Họ và tên: Huỳnh Phạm Hoàng Sơn

MSSV: 3121411184

Mục Lục

[A.BÀI TẬP THỰC HÀNH 2](#_Toc194092052)

[BÀI 1 2](#_Toc194092053)

[GIẢI THÍCH CODE 2](#_Toc194092054)

[CÁC TEST CASE 3](#_Toc194092055)

[BÀI 2 6](#_Toc194092056)

[I.GIẢI THÍCH CODE 6](#_Toc194092057)

[II.CÁC TEST CASE 7](#_Toc194092058)

[B.BÀI TẬP MỞ RỘNG 10](#_Toc194092059)

[BÀI 1 10](#_Toc194092060)

[I.GIẢI THÍCH CODE 10](#_Toc194092061)

[II.CÁC TEST CASE 11](#_Toc194092062)

[BÀI 2 13](#_Toc194092063)

[I.GIẢI THÍCH CODE 13](#_Toc194092064)

[II.CÁC TEST CASE 14](#_Toc194092065)

# A.BÀI TẬP THỰC HÀNH

## BÀI 1

### GIẢI THÍCH CODE

Lớp NganXepSoNguyen

Lớp này mô phỏng cấu trúc dữ liệu ngăn xếp (stack) để lưu trữ số nguyên.

bool rong(): Kiểm tra xem ngăn xếp có rỗng không.

void them(int giaTri): Thêm một phần tử vào đỉnh ngăn xếp.

int layRa(): Lấy phần tử trên cùng ra khỏi ngăn xếp (và xóa nó).

int dinh(): Trả về phần tử trên cùng nhưng không xóa.

void xoa(): Xóa toàn bộ ngăn xếp.

Hàm daoSo(int n) - Đảo số nguyên

Chương trình sử dụng ngăn xếp để đảo ngược số nguyên.

Cách hoạt động:

Tách từng chữ số của n, lưu vào ngăn xếp.

Lấy các chữ số ra khỏi ngăn xếp và ghép lại thành số đảo ngược.

Hàm laXauDoiXung(const string &s) - Kiểm tra xâu đối xứng

Hàm kiểm tra xem chuỗi có đối xứng hay không bằng cách sử dụng ngăn xếp.

Cách hoạt động:

Đẩy từng ký tự của chuỗi vào ngăn xếp.

Lấy từng ký tự ra và ghép lại thành chuỗi mới.

So sánh với chuỗi ban đầu.

Hàm doiSangNhiPhan(int n) - Chuyển số thập phân sang nhị phân

Hàm chuyển một số từ hệ thập phân sang hệ nhị phân bằng cách sử dụng ngăn xếp.

Cách hoạt động:

Chia số cho 2, lấy phần dư đưa vào ngăn xếp.

Lấy ra từng phần tử trong ngăn xếp để có kết quả nhị phân.

Hàm trungToSangHauTo(const string &bieuThuc) - Chuyển biểu thức trung tố sang hậu tố

Hàm này chuyển đổi một biểu thức trung tố (Infix) sang hậu tố (Postfix) bằng cách sử dụng ngăn xếp.

Quy tắc:

Toán hạng (số) được thêm trực tiếp vào kết quả.

Dấu mở ngoặc ( được đưa vào ngăn xếp.

Dấu đóng ngoặc ) lấy các toán tử ra khỏi ngăn xếp cho đến khi gặp (.

Toán tử (+, -, \*, /) lấy toán tử có độ ưu tiên cao hơn ra trước.

Hàm tinhGiaTriHauTo(const string &bieuThuc) - Tính giá trị biểu thức hậu tố

Hàm này nhận một biểu thức hậu tố và tính giá trị của nó bằng cách sử dụng ngăn xếp.

Cách hoạt động:

Số được đưa vào ngăn xếp.

Toán tử lấy hai số trên cùng của ngăn xếp để tính toán, sau đó đưa kết quả trở lại ngăn xếp.

Hàm main() - Chạy chương trình

Chương trình thực hiện các tác vụ sau:

Đảo số 1234.

