## I. Ngăn xếp và hàng đợi

Ta đã tìm hiểu các thao tác cơ bản trên DSLK và cách chúng tổ chức dữ liệu như thế nào với các hoạt động xin cấp phép bộ nhớ và xóa vùng nhớ. Ở phần này, ta sẽ tìm hiểu rõ hơn về cách lưu trữ dữ liệu và truy xuất dữ liệu thông qua hai cấu trúc dữ liệu là ngăn xếp (Stacks) và hàng đợi (Queues), đây là hai cấu trúc dữ liệu đặc biệt của danh sách tuyến tính, được dùng để lưu trữ và quản lí dữ liệu.

### 1. Stacks

#### 1.1. Định nghĩa

Stack (ngăn xếp) là cấu trúc dữ liệu trừu tượng (Abstract Data Type – ADT) được dùng để lưu trữ và quản lí dữ liệu. Stack còn thể hiểu theo một thuật ngữ khác là vùng chứa (Containers), nơi các đối tượng làm việc theo cơ chế LIFO (Last in – First out) hoặc FILO (First in – Last out), nghĩa là việc thêm một đối tượng vào stack hoặc lấy ra khỏi stack sẽ được thực hiện theo cơ chế “Vào sau ra trước”. Một ví dụ điển hỉnh đó là hình ảnh xếp các chiếc đĩa lên nhau thành một chồng, và chiếc đĩa nào càng được đặt vào sau cùng sẽ càng được lấy ra trước.

**Hình 1.1** Ví dụ minh họa về Stacks

Các đối tượng có thể được thêm vào stack bất kì lúc nào những chỉ có đối tượng vào sau cùng mới được phép lấy ra trước khỏi stack.

Hai cơ chế điển hình của stack đó là “Push” và “Pop”, trong đó:

* “Push” là cơ chế thêm/ chèn một dối tượng vào stack. Nếu stack đầy sẽ xảy ra lỗi, còn được gọi là *stack overflow.*
* “Pop” là cơ chế bớt/ xóa một dối tượng khỏi stack. Nếu stack rỗng thì lỗi sẽ xảy ra, trường hợp này đc gọi là *stack* *underflow*.

Push Pop đỉnh

đáy

**Hình 1.2** Hình ảnh mô tả hai cơ chế Push và Pop trong stack

|  |
| --- |
|  |
| **D** |
| **C** |
| **B** |
| **A** |

|  |
| --- |
|  |
|  |
| **C** |
| **B** |
| **A** |

*Pushing D Popping D*

|  |
| --- |
|  |
|  |
| **C** |
| **B** |
| **A** |

Top

Top

Top

**Hình 1.3** Mô hình hoạt động của Push và Pop

Cả hai cơ chế trên đều có xuất phát điểm là phần tử cuối cùng trong xâu (nếu xem đáy là top) và ngược lại xuất phát điểm là phần tử đầu tiên trong xấu (nếu xem đỉnh là top).

Ngoài ra stack cũng hỗ trợ các thao tác khác:

* isEmpty() : kiểm tra xem Stack có rỗng không
* Top(): trả về giá trị đầu tiên/ cuối cùng của danh sách mà không hủy nó khỏi stack.
* Size() : trả về số phần tử có trong Stacks.

#### 1.2. Ứng dụng

Stack có nhiều ứng dụng: khử đệ qui, tổ chức lưu vết các quá trình tìm kiếm theo chiều sâu và quay lui, vét cạn, ứng dụng trong các bài toán tính biểu thức,…

#### 1.3. Biểu diễn stack

**a) Biểu diễn stack qua mảng một chiều**

Ta có thể biểu diễn stack bằng cách khai báo một mảng một chiều kích thước tối đa là N. Như vậy stack có thể chứa tối đã N phần tử được đánh số bằng 0 tới N-1. Phần tử đầu tiên nằm đầu stack sẽ có chỉ số t (lúc đó trong stack có phần tử).

S0 S1 S2 S3 S(n-1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | ……….. |  |  |  |

đáy của stack đỉnh của stack

Dưới đây là cách cài đặt stack trong C++:

template<typename T>

class Stack {

private:

T arr[1000];

int top; //chỉ mục của phần tử đầu tiên trong stack

public:

Stack() : top(-1) {} //khởi tạo giá trị đầu của biến top

//điều này nghĩa là stack vừa được khai báo rỗng

bool isEmpty() {

return top == -1;

}

bool isFull()

{

return top == MAX\_SIZE - 1;

}

void push(const T& item)

{

if (isFull()) {

std::cout << "Stack overflow! Cannot push item." << std::endl;

return;

}

arr[++top] = item;

}

void pop()

{

if (isEmpty()) {

std::cout << "Stack is empty! Cannot pop item." << std::endl;

return;

}

top--;

}

T& peek()

{

if (isEmpty()) {

std::cout << "Stack is empty! Cannot peek item." << std::endl;

