**Họ tên:** Nguyễn Thị Thảo Nguyên

**MSSV:** 3121411150

**Bài 1:**

* **Đảo ngược số**
* **Ý tưởng**:
  + - Đẩy từng chữ số của số vào ngăn xếp.
    - Lấy lần lượt các chữ số ra khỏi ngăn xếp để tạo số đảo ngược.
* **Kiểm tra chuỗi đối xứng**
* **Ý tưởng**:
  + Đẩy tất cả các ký tự của chuỗi vào ngăn xếp.
  + Lấy từng ký tự từ ngăn xếp ra và so sánh với chuỗi ban đầu.
  + Nếu mọi ký tự đều khớp, chuỗi là đối xứng.
* **Chuyển từ số thập phân sang nhị phân**
* **Ý tưởng**:
* Chia số thập phân cho 2, đẩy phần dư (% 2) vào ngăn xếp.
* Tiếp tục chia cho đến khi số bằng 0.
* Lấy từng phần dư từ ngăn xếp ra và ghép lại thành chuỗi nhị phân
* **Chuyển biểu thức trung tố sang hậu tố**
* **Ý tưởng**:
  + Duyệt từng ký tự trong biểu thức.
  + Nếu ký tự là số, thêm vào kết quả.
  + Nếu ký tự là dấu ngoặc hoặc toán tử, xử lý dựa trên ưu tiên:
  + Toán tử có độ ưu tiên cao sẽ được đẩy vào ngăn xếp trước.
  + Khi gặp dấu ngoặc đóng ), lấy các toán tử ra cho đến khi gặp dấu ngoặc mở (.
  + Sau khi duyệt hết, lấy tất cả toán tử còn lại trong ngăn xếp ra.
* **Tính giá trị biểu thức hậu tố**
* **Ý tưởng**:
  + - Duyệt từng ký tự trong biểu thức hậu tố.
    - Nếu ký tự là số, đẩy vào ngăn xếp.
    - Nếu ký tự là toán tử, lấy 2 số từ ngăn xếp, thực hiện phép toán và đẩy kết quả vào lại ngăn xếp.
    - Sau khi duyệt hết, kết quả cuối cùng trong ngăn xếp là giá trị của biểu thức.
* **Mã nguồn**

#include <iostream>

#include <string>

#include <stack>

#include <cmath>

#include <cctype>

using namespace std;

#define MAX 100 // Giới hạn kích thước của ngăn xếp

// ----------------------------

// (a) Cài đặt ngăn xếp bằng mảng (StackInt)

int stackArr[MAX];

int topArr = -1;

bool IsEmptyArr() {

    return topArr == -1;

}

bool IsFullArr() {

    return topArr == MAX - 1;

}

void PushArr(int value) {

    if (IsFullArr()) {

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

    } else {

        stackArr[++topArr] = value;

    }

}

int PopArr() {

    if (IsEmptyArr()) {

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

        return -1;

    } else {

        return stackArr[topArr--];

    }

}

int PeekArr() {

    if (IsEmptyArr()) {

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

        return -1;

    } else {

        return stackArr[topArr];

    }

}

void ClearArr() {

    topArr = -1;

}

// ----------------------------

// (b) Cài đặt ngăn xếp bằng danh sách liên kết (LinkedStackInt)

struct Node {

    int data;

    Node\* next;

};

Node\* topLinked = nullptr;

bool IsEmptyLinked() {

    return topLinked == nullptr;

}

void PushLinked(int value) {

    Node\* newNode = new Node();

    newNode->data = value;

    newNode->next = topLinked;

    topLinked = newNode;

}

int PopLinked() {

    if (IsEmptyLinked()) {

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

        return -1;

    } else {

        Node\* temp = topLinked;

        int value = temp->data;

        topLinked = topLinked->next;

        delete temp;

        return value;

    }

}

int PeekLinked() {

    if (IsEmptyLinked()) {

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

        return -1;

    } else {

        return topLinked->data;

    }

}

void ClearLinked() {

    while (!IsEmptyLinked()) {

        PopLinked();

    }

}

// ----------------------------

// (c) Ứng dụng ngăn xếp để đảo số (Sử dụng ngăn xếp mảng)

void ReverseNumber(int num) {

    ClearArr();

    while (num > 0) {

        PushArr(num % 10); // Đẩy từng chữ số vào stack

        num /= 10;

