, từ ô (x1, y1) có thể đi tới ô (x2, y2) trong mê cung.

## **Bài 2: Bài toán di chuyển toa tàu (hình dưới): Các toa được đánh số từ 1 đến n, đường di chuyển có thể là các vạch đỏ. Ta cần di chuyển các toa từ A -> C sao cho tại C các toa tàu được sắp xếp các thứ tự mới nào đó. Hãy nhập vào thứ tự tại C cần có, cho biết có cách chuyển không ? Nếu có, hãy trình bày cách chuyển.**

### **TrainMovement.h**

#ifndef TRAIN\_MOVEMENT\_H

#define TRAIN\_MOVEMENT\_H

#include <vector>

#include <stack>

#include <string>

using namespace std;

class State {

private:

    vector<int> A;

    stack<int> B;

    vector<int> C;

    vector<string> moves;

public:

    State() = default;

    const vector<int>& getA() const { return A; }

    const stack<int>& getB() const { return B; }

    const vector<int>& getC() const { return C; }

    const vector<string>& getMoves() const { return moves; }

    void pushToA(int value) { A.push\_back(value); }

    void popFromA() { if (!A.empty()) A.erase(A.begin()); }

    bool isAEmpty() const { return A.empty(); }

    int frontA() const { return A.empty() ? -1 : A.front(); }

    void pushToB(int value) { B.push(value); }

    void popFromB() { if (!B.empty()) B.pop(); }

    bool isBEmpty() const { return B.empty(); }

    int topB() const { return B.empty() ? -1 : B.top(); }

    void pushToC(int value) { C.push\_back(value); }

    bool isCEqual(const vector<int>& target) const { return C == target; }

    void addMove(const string& move) { moves.push\_back(move); }

};

class TrainMovement {

private:

    int n;

    vector<int> targetOrder;

    bool isTargetReached(const State& state);

    State moveAtoB(const State& current);

    State moveAtoC(const State& current);

    State moveBtoC(const State& current);

    vector<State> getNextStates(const State& current);

public:

    TrainMovement(int numCars, const vector<int>& target);

    bool findSolution();

    void printSolution(const vector<string>& moves);

};

#endif

### **TrainMovement.cpp**

#include "TrainMovement.h"

#include <bits/stdc++.h>

TrainMovement::TrainMovement(int numCars, const vector<int>& target)

    : n(numCars), targetOrder(target) {}

bool TrainMovement::isTargetReached(const State& state) {

    return state.isCEqual(targetOrder);

}

State TrainMovement::moveAtoB(const State& current) {

    if (current.isAEmpty()) return current;

    State newState = current;

    newState.pushToB(newState.frontA());

    newState.popFromA();

    newState.addMove("Chuyen " + to\_string(newState.topB()) + " tu A -> B");

    return newState;

}

State TrainMovement::moveAtoC(const State& current) {

    if (current.isAEmpty()) return current;

    State newState = current;

    int value = newState.frontA();

    newState.pushToC(value);

    newState.popFromA();

    newState.addMove("Chuyen " + to\_string(value) + " tu A -> C");

    return newState;

}

State TrainMovement::moveBtoC(const State& current) {

    if (current.isBEmpty()) return current;

    State newState = current;

    int value = newState.topB();

    newState.pushToC(value);

    newState.popFromB();

    newState.addMove("Chuyen " + to\_string(value) + " tu B -> C");

    return newState;

}

vector<State> TrainMovement::getNextStates(const State& current) {

    vector<State> nextStates;

    if (!current.isAEmpty()) {

        nextStates.push\_back(moveAtoB(current));

        nextStates.push\_back(moveAtoC(current));

    }

    if (!current.isBEmpty()) {

        nextStates.push\_back(moveBtoC(current));

    }

    return nextStates;

}

string getStateString(const State& state) {

    stringstream ss;

    for (int val : state.getA()) ss << val << ",";

    ss << "|";

    stack<int> tempB = state.getB();

    vector<int> B;

    while (!tempB.empty()) {

        B.push\_back(tempB.top());

        tempB.pop();

