TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

THỰC HÀNH:

NGĂN XẾP VÀ HÀNG ĐỢI

SINH VIÊN THỰC HIỆN: Nguyễn Đỗ Huy – 3121411085

LỚP: DCT124C7

GVHD: ĐỖ NHƯ TÀI

Thành phố Hồ Chí Minh , Tháng 2 Năm 2025

MỤC LỤC

Bài tập thực hành

Bài 1. Ngăn xếp số nguyên

(a) Cài đặt CTDL StackInt và LinkedStackInt dùng để chứa các số nguyên, trong đó:

• Sử dụng mảng (StackInt) và danh sách liên kết (LinkedStackInt)

• Cài đặt các thao tác: InitStack, IsEmpty, IsFull, PopStack, PushStack, PeekStack, Clear

(b) Ứng dụng ngăn xếp để đảo số,

(c) Ứng dụng ngăn xếp để kiểm tra xâu đối xứng (mở rộng sang StackString),

(d) Ứng dụng ngăn xếp để đổi từ số thập phân sang nhị phân,

(e) Nhập một biểu thức từ bàn phím, hãy chuyển sang dạng hậu tố và tính giá trị biểu thức.

Bài làm

- Ý tưởng
    - Ngăn xếp (Stack) là cấu trúc dữ liệu LIFO – phần tử được thêm sau cùng sẽ được lấy ra trước.
    - Bài tập yêu cầu cài đặt bằng:
        - Mảng tĩnh (StackInt).
        - Danh sách liên kết (LinkedStackInt).
    - Mỗi cấu trúc có các thao tác: InitStack, IsEmpty, IsFull (với mảng), Push, Pop, Peek, Clear.
    - Các ứng dụng:
        - (b) Đảo số.
            - Cho một số nguyên, ta dùng stack để tách từng chữ số ra.
            - Sau đó, lấy ra từng phần tử trong stack và ghép lại – sẽ được số đảo ngược.
        - (c) Kiểm tra xâu đối xứng (StackString).
            - Duyệt từng ký tự của chuỗi và đẩy vào stack.
            - Sau đó duyệt lại chuỗi từ đầu, và so sánh từng ký tự với ký tự được pop ra từ stack.
            - Nếu tất cả trùng nhau → chuỗi đối xứng.
        - (d) Đổi số thập phân sang nhị phân.
            - Dùng phép chia liên tiếp cho 2, lấy phần dư mỗi lần chia và đẩy vào stack.
            - Sau đó pop ra từng phần tử để có chuỗi nhị phân đúng thứ tự.
        - (e) Chuyển và tính giá trị biểu thức hậu tố.
            - Duyệt từng ký tự:
                - ❖ Nếu là toán hạng → đưa thẳng vào kết quả.
                - ❖ Nếu là dấu mở ngoặc → đẩy vào stack.
                - ❖ Nếu là toán tử → so sánh độ ưu tiên với stack:

- Pop các toán tử trong stack có ưu tiên cao hơn.
- Push toán tử hiện tại vào stack.

❖ Nếu là dấu đóng ngoặc) → pop hết đến khi gặp (.

▪ Sau khi duyệt xong → pop hết các phần tử còn lại trong stack ra hậu tố.

- Mã nguồn

StackInt.cpp

(a)

```
struct StackInt {
    int top;
    int arr[MAX];
};

// Khởi tạo
void InitStack(StackInt &s) {
    s.top = -1;
}

bool IsEmptyInt(StackInt s) {
    return s.top == -1;
}

bool IsFullInt(StackInt s) {
    return s.top == MAX - 1;
}

bool PushStackInt(StackInt &s, int x) {
    if (IsFullInt(s)) return false;
    s.arr[++s.top] = x;
    return true;
}

bool PopStackInt(StackInt &s, int &x) {
    if (IsEmptyInt(s)) return false;
    x = s.arr[s.top--];
    return true;
}

bool PeekStackInt(StackInt s, int &x) {
    if (IsEmptyInt(s)) return false;
    x = s.arr[s.top];
    return true;
}
```

```
void Clear(StackInt &s) {
    s.top = -1;
}

// Stack String
struct StackString {
    int top;
    char arr[MAX];
};

void InitStack(StackString &s) { s.top = -1; }
bool IsEmpty(StackString s) { return s.top == -1; }
bool IsFull(StackString s) { return s.top == MAX - 1; }

bool PushStack(StackString &s, char c) {
    if (IsFull(s)) return false;
    s.arr[++s.top] = c;
    return true;
}

bool PopStack(StackString &s, char &c) {
    if (IsEmpty(s)) return false;
    c = s.arr[s.top--];
    return true;
}

char PeekStack(StackString s) {
    return s.arr[s.top];
}
```

