Chương 2: Cấu trúc dữ liệu tuyến tính (tiếp)

3. NGĂN XẾP (STACK)

Lý thuyết:

- Là cấu trúc dữ liệu **LIFO** (Last In, First Out - vào sau ra trước).

Các thao tác chính:

- push(x): Thêm phần tử x vào đỉnh ngăn xếp.
- pop(): Xóa phần tử ở đỉnh ngăn xếp.
- top()/peek(): Xem phần tử ở đỉnh.
- isEmpty(): Kiểm tra ngăn xếp rỗng.
- size(): Trả về số phần tử trong stack.

Ứng dụng:

- Duyệt biểu thức toán học (infix, postfix).
- Gọi hàm đệ quy.
- Quay lui (backtracking).
- Undo/Redo trong ứng dụng.

Cài đặt bằng mảng:

```
#include <iostream>
using namespace std;

const int MAX = 100;

class Stack {
private:
   int arr[MAX];
   int topIndex;

public:
   Stack() {
      topIndex = -1;
   }

   void push(int x) {
      if (topIndex >= MAX - 1) {
         cout << "Stack overflow\n";
         return;
      }
      arr[++topIndex] = x;
   }
```

```
    void pop() {
        if (topIndex < 0) {
            cout << "Stack underflow\n";
            return;
        }
        topIndex--;
    }

    int top() {
        if (topIndex < 0) {
            cout << "Stack rỗng\n";
            return -1;
        }
        return arr[topIndex];
    }

    bool isEmpty() {
        return topIndex == -1;
    }

    int size() {
        return topIndex + 1;
    }

    void printStack() {
        for (int i = topIndex; i >= 0; i--) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Stack st;
    st.push(10);
    st.push(20);
    st.push(30);

    cout << "Đỉnh stack: " << st.top() << endl;
    st.printStack(); // 30 20 10

    st.pop();
    st.printStack(); // 20 10

    cout << "Size: " << st.size() << endl;
    return 0;
}
```

4. HÀNG ĐỢI (QUEUE)

Lý thuyết:

- Là cấu trúc dữ liệu **FIFO** (First In, First Out - vào trước ra trước).

Các thao tác chính:

- enqueue(x): Thêm phần tử vào cuối hàng đợi (rear).
- dequeue(): Xóa phần tử khỏi đầu hàng đợi (front).
- front(): Truy cập phần tử ở đầu hàng.
- isEmpty(): Kiểm tra hàng đợi rỗng.
- size(): Trả về số lượng phần tử trong hàng.

Ứng dụng:

- Quản lý tiến trình trong hệ điều hành.
- Mô phỏng hàng chờ.
- Hệ thống xử lý yêu cầu theo thứ tự.

Cài đặt bằng mảng:

```
#include <iostream>

using namespace std;


const int MAX = 100;


class Queue {

private:

   int arr[MAX];

   int frontIndex, rearIndex, count;


public:

   Queue() {

      frontIndex = 0;

      rearIndex = -1;

      count = 0;

   }
```

```
void enqueue(int x) {
    if (count == MAX) {
        cout << "Queue đầy\n";
        return;
    }
    rearIndex = (rearIndex + 1) % MAX;
    arr[rearIndex] = x;
    count++;
}

void dequeue() {
    if (isEmpty()) {
        cout << "Queue rỗng\n";
        return;
    }
    frontIndex = (frontIndex + 1) % MAX;
    count--;
}

int front() {
    if (isEmpty()) {
        cout << "Queue rỗng\n";
        return -1;
    }
    return arr[frontIndex];
}

bool isEmpty() {
    return count == 0;
}
```

```
    int size() {
        return count;
    }

    void printQueue() {
        for (int i = 0; i < count; i++) {
            int index = (frontIndex + i) % MAX;
            cout << arr[index] << " ";
        }
        cout << endl;
    }
};

int main() {
    Queue q;
    q.enqueue(1);
    q.enqueue(2);
    q.enqueue(3);
    q.printQueue(); // 1 2 3

    q.dequeue();
    q.printQueue(); // 2 3

    cout << "Phần tử đầu: " << q.front() << endl;
    cout << "Kích thước: " << q.size() << endl;

    return 0;
}
```