// Trả về một tham chiếu không an toàn

// Có thể cần xử lý tình huống này một cách an toàn hơn trong ứng dụng thực tế

return arr[top];

}

return arr[top];

}

};

Trong lớp public của Stack có các phương thức sau:

* isEmpty() : kiếm tra xem stack có rỗng hay không.
* isFull() : kiểm tra xem stack có đầy hay không.
* Push() : nếu stack đầy sẽ trả về kết quả “Stack overflow” , nếu không tăng biến top lên một đơn vị và lưu giá trị tại vị trí top.
* Pop() : nếu stack rỗng sẽ trả về kết quả “Stack underflow”, nếu không sẽ giảm biến top xuống 1 đơn vị và xóa giá trị đó khỏi stack.
* Peek() : trả về giá trị của phần tử đầu tiên mà không xóa nó khỏi stack, đồng thời nếu stack rỗng sẽ trả về giá trị ngẫu nhiên không cố định và thông báo lỗi.

**b) Biễu diễn stack bằng DSLK**

Stack cũng có thể được biểu diễn qua DSLK, cụ thể là DSLK đơn. Có thể nói DSLK có những đặc điểm thích hợp để dùng stack vì mọi thao tác trên stack đều được diễn ra tại đầu stack.

**Bảng 2.2.1** Các thao tác trên DSLK đơn tương ứng với Stack

|  |  |
| --- | --- |
| **Stack** | **DSLK đơn** |
| Push | InsertHead |
| Pop | DeleteHead |
| isEmpty | isEmpty |

### 2. Queues

#### 2.1. Định nghĩa

Queues (Hàng đợi) là cấu trúc dữ liệu trừu tượng (Abstract Data Type – ADT). Giống với Stacks, thì Queues cũng được xem là một container nhưng cơ chế hoạt động lại khác với Stack. Queues hoạt động theo cơ chế FIFO (First in - First out) hoặc là LILO (Last in – Last out), nghĩa là “Vào trước ra trước”. Các đối tượng có thể được thêm vào bất kì lúc nào những chỉ có phần tử được thêm vào đầu tiên mới được phép lấy ra.

Ví dụ điển hình cho Queues đó là việc xếp hàng để mua đồ hoặc làm gì đó có thứ tự. Ai đứng trước sẽ được tính hóa đơn trước hoặc được xử lý công việc trước. Danh sách các cuộc gọi cần được trả lời cũng là một hàng đợi, hoặc danh sách các công việc cần được xử lý theo thứ tự cũng là một hàng đợi.



**Hình 2.21** Ví dụ minh họa về Queues

Giống với Stack, ở Queues cũng có hai cơ chế điển hình đó là “Enqueue” và “Dequeue”:

* Enqueue() : thêm đối tượng vào cuối hàng đợi
* Dequeue() : lấy đối tượng ở đầu và xóa khỏi hàng đợi, đồng thời trả về giá trị của nó.

**Front**  **Back**

Dequeue Enqueue

**Queue of Data**

**Hình 2.22** Mô tả hai cơ chế Enqueue và Dequeue của Queues

Ngoài ra còn có các thao tác khác như:

* isEmpty() : kiểm tra hàng đợi có rỗng không.
* Front() : trả về giá trị của phần tử đầu tiên của hàng đợi mà không hủy nó. Nếu hàng đợi rỗng sẽ xảy ra lỗi.
* Size() : trả về số phần tử có trong Queues.

#### 2.4.2.2. Ứng dụng

Queues có nhiều ứng dụng: khử đệ qui, tổ chức lưu vết các quà trình tìm kiếm theo chiều rộng và quay lui, vét cạn, tổ chức quản lý và phân phối tiến trình trong cách hệ điều hành, tổ chức bộ đệm bàn phím,…

#### 2.4.2.3. Biểu diễn Queues

***a) Biểu diễn queue qua mảng một chiều***

Khai báo một mảng một chiều A có kích thước là N phần tử, và các phần tử được đánh số tử 0 dến . Việc quyết định phần tử nào là phần tử đầu tiên là do mỗi cá nhân người lập trình lựa chọn, song phải cài đặt đúng tương ứng với cách đã được chọn.

Đây là cách cài đặt trong C++ với phần tử đầu tiên của mảng là phần tử đầu tiên của Queue.

const int MAX\_SIZE = 1000;

template<typename T>

class Queue {

private:

int front, back, size;

int arr[MAX\_SIZE];

public:

Queue() {

front = 0;

back = -1;

size = 0;

}

bool isEmpty()

{

return size == 0;

}

bool isFull()

{

return size == MAX\_SIZE;

}

void enqueue(int item)

{

if (isFull()) {

std::cout << "Queue is full. Cannot enqueue.\n";

return;

}

back = (back + 1) % MAX\_SIZE;

arr[back] = item;

size++;

}

int dequeue()

{

if (isEmpty()) {

std::cout << "Queue is empty. Cannot dequeue.\n";

return -1;

}

int item = arr[front];

front = (front + 1) % MAX\_SIZE;

size--;

return item;

