**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP HỒ CHÍ MINH**

A blue and yellow logo

AI-generated content may be incorrect.

**BÁO CÁO MÔN HỌC**

**KỸ THUẬT LẬP TRÌNH**

**BÀI TẬP LỚN: TỔNG HỢP KIẾN THỨC**

Giảng viên hướng dẫn: **ThS. TRẦN PHONG NHÃ**

Nhóm SV thực hiện:

1. **HUỲNH TRUNG NGHĨA MSSV: 6551071057**
2. **VÕ HỒNG THIÊN MSSV: 6551071078**
3. **DƯ NGUYÊN AN MSSV: 6551071001**

Lớp: **CQ.CNTT.65**

Khóa: **65**

TP Hồ Chí Minh, tháng 5 năm 2025

**LỜI CẢM ƠN**

Lời đầu tiên, em xin gửi lời chúc sức khỏe và lòng biết ơn sâu sắc tới Quý Thầy Cô Bộ môn Công nghệ Thông tin, Trường Đại học Giao thông Vận tải Phân hiệu tại Thành phố Hồ Chí Minh.

Em xin chân thành cảm ơn Quý Thầy Cô đã tạo điều kiện thuận lợi để em có thể hoàn thành báo cáo. Đặc biệt, em xin gửi lời cảm ơn sâu sắc tới thầy Trần Phong Nhã, người đã tận tình giúp đỡ, hướng dẫn em về kiến thức, định hướng, và kỹ năng cần thiết để hoàn thành bài báo cáo này.

Dù đã nỗ lực trong quá trình nghiên cứu, nhưng với hạn chế về kiến thức, chắc chắn báo cáo còn nhiều thiếu sót. Em rất mong nhận được sự đóng góp ý kiến từ Quý Thầy Cô để bài báo cáo được hoàn thiện hơn.

Cuối cùng, em xin kính chúc Quý Thầy Cô Bộ môn Công nghệ Thông tin, đặc biệt là thầy Trần Phong Nhã, luôn dồi dào sức khỏe và đạt nhiều thành công trong công việc. Em xin chân thành cảm ơn!

# **NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

|  |  |
| --- | --- |
|  | **Tp. Hồ Chí Minh, ngày.… tháng … năm 2025**  **Giảng viên hướng dẫn** |
|  | **ThS.Trần Phong Nhã** |

**MỤC LỤC**

[LỜI CẢM ƠN 1](#_Toc198894845)

[NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN 2](#_Toc198894846)

[PHẦN A: LÝ THUYẾT 7](#_Toc198894847)

[CHƯƠNG 1: HÀM (FUNCTION) 7](#_Toc198894848)

[1. Định nghĩa Hàm 7](#_Toc198894849)

[2. Cấu trúc hàm 7](#_Toc198894850)

[3. Ý nghĩa của từng phần 8](#_Toc198894851)

[4. Nguyên mẫu hàm (Function Prototype) 8](#_Toc198894852)

[5. Gọi hàm (Function Call) 8](#_Toc198894853)

[6. Hàm không trả về giá trị (Kiểu void) 9](#_Toc198894854)

[7. Truyền tham trị (Call by Value) 9](#_Toc198894855)

[8. Phạm vi biến (Scope) 9](#_Toc198894856)

[9. Lợi ích của hàm 10](#_Toc198894857)

[10. Ví dụ minh họa 10](#_Toc198894858)

[CHƯƠNG 2: CON TRỎ 11](#_Toc198894859)

[1. Khái niệm Con trỏ 11](#_Toc198894860)

[2. Khai báo con trỏ 11](#_Toc198894861)

[3. Toán tử liên quan đến con trỏ 11](#_Toc198894862)

[4. Ví dụ sử dụng con trỏ 11](#_Toc198894863)

[5. Truyền tham chiếu bằng con trỏ (Call by Reference) 11](#_Toc198894864)

[6. Ví dụ truyền tham chiếu 12](#_Toc198894865)

[7. So sánh truyền tham trị và tham chiếu 12](#_Toc198894866)

[8. Ứng dụng con trỏ trong C 12](#_Toc198894867)

[9. Lưu ý khi dùng con trỏ 12](#_Toc198894868)

[10. Ví dụ minh họa 13](#_Toc198894869)

[CHƯƠNG 3: CON TRỎ MẢNG (ARRAY POINTER) 14](#_Toc198894870)

[1. Khái niệm 14](#_Toc198894871)

[2. Phân biệt 3 khái niệm gần nhau: 14](#_Toc198894872)

[3. Khai báo con trỏ mảng 14](#_Toc198894873)

