

# Lab 2

## Mục Lục

|   |    |
|---|----|
| PHẦN 1: DANH SÁCH LIÊN KẾT .....  | 3  |
| Cơ bản .....  | 3  |
| 1. Biên dịch đoạn chương trình nêu trên. ....   | 3  |
| 2. Cho biết kết quả in ra màn hình:.....  | 3  |
| 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. ....   | 3  |
| 4. Vẽ hình danh sách liên kết theo dữ liệu được nhập ở câu 2. ....  | 3  |
| 5. Viết hàm RemoveAll xóa tất cả các phần tử trong DSLK.....  | 4  |
| Áp dụng – Nâng cao.....   | 4  |
| 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.....   | 4  |
| 2. Bổ sung chương trình mẫu cho phép tìm <b>giá trị nguyên lớn nhất</b> 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 <b>MaxList</b> .....  | 4  |
| 3. Bổ sung chương trình mẫu cho phép tính <b>số lượng các phần tử là số nguyên tố</b> của danh sách liên kết đơn gồm các giá trị nguyên. Gợi ý: tham khảo hàm Travel để viết hàm CountPrime.....  | 5  |
| 4. Bổ sung chương trình mẫu cho phép <b>thêm vào cuối</b> danh sách liên kết đơn một giá trị nguyên. Gợi ý: tham khảo hàm InsertFirst để viết hàm InsertTail.....   | 5  |
| 5. Bổ sung chương trình mẫu cho phép <b>thêm phần tử vào sau p(tham số truyền vào là 1 con trỏ)</b> danh sách liên kết đơn một giá trị nguyên.....  | 6  |
| 6. Bổ sung chương trình mẫu cho phép <b>xóa phần tử đầu</b> danh sách liên kết đơn.....   | 6  |
| 7. Bổ sung chương trình mẫu cho phép <b>xóa phần tử cuối</b> danh sách liên kết đơn.....  | 7  |
| 8. Bổ sung chương trình mẫu cho phép <b>xóa phần tử p(tham số truyền vào là 1 con trỏ) ở vị trí bất kỳ</b> danh sách liên kết đơn. ....   | 7  |
| 9. Bổ sung chương trình mẫu cho biết <b>số lượng các phần tử</b> trên danh sách liên kết đơn có giá trị trùng với giá trị <b>x</b> được cho trước. Gợi ý: tham khảo thao tác duyệt danh sách liên kết trong hàm <b>Travel</b> . ....  | 8  |
| 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. ....  | 8  |
| 11. Cho sẵn một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên và một giá trị nguyên <b>x</b> . 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 <b>x</b> và một danh sách gồm các phần tử có giá trị lớn hơn giá trị <b>x</b> ..... | 9  |
| BÀI TẬP ỨNG DỤNG .....  | 9  |
| Bài 1.....  | 9  |
| Bài 2.....  | 14 |
| PHẦN 2: STACK – QUEUE.....  | 21 |

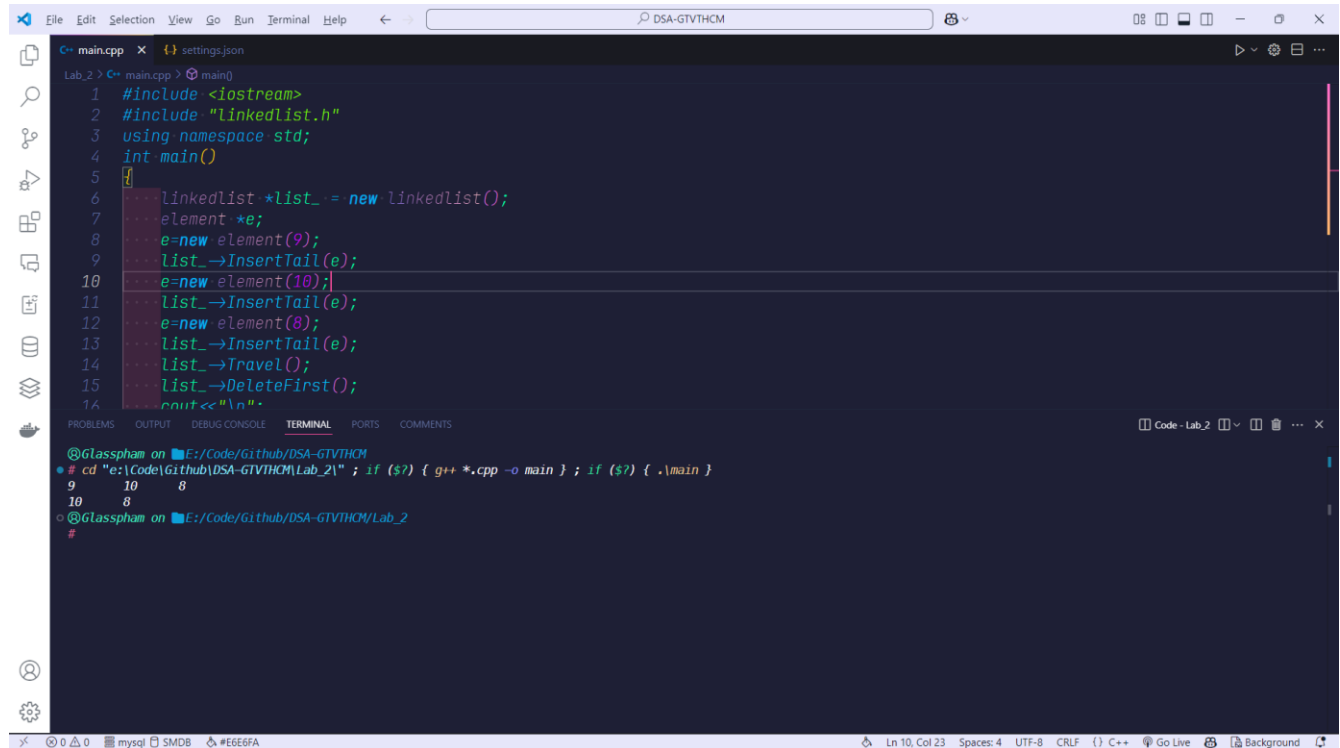
|   |    |
|---|----|
| Cơ bản .....  | 21 |
| Áp dụng – Nâng cao.....   | 26 |
| 1. Sử dụng hàm InsertTail và DeleteTail trong LinkedList để có phiên bản cài đặt Stack (thêm phần tử vào cuối danh sách và lấy phần tử ở cuối danh sách liên kết) cũng như áp dụng 1 phiên bản khác khi cài đặt Queue (thêm phần tử vào cuối danh sách liên kết và lấy phần tử ở đầu danh sách liên kết)..... | 26 |
| 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. ....   | 30 |
| 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). .... | 30 |
| BÀI TẬP ỨNG DỤNG .....  | 33 |
| Bài 1: Tìm đường trong mê cung (thực hiện loang theo chiều rộng <sử dụng queue> hoặc loang theo chiều sâu <sử dụng stack>). ....  | 33 |
| Bài 2: .....  | 38 |
| Bài 3: Tương tự yêu cầu bài 2 nhưng với hình bên dưới:.....   | 41 |

