# Chương 2: Cấu trúc dữ liệu

# 1. MẢNG (Array)

## Lý thuyết:

* Mảng là tập hợp các phần tử **cùng kiểu dữ liệu**, được lưu trữ **liên tiếp trong bộ nhớ**.
* Mỗi phần tử được truy cập thông qua **chỉ số** (bắt đầu từ 0).
* Mảng có **kích thước cố định** khi khai báo.
* Nhược điểm: **khó chèn/xóa** phần tử ở giữa vì phải dịch chuyển các phần tử.

## Cài đặt cơ bản bằng C++:

#include <iostream>

using namespace std;

int main() {

int arr[5]; // Khai báo mảng 5 phần tử

// Gán giá trị cho từng phần tử

for (int i = 0; i < 5; i++) {

arr[i] = i \* 10;

}

// In mảng

for (int i = 0; i < 5; i++) {

cout << "arr[" << i << "] = " << arr[i] << endl;

}

return 0;

}

# 2. DANH SÁCH LIÊN KẾT ĐƠN (Singly Linked List)

## Lý thuyết:

* Gồm các nút (node), mỗi nút chứa:
  + Dữ liệu (data)
  + Con trỏ trỏ đến nút kế tiếp (next)
* Ưu điểm:
  + **Kích thước động** (không cần xác định trước).
  + **Chèn/xóa dễ dàng** tại vị trí bất kỳ.
* Nhược điểm: Truy cập phần tử **không ngẫu nhiên**, phải duyệt từ đầu danh sách.

## Cài đặt cơ bản bằng C++:

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = nullptr;

}

// Thêm phần tử vào đầu danh sách

void insertFirst(int value) {

Node\* newNode = new Node{value, head};

head = newNode;

}

// Thêm phần tử vào cuối danh sách

void insertLast(int value) {

Node\* newNode = new Node{value, nullptr};

if (head == nullptr) {

head = newNode;

return;

}

Node\* temp = head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

}

// Thêm phần tử vào vị trí k (tính từ 0)

void insertAt(int pos, int value) {

if (pos == 0) {

insertFirst(value);

return;

}

Node\* temp = head;

for (int i = 0; i < pos - 1 && temp != nullptr; i++)

temp = temp->next;

if (temp == nullptr) {

cout << "Vị trí không hợp lệ\n";

return;

}

Node\* newNode = new Node{value, temp->next};

temp->next = newNode;

}

// Xóa phần tử đầu tiên

void deleteFirst() {

if (head == nullptr) return;

Node\* temp = head;

head = head->next;

delete temp;

}

// Xóa phần tử cuối cùng

void deleteLast() {

if (head == nullptr) return;

if (head->next == nullptr) {

delete head;

head = nullptr;

return;

}

Node\* temp = head;

while (temp->next->next != nullptr)

temp = temp->next;

delete temp->next;

temp->next = nullptr;

}

// Xóa phần tử có giá trị cụ thể

void deleteByValue(int value) {

if (head == nullptr) return;

if (head->data == value) {

deleteFirst();

return;

}

Node\* temp = head;

while (temp->next != nullptr && temp->next->data != value)

temp = temp->next;

if (temp->next == nullptr) {

cout << "Không tìm thấy giá trị.\n";

return;

}

Node\* toDelete = temp->next;

temp->next = toDelete->next;

delete toDelete;

}

// Tìm kiếm giá trị

bool search(int value) {

Node\* temp = head;

while (temp != nullptr) {

if (temp->data == value) return true;

temp = temp->next;

}

return false;

}

// Đếm số nút

int count() {

int cnt = 0;

Node\* temp = head;

while (temp != nullptr) {

cnt++;

temp = temp->next;

}

return cnt;

}

// In danh sách

void printList() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL\n";

}

// Giải phóng bộ nhớ khi xóa danh sách

~LinkedList() {

while (head != nullptr) {

deleteFirst();

}

}

};

int main() {

LinkedList list;

list.insertFirst(10);

list.insertLast(20);

list.insertLast(30);

list.insertAt(1, 15); // chèn 15 vào vị trí 1

list.printList(); // 10 -> 15 -> 20 -> 30 -> NULL

list.deleteByValue(20);

list.printList(); // 10 -> 15 -> 30 -> NULL

cout << "Tìm 15? " << (list.search(15) ? "Có\n" : "Không\n"); // Có

cout << "Số nút: " << list.count() << endl; // 3

list.deleteFirst();

list.deleteLast();

list.printList(); // 15 -> NULL

return 0;

}