}

int front()

{

if (isEmpty()) {

std::cout << "Queue is empty. Cannot peek.\n";

return -1;

}

return arr[front];

}

};

Nhưng khác với ở Stack, vì khi thực hiện các thao tác thêm hoặc xóa các phần tử, ta chỉ tác động đến một phần đầu hoặc cuối của mảng, nghĩa là thêm/ xóa các phần tử cuối theo cơ chế LIFO. Dưới đây là minh họa khi thực hiện ở Queue.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** |  |  |

**(A)** Front Back

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** |  |

Front **(B)** Back

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **B** | **C** | **D** | **E** |  |

FrontBack **(C)**

**Hình 2.23** Mô hình hoạt động Enqueue và Dequeue

**A.** Một queue trước khi Enqueue và Dequeue

**B.** Sau khi Enqueue

**C.** Sau khi Dequeue

Tuy nhiên với cách tổ chức này thì sau một số lần thực hiện Enqueue và Dequeue thì Queue sẽ di chuyển khắp không gian nhớ. Do đó điều này đã được khắc phục bằng cách coi không gian nhớ dành cho queue được tổ chức theo kiểu vòng tròn. Giải thuật bổ sung và loại bỏ trên queue có sự thay đổi như sau

const int MAX\_SIZE = 1000;

template<typename T>

class Queue {

private:

int front, back, size;

int arr[MAX\_SIZE+1];

public:

Queue() {

front = 0;

back = 0;

size = 0;

}

bool isEmpty()

{

return size == 0;

}

bool isFull()

{

return size == MAX\_SIZE;

}

void enqueue(int item)

{

if (isFull()) {

std::cout << "Queue is full. Cannot enqueue.\n";

return;

}

back = (back + 1) % (MAX\_SIZE+1);

arr[back] = item;

size++;

}

int dequeue()

{

if (isEmpty()) {

std::cout << "Queue is empty. Cannot dequeue.\n";

return -1;

}

int item = arr[front];

front = (front + 1) % (MAX\_SZIE+1);

size--;

return item;

}

int peek()

{

if (isEmpty()) {

std::cout << "Queue is empty. Cannot peek.\n";

return -1;

}

return arr[front];

}

};

***b) biểu diễn Queue qua DSLK đơn***

Giống với Stack, Queue cũng có thể được biểu diễn qua DSLK đơn với các thao tác quen thuộc.

**Bảng 2.2** Các thao tác trên DSLK đơn tương ứng với Queue

|  |  |
| --- | --- |
| **Queues** | **DSLK đơn** |
| Enqueue | InsertTail |
| Dequeue | DeleteHead |
| isEmpty | isEmpty |

## II. BT Thực hành

### Bài 1.

#include <iostream>

#include <string>

#define MAX 100

class StackInt

{

private:

    int arr[MAX];

    int top;

public:

    StackInt()

    {

        top = -1;

    }

    void InitStack()

    {

        top = -1;

    }

    bool IsEmpty()

    {

        return top == -1;

    }

    bool IsFull()

    {

        return top == MAX - 1;

    }

    void Push(int value)

    {

        if (IsFull())

        {

            std::cout << "Stack is full!" << std::endl;

        }

        else

        {

            arr[++top] = value;

        }

    }

    int Pop()

    {

        if (IsEmpty())

        {

            std::cout << "Stack is empty!" << std::endl;

            return -1;

        }

        else

        {

            return arr[top--];

        }

    }

    int Peek()

    {

        if (IsEmpty())

        {

            std::cout << "Stack is empty!" << std::endl;

            return -1;

        }

        else

        {

            return arr[top];

        }

    }

    void Clear()

    {

        top = -1;

    }

};

class StackString

{

private:

    char \*arr;

    int top;

    int capacity;

public:

    StackString(int size)

    {

        capacity = size;

        arr = new char[capacity];

        top = -1;

    }

    ~StackString()

    {

        delete[] arr;

    }

    void InitStack()

    {

        top = -1;

    }

    bool IsEmpty()

    {

        return top == -1;

    }

    bool IsFull()

    {

        return top == capacity - 1;

    }

    void Push(char value)

    {

        if (IsFull())

        {

            std::cout << "Stack is full!" << std::endl;

        }

        else

        {

            arr[++top] = value;

        }

    }

    char Pop()

    {

        if (IsEmpty())

        {

            std::cout << "Stack is empty!" << std::endl;

            return '\0';

        }

        else

        {

            return arr[top--];

        }

    }

    char Peek()

    {

        if (IsEmpty())

        {

            std::cout << "Stack is empty!" << std::endl;

            return '\0';

        }

        else

        {

            return arr[top];

        }

    }

    void Clear()

    {

        top = -1;

    }

};

class Node

{

public:

    int data;

    Node \*next;

};

class LinkedStackInt

{

private:

    Node \*top;

public:

    LinkedStackInt()

    {

        top = nullptr;

    }

    void InitStack()

    {

        top = nullptr;