(b)

```
// (b) Ứng dụng ngăn xếp để đảo số
void DaoSo_StackArray(int n) {
    StackInt s;
    InitStack(s);
    int temp = n;

    if (temp == 0) {
        cout << "So dao nguoc cua 0 la: 0" << endl;
        return;
    }
    cout << "So dao nguoc cua " << n << " la: ";
    // Push từng chữ số vào stack
    while (temp > 0) {
        int digit = temp % 10;
        cout << digit;
        PushStackInt(s, digit);
```

```
        temp /= 10;
    }
    cout << endl;
}
```

(c)

```
// (c) Ứng dụng ngăn xếp để kiểm tra xâu đối xứng
bool KiemTraDoiXung_StackArray(string str) {
    StackString s;
    InitStack(s);

    for (char c : str) {
        PushStack(s, c);
    }

    for (char c : str) {
        char topChar;
        PopStack(s, topChar);
        if (c != topChar)
            return false;
    }

    return true;
}
```

(d)

```
// (d) Ứng dụng ngăn xếp để đổi từ số thập phân sang nhị phân
void DoiNhiPhan_StackArray(int n) {
    StackInt s;
    InitStack(s);

    int temp = n;
    if (n == 0) {
        cout << "0";
        return;
    }

    // Chia 2 và đẩy dư vào stack
    while (temp > 0) {
        PushStackInt(s, temp % 2);
        temp /= 2;
    }

    // In ra nhị phân
    cout << "So nhi phan cua " << n << " la: ";
    while (!IsEmptyInt(s)) {
        int bit;
```

```
        PopStackInt(s, bit);
        cout << bit;
    }
    cout << endl;
}
```

(e)

```
// (e) Nhập một biểu thức từ bàn phím, hãy chuyển sang dạng hậu tố và tính giá trị biểu thức.
// Chuyển trung tố → hậu tố
bool LaToanHang(char c) {
    return isdigit(c); // hoặc isalpha(c)
}

int UuTien(char op) {
    if (op == '(') return 0;
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return -1;
}

string InfixToPostfix_Array(string infix) {
    StackString st;
    InitStack(st);
    string postfix = "";

    for (char c : infix) {
        if (LaToanHang(c)) {
            postfix += c;
        }
        else if (c == '(') {
            PushStack(st, c);
        }
        else if (c == ')') {
            char t;
            while (!IsEmpty(st) && PeekStack(st) != '(') {
                PopStack(st, t);
                postfix += t;
            }
            PopStack(st, t); // bỏ '('
        }
        else {
            while (!IsEmpty(st) && UuTien(c) <= UuTien(PeekStack(st))) {
                char t;
                PopStack(st, t);
                postfix += t;
            }
```

```
            PushStack(st, c);
        }
    }

    char t;
    while (!IsEmpty(st)) {
        PopStack(st, t);
        postfix += t;
    }

    return postfix;
}

// Tính biểu thức hậu tố
int TinhHauTo_Array(string postfix) {
    StackInt st;
    InitStack(st);

    for (char c : postfix) {
        if (isdigit(c)) {
            PushStackInt(st, c - '0');
        } else {
            int b, a;
            PopStackInt(st, b);
            PopStackInt(st, a);
            int kq;
            switch (c) {
                case '+': kq = a + b; break;
                case '-': kq = a - b; break;
                case '*': kq = a * b; break;
                case '/': kq = a / b; break;
            }
            PushStackInt(st, kq);
        }
    }

    int kq;
    PopStackInt(st, kq);
    return kq;
}
```