## PHẦN 1: DANH SÁCH LIÊN KẾT

Cơ bản

Yêu cầu:

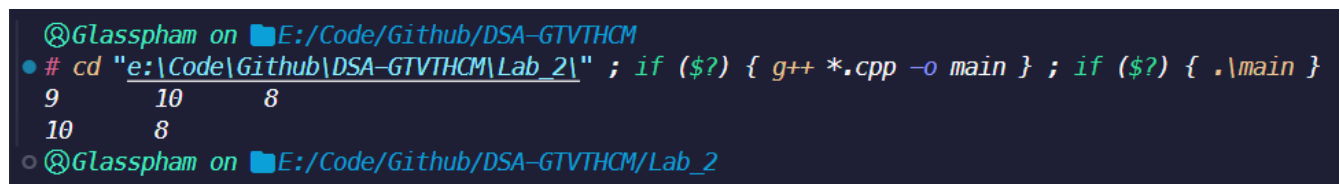
1. Biên dịch đoạn chương trình nêu trên.



```
1 #include <iostream>
2 #include "linkedList.h"
3 using namespace std;
4 int main()
5 {
6     linkedlist *list_ = new linkedlist();
7     element *e;
8     e=new element(9);
9     list_→InsertTail(e);
10    e=new element(10);
11    list_→InsertTail(e);
12    e=new element(8);
13    list_→InsertTail(e);
14    list_→Travel();
15    list_→DeleteFirst();
16    cout<<"\n";
17 }
```

```
@Glasspham on E:/Code/Github/DSA-GTVTHCM
# cd "E:/Code/Github/DSA-GTVTHCM/Lab_2\" ; if ($?) { g++ *.cpp -o main } ; if ($?) { .\main }
9      10      8
10      8
@Glasspham on E:/Code/Github/DSA-GTVTHCM/Lab_2
#
```

2. Cho biết kết quả in ra màn hình:



```
@Glasspham on E:/Code/Github/DSA-GTVTHCM
# cd "E:/Code/Github/DSA-GTVTHCM/Lab_2\" ; if ($?) { g++ *.cpp -o main } ; if ($?) { .\main }
9      10      8
10      8
@Glasspham on E:/Code/Github/DSA-GTVTHCM/Lab_2
#
```

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 nhập vào danh sách liên kết theo phương thức **InsertTail()**, tức là thêm phần tử mới vào cuối danh sách. Điều này dẫn đến thứ tự các phần tử trong danh sách khi duyệt (hàm Travel()) sẽ **giữ nguyên thứ tự như khi nhập**.

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



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

```
void linkedlist::RemoveAll() {  
    element* p = head;  
    while (p != nullptr) {  
        element* temp = p;  
        p = p->Getpointer(); // Di chuyển đến phần tử kế tiếp  
        delete temp;        // Xóa phần tử hiện tại  
    }  
    head = tail = nullptr; // Đặt danh sách về rỗng  
    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*.

```
int linkedlist::SumList() {  
    int sum = 0;  
    element* p = head;  
    while (p != nullptr) {  
        sum += p->Getdata(); // Cộng giá trị của từng phần tử vào tổng  
        p = p->Getpointer(); // Di chuyển đến phần tử kế tiếp  
    }  
    return sum;  
}
```

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***.

```
int linkedlist::MaxList() {  
    if (head == nullptr) {  
        cout << "List Empty!\n";  
        return -1; // Trả về giá trị nhỏ nhất nếu danh sách rỗng  
    }  
    int maxVal = head->Getdata(); // Giả sử phần tử đầu tiên là lớn nhất
```

```

element* p = head->Getpointer();

while (p != nullptr) {
    if (p->Getdata() > maxVal)
        maxVal = p->Getdata(); // Cập nhật giá trị lớn nhất
    p = p->Getpointer(); // Di chuyển đến phần tử tiếp theo
}

return maxVal;
}

```

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.

```

bool isPrime(int n) {
    if (n < 2) return false;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) return false;
    return true;
}

int linkedlist::CountPrime() {
    int count = 0;
    element* p = head;
    while (p != nullptr) {
        if (isPrime(p->Getdata())) count++;
        p = p->Getpointer();
    }
    return count;
}

```

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.

```

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++;
}

```

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.