    }

    int reversedNum = 0;

    int multiplier = 1;

    while (!IsEmptyArr()) {

        reversedNum += PopArr() \* multiplier;

        multiplier \*= 10;

    }

    cout << "Reversed number: " << reversedNum << endl;

}

// ----------------------------

// (d) Ứng dụng ngăn xếp để kiểm tra xâu đối xứng (StackString)

bool IsPalindrome(const string& str) {

    ClearArr();

    int len = str.length();

    // Đẩy các ký tự vào ngăn xếp

    for (int i = 0; i < len; i++) {

        PushArr(str[i]);

    }

    // Kiểm tra tính đối xứng

    for (int i = 0; i < len; i++) {

        if (str[i] != PopArr()) {

            return false;

        }

    }

    return true;

}

// ----------------------------

// (e) Ứng dụng ngăn xếp để đổi từ số thập phân sang nhị phân

void DecimalToBinary(int num) {

    ClearArr();

    while (num > 0) {

        PushArr(num % 2);

        num /= 2;

    }

    cout << "Binary: ";

    while (!IsEmptyArr()) {

        cout << PopArr();

    }

    cout << endl;

}

// ----------------------------

// (f) Chuyển đổi biểu thức từ tiền tố sang hậu tố và tính giá trị biểu thức

// Hàm kiểm tra nếu một ký tự là toán tử

bool IsOperator(char c) {

    return (c == '+' || c == '-' || c == '\*' || c == '/');

}

// Hàm tính toán

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;

    }

}

// Hàm kiểm tra độ ưu tiên của toán tử

int Precedence(char op) {

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

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

    return 0;

}

// Chuyển đổi biểu thức từ trung tố sang hậu tố

string InfixToPostfix(string expr) {

    stack<char> operators;

    string postfix = "";

    for (int i = 0; i < expr.length(); i++) {

        char c = expr[i];

        if (isdigit(c)) {

            postfix += c;

        } else if (c == '(') {

            operators.push(c);

        } else if (c == ')') {

            while (!operators.empty() && operators.top() != '(') {

                postfix += operators.top();

                operators.pop();

            }

            operators.pop();

        } else if (IsOperator(c)) {

            while (!operators.empty() && Precedence(operators.top()) >= Precedence(c)) {

                postfix += operators.top();

                operators.pop();

            }

            operators.push(c);

        }

    }

    while (!operators.empty()) {

        postfix += operators.top();

        operators.pop();

    }

    return postfix;

}

// Tính giá trị của biểu thức hậu tố

int EvaluatePostfix(string expr) {

    stack<int> values;

    for (int i = 0; i < expr.length(); i++) {

        char c = expr[i];

        if (isdigit(c)) {

            values.push(c - '0');

        } else if (IsOperator(c)) {

            int val2 = values.top(); values.pop();

            int val1 = values.top(); values.pop();

            values.push(ApplyOperator(val1, val2, c));

        }

    }

    return values.top();

}

// ----------------------------

// Hàm main

int main() {

    int choice, num;

    string expr, str;

    while (true) {

        cout << "Menu:" << endl;

        cout << "1. Reverse a number using stack" << endl;

        cout << "2. Check if a string is palindrome using stack" << endl;

        cout << "3. Convert decimal to binary using stack" << endl;

        cout << "4. Infix to Postfix conversion and evaluate the expression" << endl;

        cout << "0. Exit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter a number: ";

            cin >> num;

            ReverseNumber(num);

        } else if (choice == 2) {

            cout << "Enter a string: ";

            cin >> str;

            if (IsPalindrome(str)) {

                cout << "The string is a palindrome." << endl;

            } else {

                cout << "The string is not a palindrome." << endl;

            }

        } else if (choice == 3) {

            cout << "Enter a decimal number: ";

            cin >> num;

            DecimalToBinary(num);

        } else if (choice == 4) {

            cout << "Enter an infix expression (e.g., 3+(2\*5)): ";

            cin >> expr;

            string postfix = InfixToPostfix(expr);

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

            cout << "Result of the postfix expression: " << EvaluatePostfix(postfix) << endl;

        } else if (choice == 0) {

            cout << "Exiting program..." << endl;

            break;

        } else {

            cout << "Invalid choice!" << endl;

        }

    }

    return 0;

