Source Code

Bài 1

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

typedef struct Stack {

Node \*top;

} Stack;

// Khởi tạo stack rỗng

void Init(Stack \*s) {

s->top = NULL;

}

// Kiểm tra rỗng

int isEmpty(Stack \*s) {

return (s->top == NULL);

}

// Push: thêm phần tử vào đỉnh

int Push(Stack \*s, int x) {

Node \*p = (Node\*)malloc(sizeof(Node));

if (p == NULL) {

return 0; // thất bại

}

p->data = x;

p->next = s->top;

s->top = p;

return 1; // thành công

}

// Peek: xem phần tử đỉnh mà không xóa

// trả -1 nếu rỗng (giả sử dữ liệu không âm - nếu cần, có thể dùng cờ lỗi)

int Peek(Stack \*s, int \*ok) {

if (isEmpty(s)) {

if (ok) \*ok = 0;

return -1;

}

if (ok) \*ok = 1;

return s->top->data;

}

// Pop: lấy phần tử đỉnh và xóa node, trả ra giá trị. Nếu rỗng trả -1 và ok=0

int Pop(Stack \*s, int \*ok) {

if (isEmpty(s)) {

if (ok) \*ok = 0;

return -1;

}

Node \*p = s->top;

int v = p->data;

s->top = p->next;

free(p);

if (ok) \*ok = 1;

return v;

}

// Giải phóng toàn bộ node trong stack (important để tránh memory leak)

void Destroy(Stack \*s) {

Node \*p = s->top;

while (p != NULL) {

Node \*tmp = p;

p = p->next;

free(tmp);

}

s->top = NULL;

}

// Đổi số thập phân sang nhị phân và in ra màn hình

// Xử lý số 0 và số âm

void DoiSangNhiPhan\_In(int n) {

Stack s;

Init(&s);

if (n == 0) {

printf("0\n");

return;

}

int negative = 0;

long nl = n; // để tránh overflow khi n = INT\_MIN

if (nl < 0) {

negative = 1;

nl = -nl; // chuyển sang dương

}

// Lưu các phần dư

while (nl > 0) {

int r = (int)(nl % 2);

if (!Push(&s, r)) {

fprintf(stderr, "Loi cap phat bo nho khi Push!\n");

Destroy(&s);

return;

}

nl /= 2;

}

// In kết quả

if (negative) putchar('-');

int ok;

while (!isEmpty(&s)) {

int bit = Pop(&s, &ok);

if (ok) putchar('0' + bit);

else break;

}

putchar('\n');

Destroy(&s);

}

// Một phiên bản trả kết quả vào buffer (cần cấp bộ nhớ đủ lớn)

int DoiSangNhiPhan\_Buffer(int n, char \*buf, size\_t buflen) {

if (buf == NULL || buflen == 0) return 0;

Stack s;

Init(&s);

if (n == 0) {

if (buflen < 2) return 0;

buf[0] = '0'; buf[1] = '\0';

return 1;

}

int negative = 0;

long nl = n;

if (nl < 0) {

negative = 1;

nl = -nl;

}

while (nl > 0) {

int r = (int)(nl % 2);

if (!Push(&s, r)) {

Destroy(&s);

return 0;

}

nl /= 2;

}

size\_t idx = 0;

if (negative) {

if (idx+1 >= buflen) { Destroy(&s); return 0; }

buf[idx++] = '-';

}

int ok;

while (!isEmpty(&s)) {

if (idx+1 >= buflen) { Destroy(&s); return 0; } // không đủ chỗ

int bit = Pop(&s, &ok);

if (!ok) break;

buf[idx++] = '0' + bit;

}

if (idx >= buflen) { Destroy(&s); return 0; }

buf[idx] = '\0';

Destroy(&s);

return 1;

}

int main(void) {

int n;

printf("Nhap so thap phan (co the am): ");

if (scanf("%d", &n) != 1) {

printf("Nhap khong hop le!\n");

return 1;

}

printf("Ket qua (in): ");

DoiSangNhiPhan\_In(n);

// Ví dụ dùng buffer

char out[128];

if (DoiSangNhiPhan\_Buffer(n, out, sizeof(out))) {

printf("Ket qua (buffer): %s\n", out);

} else {

printf("Chuyen sang nhi phan (buffer) that bai!\n");

}

return 0;

}

Code hoàn chỉnh

#include <iostream>

#include <string>

using namespace std;

// ... (Tất cả các hàm đã triển khai ở trên)