    }

    for (int val : B) ss << val << ",";

    ss << "|";

    for (int val : state.getC()) ss << val << ",";

    return ss.str();

}

bool TrainMovement::findSolution() {

    State initialState;

    for (int i = 1; i <= n; i++) {

        initialState.pushToA(i);

    }

    queue<State> queue;

    set<string> visited;

    queue.push(initialState);

    visited.insert(getStateString(initialState));

    while (!queue.empty()) {

        State current = queue.front();

        queue.pop();

        if (isTargetReached(current)) {

            printSolution(current.getMoves());

            return true;

        }

        vector<State> nextStates = getNextStates(current);

        for (const State& next : nextStates) {

            string stateStr = getStateString(next);

            if (visited.find(stateStr) == visited.end()) {

                visited.insert(stateStr);

                queue.push(next);

            }

        }

    }

    return false;

}

void TrainMovement::printSolution(const vector<string>& moves) {

    cout << "Giai phap tim thay voi " << moves.size() << " buoc:" << endl;

    for (const string& move : moves) {

        cout << move << endl;

    }

}

### **main.cpp**

#include "TrainMovement.h"

#include <iostream>

using namespace std;

int main() {

    int n;

    cout << "Nhap so toa tau: ";

    cin >> n;

    vector<int> targetOrder(n);

    cout << "Nhap thu tu dich (cach nhau boi dau cach): ";

    for (int i = 0; i < n; i++) {

        cin >> targetOrder[i];

    }

    TrainMovement solver(n, targetOrder);

    if (!solver.findSolution()) {

        cout << "Khong tim thay giai phap!" << endl;

    }

    return 0;

}

### **Trình bày cách chuyển**

Cho là n = 4, thứ tự cần có (1, 3, 4, 2)

A: 1 2 3 4

B: empty

C: empty

Chuyển 1 từ A->C:  
A: 2 3 4

B: rỗng

C: 1

Chuyển 2 từ A->B:

A: 3 4

B: 2

C: 1

Chuyển 3 từ A->C:

A: 4

B: 2

C: 1 3

Chuyển 4 từ A->C:

A: rỗng

B: 2

C: 1 3 4

Chuyển 2 từ B->C:

A: rỗng

B: rỗng

C: 1 3 4 2

## **Bài 3: Tương tự yêu cầu bài 3 nhưng với hình bên dưới:**

### **TrainMovement.h**

#ifndef TRAIN\_MOVEMENT\_H

#define TRAIN\_MOVEMENT\_H

#include <vector>

#include <queue>

#include <string>

using namespace std;

class State {

private:

    vector<int> A;

    vector<int> B;

    vector<int> C;

    vector<string> moves;

public:

    State() = default;

    const vector<int>& getA() const { return A; }

    const vector<int>& getB() const { return B; }

    const vector<int>& getC() const { return C; }

    const vector<string>& getMoves() const { return moves; }

    void pushToA(int value) { A.push\_back(value); }

    void popFromA() { if (!A.empty()) A.erase(A.begin()); }

    bool isAEmpty() const { return A.empty(); }

    int frontA() const { return A.empty() ? -1 : A.front(); }

    void pushToB(int value) { B.push\_back(value); }

    void popFromB() { if (!B.empty()) B.erase(B.begin()); }

    bool isBEmpty() const { return B.empty(); }

    int frontB() const { return B.empty() ? -1 : B.front(); }

    void pushToC(int value) { C.push\_back(value); }

    bool isCEqual(const vector<int>& target) const { return C == target; }

    void addMove(const string& move) { moves.push\_back(move); }

};

class TrainMovement {

private:

    int n;

    vector<int> targetOrder;

    bool isTargetReached(const State& state);

    State moveAtoB(const State& current);

    State moveAtoC(const State& current);

    State moveBtoC(const State& current);

    vector<State> getNextStates(const State& current);

public:

    TrainMovement(int numCars, const vector<int>& target);

    bool findSolution();

    void printSolution(const vector<string>& moves);

};