    }

    bool IsEmpty()

    {

        return top == nullptr;

    }

    void Push(int value)

    {

        Node \*newNode = new Node;

        newNode->data = value;

        newNode->next = top;

        top = newNode;

    }

    int Pop()

    {

        if (IsEmpty())

        {

            std::cout << "Stack is empty!" << std::endl;

            return -1;

        }

        else

        {

            Node \*temp = top;

            int value = top->data;

            top = top->next;

            delete temp;

            return value;

        }

    }

    int Peek()

    {

        if (IsEmpty())

        {

            std::cout << "Stack is empty!" << std::endl;

            return -1;

        }

        else

        {

            return top->data;

        }

    }

    void Clear()

    {

        while (!IsEmpty())

        {

            Pop();

        }

    }

};

void ReverseNumber(int num)

{

    StackInt stack;

    while (num > 0)

    {

        stack.Push(num % 10);

        num /= 10;

    }

    while (!stack.IsEmpty())

    {

        std::cout << stack.Pop();

    }

}

bool IsPalindrome(const std::string &str)

{

    StackString stack(str.length());

    for (char c : str)

    {

        stack.Push(c);

    }

    for (char c : str)

    {

        if (c != stack.Pop())

        {

            return false;

        }

    }

    return true;

}

void DecimalToBinary(int num)

{

    StackInt stack;

    while (num > 0)

    {

        stack.Push(num % 2);

        num /= 2;

    }

    while (!stack.IsEmpty())

    {

        std::cout << stack.Pop();

    }

}

bool isDigit(char c)

{

    return c >= '0' && c <= '9';

}

int precedence(char op)

{

    if (op == '+' || op == '-')

        return 1;

    if (op == '\*' || op == '/')

        return 2;

    return 0;

}

std::string InfixToPostfix(const std::string &infix)

{

    StackString opStack(infix.length());

    std::string postfix;

    for (char c : infix)

    {

        if (isDigit(c))

        {

            postfix += c;

        }

        else if (c == '(')

        {

            opStack.Push(c);

        }

        else if (c == ')')

        {

            while (!opStack.IsEmpty() && opStack.Peek() != '(')

            {

                postfix += opStack.Pop();

            }

            opStack.Pop(); // Remove '('

        }

        else if (c == '+' || c == '-' || c == '\*' || c == '/')

        {

            while (!opStack.IsEmpty() && precedence(opStack.Peek()) >= precedence(c))

            {

                postfix += opStack.Pop();

            }

            opStack.Push(c);

        }

    }

    while (!opStack.IsEmpty())

    {

        postfix += opStack.Pop();

    }

    return postfix;

}

int evaluatePostfix(const std::string &postfix)

{

    StackInt stack;

    for (char c : postfix)

    {

        if (isDigit(c))

        {

            stack.Push(c - '0');

        }

        else

        {

            int b = stack.Pop();

            int a = stack.Pop();

            int result = applyOperator(a, b, c);

            stack.Push(result);

        }

    }

    return stack.Pop();

}

int applyOperator(int a, int b, char op)

{

    switch (op)

    {

    case '+':

        return a + b;

    case '-':

        return a - b;

    case '\*':

        return a \* b;

    case '/':

        return a / b;

    default:

        return 0;

    }

}

int main()

{

    std::string infix;

    std::cout << "Enter an infix expression: ";

    std::cin >> infix;

    std::string postfix = InfixToPostfix(infix);

    std::cout << "Postfix expression: " << postfix << std::endl;

    int result = evaluatePostfix(postfix);

    std::cout << "Evaluation result: " << result << std::endl;

    int num = 12345;

    std::cout << "Reversed number: ";

    ReverseNumber(num);

    std::cout << std::endl;

    std::string str = "racecar";

    std::cout << "Is palindrome: " << (IsPalindrome(str) ? "Yes" : "No") << std::endl;

    num = 29;

    std::cout << "Binary of " << num << ": ";

    DecimalToBinary(num);

    std::cout << std::endl;

    return 0;

}

### Bài 2.

#include <iostream>

#include <string>

#include <cmath>

class QueueInt

{

private:

    int \*arr;

    int front;

    int rear;

    int size;

    int capacity;

public:

    QueueInt(int capacity)

    {

        this->capacity = capacity;

        arr = new int[capacity];

        front = 0;

        rear = -1;

        size = 0;

    }

    bool isEmpty()

    {

        return size == 0;

    }

    bool isFull()

    {

        return size == capacity;

    }

    void EnQueue(int value)

    {

        if (isFull())

        {

            std::cout << "Queue is Full!" << std::endl;

            return;

        }

        rear = (rear + 1) % capacity;

        arr[rear] = value;

        size++;

    }

    int DeQueue()

    {

        if (isEmpty())

        {

            std::cout << "Queue is Empty!" << std::endl;

            return -1;

        }

        int value = arr[front];

        front = (front + 1) % capacity;

        size--;

        return value;

    }

    int Peak()