Kiểm tra đối xứng của "madam".

Chuyển 10 sang nhị phân.

Chuyển "3+5\*2" sang hậu tố và tính giá trị của nó.

### CÁC TEST CASE

Kiểm tra đảo số (daoSo)

| Đầu vào | Kết quả mong đợi |
| --- | --- |
| 1234 | 4321 |
| 100 | 1 |
| 98765 | 56789 |

Lệnh test:

cout << "Dao so 1234: " << daoSo(1234) << endl;

cout << "Dao so 100: " << daoSo(100) << endl;

cout << "Dao so 98765: " << daoSo(98765) << endl;

Kiểm tra xâu đối xứng (laXauDoiXung)

| Đầu vào | Kết quả mong đợi |
| --- | --- |
| "madam" | 1 (true) |
| "hello" | 0 (false) |
| "racecar" | 1 (true) |

Lệnh test:

cout << "Xau 'madam' doi xung: " << laXauDoiXung("madam") << endl;

cout << "Xau 'hello' doi xung: " << laXauDoiXung("hello") << endl;

cout << "Xau 'racecar' doi xung: " << laXauDoiXung("racecar") << endl;

Chuyển số thập phân sang nhị phân (doiSangNhiPhan)

| Đầu vào | Kết quả mong đợi |
| --- | --- |
| 10 | "1010" |
| 5 | "101" |
| 32 | "100000" |

Lệnh test:

cout << "10 sang nhi phan: " << doiSangNhiPhan(10) << endl;

cout << "5 sang nhi phan: " << doiSangNhiPhan(5) << endl;

cout << "32 sang nhi phan: " << doiSangNhiPhan(32) << endl;

Chuyển trung tố sang hậu tố (trungToSangHauTo)

| Đầu vào | Kết quả mong đợi |
| --- | --- |
| "3+5\*2" | "352\*+" |
| "(1+2)\*3" | "12+3\*" |
| "8/(4-2)" | "842-/" |

Lệnh test:

cout << "Trung to '3+5\*2' sang hau to: " << trungToSangHauTo("3+5\*2") << endl;

cout << "Trung to '(1+2)\*3' sang hau to: " << trungToSangHauTo("(1+2)\*3") << endl;

cout << "Trung to '8/(4-2)' sang hau to: " << trungToSangHauTo("8/(4-2)") << endl;

Tính giá trị biểu thức hậu tố (tinhGiaTriHauTo)

| Đầu vào | Kết quả mong đợi |
| --- | --- |
| "352\*+" | 13 |
| "12+3\*" | 9 |
| "842-/" | 4 |

Lệnh test:

cout << "Gia tri bieu thuc hau to '352\*+': " << tinhGiaTriHauTo("352\*+") << endl;

cout << "Gia tri bieu thuc hau to '12+3\*': " << tinhGiaTriHauTo("12+3\*") << endl;

cout << "Gia tri bieu thuc hau to '842-/': " << tinhGiaTriHauTo("842-/") << endl;

## BÀI 2

### I.GIẢI THÍCH CODE

Dưới đây là giải thích từng phần của bài toán mà không kèm ví dụ:

Cấu trúc dữ liệu hàng đợi

Hàng đợi (Queue) là một cấu trúc dữ liệu kiểu FIFO (First-In-First-Out), nghĩa là phần tử vào trước sẽ được lấy ra trước.

Hàng đợi sử dụng mảng (QueueInt)

Sử dụng mảng tĩnh hoặc động để quản lý hàng đợi.

Các thao tác chính:

Khởi tạo hàng đợi (InitQueue).

Kiểm tra hàng đợi có rỗng không (IsEmpty).

Kiểm tra hàng đợi có đầy không (IsFull).

Thêm phần tử vào cuối hàng đợi (PushQueue).

Lấy phần tử ra khỏi đầu hàng đợi (PopQueue).

Xem phần tử đầu hàng đợi mà không xóa (PeekQueue).

Xóa toàn bộ hàng đợi (Clear).