```

void linkedlist::insertAfter(element *e, element *p) {
    element *tmp = this->head;
    while (tmp != nullptr) {
        if (tmp->Getdata() == e->Getdata()) {
            p->Setpointer(tmp->Getpointer());
            tmp->Setpointer(p);
            nNum++;
            return;
        }
        tmp = tmp->Getpointer();
    }
}

```

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

```

bool linkedlist::DeleteFirst() {
    if (this->head == nullptr) return false;
    else {
        element *p = this->head;
        this->head = this->head->Getpointer();
        delete p;
        --nNum;
        return true;
    }
}

```

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

```
bool linkedlist::DeleteLast() {  
    if (this->head == nullptr) return false;  
    if (this->head == this->tail) {  
        delete this->head;  
        this->head = this->tail = nullptr;  
        nNum = 0;  
        return true;  
    }  
    element *p = this->head;  
    while (p->Getpointer() != this->tail) p = p->Getpointer();  
    delete this->tail;  
    this->tail = p;  
    this->tail->Setpointer(nullptr);  
    nNum--;  
    return true;  
}
```

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.

```
bool linkedlist::DeleteAfter(element *e) {  
    element *p = this->head;  
    while (p != nullptr) {  
        if (p->Getdata() == e->Getdata()) {  
            element *temp = p->Getpointer();  
            p->Setpointer(temp->Getpointer());  
            delete temp;  
            --nNum;  
            return true;  
        }  
        p = p->Getpointer();  
    }
```

```

    }

    return false;
}

```

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**.

```

int linkedlist::CountX(int x) {

    int count = 0;

    element* p = this->head;

    while (p != nullptr) {

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

            count++;

        p = p->Getpointer();

    }

    return count;

}

```

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.

```

void linkedlist::InsertFirstUnique(element *e) {

    element *p = head;

    while (p != nullptr) {

        if (p->Getdata() == e->Getdata()) {

            cout << "Element already exists!\n";

            return;

        }

        p = p->Getpointer();

    }

    InsertFirst(e);

}

```



```

void linkedlist::InsertTailUnique(element *e) {
    element *p = head;
    while (p != nullptr) {
        if (p->Getdata() == e->Getdata()) {
            cout << "Element already exists!\n";
            return;
        }
        p = p->Getpointer();
    }
    InsertTail(e);
}

```

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  $x$ .

```

void linkedlist::splitList(int x, linkedlist &lessList, linkedlist &greaterList) {
    element *p = this->head;
    while (p != nullptr) {
        element *newNode = new element(p->Getdata());
        if (p->Getdata() < x) lessList.InsertTail(newNode);
        else greaterList.InsertTail(newNode);
        p = p->Getpointer();
    }
}

```

## BÀI TẬP ỨNG DỤNG

### Bài 1.

Đề xuất cấu trúc dữ liệu thích hợp để biểu diễn đa thức  $(a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0)$  bằng danh sách liên kết (đơn và kép). Cài đặt các thao tác trên danh sách liên kết đơn biểu diễn đa thức:

- In đa thức
- Tính giá trị đa thức (với giá trị  $x$  nhập vào)
- Cộng hai đa thức
- Nhân hai đa thức

// File: Polynomial.h

```
#pragma once

#ifndef _POLYNOMIAL_H_
#define _POLYNOMIAL_H_

class Term {
public:
    int coeff; // Hệ số
    int exp;   // Số mũ

    Term* next;

    Term(int c, int e) : coeff(c), exp(e), next(nullptr) {}
};

class Polynomial {
private:
    Term* head;
public:
    Polynomial();
    ~Polynomial();

    void insertTerm(int coeff, int exp);

    void printPolynomial();

    int evaluatePolynomial(int x);

    Polynomial addPolynomials(const Polynomial& p);

    Polynomial multiplyPolynomials(const Polynomial& p);
};

#endif // _POLYNOMIAL_H_
```

// File: Polynomial.cpp

```
#include "Polynomial.h"

#include <iostream>

#include <cmath>

using namespace std;

Polynomial::Polynomial() { head = nullptr; }
```

```

Polynomial::~Polynomial() {
    Term* temp;
    while (head) {
        temp = head;
        head = head->next;
        delete temp;
    }
}

void Polynomial::insertTerm(int coeff, int exp) {
    if (coeff == 0) return;
    Term* newNode = new Term(coeff, exp);
    if (!head || head->exp < exp) {
        newNode->next = head;
        head = newNode;
        return;
    }
    Term* temp = head;
    Term* prev = nullptr;
    while (temp && temp->exp > exp) {
        prev = temp;
        temp = temp->next;
    }
    if (temp && temp->exp == exp) {
        temp->coeff += coeff;
        if (temp->coeff == 0) { // Nếu hệ số bằng 0, xóa node
            if (prev) prev->next = temp->next;
            else head = temp->next;
            delete temp;
        }
    }
    return;
}

```

```

    }

    newNode->next = temp;
    if (prev) prev->next = newNode;
}

void Polynomial::printPolynomial() {
    if (!head) {
        cout << "0" << endl;
        return;
    }

    Term* temp = head;
    while (temp) {
        cout << temp->coeff;
        if(temp->exp == 1) cout << "x";
        else if(temp->exp != 0) cout << "x^" << temp->exp;
        if (temp->next) cout << " + ";
        temp = temp->next;
    }
    cout << endl;
}

int Polynomial::evaluatePolynomial(int x) {
    int result = 0;
    Term* temp = head;
    while (temp) {
        result += temp->coeff * pow(x, temp->exp);
        temp = temp->next;
    }
    return result;
}

Polynomial Polynomial::addPolynomials(const Polynomial& other) {
    Polynomial result;

```

```

Term* p1 = head;
Term* p2 = other.head;
while (p1) {
    result.insertTerm(p1->coeff, p1->exp);
    p1 = p1->next;
}
while (p2) {
    result.insertTerm(p2->coeff, p2->exp);
    p2 = p2->next;
}
return result;
}