    {

        if (isEmpty())

        {

            std::cout << "Queue is Empty!" << std::endl;

            return -1;

        }

        return arr[front];

    }

    void Clear()

    {

        front = 0;

        rear = -1;

        size = 0;

    }

    ~QueueInt()

    {

        delete[] arr;

    }

};

struct Node

{

    int data;

    Node \*next;

};

class LinkedQueueInt

{

private:

    Node \*front;

    Node \*rear;

public:

    LinkedQueueInt()

    {

        front = nullptr;

        rear = nullptr;

    }

    bool IsEmpty()

    {

        return front == nullptr;

    }

    void Enqueue(int value)

    {

        Node \*newNode = new Node();

        newNode->data = value;

        newNode->next = nullptr;

        if (IsEmpty())

        {

            front = rear = newNode;

        }

        else

        {

            rear->next = newNode;

            rear = newNode;

        }

    }

    int Dequeue()

    {

        if (IsEmpty())

        {

            std::cout << "Queue is empty!" << std::endl;

            return -1;

        }

        Node \*temp = front;

        int value = front->data;

        front = front->next;

        if (front == nullptr)

            rear = nullptr;

        delete temp;

        return value;

    }

    int Peek()

    {

        if (IsEmpty())

        {

            std::cout << "Queue is empty!" << std::endl;

            return -1;

        }

        return front->data;

    }

    void Clear()

    {

        while (!IsEmpty())

        {

            Dequeue();

        }

    }

    ~LinkedQueueInt()

    {

        Clear();

    }

};

struct Dancer

{

    std::string Name;

    std::string Sex;

};

class QueueDancer

{

private:

    Dancer \*arr;

    int front;

    int rear;

    int capacity;

    int size;

public:

    QueueDancer(int cap = 10)

    {

        capacity = cap;

        arr = new Dancer[capacity];

        front = 0;

        rear = -1;

        size = 0;

    }

    QueueDancer(const QueueDancer &other)

    {

        capacity = other.capacity;

        size = other.size;

        front = other.front;

        rear = other.rear;

        arr = new Dancer[capacity];

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

        {

            arr[i] = other.arr[i];

        }

    }

    bool IsEmpty()

    {

        return size == 0;

    }

    bool IsFull()

    {

        return size == capacity;

    }

    void Enqueue(Dancer value)

    {

        if (IsFull())

        {

            std::cout << "Queue is full!" << std::endl;

            return;

        }

        rear = (rear + 1) % capacity;

        arr[rear] = value;

        size++;

    }

    Dancer Dequeue()

    {

        if (IsEmpty())

        {

            std::cout << "Queue is empty!" << std::endl;

            return {"", ""};

        }

        Dancer value = arr[front];

        front = (front + 1) % capacity;

        size--;

        return value;

    }

    Dancer Peek()

    {

        if (IsEmpty())

        {

            std::cout << "Queue is empty!" << std::endl;

            return {"", ""};

        }

        return arr[front];

    }

    void Clear()

    {

        front = 0;

        rear = -1;

        size = 0;

    }

    int GetSize()

    {

        return size;

    }

    void ShowQueue()

    {

        if (IsEmpty())

        {

            std::cout << "Queue is empty!" << std::endl;

            return;

        }

        int index = front;

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

        {

            std::cout << arr[index].Name << std::endl;

            index = (index + 1) % capacity;

        }

    }

    ~QueueDancer()

    {

        delete[] arr;

    }

};

int getMaxDigits(int arr[], int n)

{

    int max = arr[0];

    for (int i = 1; i < n; i++)

    {

        if (arr[i] > max)

            max = arr[i];

    }

    return max == 0 ? 1 : static\_cast<int>(log10(max)) + 1;

}

int getDigit(int num, int pos)

{

    return (num / static\_cast<int>(pow(10, pos))) % 10;

}

void RadixSort(int arr[], int n)

{

    QueueInt \*queues[10];

    for (int i = 0; i < 10; i++)

    {

        queues[i] = new QueueInt(n);

    }

    int maxDigits = getMaxDigits(arr, n);

    for (int pos = 0; pos < maxDigits; pos++)

    {

        for (int i = 0; i < n; i++)

        {

            int digit = getDigit(arr[i], pos);

            queues[digit]->EnQueue(arr[i]);

        }

        int index = 0;

        for (int digit = 0; digit < 10; digit++)

        {

            while (!queues[digit]->isEmpty())

            {

                arr[index++] = queues[digit]->DeQueue();

            }

        }

    }

    for (int i = 0; i < 10; i++)

    {

        delete queues[i];

    }

}

void printArray(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        std::cout << arr[i] << " ";

    }

    std::cout << std::endl;

}

void NewDancers(QueueDancer &male, QueueDancer &female)

{

    if (!male.IsEmpty() && !female.IsEmpty())

    {

        Dancer m = male.Dequeue();

        Dancer w = female.Dequeue();

        std::cout << "Cap dien vien: " << m.Name << " va " << w.Name << std::endl;