// Hàm menu để test toàn bộ chức năng

void showMenu() {

cout << "\n" << string(50, '=') << endl;

cout << " CHƯƠNG TRÌNH DEMO STACK" << endl;

cout << string(50, '=') << endl;

cout << "1. Test các thao tác stack cơ bản" << endl;

cout << "2. Chuyển đổi số thập phân → nhị phân" << endl;

cout << "3. Demo với các số có sẵn" << endl;

cout << "4. Hiển thị stack hiện tại" << endl;

cout << "0. Thoát" << endl;

cout << string(50, '=') << endl;

cout << "Lựa chọn: ";

}

// Hàm test thao tác cơ bản

void testBasicOperations() {

cout << "\n--- TEST THAO TÁC CƠ BẢN ---" << endl;

Stack s;

initStack(s);

isEmptyDetailed(s);

// Test push

push(s, 10);

push(s, 20);

push(s, 30);

displayStack(s);

// Test pop

int val;

pop(s, val);

displayStack(s);

pop(s, val);

displayStack(s);

// Test stack rỗng

while (pop(s, val)) {

cout << "Đã lấy: " << val << endl;

}

isEmptyDetailed(s);

}

// Hàm demo với số có sẵn

void demoPredefinedNumbers() {

cout << "\n--- DEMO VỚI SỐ CÓ SẴN ---" << endl;

int testNumbers[] = {0, 1, 2, 5, 10, 15, 16, 255, 1024};

int count = sizeof(testNumbers) / sizeof(testNumbers[0]);

for (int i = 0; i < count; i++) {

decimalToBinary(testNumbers[i]);

}

}

int main() {

Stack mainStack;

initStack(mainStack);

int choice;

do {

showMenu();

cin >> choice;

switch (choice) {

case 1:

testBasicOperations();

break;

case 2: {

int number;

cout << "Nhập số thập phân: ";

cin >> number;

decimalToBinary(number);

break;

}

case 3:

demoPredefinedNumbers();

break;

case 4:

displayStack(mainStack);

break;

case 0:

cout << "Kết thúc chương trình!" << endl;

break;

default:

cout << "Lựa chọn không hợp lệ!" << endl;

}

} while (choice != 0);

return 0;

}

**BÀI TẬP 2: QUEUE - HÀNG ĐỢI SỬ DỤNG DANH SÁCH LIÊN KẾT ĐƠN**

#include <iostream>

using namespace std;

// Định nghĩa cấu trúc và các hàm như trên

// Hàm hiển thị toàn bộ queue

void displayQueue(Queue q) {

if (isEmpty(q)) {

cout << "Queue rong!" << endl;

return;

}

cout << "Noi dung queue (tu front -> rear): ";

Node\* current = q.front;

while (current != nullptr) {

cout << current->data;

if (current->next != nullptr) {

cout << " -> ";

}

current = current->next;

}

cout << " [Count: " << q.count << "]" << endl;

}

// Hàm xem phần tử đầu queue không xóa

int front(Queue q) {

if (isEmpty(q)) {

cout << "Queue rong!" << endl;

return -1;

}

return q.front->data;

}

// Hàm demo các thao tác cơ bản

void demoBasicOperations() {

cout << "\n" << "DEMO CAC THAO TAC CO BAN TREN QUEUE" << endl;

Queue q;

initQueue(q);

// Test queue rỗng

isEmptyDetailed(q);

// Test thêm phần tử

cout << "\nThem phan tu vao queue" << endl;

enqueue(q, 10);

enqueue(q, 20);

enqueue(q, 30);

displayQueue(q);

// Test xem phần tử đầu

cout << "\nXem phan tu dau queue" << endl;

cout << "Phan tu dau queue: " << front(q) << endl;

// Test lấy phần tử

cout << "\nLay phan tu ra khoi queue" << endl;

int value;

dequeue(q, value);

cout << "Da lay: " << value << endl;

displayQueue(q);

dequeue(q, value);

cout << "Da lay: " << value << endl;

displayQueue(q);

// Test queue sau khi lấy hết

cout << "\nKiem tra queue rong" << endl;

dequeue(q, value);

cout << "Da lay: " << value << endl;

displayQueue(q);

isEmptyDetailed(q);

// Test thêm lại sau khi rỗng

cout << "\nThem lai sau khi rong" << endl;

enqueue(q, 40);

enqueue(q, 50);

displayQueue(q);

}

// Hàm menu tương tác

void showMenu() {

cout << "\nCHUONG TRINH QUAN LY QUEUE" << endl;

cout << "1. Khoi tao queue" << endl;

cout << "2. Kiem tra queue rong" << endl;

cout << "3. Them phan tu (Enqueue)" << endl;

cout << "4. Lay phan tu (Dequeue)" << endl;

cout << "5. Xem phan tu dau (Front)" << endl;

cout << "6. Hien thi queue" << endl;

cout << "7. Demo tu dong" << endl;

cout << "0. Thoat" << endl;

cout << "Lua chon: ";