Hàng đợi sử dụng danh sách liên kết (LinkedQueueInt)

Sử dụng danh sách liên kết để lưu trữ phần tử.

Có hai con trỏ:

dau: trỏ đến phần tử đầu tiên.

cuoi: trỏ đến phần tử cuối cùng.

Các thao tác:

Thêm phần tử vào cuối hàng đợi, cập nhật cuoi.

Xóa phần tử khỏi đầu hàng đợi, cập nhật dau.

Kiểm tra hàng đợi có rỗng không.

Xóa toàn bộ hàng đợi bằng cách duyệt và giải phóng bộ nhớ.

Ứng dụng hàng đợi: Xếp lịch cặp múa

Bài toán yêu cầu quản lý danh sách nam và nữ để ghép cặp nhảy.

Các bước thực hiện:

Khởi tạo hai hàng đợi chứa danh sách vũ công nam và nữ.

Duyệt qua hai hàng đợi, lấy từng cặp nam - nữ ra và ghép đôi.

Nếu còn dư nam hoặc nữ sau khi ghép, thông báo có người chưa có cặp.

Ứng dụng hàng đợi: Thuật toán Radix Sort

Radix Sort là thuật toán sắp xếp số nguyên dựa trên từng chữ số từ nhỏ đến lớn.

Các bước thực hiện:

Tìm số có nhiều chữ số nhất để xác định số lần duyệt.

Duyệt từng chữ số từ hàng đơn vị đến hàng lớn nhất:

Dùng 10 hàng đợi để phân loại số theo chữ số đang xét.

Thêm các số vào hàng đợi tương ứng với chữ số tại vị trí đang xét.

Gom lại danh sách số từ các hàng đợi để tạo mảng mới.

Lặp lại quá trình trên cho đến khi sắp xếp xong toàn bộ dãy số

### II.CÁC TEST CASE

Dưới đây là các test case để kiểm tra từng phần của chương trình:

Kiểm tra hàng đợi (QueueInt và LinkedQueueInt)

Test khả năng thêm, xóa và kiểm tra rỗng của hàng đợi.

Test Case 1: Kiểm tra hàng đợi rỗng

Dữ liệu vào: Không thêm phần tử nào.

Kết quả mong đợi: Hàng đợi rỗng.

Test Case 2: Thêm và lấy phần tử

Dữ liệu vào: Thêm các số 5, 10, 15 vào hàng đợi.

Kết quả mong đợi:

Lấy phần tử ra: 5 (FIFO).

Lấy tiếp: 10, rồi 15.

Hàng đợi rỗng sau khi lấy hết.

Test Case 3: Xóa toàn bộ hàng đợi

Dữ liệu vào: Thêm 1, 2, 3, sau đó xóa toàn bộ.

Kết quả mong đợi: Hàng đợi rỗng sau khi xóa.

Kiểm tra xếp lịch cặp múa

Test khả năng ghép cặp nam – nữ.

Test Case 1: Số nam và nữ bằng nhau

Dữ liệu vào:

Nam: Minh, An, Hoàng.

Nữ: Lan, Mai, Hương.

Kết quả mong đợi:

Minh - Lan

An - Mai

Hoàng - Hương

Test Case 2: Thiếu nữ

Dữ liệu vào:

Nam: Minh, An, Hoàng.

Nữ: Lan, Mai.

Kết quả mong đợi:

Minh - Lan

An - Mai

Thông báo: Thiếu nữ, Hoàng chưa có cặp.

Test Case 3: Thiếu nam

Dữ liệu vào:

Nam: Minh, An.

Nữ: Lan, Mai, Hương.

Kết quả mong đợi:

Minh - Lan

An - Mai

Thông báo: Thiếu nam, Hương chưa có cặp.

Kiểm tra thuật toán Radix Sort

Test khả năng sắp xếp số bằng Radix Sort.

Test Case 1: Dãy số ngẫu nhiên

Dữ liệu vào: [170, 45, 75, 90, 802, 24, 2, 66]

Kết quả mong đợi: [2, 24, 45, 66, 75, 90, 170, 802].