    }

    else if (!male.IsEmpty() && female.IsEmpty())

    {

        std::cout << "Dang cho dien vien nu." << std::endl;

    }

    else if (!female.IsEmpty() && male.IsEmpty())

    {

        std::cout << "Dang cho dien vien nam." << std::endl;

    }

}

void HeadOfLine(QueueDancer &male, QueueDancer &female)

{

    if (!male.IsEmpty() && !female.IsEmpty())

    {

        Dancer m = male.Peek();

        Dancer w = female.Peek();

        std::cout << "Cap dien vien ke tiep: " << m.Name << "\t" << w.Name << std::endl;

    }

    else if (!male.IsEmpty())

    {

        Dancer m = male.Peek();

        std::cout << "Dien vien nam ke tiep: " << m.Name << std::endl;

    }

    else if (!female.IsEmpty())

    {

        Dancer w = female.Peek();

        std::cout << "Dien vien nu ke tiep: " << w.Name << std::endl;

    }

}

void StartDancing(QueueDancer &male, QueueDancer &female)

{

    std::cout << "Cac cap dien vien:" << std::endl;

    while (!male.IsEmpty() && !female.IsEmpty())

    {

        Dancer m = male.Dequeue();

        Dancer w = female.Dequeue();

        std::cout << m.Name << "\t" << w.Name << std::endl;

    }

    if (!male.IsEmpty())

    {

        std::cout << "Con " << male.GetSize() << " dien vien nam dang cho.\n";

    }

    if (!female.IsEmpty())

    {

        std::cout << "Con " << female.GetSize() << " dien vien nu dang cho.\n";

    }

}

void FormLines(QueueDancer &male, QueueDancer &female)

{

    std::string dancers[] = {"F Trang", "M Truc", "M Thien", "M Bao", "F Thu", "M Tien",

                             "F Thuy", "M Nghia", "F Thao", "M Phuoc", "M Hung", "F Vy"};

    int n = sizeof(dancers) / sizeof(dancers[0]);

    for (int i = 0; i < n; i++)

    {

        Dancer d;

        std::string person = dancers[i];

        d.Sex = person.substr(0, 1);

        d.Name = person.substr(2);

        if (d.Sex == "M")

        {

            male.Enqueue(d);

        }

        else

        {

            female.Enqueue(d);

        }

    }

}

int main()

{

    std::cout << "Testing QueueInt (Array-based Queue)\n";

    QueueInt intQueue(5);

    intQueue.EnQueue(10);

    intQueue.EnQueue(20);

    intQueue.EnQueue(30);

    std::cout << "Front element: " << intQueue.Peak() << std::endl;

    std::cout << "Dequeued: " << intQueue.DeQueue() << std::endl;

    std::cout << "Dequeued: " << intQueue.DeQueue() << std::endl;

    std::cout << "\nTesting LinkedQueueInt (Linked List-based Queue)\n";

    LinkedQueueInt linkedQueue;

    linkedQueue.Enqueue(100);

    linkedQueue.Enqueue(200);

    std::cout << "Front element: " << linkedQueue.Peek() << std::endl;

    std::cout << "Dequeued: " << linkedQueue.Dequeue() << std::endl;

    std::cout << "\nTesting Radix Sort with QueueInt\n";

    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};

    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Original array: ";

    printArray(arr, n);

    RadixSort(arr, n);

    std::cout << "Sorted array: ";

    printArray(arr, n);

    std::cout << "\nTesting QueueDancer (Pairing System)\n";

    QueueDancer maleQueue, femaleQueue;

    FormLines(maleQueue, femaleQueue);

    std::cout << "\n--- Male Dancers ---\n";

    maleQueue.ShowQueue();

    std::cout << "\n--- Female Dancers ---\n";

    femaleQueue.ShowQueue();

    std::cout << "\nPairing Dancers...\n";

    StartDancing(maleQueue, femaleQueue);

    return 0;

}

### III. BT Mở rộng

### Bài 1.

#include <iostream>

#include <string>

template <typename T>

class Stack

{

private:

    T \*arr;

    int capacity;

    int topIndex;

public:

    Stack(int size = 100)

    {

        capacity = size;

        arr = new T[capacity];

        topIndex = -1;

    }

    ~Stack()

    {

        delete[] arr;

    }

    bool isEmpty()

    {

        return topIndex == -1;

    }

    bool isFull()

    {

        return topIndex == capacity - 1;

    }

    void push(T value)

    {

        if (isFull())

        {

            std::cout << "Stack is full!\n";

            return;

        }

        arr[++topIndex] = value;

    }

    T pop()

    {

        if (isEmpty())

        {

            std::cout << "Stack is empty!\n";

            return T();

        }

        return arr[topIndex--];

    }

    T top()

    {

        if (isEmpty())

        {

            std::cout << "Stack is empty!\n";

            return T();

        }

        return arr[topIndex];

    }

};

long long FibonacciRecursive(int n)