Polynomial Polynomial::multiplyPolynomials(const Polynomial& other) {
    Polynomial result;
    for (Term* i = head; i; i = i->next)
        for (Term* j = other.head; j; j = j->next)
            result.insertTerm(i->coeff * j->coeff, i->exp + j->exp);
    return result;
}

```

// File: main.cpp

```

#include "Polynomial.h"
#include <iostream>
using namespace std;
int main() {
    Polynomial poly1, poly2;
    // Nhập đa thức 1
    poly1.insertTerm(3, 2);
    poly1.insertTerm(5, 3);
    poly1.insertTerm(2, 0);
    cout << "Đa thức 1: ";
}

```

```

poly1.printPolynomial();

// Nhập đa thức 2
poly2.insertTerm(4, 1);
poly2.insertTerm(1, 0);
cout << "Đa thức 2: ";
poly2.printPolynomial();

// Cộng hai đa thức
Polynomial sum = poly1.addPolynomials(poly2);
cout << "Tổng hai đa thức: ";
sum.printPolynomial();

// Nhân hai đa thức
Polynomial product = poly1.multiplyPolynomials(poly2);
cout << "Tích hai đa thức: ";
product.printPolynomial();

// Tính giá trị đa thức tại x = 2
cout << "Giá trị đa thức 1 tại x = " << 2 << " là: " << poly1.evaluatePolynomial(2) << endl;

return 0;
}

```

## Bài 2.

Thông tin của một quyền 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 quyền sách có trong thư viện (được nhập từ bàn phím).
  - b. Cho biết số lượng các quyền 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 quyền sách nào.

```
// File: Library.h
```

```
#pragma once
```

```

#ifndef _LIBRARY_H_
#define _LIBRARY_H_
#include <iostream>
#include <string>
using namespace std;
const int MAX_AUTHORS = 5;
struct Book {
    string title;
    string authors[MAX_AUTHORS];
    int authorCount;
    string publisher;
    int year;
    Book* next;
    Book(string t, string p, int y) : title(t), publisher(p), year(y), authorCount(0), next(nullptr) {}
};
class Library {
private:
    Book* head;
public:
    Library();
    ~Library();
    void addBook();
    int countBooksByAuthor(string author);
    void findBooksByPublisherAndYear(string publisher, int year);
    void displayBooks();
};
#endif // _LIBRARY_H_
// File: Library.cpp
#include "Library.h"
Library::Library() { head = nullptr; }

```

```

Library::~~Library() {
    while (head) {
        Book* temp = head;
        head = head->next;
        delete temp;
    }
}

void Library::addBook() {
    string title, publisher;
    int year, numAuthors;
    cin.ignore();
    cout << "Nhap ten sach: ";
    getline(cin, title);
    cout << "Nhap so luong tac gia (toi da 5): ";
    cin >> numAuthors;
    cin.ignore();
    if (numAuthors > MAX_AUTHORS) numAuthors = MAX_AUTHORS;
    string authors[MAX_AUTHORS];
    for (int i = 0; i < numAuthors; i++) {
        cout << "Nhap ten tac gia " << i + 1 << ": ";
        getline(cin, authors[i]);
    }
    cout << "Nhap nha xuất bản: ";
    getline(cin, publisher);
    cout << "Nhap năm xuất bản: ";
    cin >> year;

    Book* newBook = new Book(title, publisher, year);
    newBook->authorCount = numAuthors;
    for (int i = 0; i < numAuthors; i++) newBook->authors[i] = authors[i];
    newBook->next = head;
}

```



```

    head = newBook;
}

int Library::countBooksByAuthor(string author) {
    int count = 0;
    Book* temp = head;
    while (temp) {
        for (int i = 0; i < temp->authorCount; i++) {
            if (temp->authors[i] == author) {
                count++;
                break;
            }
        }
        temp = temp->next;
    }
    return count;
}

void Library::findBooksByPublisherAndYear(string publisher, int year) {
    Book* temp = head;
    bool found = false;
    while (temp) {
        if (temp->publisher == publisher && temp->year == year) {
            cout << "- " << temp->title << endl;
            found = true;
        }
        temp = temp->next;
    }
    if (!found) cout << "Khong co sach nao duoc tim thay!" << endl;
}

void Library::displayBooks() {
    Book* temp = head;

```

```

while (temp) {
    cout << "Ten sach: " << temp->title << "\nTac gia: ";
    for (int i = 0; i < temp->authorCount; i++) {
        cout << temp->authors[i];
        if (i < temp->authorCount - 1) cout << ", ";
    }
    cout << "\nNha xuat ban: " << temp->publisher << "\nNam xuat ban: " << temp->year << endl;
    cout << "-----\n";
    temp = temp->next;
}
}

// File: main.cpp

#include "Library.h"

void menu() {
    cout << "\n===== MENU =====\n";
    cout << "1. Them sach\n";
    cout << "2. Hien thi sach\n";
    cout << "3. Dem so sach cua mot tac gia\n";
    cout << "4. Tim sach theo nam va NXB\n";
    cout << "0. Thoat\n";
    cout << "Lua chon: ";
}

int main() {
    Library lib;
    int choice;
    do {
        menu();
        cin >> choice;
        cout << "-----\n";
        switch (choice) {

```

case 1:

```
lib.addBook();
```

```
break;
```

case 2:

```
lib.displayBooks();
```

```
break;
```

case 3: {

```
cin.ignore();
```

```
cout << "Nhap ten tac gia: ";
```

```
string author; getline(cin, author);
```

```
cout << "So sach cua tac gia " << author << ": " << lib.countBooksByAuthor(author) << endl;
```