#endif

### **TrainMovement.cpp**

#include "TrainMovement.h"

#include <bits/stdc++.h>

TrainMovement::TrainMovement(int numCars, const vector<int>& target)

    : n(numCars), targetOrder(target) {}

bool TrainMovement::isTargetReached(const State& state) {

    return state.isCEqual(targetOrder);

}

State TrainMovement::moveAtoB(const State& current) {

    if (current.isAEmpty()) return current;

    State newState = current;

    newState.pushToB(newState.frontA());

    newState.popFromA();

    newState.addMove("Chuyen " + to\_string(newState.frontB()) + " tu A -> B");

    return newState;

}

State TrainMovement::moveAtoC(const State& current) {

    if (current.isAEmpty()) return current;

    State newState = current;

    int value = newState.frontA();

    newState.pushToC(value);

    newState.popFromA();

    newState.addMove("Chuyen " + to\_string(value) + " tu A -> C");

    return newState;

}

State TrainMovement::moveBtoC(const State& current) {

    if (current.isBEmpty()) return current;

    State newState = current;

    int value = newState.frontB();

    newState.pushToC(value);

    newState.popFromB();

    newState.addMove("Chuyen " + to\_string(value) + " tu B -> C");

    return newState;

}

string getStateString(const State& state) {

    stringstream ss;

    for (int val : state.getA()) ss << val << ",";

    ss << "|";

    for (int val : state.getB()) ss << val << ",";

    ss << "|";

    for (int val : state.getC()) ss << val << ",";

    return ss.str();

}

vector<State> TrainMovement::getNextStates(const State& current) {

    vector<State> nextStates;

    if (!current.isAEmpty()) {

        nextStates.push\_back(moveAtoB(current));

        nextStates.push\_back(moveAtoC(current));

    }

    if (!current.isBEmpty()) {

        nextStates.push\_back(moveBtoC(current));

    }

    return nextStates;

}

bool TrainMovement::findSolution() {

    State initialState;

    for (int i = 1; i <= n; i++) {

        initialState.pushToA(i);

    }

    queue<State> queue;

    set<string> visited;

    queue.push(initialState);

    visited.insert(getStateString(initialState));

    while (!queue.empty()) {

        State current = queue.front();

        queue.pop();

        if (isTargetReached(current)) {

            printSolution(current.getMoves());

            return true;

        }

        vector<State> nextStates = getNextStates(current);

        for (const State& next : nextStates) {

            string stateStr = getStateString(next);

            if (visited.find(stateStr) == visited.end()) {

                visited.insert(stateStr);

                queue.push(next);

            }

        }

    }

    return false;

}

void TrainMovement::printSolution(const vector<string>& moves) {

    cout << "Co " << moves.size() << " buoc:" << endl;

    for (const string& move : moves) {

        cout << move << endl;

    }

}

### **main.cpp**

#include "TrainMovement.h"

#include <iostream>

using namespace std;

int main() {

    int n;

    cout << "Nhap so luong toa: ";

    cin >> n;

    vector<int> targetOrder(n);

    cout << "Nhap thu tu toa (cach nhau boi dau cach): ";

    for (int i = 0; i < n; i++) {

        cin >> targetOrder[i];

    }

    TrainMovement solver(n, targetOrder);

    if (!solver.findSolution()) {

        cout << "Khong co giai phap!" << endl;

    }

    return 0;

}

### **Trình bày cách chuyển**

Cho là n = 4, thứ tự cần có (1, 4, 2, 3)

A: 1 2 3 4

B: empty

C: empty

Chuyển 1 từ A->C:

A: 2 3 4

B: rỗng

C: 1

Chuyển 2 từ A->B:

A: 3 4

B: 2

C: 1

Chuyển 3 từ A->B:

A: 4

B: 2 3

C: 1

Chuyển 4 từ A->C:

A: rỗng

B: 2 3

C: 1 4

Chuyển 2 từ B->C:

A: rỗng

B: 3

C: 1 4 2

Chuyển 3 từ B->C:

A: rỗng

B: rỗng

C: 1 4 2 3