{

    if (n <= 1)

        return n;

    return FibonacciRecursive(n - 1) + FibonacciRecursive(n - 2);

}

long long FibonacciIterativeWithStack(int n)

{

    if (n <= 1)

        return n;

    Stack<std::pair<int, long long>> stk;

    stk.push({n, 0});

    long long result = 0;

    while (!stk.isEmpty())

    {

        auto current = stk.pop();

        int currentN = current.first;

        long long value = current.second;

        if (currentN <= 1)

        {

            result += currentN;

            continue;

        }

        if (value == 0)

        {

            stk.push({currentN, 1});

            stk.push({currentN - 1, 0});

        }

        else

        {

            stk.push({currentN - 2, 0});

        }

    }

    return result;

}

int ReverseNumberRecursive(int num, int reversed = 0)

{

    if (num == 0)

        return reversed;

    int digit = num % 10;

    return ReverseNumberRecursive(num / 10, reversed \* 10 + digit);

}

int ReverseNumberWithStack(int num)

{

    Stack<int> digits;

    while (num > 0)

    {

        digits.push(num % 10);

        num /= 10;

    }

    int reversed = 0;

    int place = 1;

    while (!digits.isEmpty())

    {

        reversed += digits.top() \* place;

        digits.pop();

        place \*= 10;

    }

    return reversed;

}

void HanoiTowerRecursive(int n, char source, char auxiliary, char destination)

{

    if (n == 1)

    {

        std::cout << "Move disk 1 from " << source << " to " << destination << "\n";

        return;

    }

    HanoiTowerRecursive(n - 1, source, destination, auxiliary);

    std::cout << "Move disk " << n << " from " << source << " to " << destination << "\n";

    HanoiTowerRecursive(n - 1, auxiliary, source, destination);

}

void HanoiTowerWithStack(int n, char source, char auxiliary, char destination)

{

    struct Move

    {

        int numDisks;

        char src, aux, dest;

        int state;

    };

    Stack<Move> stk;

    stk.push({n, source, auxiliary, destination, 0});

    while (!stk.isEmpty())

    {

        Move current = stk.pop();

        if (current.numDisks == 1)

        {

            std::cout << "Move disk 1 from " << current.src << " to " << current.dest << "\n";

            continue;

        }

        if (current.state == 0)

        {

            stk.push({current.numDisks, current.src, current.aux, current.dest, 1});

            stk.push({current.numDisks - 1, current.src, current.dest, current.aux, 0});

        }

        else if (current.state == 1)

        {

            std::cout << "Move disk " << current.numDisks << " from " << current.src << " to " << current.dest << "\n";

            stk.push({current.numDisks, current.src, current.aux, current.dest, 2});

        }

        else

        {

            stk.push({current.numDisks - 1, current.aux, current.src, current.dest, 0});

        }

    }

}

int main()

{

    std::cout << "Testing Fibonacci\n";

    for (int i = 0; i <= 10; i++)

    {

        std::cout << "Fibonacci(" << i << ") = " << FibonacciRecursive(i) << " (Recursive)\n";

        std::cout << "Fibonacci(" << i << ") = " << FibonacciIterativeWithStack(i) << " (Stack)\n";

    }

    std::cout << "\nTesting Reverse Number\n";

    int numbers[] = {12345, 100, 54321, 0};

    for (int num : numbers)

    {

        std::cout << "Original: " << num << ", Reversed (Recursive): " << ReverseNumberRecursive(num) << "\n";

        std::cout << "Original: " << num << ", Reversed (Stack): " << ReverseNumberWithStack(num) << "\n";

    }

    std::cout << "\nTesting Hanoi Tower (Recursive) with 3 disks\n";

    HanoiTowerRecursive(3, 'A', 'B', 'C');

    std::cout << "\nTesting Hanoi Tower (Stack) with 3 disks\n";

    HanoiTowerWithStack(3, 'A', 'B', 'C');

    return 0;

}

### Bài 2.

#include <iostream>

#include <string>

#include <fstream>

template <typename T>

class Queue

{

private:

    T \*arr;

    int front;

    int rear;

    int size;

    int capacity;

public:

    Queue(int cap = 100)

    {

        capacity = cap;

        arr = new T[capacity];

        front = 0;

        rear = -1;

        size = 0;

    }

    ~Queue()

    {

        delete[] arr;

    }

    bool isEmpty()

    {

        return size == 0;

    }

    bool isFull()

    {

        return size == capacity;

    }

    void enqueue(T value)

    {

        if (isFull())

        {

            std::cout << "Queue is full!\n";

            return;

        }

        rear = (rear + 1) % capacity;

        arr[rear] = value;

        size++;

    }

    T dequeue()

    {

        if (isEmpty())

        {

            std::cout << "Queue is empty!\n";

            return T();

        }

        T value = arr[front];

        front = (front + 1) % capacity;

        size--;

        return value;

    }

};

struct Point

{

    int x, y;

};

void findPathInMaze(int \*\*maze, int N, int M, Point start, Point end)