[4. Gán con trỏ mảng 14](#_Toc198894874)

[5. Truy cập phần tử thông qua con trỏ mảng 14](#_Toc198894875)

[6. Tại sao dùng con trỏ mảng? 14](#_Toc198894876)

[7. Ví dụ: truyền mảng vào hàm dùng con trỏ mảng 15](#_Toc198894877)

[8. Con trỏ tới mảng 2 chiều 15](#_Toc198894878)

[9. Duyệt toàn bộ mảng 2 chiều bằng con trỏ mảng 15](#_Toc198894879)

[10. Lưu ý 15](#_Toc198894880)

[11. So sánh với các loại con trỏ/mảng khác 16](#_Toc198894881)

[12. Ví dụ minh họa 16](#_Toc198894882)

[CHƯƠNG 4: MẢNG CON TRỎ (Pointer Array) 16](#_Toc198894883)

[1. Khái niệm mảng con trỏ 16](#_Toc198894884)

[2. Cú pháp khai báo 17](#_Toc198894885)

[3. Minh họa cơ bản với chuỗi 17](#_Toc198894886)

[4. So sánh mảng chuỗi tĩnh và mảng con trỏ 17](#_Toc198894887)

[5. Truy cập dữ liệu qua mảng con trỏ 18](#_Toc198894888)

[6. Cấp phát động mảng con trỏ 18](#_Toc198894889)

[7. Mảng con trỏ hai chiều (Mảng con trỏ trỏ tới mảng) 19](#_Toc198894890)

[8. Lưu ý khi dùng mảng con trỏ 19](#_Toc198894891)

[9. Ứng dụng thực tế 19](#_Toc198894892)

[10. Ví dụ minh họa 19](#_Toc198894893)

[CHƯƠNG 5: CON TRỎ HÀM (FUNCTION POINTER) 21](#_Toc198894894)

[1. Khái niệm 21](#_Toc198894895)

[2. Cú pháp khai báo 21](#_Toc198894896)

[3. Gán con trỏ cho hàm 21](#_Toc198894897)

[4. Gọi hàm qua con trỏ 21](#_Toc198894898)

[5. Truyền hàm qua con trỏ hàm 21](#_Toc198894899)

[6. Mảng con trỏ hàm (Menu động) 22](#_Toc198894900)

[7. Con trỏ tới hàm không tham số 22](#_Toc198894901)

[8. Lưu ý 22](#_Toc198894902)

[9. Dùng typedef với con trỏ hàm 22](#_Toc198894903)

[10. Ứng dụng thực tế 23](#_Toc198894904)

[11. Ví dụ minh họa 23](#_Toc198894905)

[CHƯƠNG 6: CẤP PHÁT ĐỘNG 24](#_Toc198894906)

[1. Khái niệm 24](#_Toc198894907)

[2. Thư viện cần dùng 24](#_Toc198894908)

[3. Các hàm cấp phát động 24](#_Toc198894909)

[4. Cấp phát động biến đơn 24](#_Toc198894910)

[5. Cấp phát động mảng 1 chiều 24](#_Toc198894911)

[6. Cấp phát động mảng 2 chiều //Cấp phát mảng int a[m][n] 25](#_Toc198894912)

[7. Cấp phát với calloc 26](#_Toc198894913)

[8. Thay đổi kích thước với realloc 26](#_Toc198894914)

[9. Giải phóng bộ nhớ 26](#_Toc198894915)

[10. Lưu ý quan trọng 26](#_Toc198894916)

[11. Ví dụ minh họa 26](#_Toc198894917)

[CHƯƠNG 7: XỬ LÝ TỆP (FILE) 28](#_Toc198894918)

[1. Tệp và lợi ích của tệp 28](#_Toc198894919)

[2. Chế độ mở tệp 28](#_Toc198894920)

[3. Thao tác trên tệp 29](#_Toc198894921)

[4. Ví dụ minh họa 31](#_Toc198894922)

[CHƯƠNG 8: KIỂU CẤU TRÚC (STRUCT) 33](#_Toc198894923)

[1. Giới thiệu 33](#_Toc198894924)

[2. Định nghĩa cấu trúc 33](#_Toc198894925)

[3. Khai báo biến 34](#_Toc198894926)

[4. Toán tử . và -> 34](#_Toc198894927)

[5. Ví dụ minh họa: Quản lý sinh viên 35](#_Toc198894928)

[CHƯƠNG 9: DANH SÁCH LIÊN KẾT 37](#_Toc198894929)