}

**Bài 2**

* **Ứng dụng xếp lịch cặp múa nam/nữ**
* **Ý tưởng** 
  + - Có hai hàng đợi, một cho danh sách các bạn nam, một cho danh sách các bạn nữ.
    - Ghép đôi các bạn nam và nữ lần lượt từ đầu hàng, in kết quả ghép đôi.
    - Nếu một trong hai hàng đợi còn lại phần tử, in ra các bạn chưa được ghép cặp.
* **Ứng dụng hàng đợi trong RadixSort**
* **Ý tưởng** 
  + RadixSort là thuật toán sắp xếp không cần so sánh (non-comparison sorting).
  + Số nguyên được sắp xếp dựa trên từng chữ số của nó (từ hàng đơn vị, chục, trăm...).
  + Dùng 10 hàng đợi để phân loại các số theo chữ số tương ứng.
* **Mã nguồn**

#include <iostream>

#include <cstring>

#include <queue>

#include <cmath>

using namespace std;

#define MAX 100

// ==============================

// Cài đặt QueueInt dùng mảng

int stackArr[MAX];

int topArr = -1;

void InitStackArr() {

    topArr = -1;

}

bool IsEmptyArr() {

    return topArr == -1;

}

bool IsFullArr() {

    return topArr == MAX - 1;

}

void PushStackArr(int value) {

    if (IsFullArr()) {

        cout << "Error: Stack is full! Cannot push value." << endl;

    } else {

        stackArr[++topArr] = value;

    }

}

int PopStackArr() {

    if (IsEmptyArr()) {

        cout << "Error: Stack is empty! Cannot pop value." << endl;

        return -1;

    } else {

        return stackArr[topArr--];

    }

}

int PeekStackArr() {

    if (IsEmptyArr()) {

        cout << "Error: Stack is empty! Cannot peek value." << endl;

        return -1;

    } else {

        return stackArr[topArr];

    }

}

void ClearStackArr() {

    topArr = -1;

}

// ==============================

// Cài đặt LinkedQueueInt dùng danh sách liên kết

struct Node {

    int data;

    Node\* next;

};

Node\* topLinked = nullptr;

void InitStackLinked() {

    topLinked = nullptr;

}

bool IsEmptyLinked() {

    return topLinked == nullptr;

}

void PushStackLinked(int value) {

    Node\* newNode = new Node();

    if (!newNode) {

        cout << "Error: Memory allocation failed!" << endl;

        return;

    }

    newNode->data = value;

    newNode->next = topLinked;

    topLinked = newNode;

}

int PopStackLinked() {

    if (IsEmptyLinked()) {

        cout << "Error: Stack is empty! Cannot pop value." << endl;

        return -1;

    } else {

        Node\* temp = topLinked;

        int value = temp->data;

        topLinked = topLinked->next;

        delete temp;

        return value;

    }

}

int PeekStackLinked() {

    if (IsEmptyLinked()) {

        cout << "Error: Stack is empty! Cannot peek value." << endl;

        return -1;

    } else {

        return topLinked->data;

    }

}

void ClearStackLinked() {

    while (!IsEmptyLinked()) {

        PopStackLinked();

    }

}

// ==============================

// Thuật toán RadixSort sử dụng Queue

int GetDigit(int num, int digitPos) {

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

}

void RadixSort(int arr[], int n) {

    if (n <= 0) {

        cout << "Error: Array size must be greater than 0!" << endl;

        return;

    }

    int maxElem = arr[0];

    for (int i = 1; i < n; i++) {

        if (arr[i] > maxElem) {

            maxElem = arr[i];

        }

    }

    int maxDigits = 0;

    while (maxElem > 0) {

        maxDigits++;

        maxElem /= 10;

    }

    for (int d = 0; d < maxDigits; d++) {

        queue<int> buckets[10];

        for (int i = 0; i < n; i++) {

            int digit = GetDigit(arr[i], d);

            buckets[digit].push(arr[i]);

        }

        int idx = 0;

        for (int i = 0; i < 10; i++) {

            while (!buckets[i].empty()) {

                arr[idx++] = buckets[i].front();

                buckets[i].pop();

            }

        }

    }

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << endl;

}

// ==============================

// Ứng dụng quản lý xếp hàng múa

struct Dancer {

    char Name[100];

    char Sex[10];

};

void GetName(Dancer &d, const char \*s) {

    strcpy(d.Name, s);