LinkedStackInt.cpp

(a)

```
struct Node {
    int info;
```

```
    Node *next;
};

struct LinkedStackInt {
    Node *top;
};

// Khởi tạo
void InitStack(LinkedStackInt &s) {
    s.top = nullptr;
}

bool IsEmpty(LinkedStackInt s) {
    return s.top == nullptr;
}

void PushStack(LinkedStackInt &s, int x) {
    Node *p = new Node{x, s.top};
    s.top = p;
}

bool PopStack(LinkedStackInt &s, int &x) {
    if (IsEmpty(s)) return false;
    Node *p = s.top;
    x = p->info;
    s.top = s.top->next;
    delete p;
    return true;
}

bool PeekStack(LinkedStackInt s, int &x) {
    if (IsEmpty(s)) return false;
    x = s.top->info;
    return true;
}

void Clear(LinkedStackInt &s) {
    Node *p;
    while (s.top != nullptr) {
        p = s.top;
        s.top = s.top->next;
        delete p;
    }
}

// LinkedStackString
struct CharNode {
```

```
    char info;
    CharNode *next;
};

struct LinkedStackString {
    CharNode *top;
};

void InitStack(LinkedStackString &s) { s.top = nullptr; }
bool IsEmpty(LinkedStackString s) { return s.top == nullptr; }

void PushStack(LinkedStackString &s, char c) {
    CharNode *p = new CharNode{c, s.top};
    s.top = p;
}

bool PopStack(LinkedStackString &s, char &c) {
    if (IsEmpty(s)) return false;
    CharNode *p = s.top;
    c = p->info;
    s.top = p->next;
    delete p;
    return true;
}

char PeekStack(LinkedStackString s) {
    if (!IsEmpty(s)) return s.top->info;
    return '\0'; // hoặc throw nếu muốn
}
```

(b)

```
// (b) Ứng dụng ngăn xếp để đảo số
void DaoSo_LinkedStack(int n) {
    LinkedStackInt s;
    InitStack(s);
    int temp = n;

    cout << "So dao nguoc cua " << n << " la: ";

    // Push từng chữ số
    while (temp > 0) {
        int digit = temp % 10;
        cout << digit;
        PushStack(s, temp % 10);
        temp /= 10;
    }
    cout << endl;
```

```
}
```

(c)

```
// (c) Ứng dụng ngăn xếp để kiểm tra xâu đối xứng
bool KiemTraDoiXung_LinkedStack(string str) {
    LinkedStackString s;
    InitStack(s);

    for (char c : str) {
        PushStack(s, c);
    }

    for (char c : str) {
        char topChar;
        PopStack(s, topChar);
        if (c != topChar)
            return false;
    }

    return true;
}
```

(d)

```
// (d) Ứng dụng ngăn xếp để đổi từ số thập phân sang nhị phân
void DoiNhiPhan_LinkedStack(int n) {
    LinkedStackInt s;
    InitStack(s);

    int temp = n;
    if (n == 0) {
        cout << "0";
        return;
    }

    while (temp > 0) {
        PushStack(s, temp % 2);
        temp /= 2;
    }

    cout << "So nhi phan cua " << n << " la: ";
    while (!IsEmpty(s)) {
        int bit;
        PopStack(s, bit);
        cout << bit;
    }
    cout << endl;
}
```