Test Case 2: Dãy số đã sắp xếp

Dữ liệu vào: [1, 2, 3, 4, 5]

Kết quả mong đợi: [1, 2, 3, 4, 5].

Test Case 3: Dãy số giảm dần

Dữ liệu vào: [9, 8, 7, 6, 5]

Kết quả mong đợi: [5, 6, 7, 8, 9].

Test Case 4: Dãy có số lặp lại

Dữ liệu vào: [10, 10, 5, 5, 20, 20]

Kết quả mong đợi: [5, 5, 10, 10, 20, 20].

# B.BÀI TẬP MỞ RỘNG

## BÀI 1

### I.GIẢI THÍCH CODE

Tính số Fibonacci

Bài toán yêu cầu tính số Fibonacci thứ n, với công thức:

F(0)=0F(0) = 0

F(1)=1F(1) = 1

F(n)=F(n−1)+F(n−2)F(n) = F(n-1) + F(n-2) (với n≥2n \geq 2)

Cách 1: Đệ quy

Hàm tự gọi lại chính nó với hai giá trị nhỏ hơn (n-1 và n-2).

Khi n bằng 0 hoặc 1, trả về trực tiếp kết quả.

Nhược điểm: Gọi đệ quy chồng chất, tính toán dư thừa, độ phức tạp O(2^n).

Cách 2: Khử đệ quy dùng Stack

Dùng một vòng lặp để tính Fibonacci từ dưới lên.

Hai biến a và b lưu hai giá trị trước đó, cộng lại để có giá trị mới.

Dùng stack để lưu trữ số Fibonacci lớn hơn trong quá trình tính toán.

Độ phức tạp O(n), tối ưu hơn đệ quy.

Đảo ngược số

Bài toán yêu cầu lấy từng chữ số của số nguyên và sắp xếp lại theo thứ tự ngược lại.

Cách 1: Đệ quy

Mỗi lần lấy chữ số cuối (n % 10), đẩy vào số đảo ngược đang có.

Chia nhỏ số bằng cách loại bỏ chữ số cuối (n / 10).

Khi n bằng 0, trả về kết quả cuối cùng.

Cách 2: Khử đệ quy dùng Stack

Đẩy từng chữ số vào Stack để lưu trữ theo thứ tự ban đầu.

Lấy từng chữ số từ Stack ra, ghép lại thành số đảo ngược.

Stack giúp đảm bảo thứ tự đảo ngược mà không cần dùng đệ quy.

Bài toán Tháp Hà Nội

Tháp Hà Nội có n đĩa xếp theo thứ tự nhỏ trên lớn ở cột xuất phát. Mục tiêu là di chuyển toàn bộ đĩa sang cột đích, tuân theo các quy tắc:

Chỉ di chuyển từng đĩa một lần.

Không được đặt đĩa lớn lên trên đĩa nhỏ.

Chỉ được dùng một cột trung gian.

Cách 1: Đệ quy

Chia nhỏ bài toán bằng cách:

Di chuyển n-1 đĩa từ cột xuất phát sang cột trung gian.

Di chuyển đĩa lớn nhất sang cột đích.

Di chuyển n-1 đĩa từ cột trung gian sang cột đích.

Khi chỉ còn một đĩa, di chuyển trực tiếp từ cột xuất phát sang cột đích.

Cách 2: Khử đệ quy dùng Stack

Dùng Stack để mô phỏng các bước di chuyển như trong đệ quy.

Mỗi lần lấy một thao tác từ Stack để thực hiện và thêm các thao tác mới theo thứ tự cần thiết.

Stack giúp tránh việc gọi đệ quy chồng chất và dễ kiểm soát quá trình di chuyển.