}

int main() {

Queue q;

initQueue(q);

int choice, value;

do {

showMenu();

cin >> choice;

switch (choice) {

case 1:

initQueue(q);

break;

case 2:

isEmptyDetailed(q);

break;

case 3:

cout << "Nhap gia tri can them: ";

cin >> value;

enqueue(q, value);

break;

case 4:

if (dequeue(q, value)) {

cout << "Da lay duoc: " << value << endl;

}

break;

case 5:

if (!isEmpty(q)) {

cout << "Phan tu dau queue: " << front(q) << endl;

}

break;

case 6:

displayQueue(q);

break;

case 7:

demoBasicOperations();

break;

case 0:

cout << "Ket thuc chuong trinh!" << endl;

break;

default:

cout << "Lua chon khong hop le!" << endl;

}

} while (choice != 0);

// Giải phóng bộ nhớ còn lại

while (!isEmpty(q)) {

dequeue(q);

}

return 0;

}

**BÀI TẬP 3: STACK SỬ DỤNG MẢNG**

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

#define MAX 100

struct ArrayStack {

int data[MAX];

int top;

ArrayStack() : top(-1) {}

};

// Các hàm đã định nghĩa ở trên

void initStack(ArrayStack &s) {

s.top = -1;

}

bool isEmpty(ArrayStack s) {

return s.top == -1;

}

bool isFull(ArrayStack s) {

return s.top == MAX - 1;

}

bool push(ArrayStack &s, int value) {

if (isFull(s)) {

cout << "FULL" << endl;

return false;

}

s.top++;

s.data[s.top] = value;

return true;

}

bool pop(ArrayStack &s, int &value) {

if (isEmpty(s)) {

cout << "EMPTY" << endl;

return false;

}

value = s.data[s.top];

s.top--;

return true;

}

bool top(ArrayStack s, int &value) {

if (isEmpty(s)) {

cout << "EMPTY" << endl;

return false;

}

value = s.data[s.top];

return true;

}

int size(ArrayStack s) {

return s.top + 1;

}

// Hàm xử lý lệnh

void processCommand(ArrayStack &stack, const string &command) {

stringstream ss(command);

string cmd;

ss >> cmd;

if (cmd == "PUSH") {

int value;

ss >> value;

push(stack, value);

}

else if (cmd == "POP") {

int value;

if (pop(stack, value)) {

cout << value << endl;

}

}

else if (cmd == "TOP") {

int value;

if (top(stack, value)) {

cout << value << endl;

}

}

else if (cmd == "SIZE") {

cout << size(stack) << endl;

}

else if (cmd == "END") {

// Kết thúc chương trình

return;

}

}

// Hàm main

int main() {

ArrayStack stack;

initStack(stack);

string command;

cout << "Nhap cac lenh (END de ket thuc):" << endl;

while (true) {

getline(cin, command);

if (command == "END") {

break;

}

processCommand(stack, command);

}

return 0;

}

**BÀI TẬP 4: KIỂM TRA DẤU NGOẶC & HTML-LIKE TAGS BẰNG STACK**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

#define MAX 1000

// Định nghĩa các cấu trúc và hàm như trên

// Phiên bản tối ưu sử dụng vector thay vì mảng

class BracketChecker {

private:

vector<char> stack;

public:

string check(const string &expression) {

stack.clear();

for (char c : expression) {

if (c == '(' || c == '[' || c == '{') {

stack.push\_back(c);

}

else if (c == ')' || c == ']' || c == '}') {

if (stack.empty()) return "NO";

char topChar = stack.back();

stack.pop\_back();

if (!isMatchingPair(topChar, c)) {

return "NO";

}

}

}

return stack.empty() ? "YES" : "NO";

}

private:

bool isMatchingPair(char open, char close) {

return (open == '(' && close == ')') ||

(open == '[' && close == ']') ||

(open == '{' && close == '}');

}

};

class HTMLTagChecker {

private:

vector<string> stack;

public:

string check(const string &html) {

stack.clear();

int i = 0;

int n = html.length();

while (i < n) {

if (html[i] == '<') {

if (i + 1 < n && html[i + 1] == '/') {

// Closing tag

int j = i + 2;

while (j < n && html[j] != '>') j++;

if (j >= n) return "NO";

string tagName = html.substr(i + 2, j - i - 2);

if (stack.empty() || stack.back() != tagName) {

return "NO";

}

stack.pop\_back();

i = j + 1;

}

else {

// Opening tag

int j = i + 1;

while (j < n && html[j] != '>') j++;

if (j >= n) return "NO";

string tagName = html.substr(i + 1, j - i - 1);

stack.push\_back(tagName);

i = j + 1;

}

}

else {

i++;