(e)

```
// (e) Nhập một biểu thức từ bàn phím, hãy chuyển sang dạng hậu tố và tính giá trị biểu thức.
// Chuyển infix → postfix
bool LaToanHang(char c) {
    return isdigit(c); // hoặc isalpha(c)
}

int UuTien(char op) {
    if (op == '(') return 0;
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return -1;
}

string InfixToPostfix_Linked(string infix) {
    LinkedStackString st;
    InitStack(st);
    string postfix = "";

    for (char c : infix) {
        if (LaToanHang(c)) {
            postfix += c;
        }
        else if (c == '(') {
            PushStack(st, c);
        }
        else if (c == ')') {
            char t;
            while (!IsEmpty(st) && PeekStack(st) != '(') {
                PopStack(st, t);
                postfix += t;
            }
            PopStack(st, t); // remove '('
        }
        else {
            while (!IsEmpty(st) && UuTien(c) <= UuTien(PeekStack(st))) {
                char t;
                PopStack(st, t);
                postfix += t;
            }
            PushStack(st, c);
        }
    }

    char t;
```

```
    while (!IsEmpty(st)) {
        PopStack(st, t);
        postfix += t;
    }

    return postfix;
}
// Tính giá trị hậu tố
int TinhHauTo_Linked(string postfix) {
    LinkedStackInt st;
    InitStack(st);

    for (char c : postfix) {
        if (isdigit(c)) {
            PushStack(st, c - '0');
        } else {
            int b, a;
            PopStack(st, b);
            PopStack(st, a);
            int kq;
            switch (c) {
                case '+': kq = a + b; break;
                case '-': kq = a - b; break;
                case '*': kq = a * b; break;
                case '/': kq = a / b; break;
            }
            PushStack(st, kq);
        }
    }

    int kq;
    PopStack(st, kq);
    return kq;
}
```

Bài 2. Hàng đợi số nguyên

(a) Cài đặt CTDL QueueInt và LinkedQueueInt dùng để chứa các số nguyên, trong đó:

• Sử dụng mảng (StackInt) và danh sách liên kết (LinkedStackInt)

• Cài đặt các thao tác: InitStack, IsEmpty, IsFull, PopStack, PushStack, PeekStack, Clear

(b) Ứng dụng hàng đợi để làm bài toán xếp lịch cặp múa nam/nữ (như trong bài giảng),

(c) Ứng dụng hàng đợi để cài thuật toán RadixSort

Bài làm

- Ý tưởng
    - Câu a.
        - Hàng đợi là cấu trúc FIFO (vào trước – ra trước).
        - Mảng: dùng front, rear, count, size.
        - DSLK: dùng 2 con trỏ front, rear và count.
        - Cài đặt đầy đủ: khởi tạo, kiểm tra rỗng, đầy, thêm, lấy, xem đầu, xóa hàng.
    - Câu b.
        - Có 2 hàng đợi:
            - QueueNam: chứa các bạn nam.
            - QueueNu: chứa các bạn nữ.
        - Mỗi cặp gồm 1 nam và 1 nữ sẽ vào sàn nhảy cùng lúc.
        - Duyệt theo từng cặp đầu tiên trong 2 hàng → ghép cặp → in ra.
        - Nếu còn dư nam hoặc nữ → in ra ai đang chờ.
    - Câu c.
        - Radix Sort là thuật toán sắp xếp các số nguyên dựa trên từng chữ số, bắt đầu từ hàng đơn vị, rồi đến hàng chục, trăm,...
        - Dùng 10 hàng đợi (bin[0..9]) để phân loại các số theo chữ số ở vị trí đang xét.
        - 2 vòng chính:
            - Vòng 1: sắp theo hàng đơn vị.
            - Vòng 2: sắp theo hàng chục (và tiếp tục nếu có hàng trăm,...).
- Mã nguồn

(a)

QueueInt

```
const int MAX = 100;

struct QueueInt {
    int arr[MAX];
    int front, rear, count, size;
```

```
};

void InitQueue(QueueInt &q, int size = MAX) {
    q.front = q.rear = -1;
    q.count = 0;
    q.size = size;
}

bool IsEmpty(QueueInt q) {
    return q.front == -1;
}

bool IsFull(QueueInt q) {
    return q.count == q.size;
}

bool Enqueue(QueueInt &q, int x) {
    if (IsFull(q)) return false;

    if (q.front == -1) q.front = 0;

    q.rear = (q.rear + 1) % q.size;
    q.arr[q.rear] = x;
    q.count++;
    return true;
}

bool Dequeue(QueueInt &q, int &x) {
    if (IsEmpty(q)) return false;

    x = q.arr[q.front];
    if (q.front == q.rear) // chỉ còn 1 phần tử
        q.front = q.rear = -1;
    else
        q.front = (q.front + 1) % q.size;

    q.count--;
    return true;
}

bool Peek(QueueInt q, int &x) {
    if (IsEmpty(q)) return false;
    x = q.arr[q.front];
    return true;
}

void Clear(QueueInt &q) {
```

```
    q.front = q.rear = -1;
    q.count = 0;
}
```