```
break;
```

```
}
```

case 4: {

```
cout << "Nhap nam xuất bản: ";
```

```
int year; cin >> year;
```

```
cin.ignore();
```

```
cout << "Nhap nhà xuất bản: ";
```

```
string publisher; getline(cin, publisher);
```

```
cout << "Các sách của NXB " << publisher << " trong năm " << year << ":\n";
```

```
lib.findBooksByPublisherAndYear(publisher, year);
```

```
break;
```

```
}
```

case 0:

```
cout << "Thoát chương trình...\n";
```

```
break;
```

default:

```
cout << "Lựa chọn không hợp lệ!\n";
```

```
break;
```

```
}
```

```
} while (choice != 0);  
return 0;  
}
```

## PHẦN 2: STACK – QUEUE

### Cơ bản

1. Sử dụng danh sách liên kết để cài đặt cấu trúc Stack, Queue.
2. Sử dụng các hàm PushStack, PopStack, EnQueue, DeQueue để cài đặt.
  - a. Về Stack: Trong hàm main, thực hiện việc thêm vào 3 giá trị do người dùng nhập vào (thực hiện 3 lệnh thêm phần tử vào stack), sau đó thực hiện 4 lần lệnh lấy giá trị phần tử ra khỏi stack, nếu có, in giá trị phần tử ra màn hình, nếu không có (stack rỗng), in ra màn hình “STACK RONG, KHONG LAY DUOC PHAN TU”.
  - b. Về Queue: Trong hàm main, thực hiện việc thêm vào 3 giá trị do người dùng nhập vào (thực hiện 3 lần lệnh thêm phần tử vào queue), sau đó thực hiện 4 lần lệnh lấy giá trị phần tử ra khỏi queue, nếu có, in giá trị phần tử ra màn hình, nếu không có (queue rỗng), in ra màn hình “QUEUE RONG, KHONG LAY DUOC PHAN TU”.

```
#include <iostream>

using namespace std;

// Cấu trúc một phần tử của danh sách liên kết
class Node {
public:
    int data;
    Node* next;

    Node(int x) : data(x), next(nullptr) {}
};

// Lớp danh sách liên kết
class LinkedList {
private:
    Node* head;
    Node* tail;
public:
    LinkedList() : head(nullptr), tail(nullptr) {}
    ~LinkedList() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
    }
};
```

```

    }
}

// Thêm phần tử vào đầu danh sách (dùng cho Stack)
void insertFirst(int x) {
    Node* newNode = new Node(x);
    if (!head) tail = newNode;
    newNode->next = head;
    head = newNode;
}

// Thêm phần tử vào cuối danh sách (dùng cho Queue)
void insertLast(int x) {
    Node* newNode = new Node(x);
    if (!head) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
}

// Xóa phần tử đầu danh sách (dùng cho Stack & Queue)
int deleteFirst() {
    if (!head) return -1; // Danh sách rỗng
    Node* temp = head;
    int value = temp->data;
    head = head->next;
    if (!head) tail = nullptr;
    delete temp;
    return value;
}

// Xóa phần tử cuối danh sách (dùng cho Queue)

```

```

int deleteLast() {
    if (!head) return -1; // Danh sách rỗng
    if (head == tail) { // Nếu chỉ có 1 phần tử
        int value = head->data;

        delete head;

        head = tail = nullptr;

        return value;
    }

    Node* temp = head;
    while (temp->next != tail) {
        temp = temp->next;
    }

    int value = tail->data;

    delete tail;

    tail = temp;
    tail->next = nullptr;

    return value;
}

// In danh sách
void printList() {
    Node* temp = head;

    while (temp) {
        cout << temp->data << " ";

        temp = temp->next;
    }

    cout << endl;
}

bool isEmpty() { return head == nullptr; }
};

```

// Lớp Stack (Ngăn xếp)

```
class Stack {
private:
    LinkedList list;
public:
    void push(int x) { list.insertFirst(x); }
    int pop() {
        int value = list.deleteFirst();
        if (value == -1) cout << "STACK RONG, KHONG LAY DUOC PHAN TU" << endl;
        return value;
    }
    void printStack() { list.printList(); }
};

// Lớp Queue (Hàng đợi)
class Queue {
private:
    LinkedList list;
public:
    void enqueue(int x) { list.insertLast(x); }
    int dequeue() {
        int value = list.deleteFirst();
        if (value == -1) cout << "QUEUE RONG, KHONG LAY DUOC PHAN TU" << endl;
        return value;
    }
    void printQueue() { list.printList(); }
};

// Hàm main để kiểm tra chương trình
int main() {
    Stack stack;
    Queue queue;
    cout << "Nhap 3 gia tri vao Stack: ";
```



```
for (int i = 0; i < 3; i++) {  
    int x;  
    cin >> x;  
    stack.push(x);  
}  
cout << "Stack sau khi them: ";  
stack.printStack();  
cout << "Pop 4 lan tu Stack:\n";  
for (int i = 0; i < 4; i++) {  
    int value = stack.pop();  
    if (value != -1) cout << "Lay ra: " << value << endl;  
}  
cout << "Nhap 3 gia tri vao Queue: ";  
for (int i = 0; i < 3; i++) {  
    int x;  
    cin >> x;  
    queue.enqueue(x);  
}  
cout << "Queue sau khi them: ";  
queue.printQueue();  
cout << "Dequeue 4 lan tu Queue:\n";  
for (int i = 0; i < 4; i++) {  
    int value = queue.dequeue();  
    if (value != -1) cout << "Lay ra: " << value << endl;  
}  
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).

```
#include <iostream>

using namespace std;

// Cấu trúc một phần tử của danh sách liên kết
class Node {
public:
    int data;
    Node* next;
    Node(int x) : data(x), next(nullptr) {}
};

// Lớp danh sách liên kết
class LinkedList {
private:
    Node* head;
    Node* tail;
public:
    LinkedList() : head(nullptr), tail(nullptr) {}

    ~LinkedList() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
    }

    // Thêm phần tử vào cuối danh sách (Insert Tail)
    void insertTail(int x) {
        Node* newNode = new Node(x);
```

```

    if (!head) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
}

// Xóa phần tử cuối danh sách (Delete Tail)
int deleteTail() {
    if (!head) return -1; // Danh sách rỗng
    if (head == tail) { // Nếu chỉ có 1 phần tử
        int value = head->data;
        delete head;
        head = tail = nullptr;
        return value;
    }

    Node* temp = head;
    while (temp->next != tail) {
        temp = temp->next;
    }

    int value = tail->data;
    delete tail;
    tail = temp;
    tail->next = nullptr;
    return value;
}

// Xóa phần tử đầu danh sách (Delete Head)
int deleteHead() {
    if (!head) return -1;
    Node* temp = head;