### II.CÁC TEST CASE

Test Cases để thử chương trình

Tính số Fibonacci

Test Case 1

Input: n = 0  
 Output: 0

Test Case 2

Input: n = 1  
 Output: 1

Test Case 3

Input: n = 5  
Output: 5

Test Case 4

Input: n = 10  
Output: 55

Đảo ngược số

Test Case 1

Input: n = 1234  
Output: 4321

Test Case 2

Input: n = 100  
Output: 1

Test Case 3

Input: n = 987654321  
Output: 123456789

Test Case 4

Input: n = 0  
Output: 0

Bài toán Tháp Hà Nội

Test Case 1

Input: n = 1  
Output:

Di chuyen dia tu A den C

Test Case 2

Input: n = 2  
Output:

Di chuyen dia tu A den B

Di chuyen dia tu A den C

Di chuyen dia tu B den C

Test Case 3

Input: n = 3  
Output:

Di chuyen dia tu A den C

Di chuyen dia tu A den B

Di chuyen dia tu C den B

Di chuyen dia tu A den C

Di chuyen dia tu B den A

Di chuyen dia tu B den C

Di chuyen dia tu A den C

## BÀI 2

### I.GIẢI THÍCH CODE

Giải thích thuật toán tìm đường trong mê cung bằng BFS (Hàng đợi - Queue)

Nhập dữ liệu đầu vào

Đọc kích thước mê cung N x M (số dòng x số cột).

Đọc vị trí xuất phát (x0, y0) và vị trí đích (x1, y1).

Đọc ma trận N x M, trong đó:

0 là ô có thể đi qua.

1 là ô có chướng ngại vật (không đi qua được).

Ý tưởng thuật toán BFS (Duyệt theo từng lớp)

BFS sử dụng hàng đợi (queue) để duyệt tất cả các ô có thể đi đến từ một vị trí, theo từng lớp, đảm bảo tìm được đường đi ngắn nhất.

Bắt đầu từ vị trí (x0, y0), lần lượt xét các ô kề bên theo thứ tự: trái, phải, lên, xuống.

Nếu một ô có thể đi được (không có chướng ngại vật, chưa đi qua), đưa ô đó vào hàng đợi và đánh dấu đã thăm.

Tiếp tục duyệt các ô trong hàng đợi cho đến khi:

Tìm thấy điểm đích (x1, y1): Truy vết ngược lại để in đường đi.

Hàng đợi trống: Không có đường đi.

Mảng truy vết đường đi

Dùng một mảng truyVet để lưu vị trí trước đó của mỗi ô được duyệt, giúp xây dựng lộ trình khi tìm thấy điểm đích.

Khi tìm thấy (x1, y1), truy ngược lại từ điểm đích về điểm xuất phát để lấy đường đi.

Xuất kết quả

Nếu tìm thấy đường đi:

In số bước cần đi.

In danh sách tọa độ từ điểm xuất phát đến điểm đích.

Nếu không tìm thấy đường đi, in 0.

### II.CÁC TEST CASE

Dưới đây là một số test case để bạn có thể thử với chương trình tìm đường trong mê cung bằng BFS:

Test Case 1: Có đường đi đơn giản

Input

5 6 0 0 4 5

0 1 0 1 0 0

0 0 0 1 1 0

1 1 0 0 0 0

0 0 1 1 1 0

1 0 0 0 0 0

Output

7

0 0

1 0

1 1

1 2

2 2

3 2

3 3

4 5

Có đường đi từ (0,0) đến (4,5).

Test Case 2: Không có đường đi (bị chặn hoàn toàn)

Input

4 4 0 0 3 3

0 1 1 1

1 1 0 1

1 1 0 1

1 1 1 1

Output

0

Không có đường đi vì điểm đích bị chặn.

Test Case 3: Điểm xuất phát và điểm đích trùng nhau

Input

3 3 1 1 1 1

0 0 0

0 0 0

0 0 0

Output

1

1 1

Không cần di chuyển, in ra chính điểm xuất phát.

Test Case 4: Mê cung có nhiều chướng ngại vật

Input

6 6 0 0 5 5

0 1 0 0 1 0

0 1 1 0 1 0

0 0 0 0 0 0

1 1 1 1 1 0

0 0 0 0 1 0

1 1 1 0 0 0

Output

10

0 0

1 0

2 0

2 1

2 2

2 3

2 4

3 4

4 4

5 5

Có thể tìm được đường đi dài nhất.