LinkedStackInt

```
struct Node {
    int info;
    Node *next;
};

struct LinkedQueueInt {
    Node *front;
    Node *rear;
    int count;
};

void InitQueue(LinkedQueueInt &q) {
    q.front = q.rear = nullptr;
    q.count = 0;
}

bool IsEmpty(LinkedQueueInt q) {
    return q.front == nullptr;
}

void Enqueue(LinkedQueueInt &q, int x) {
    Node *p = new Node{x, nullptr};
    if (IsEmpty(q))
        q.front = q.rear = p;
    else {
        q.rear->next = p;
        q.rear = p;
    }
    q.count++;
}

bool Dequeue(LinkedQueueInt &q, int &x) {
    if (IsEmpty(q)) return false;
    Node *p = q.front;
    x = p->info;
    q.front = p->next;
    if (q.front == nullptr) q.rear = nullptr;
    delete p;
    q.count--;
    return true;
}
```

```
bool Peek(LinkedQueueInt q, int &x) {
    if (IsEmpty(q)) return false;
    x = q.front->info;
    return true;
}

void Clear(LinkedQueueInt &q) {
    while (!IsEmpty(q)) {
        int x;
        Dequeue(q, x);
    }
}
```

(b)

```
// ==== CẤU TRÚC DANCER VÀ QUEUE ====

struct Dancer {
    char Name[100];
    char Sex; // 'M' hoặc 'F'
};

struct Node {
    Dancer data;
    Node* next;
};

struct Queue {
    Node *front, *rear;
    int count;
};

void InitQueue(Queue &q) {
    q.front = q.rear = nullptr;
    q.count = 0;
}

bool IsEmpty(Queue q) {
    return q.front == nullptr;
}

void Enqueue(Queue &q, Dancer d) {
    Node* p = new Node{d, nullptr};
    if (IsEmpty(q))
        q.front = q.rear = p;
    else {
        q.rear->next = p;
        q.rear = p;
```

```
    }
    q.count++;
}

bool Dequeue(Queue &q, Dancer &d) {
    if (IsEmpty(q)) return false;
    Node* p = q.front;
    d = p->data;
    q.front = p->next;
    if (q.front == nullptr) q.rear = nullptr;
    delete p;
    q.count--;
    return true;
}

bool Peek(Queue q, Dancer &d) {
    if (IsEmpty(q)) return false;
    d = q.front->data;
    return true;
}

void ShowQueue(Queue q) {
    Node* p = q.front;
    while (p != nullptr) {
        cout << p->data.Name << " ";
        p = p->next;
    }
    cout << endl;
}

// Đưa vào hàng đợi từ danh sách tên
void FormLines(Queue &male, Queue &female) {
    const char* 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 = 12;

    for (int i = 0; i < n; i++) {
        Dancer d;
        d.Sex = dancers[i][0]; // M hoặc F
        strcpy(d.Name, dancers[i] + 2);

        if (d.Sex == 'M') Enqueue(male, d);
        else Enqueue(female, d);
    }
```

```
}

// Ghép tối đa 4 cặp
void StartDancing(Queue &male, Queue &female) {
    cout << "\n--- CAC CAP NHAY TREN SAN ---\n";

    for (int i = 0; i < 4; i++) {
        if (male.count > 0 && female.count > 0) {
            Dancer m, f;
            Dequeue(male, m);
            Dequeue(female, f);
            cout << m.Name << " - " << f.Name << endl;
        }
    }

    if (male.count > 0 || female.count > 0) {
        cout << "\n--- NHUNG NGUOI DANG CHO ---\n";
        if (male.count > 0) {
            Dancer m;
            Peek(male, m);
            cout << "Dien vien nam dang cho: " << m.Name << endl;
        }
        if (female.count > 0) {
            Dancer f;
            Peek(female, f);
            cout << "Dien vien nu dang cho: " << f.Name << endl;
        }
    }
}

// Main
int main() {
    Queue males, females;
    InitQueue(males);
    InitQueue(females);

    FormLines(males, females);

    cout << "Danh sach dien vien nam: ";
    ShowQueue(males);
    cout << "Danh sach dien vien nu: ";
    ShowQueue(females);

    StartDancing(males, females);

    return 0;
}
```