```

```

    int value = temp->data;

    head = head->next;

    if (!head) tail = nullptr;

    delete temp;

    return value;
}

// In danh sách
void printList() {
    Node* temp = head;

    while (temp) {
        cout << temp->data << " ";

        temp = temp->next;
    }

    cout << endl;
}

bool isEmpty() { return head == nullptr; }
};

// Lớp Stack (Ngăn xếp) - Cài đặt với InsertTail & DeleteTail
class Stack {
private:
    LinkedList list;
public:
    void push(int x) { list.insertTail(x); }

    int pop() {
        int value = list.deleteTail();

        if (value == -1) cout << "STACK RONG, KHONG LAY DUOC PHAN TU" << endl;

        return value;
    }

    void printStack() { list.printList(); }
};

```

*// Lớp Queue (Hàng đợi) - Cài đặt với InsertTail & DeleteHead*

```
class Queue {  
private:  
    LinkedList list;  
public:  
    void enqueue(int x) { list.insertTail(x); }  
    int dequeue() {  
        int value = list.deleteHead();  
        if (value == -1) cout << "QUEUE RONG, KHONG LAY DUOC PHAN TU" << endl;  
        return value;  
    }  
    void printQueue() { list.printList(); }  
};
```

*// Hàm main để kiểm tra chương trình*

```
int main() {  
    Stack stack;  
    Queue queue;  
    cout << "Nhap 3 gia tri vao Stack: ";  
    for (int i = 0; i < 3; i++) {  
        int x;  
        cin >> x;  
        stack.push(x);  
    }  
    cout << "Stack sau khi them: ";  
    stack.printStack();  
    cout << "Pop 4 lan tu Stack:\n";  
    for (int i = 0; i < 4; i++) {  
        int value = stack.pop();  
        if (value != -1) cout << "Lay ra: " << value << endl;  
    }  
}
```

```

cout << "Nhap 3 gia tri vao Queue: ";
for (int i = 0; i < 3; i++) {
    int x;
    cin >> x;
    queue.enqueue(x);
}
cout << "Queue sau khi them: ";
queue.printQueue();
cout << "Dequeue 4 lan tu Queue:\n";
for (int i = 0; i < 4; i++) {
    int value = queue.dequeue();
    if (value != -1) cout << "Lay ra: " << value << endl;
}
return 0;
}

```

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.

| Cách cài đặt                       | Stack (LIFO - Last In First Out)             | Queue (FIFO - First In First Out)     |
|------------------------------------|--|---------------------------------------|
| <b>Cách 1 – Chương trình mẫu</b>   | Nhanh hơn vì chèn/xóa đều ở đầu danh sách    | Nhanh hơn vì chèn ở cuối, xóa ở đầu   |
| <b>Cách 2 - Áp dụng – Nâng cao</b> | Chậm hơn vì phải duyệt danh sách để xóa cuối | Tương tự cách 1, hiệu suất ngang nhau |

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).

```

#include <iostream>

using namespace std;

class Node {
public:
    int data;

```

```
Node* next;

Node(int value) : data(value), next(nullptr) {}

};

class Stack {
private:
    Node* top;
public:
    Stack() : top(nullptr) {}

    ~Stack() {
        while (!isEmpty()) {
            pop();
        }
    }

    void push(int value) {
        Node* newNode = new Node(value);
        newNode->next = top;
        top = newNode;
    }

    void pop() {
        if (isEmpty()) return;
        Node* temp = top;
        top = top->next;
        delete temp;
    }

    int peek() {
        if (isEmpty()) throw runtime_error("Stack is empty");
        return top->data;
    }

    bool isEmpty() {
```

```
        return top == nullptr;
    }
};

void convertToBinary(int num) {
    Stack stack;
    if (num == 0) {
        cout << "0";
        return;
    }
    while (num > 0) {
        stack.push(num % 2);
        num /= 2;
    }
    while (!stack.isEmpty()) {
        cout << stack.peek();
        stack.pop();
    }
    cout << endl;
}

int main() {
    cout << "Nhap so can chuyen doi: ";
    int num; cin >> num;
    cout << "So nhi phan: ";
    convertToBinary(num);
    return 0;
}
```