}

const char\* ToString(Dancer &d) {

    return d.Name;

}

struct Queue {

    queue<Dancer> q;

    bool IsEmpty() {

        return q.empty();

    }

    void Enqueue(const Dancer &d) {

        q.push(d);

    }

    Dancer Dequeue() {

        if (q.empty()) {

            cout << "Error: Queue is empty! Cannot dequeue." << endl;

            return {"", ""};

        }

        Dancer d = q.front();

        q.pop();

        return d;

    }

    Dancer Peek() {

        if (q.empty()) {

            cout << "Error: Queue is empty! Cannot peek." << endl;

            return {"", ""};

        }

        return q.front();

    }

    int Count() {

        return q.size();

    }

    void ShowQueue() {

        if (IsEmpty()) {

            cout << "Hang doi trong." << endl;

            return;

        }

        queue<Dancer> tempQueue = q;

        while (!tempQueue.empty()) {

            Dancer d = tempQueue.front();

            tempQueue.pop();

            cout << d.Name << endl;

        }

    }

};

void NewDancers(Queue &male, Queue &female) {

    if (male.Count() > 0 && female.Count() > 0) {

        Dancer m = male.Dequeue();

        Dancer w = female.Dequeue();

        printf("Cap dien vien: %s va %s\n", m.Name, w.Name);

    } else if (male.Count() > 0 && female.Count() == 0) {

        printf("Dang cho dien vien nu.\n");

    } else if (female.Count() > 0 && male.Count() == 0) {

        printf("Dang cho dien vien nam.\n");

    } else {

        printf("Khong co dien vien nao trong hang cho.\n");

    }

}

void StartDancing(Queue &male, Queue &female) {

    printf("Cac cap dien vien:\n");

    while (male.Count() > 0 && female.Count() > 0) {

        Dancer m = male.Dequeue();

        Dancer w = female.Dequeue();

        printf("%s \t %s\n", m.Name, w.Name);

    }

    if (male.Count() > 0) {

        printf("Van con %d dien vien nam trong hang cho:\n", male.Count());

        male.ShowQueue();

    }

    if (female.Count() > 0) {

        printf("Van con %d dien vien nu trong hang cho:\n", female.Count());

        female.ShowQueue();

    }

}

void FormLines(Queue &male, Queue &female) {

    int n;

    cout << "Nhap so luong dien vien: ";

    cin >> n;

    if (n <= 0) {

        cout << "Error: So luong dien vien phai lon hon 0!" << endl;

        return;

    }

    Dancer d;

    for (int i = 0; i < n; i++) {

        cout << "Nhap thong tin dien vien thu " << i + 1 << " (gioi tinh M/F va ten): ";

        cin >> d.Sex >> d.Name;

        if (strcmp(d.Sex, "M") == 0) {

            male.Enqueue(d);

        } else if (strcmp(d.Sex, "F") == 0) {

            female.Enqueue(d);

        } else {

            cout << "Error: Gioi tinh khong hop le! Vui long nhap lai." << endl;

            i--; // Retry current input

        }

    }

}

int main() {

    Queue males;

    Queue females;

    int choice = 0;

    while (true) {

        cout << "==============================\n";

        cout << "Menu\n";

        cout << "1. Quan ly xep hang mua\n";

        cout << "2. RadixSort\n";

        cout << "3. Thoat\n";

        cout << "==============================\n";

        cout << "Nhap lua chon cua ban: ";

        cin >> choice;

        if (choice == 1) {

            FormLines(males, females);

            cout << "\nDanh sach dien vien nam:\n";

            males.ShowQueue();

            cout << "\nDanh sach dien vien nu:\n";

            females.ShowQueue();

            StartDancing(males, females);

        } else if (choice == 2) {

            int n;

            cout << "Nhap so luong phan tu: ";

            cin >> n;

            if (n <= 0) {

                cout << "Error: So luong phan tu phai lon hon 0!" << endl;

                continue;

            }

            int arr[n];

            cout << "Nhap cac phan tu: ";

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

                cin >> arr[i];

            }

            RadixSort(arr, n);

        } else if (choice == 3) {

            cout << "Thoat chuong trinh. Tam biet!\n";

            break;

        } else {

            cout << "Error: Lua chon khong hop le. Vui long nhap lai." << endl;

        }

    }

    return 0;

}