{

    if (maze[start.x][start.y] == 1 || maze[end.x][end.y] == 1)

    {

        std::cout << "No path exists (start or end is blocked)!\n";

        return;

    }

    int dx[] = {0, 0, 1, -1};

    int dy[] = {1, -1, 0, 0};

    Queue<Point> q;

    bool visited[100][100] = {false};

    Point parent[100][100];

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            parent[i][j] = {-1, -1};

        }

    }

    q.enqueue(start);

    visited[start.x][start.y] = true;

    bool found = false;

    while (!q.isEmpty())

    {

        Point current = q.dequeue();

        if (current.x == end.x && current.y == end.y)

        {

            found = true;

            break;

        }

        for (int i = 0; i < 4; i++)

        {

            int newX = current.x + dx[i];

            int newY = current.y + dy[i];

            if (newX >= 0 && newX < N && newY >= 0 && newY < M && !visited[newX][newY] && maze[newX][newY] == 0)

            {

                q.enqueue({newX, newY});

                visited[newX][newY] = true;

                parent[newX][newY] = current;

            }

        }

    }

    if (!found)

    {

        std::cout << "No path exists!\n";

        return;

    }

    Point path[1000];

    int pathLength = 0;

    Point current = end;

    while (current.x != -1 && current.y != -1)

    {

        path[pathLength++] = current;

        current = parent[current.x][current.y];

    }

    std::cout << "Path from (" << start.x << "," << start.y << ") to (" << end.x << "," << end.y << "):\n";

    for (int i = pathLength - 1; i >= 0; i--)

    {

        std::cout << "(" << path[i].x << "," << path[i].y << ")";

        if (i > 0)

            std::cout << " -> ";

    }

    std::cout << "\n";

}

void findPathInGrid(int grid[5][5], int N, int M)

{

    Point start = {-1, -1}, end = {-1, -1};

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            if (grid[i][j] == 2)

                start = {i, j};

            if (grid[i][j] == 6)

                end = {i, j};

        }

    }

    if (start.x == -1 || end.x == -1)

    {

        std::cout << "Start or end point not found!\n";

        return;

    }

    int dx[] = {0, 0, 1, -1};

    int dy[] = {1, -1, 0, 0};

    Queue<Point> q;

    bool visited[5][5] = {false};

    Point parent[5][5];

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            parent[i][j] = {-1, -1};

        }

    }

    q.enqueue(start);

    visited[start.x][start.y] = true;

    bool found = false;

    while (!q.isEmpty())

    {

        Point current = q.dequeue();

        if (current.x == end.x && current.y == end.y)

        {

            found = true;

            break;

        }

        for (int i = 0; i < 4; i++)

        {

            int newX = current.x + dx[i];

            int newY = current.y + dy[i];

            if (newX >= 0 && newX < N && newY >= 0 && newY < M && !visited[newX][newY] && grid[newX][newY] != 1)

            {

                q.enqueue({newX, newY});

                visited[newX][newY] = true;

                parent[newX][newY] = current;

            }

        }

    }

    if (!found)

    {

        std::cout << "No path exists!\n";

        return;

    }

    Point path[100];

    int pathLength = 0;

    Point current = end;

    while (current.x != -1 && current.y != -1)

    {

        path[pathLength++] = current;

        current = parent[current.x][current.y];

    }

    std::cout << "Path from (" << start.x << "," << start.y << ") to (" << end.x << "," << end.y << "):\n";

    for (int i = pathLength - 1; i >= 0; i--)

    {

        std::cout << "(" << path[i].x << "," << path[i].y << ")";

        if (i > 0)

            std::cout << " -> ";

    }

    std::cout << "\n";

}

int main()

{

    std::cout << "Testing Maze (BFS with Queue)\n";

    std::ifstream mazeFile("maze.txt");

    if (!mazeFile)

    {

        std::cout << "Cannot open maze.txt!\n";

        return 1;

    }

    int N, M;

    mazeFile >> N >> M;

    int \*\*maze = new int \*[N];

    for (int i = 0; i < N; i++)

    {

        maze[i] = new int[M];

        for (int j = 0; j < M; j++)

        {

            mazeFile >> maze[i][j];

        }

    }

    Point start, end;

    mazeFile >> start.x >> start.y >> end.x >> end.y;

    mazeFile.close();

    findPathInMaze(maze, N, M, start, end);

    for (int i = 0; i < N; i++)

    {

        delete[] maze[i];

    }

    delete[] maze;

    std::cout << "\nTesting Grid Path (BFS with Queue)\n";

    std::ifstream gridFile("grid.txt");

    if (!gridFile)

    {

        std::cout << "Cannot open grid.txt!\n";

        return 1;

    }

    int grid[5][5];

    for (int i = 0; i < 5; i++)

    {

        for (int j = 0; j < 5; j++)

        {

            gridFile >> grid[i][j];

        }

    }

    gridFile.close();

    findPathInGrid(grid, 5, 5);

    return 0;

}