## 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  $m \times n$ , mỗi phần tử là số 0 hoặc 1. Giá trị 1 : có thể đi tới và giá trị 0 : không thể đi tới được. 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.

```
#include <iostream>

#include <vector>

using namespace std;

// Định nghĩa 4 hướng di chuyển: Trái, Phải, Lên, Xuống
const int dx[4] = {-1, 1, 0, 0};
const int dy[4] = {0, 0, -1, 1};

int m, n;

vector<vector<int>>> maze;

// Class Node chung cho cả Stack và Queue
template<typename T>
class Node {
public:
    T x, y;
    Node* next;
    Node(T x, T y) : x(x), y(y), next(nullptr) {}
};

// Class Queue tự triển khai
template<typename T>
class Queue {
private:
    Node<T>* front;
    Node<T>* rear;
public:
    Queue() : front(nullptr), rear(nullptr) {}
    bool empty() { return front == nullptr; }
```

```

void push(T x, T y) {
    Node<T>* newNode = new Node<T>(x, y);
    if (rear) rear->next = newNode;
    else front = newNode;
    rear = newNode;
}

void pop() {
    if (front) {
        Node<T>* temp = front;
        front = front->next;
        if (!front) rear = nullptr;
        delete temp;
    }
}

Node<T>* peek() { return front; }
};

// Class Stack tự triển khai
template<typename T>
class Stack {
private:
    Node<T>* top;
public:
    Stack() : top(nullptr) {}
    bool empty() { return top == nullptr; }
    void push(T x, T y) {
        Node<T>* newNode = new Node<T>(x, y);
        newNode->next = top;
        top = newNode;
    }
    void pop() {

```

```

    if (top) {
        Node<T>* temp = top;

        top = top->next;

        delete temp;
    }
}

Node<T>* peek() { return top; }
};

// Hàm kiểm tra một ô có hợp lệ không
bool isValid(int x, int y, const vector<vector<bool>>& visited) {
    return (x >= 0 && x < m && y >= 0 && y < n && maze[x][y] == 1 && !visited[x][y]);
}

// BFS (dùng Queue)
bool bfsPathExists(int x1, int y1, int x2, int y2) {
    vector<vector<bool>> visited(m, vector<bool>(n, false));

    Queue<int> q;
    q.push(x1, y1);
    visited[x1][y1] = true;

    while (!q.empty()) {
        Node<int>* current = q.peek();

        int x = current->x, y = current->y;

        q.pop();

        if (x == x2 && y == y2) return true;

        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i], ny = y + dy[i];

            if (isValid(nx, ny, visited)) {
                visited[nx][ny] = true;

                q.push(nx, ny);
            }
        }
    }
}

```

```

    }
}

return false;
}

// DFS (dùng Stack)
bool dfsPathExists(int x1, int y1, int x2, int y2) {
    vector<vector<bool>> visited(m, vector<bool>(n, false));

    Stack<int> s;

    s.push(x1, y1);

    visited[x1][y1] = true;

    while (!s.empty()) {
        Node<int>* current = s.peek();

        int x = current->x, y = current->y;

        s.pop();

        if (x == x2 && y == y2) return true;

        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i], ny = y + dy[i];

            if (isValid(nx, ny, visited)) {
                visited[nx][ny] = true;

                s.push(nx, ny);
            }
        }
    }

    return false;
}

// Hàm in mê cung
void printMaze(const vector<vector<int>>& maze) {
    for (const auto& row : maze) {
        for (int cell : row)
            cout << cell << " ";
    }
}

```

```

        cout << endl;
    }
}

// Hàm main
int main() {
    cout << "Nhap kích thước ma trận (m n): ";
    cin >> m >> n;
    maze.resize(m, vector<int>(n));

    cout << "Nhap ma trận (0: không đi được, 1: có thể đi):\n";
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            cin >> maze[i][j];

    cout << "Ma trận vừa nhập:\n";
    printMaze(maze);
    int x1, y1, x2, y2;
    cout << "Nhap tọa độ bắt đầu (x1 y1): ";
    cin >> x1 >> y1;
    cout << "Nhap tọa độ đích (x2 y2): ";
    cin >> x2 >> y2;
    if (maze[x1][y1] == 0 || maze[x2][y2] == 0) {
        cout << "Không thể đi từ vị trí bắt đầu hoặc đích do là tường.\n";
        return 0;
    }
    cout << "Kiểm tra bằng BFS (Queue): ";
    if (bfsPathExists(x1, y1, x2, y2))
        cout << "CÓ đường đi.\n";
    else cout << "KHÔNG có đường đi.\n";
    cout << "Kiểm tra bằng DFS (Stack): ";
    if (dfsPathExists(x1, y1, x2, y2))
        cout << "CÓ đường đi.\n";
}

```

```

else cout << "KHONG co duong di.\n";

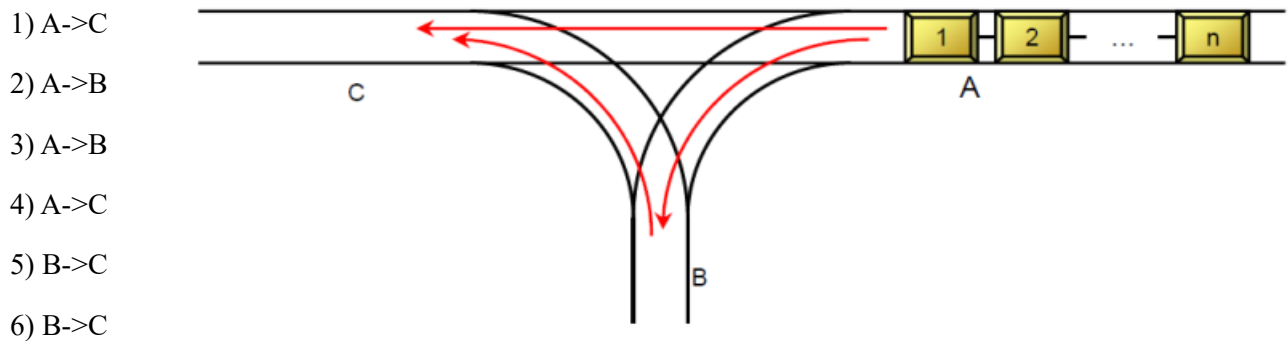
return 0;
}

```

## 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.

Ví dụ: n = 4 và thứ tự cần có (1, 4, 3, 2)



```

#include <iostream>
#include <vector>
using namespace std;
// Tự cài đặt ngăn xếp (stack)
template <typename T>
class Node {
public:
    T data;
    Node* next;
    Node(T val) : data(val), next(nullptr) {}
};
template <typename T>
class Stack {
private:
    Node<T>* topNode;
    int size;
public:

```

```

Stack() : topNode(nullptr), size(0) {}

bool empty() { return size == 0; }

void push(T val) {
    Node<T>* newNode = new Node<T>(val);
    newNode->next = topNode;
    topNode = newNode;
    size++;
}

void pop() {
    if (empty()) return;
    Node<T>* temp = topNode;
    topNode = topNode->next;
    delete temp;
    size--;
}

T top() {
    if (!empty()) return topNode->data;
    throw runtime_error("Stack is empty");
}
};

// Hàm kiểm tra xem có thể đạt được thứ tự mong muốn tại C hay không
bool solveRailwayShunting(int n, vector<int>& targetOrder) {
    Stack<int> A, B, C;
    for (int i = n; i >= 1; i--) A.push(i);
    int targetIndex = 0;
    vector<string> steps;
    while (!A.empty() || !B.empty()) {
        if (!B.empty() && B.top() == targetOrder[targetIndex]) {
            C.push(B.top()); B.pop();
            steps.push_back("B->C");
        }
    }
}

```

```

        targetIndex++;
    }
    else if (!A.empty() && A.top() == targetOrder[targetIndex]) {
        C.push(A.top()); A.pop();
        steps.push_back("A->C");
        targetIndex++;
    }
    else if (!A.empty()) {
        B.push(A.top()); A.pop();
        steps.push_back("A->B");
    }
    else {
        cout << "Khong the sap xep theo thu tu mong muon!\n";
        return false;
    }
}

cout << "Co the sap xep! Cac buoc di chuyen la:\n";
for (const string& step : steps)
    cout << step << endl;
return true;
}

int main() {
    cout << "Nhap so toa tau: ";
    int n; cin >> n;
    vector<int> targetOrder(n);

    cout << "Nhap thu tu mong muon tai C: ";
    for (int i = 0; i < n; i++) cin >> targetOrder[i];

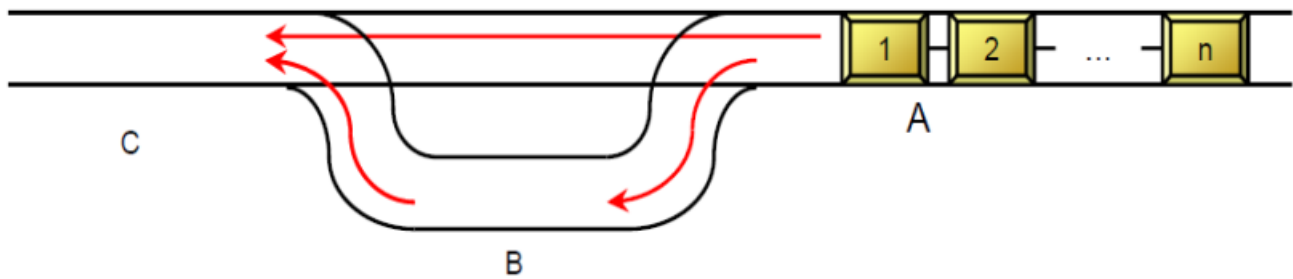
    solveRailwayShunting(n, targetOrder);

    return 0;
}

```



Bài 3: Tương tự yêu cầu bài 2 nhưng với hình bên dưới:



```
#include <iostream>

#include <vector>

using namespace std;

// Tự cài đặt hàng đợi (queue)

template <typename T>

class Node {

public:

    T data;

    Node* next;

    Node(T val) : data(val), next(nullptr) {}

};

template <typename T>

class Queue {

private:

    Node<T>* frontNode;

    Node<T>* rearNode;

    int size;

public:

    Queue() : frontNode(nullptr), rearNode(nullptr), size(0) {}

    bool empty() { return size == 0; }

    void push(T val) {

        Node<T>* newNode = new Node<T>(val);

        if (rearNode) rearNode->next = newNode;
```

```

    rearNode = newNode;

    if (!frontNode) frontNode = newNode;

    size++;
}

void pop() {
    if (empty()) return;

    Node<T>* temp = frontNode;

    frontNode = frontNode->next;

    if (!frontNode) rearNode = nullptr;

    delete temp;

    size--;
}

T front() {
    if (!empty()) return frontNode->data;

    throw runtime_error("Queue is empty");
}
};

// Hàm kiểm tra xem có thể đạt được thứ tự mong muốn tại C hay không
bool solveRailwayReordering(int n, vector<int>& targetOrder) {
    Queue<int> A, B, C;

    for (int i = 1; i <= n; i++) A.push(i);

    int targetIndex = 0;

    vector<string> steps;

    while (!A.empty() || !B.empty()) {
        if (!B.empty() && B.front() == targetOrder[targetIndex]) {
            C.push(B.front()); B.pop();

            steps.push_back("B->C");

            targetIndex++;
        }

        else if (!A.empty() && A.front() == targetOrder[targetIndex]) {

```

```

        C.push(A.front()); A.pop();

        steps.push_back("A->C");

        targetIndex++;
    }

    else if (!A.empty()) {
        B.push(A.front()); A.pop();

        steps.push_back("A->B");
    }

    else {
        cout << "Khong the sap xep theo thu tu mong muon!\n";

        return false;
    }
}

cout << "Co the sap xep! Cac buoc di chuyen la:\n";
for (const string& step : steps)
    cout << step << endl;

return true;
}

int main() {
    cout << "Nhap so toa tau: ";

    int n; cin >> n;

    vector<int> targetOrder(n);

    cout << "Nhap thu tu mong muon tai C: ";

    for (int i = 0; i < n; i++) cin >> targetOrder[i];

    solveRailwayReordering(n, targetOrder);

    return 0;
}

```