(c)

```
struct Node {
    int data;
    Node *next;
};

struct Queue {
    Node *front, *rear;
    int count;
};

void InitQueue(Queue &q) {
    q.front = q.rear = nullptr;
    q.count = 0;
}

bool IsEmpty(Queue q) {
    return q.front == nullptr;
}

void Enqueue(Queue &q, int x) {
    Node *p = new Node{x, nullptr};
    if (IsEmpty(q)) q.front = q.rear = p;
    else {
        q.rear->next = p;
        q.rear = p;
    }
    q.count++;
}

int Dequeue(Queue &q) {
    if (IsEmpty(q)) return -1;
    Node *p = q.front;
    int x = p->data;
    q.front = p->next;
    if (q.front == nullptr) q.rear = nullptr;
    delete p;
    q.count--;
    return x;
}

// Hàm hiển thị mảng
void DisplayArray(int a[], int n) {
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
```

```
}

// Bước RadixSort với từng chữ số
void RadixSort(Queue bin[], int a[], int n, int digit) {
    for (int i = 0; i < n; i++) {
        int num = (digit == 1) ? a[i] % 10 : (a[i] / 10) % 10;
        Enqueue(bin[num], a[i]);
    }

    // Gom lại dãy từ các bin
    int idx = 0;
    for (int i = 0; i < 10; i++) {
        while (!IsEmpty(bin[i])) {
            a[idx++] = Dequeue(bin[i]);
        }
    }
}

// Main
int main() {
    int a[] = {91, 46, 85, 15, 92, 35, 31, 22};
    int n = sizeof(a) / sizeof(int);

    Queue bin[10];
    for (int i = 0; i < 10; i++) InitQueue(bin[i]);

    cout << "Day ban dau: ";
    DisplayArray(a, n);

    // Bước 1: theo hàng đơn vị
    RadixSort(bin, a, n, 1);
    cout << "Sau buoc 1 (hang don vi): ";
    DisplayArray(a, n);

    // Bước 2: theo hàng chục
    RadixSort(bin, a, n, 10);
    cout << "Sau buoc 2 (hang chuc): ";
    DisplayArray(a, n);

    return 0;
}
```

Bài tập mở rộng

Bài 1. Dùng stack khử đệ qui
(a) Viết chương trình đệ qui / khử đệ qui (dùng stack) cho bài toán tính số Fibonancy, đảo ngược số

(b) Viết chương trình đệ qui / khử đệ qui (dùng stack) cho bài toán đảo ngược số

(c) HanoiTower là một tháp có nhiều tầng, tầng nhỏ nằm trên tầng lớn. Sau đó viết phương thức di chuyển tháp này từ vị trí 1 đến vị trí 3 thông qua vị trí trung gian 2; mỗi lần chỉ được di chuyển tầng trên cùng của tháp và tại mỗi vị trí 1, 2, và 3 đều là tháp. Hãy viết chương trình đệ qui / khử đệ qui cho bài toán tháp Hà Nội.