}

}

return stack.empty() ? "YES" : "NO";

}

};

// Hàm main với menu

int main() {

BracketChecker bracketChecker;

HTMLTagChecker htmlChecker;

int choice;

string input;

do {

cout << "\n=== KIEM TRA STACK ===" << endl;

cout << "1. Kiem tra dau ngoac" << endl;

cout << "2. Kiem tra the HTML" << endl;

cout << "0. Thoat" << endl;

cout << "Lua chon: ";

cin >> choice;

cin.ignore(); // Xóa ký tự newline

switch (choice) {

case 1:

cout << "Nhap chuoi ngoac: ";

getline(cin, input);

cout << "Ket qua: " << bracketChecker.check(input) << endl;

break;

case 2:

cout << "Nhap chuoi HTML: ";

getline(cin, input);

cout << "Ket qua: " << htmlChecker.check(input) << endl;

break;

case 0:

cout << "Tam biet!" << endl;

break;

default:

cout << "Lua chon khong hop le!" << endl;

}

} while (choice != 0);

return 0;

}

**BÀI TẬP 5: CHUYỂN TRUNG TỐ → HẬU TỐ & TÍNH GIÁ TRỊ**

#include <iostream>

#include <string>

#include <vector>

#include <stack>

#include <cctype>

#include <cmath>

#include <sstream>

using namespace std;

class InfixCalculator {

private:

int precedence(char op) {

if (op == '^') return 3;

if (op == '\*' || op == '/') return 2;

if (op == '+' || op == '-') return 1;

return 0;

}

bool isOperator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

}

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

switch(op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/':

if (b == 0) throw runtime\_error("Loi: chia cho 0");

return a / b;

case '^': return (int)pow(a, b);

default: return 0;

}

}

public:

vector<string> tokenize(const string& expression) {

vector<string> tokens;

string currentToken = "";

for (char c : expression) {

if (c == ' ') {

if (!currentToken.empty()) {

tokens.push\_back(currentToken);

currentToken = "";

}

}

else if (isOperator(c) || c == '(' || c == ')') {

if (!currentToken.empty()) {

tokens.push\_back(currentToken);

currentToken = "";

}

tokens.push\_back(string(1, c));

}

else {

currentToken += c;

}

}

if (!currentToken.empty()) {

tokens.push\_back(currentToken);

}

return tokens;

}

vector<string> infixToPostfix(const vector<string>& tokens) {

vector<string> output;

stack<string> operatorStack;

for (const string& token : tokens) {

if (isdigit(token[0]) || (token.length() > 1 && isdigit(token[1]))) {

output.push\_back(token);

}

else if (isOperator(token[0])) {

while (!operatorStack.empty() &&

isOperator(operatorStack.top()[0]) &&

precedence(operatorStack.top()[0]) >= precedence(token[0])) {

output.push\_back(operatorStack.top());

operatorStack.pop();

}

operatorStack.push(token);

}

else if (token == "(") {

operatorStack.push(token);

}

else if (token == ")") {

while (!operatorStack.empty() && operatorStack.top() != "(") {

output.push\_back(operatorStack.top());

operatorStack.pop();

}

if (!operatorStack.empty()) {

operatorStack.pop();

}

}

}

while (!operatorStack.empty()) {

output.push\_back(operatorStack.top());

operatorStack.pop();

}

return output;

}

int evaluatePostfix(const vector<string>& postfix) {

stack<int> valueStack;

for (const string& token : postfix) {

if (isdigit(token[0]) || (token.length() > 1 && isdigit(token[1]))) {

valueStack.push(stoi(token));

}

else if (isOperator(token[0])) {

int b = valueStack.top(); valueStack.pop();

int a = valueStack.top(); valueStack.pop();

int result = applyOperator(a, b, token[0]);

valueStack.push(result);

}

}

return valueStack.top();

}

void processExpression(const string& expression) {

try {

// Bước 1: Tách token

vector<string> tokens = tokenize(expression);

// Bước 2: Chuyển sang hậu tố

vector<string> postfix = infixToPostfix(tokens);

// Bước 3: Hiển thị kết quả hậu tố

cout << "Bieu thuc hau to: ";

for (size\_t i = 0; i < postfix.size(); ++i) {

cout << postfix[i];

if (i < postfix.size() - 1) cout << " ";

}

cout << endl;

// Bước 4: Tính giá trị

int result = evaluatePostfix(postfix);

cout << "Gia tri: " << result << endl;

} catch (const exception& e) {

cout << "Loi: " << e.what() << endl;

}

}

};

int main() {

InfixCalculator calculator;

string expression;

cout << "Nhap bieu thuc trung to: ";

getline(cin, expression);

calculator.processExpression(expression);

return 0;

}