Bài làm

- Ý tưởng
    - Đệ quy
        - Câu a.
            - Hàm Fibonacci(n) được định nghĩa theo công thức:
              $$F(n) = F(n-1) + F(n-2),\ F(0) = 0,\ F(1) = 1$$
            - Gọi đệ quy đến khi n == 0 hoặc n == 1, sau đó kết quả được cộng dồn quay ngược lại.
        - Câu b.
            - Lấy chữ số cuối n % 10 và in ra trước.
            - Gọi lại hàm với phần còn lại n / 10.
            - Khi còn 1 chữ số (n < 10), in ra và kết thúc.
        - Câu c.
            - Để di chuyển n đĩa từ cọc A → C (qua B):
                - ❖ Di chuyển n - 1 đĩa từ A → B (qua C).
                - ❖ Di chuyển đĩa lớn nhất n từ A → C.
                - ❖ Di chuyển n - 1 đĩa từ B → C (qua A).
            - Áp dụng đệ quy cho mỗi bước.
    - Khử đệ quy bằng stack
        - Câu a.
            - Dùng stack để mô phỏng quá trình gọi đệ quy từ n giảm về 2.
            - Bắt đầu với a = 0, b = 1, rồi duyệt lại stack để cộng dồn như công thức Fibonacci.
            - Mỗi lần lặp: f(n) = f(n-1) + f(n-2) → thực hiện tương đương khi pop() ra khỏi stack.
        - Câu b.
            - Dùng stack để lưu từng chữ số của số n.
            - Sau đó lần lượt pop ra và in, nhờ đó số được in ngược thứ tự ban đầu.
        - Câu c.

- Tạo struct Frame để mô phỏng từng "lời gọi hàm".
- Stack giữ các Frame, trong đó mỗi Frame chứa:
  - Số lượng đĩa n
  - Nguồn, trung gian, đích (A, B, C)
  - step đánh dấu trạng thái xử lý: gọi lần đầu, đã in, gọi lần 2.
- Dựa vào step, mô phỏng lần lượt từng bước như đệ quy thật.

- Mã nguồn

Đệ quy

(a)

```
// (a) Đệ quy tính Fibonacci
int Fibonacci(int n) {
    if (n <= 1) return n;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

(b)

```
// (b) Đệ quy đảo ngược số
void DaoNguoc_DeQuy(int n) {
    if (n < 10) {
        cout << n;
        return;
    }
    cout << n % 10;
    DaoNguoc_DeQuy(n / 10);
}
```

(c)

```
// (c) Đệ quy giải bài toán Tháp Hà Nội
void Hanoi_DeQuy(int n, char A, char B, char C) {
    if (n == 1) {
        cout << "Di chuyen dia 1 tu " << A << " sang " << C << endl;
        return;
    }
    Hanoi_DeQuy(n - 1, A, C, B);
    cout << "Di chuyen dia " << n << " tu " << A << " sang " << C << endl;
    Hanoi_DeQuy(n - 1, B, A, C);
}
```

Khử đệ quy bằng stack

(a)

```
// Khử đệ quy tính Fibonacci bằng stack
```

```cpp
int Fibonacci_Stack(int n) {
    if (n <= 1) return n;

    stack<int> s;
    for (int i = n; i > 1; i--)
        s.push(i);

    int a = 0, b = 1;
    while (!s.empty()) {
        int temp = b;
        b = a + b;
        a = temp;
        s.pop();
    }
    return b;
}
```

(b)

```cpp
// Khử đệ quy đảo ngược số bằng stack
void DaoNguoc_Stack(int n) {
    stack<int> s;

    while (n > 0) {
        s.push(n % 10);
        n /= 10;
    }

    while (!s.empty()) {
        cout << s.top();
        s.pop();
    }
}
```

(c)

```cpp
// Khử đệ quy giải bài toán Tháp Hà Nội bằng stack
struct Frame {
    int n;
    char A, B, C;
    int step;
};

void Hanoi_Stack(int n, char A, char B, char C) {
    stack<Frame> s;
    s.push({n, A, B, C, 0});

    while (!s.empty()) {
        Frame &f = s.top();
```

```
        if (f.n == 1) {
            cout << "Di chuyen dia 1 tu " << f.A << " sang " << f.C << endl;
            s.pop();
            continue;
        }

        if (f.step == 0) {
            s.push({f.n - 1, f.A, f.C, f.B, 0});
            f.step++;
        }
        else if (f.step == 1) {
            cout << "Di chuyen dia " << f.n << " tu " << f.A << " sang " << f.C <<
endl;
            f.step++;
        }
        else if (f.step == 2) {
            s.push({f.n - 1, f.B, f.A, f.C, 0});
            f.step++;
        }
        else {
            s.pop();
        }
    }
}
```