[1. Cấu trúc tự tham chiếu 37](#_Toc198894930)

[2. Danh sách liên kết đơn (Single Linked List) 37](#_Toc198894931)

[3. Các thao tác cơ bản trên danh sách liên kết đơn 38](#_Toc198894932)

[4. Danh sách liên kết đôi (Double Linked List) 40](#_Toc198894933)

[5. So sánh: Mảng – DSLK đơn – DSLK đôi 41](#_Toc198894934)

[6. Ứng dụng 41](#_Toc198894935)

[7. Ví dụ minh họa 41](#_Toc198894936)

[PHẦN B: ỨNG DỤNG 45](#_Toc198894937)

[1. Mục Đích 45](#_Toc198894938)

[2. Đặt Vấn Đề 45](#_Toc198894939)

[3. Cấu Trúc Chương Trình 45](#_Toc198894940)

[4. Chức Năng 47](#_Toc198894941)

[5.  Kiến Thức Vận Dụng 48](#_Toc198894942)

[6. Hướng Mở Rộng 48](#_Toc198894943)

**PHẦN A: LÝ THUYẾT**

## **CHƯƠNG 1: HÀM (FUNCTION)**

### **Định nghĩa Hàm**

* Hàm (Function) là một khối mã lệnh được đặt tên, thực hiện một nhiệm vụ nhất định trong chương trình.
* Hàm giúp chia nhỏ chương trình thành các phần riêng biệt, dễ quản lý, dễ đọc và dễ bảo trì.
* Mỗi hàm có thể nhận dữ liệu đầu vào (tham số) và có thể trả về kết quả (giá trị trả về).
* Sử dụng hàm giúp tái sử dụng mã lệnh và giảm thiểu sự lặp lại trong chương trình.

### **Cấu trúc hàm**

Một hàm trong C thường có các phần cơ bản:

* Kiểu trả về (Return Type): cho biết hàm trả về kiểu dữ liệu gì (int, float, char, void...).
* Tên hàm: tên gọi để gọi hàm đó từ nơi khác.
* Danh sách tham số: các biến nhận dữ liệu đầu vào khi hàm được gọi. Nếu không có tham số, để trống hoặc ghi void.
* Phần thân hàm: chứa các câu lệnh xử lý.

Cú pháp tổng quát:

<kiểu\_trả\_về> <tên\_hàm>(<danh\_sách\_tham\_số>) {

// Các câu lệnh xử lý

return <giá\_trị>;

// Nếu có kiểu trả về khác void

}

### **Ý nghĩa của từng phần**

* Kiểu trả về:
  + Nếu hàm thực hiện tính toán và trả kết quả, kiểu trả về sẽ là kiểu dữ liệu của kết quả đó (ví dụ int, float).
  + Nếu hàm không trả kết quả gì, kiểu trả về là void.
* Tên hàm:
  + Là tên định danh hàm, theo quy tắc đặt tên biến trong C.
* Danh sách tham số:
  + Chứa biến đầu vào cho hàm, được truyền khi gọi hàm.
  + Giúp hàm nhận dữ liệu từ nơi gọi, làm hàm linh hoạt hơn.
* Thân hàm:
  + Nơi đặt các câu lệnh để thực hiện chức năng mong muốn.
* Lệnh return:
  + Kết thúc hàm và trả về giá trị cho nơi gọi (nếu có).

### **Nguyên mẫu hàm (Function Prototype)**

* Để sử dụng hàm trong main() hoặc hàm khác mà chưa định nghĩa hàm trước, ta cần khai báo nguyên mẫu hàm.
* Nguyên mẫu bao gồm kiểu trả về, tên hàm, kiểu các tham số.
* Giúp trình biên dịch kiểm tra đúng kiểu dữ liệu và số lượng tham số khi gọi hàm.

Ví dụ: int tinhTong(int a, int b);

### **Gọi hàm (Function Call)**

* Khi cần dùng hàm, ta gọi theo cú pháp:

<tên\_hàm>(<danh\_sách\_đối\_số>);

* Đối số truyền vào phải tương ứng với kiểu và số lượng tham số khai báo.
* Hàm thực hiện lệnh và trả kết quả (nếu có).

### **Hàm không trả về giá trị (Kiểu void)**

* Hàm có kiểu trả về void nghĩa là không trả kết quả về nơi gọi.
* Thường dùng cho các hàm thực hiện công việc như in ra màn hình, cập nhật biến toàn cục, hoặc thực hiện hành động.

Ví dụ:

void hienThi() {

printf("Xin chao!\n");

}

### **Truyền tham trị (Call by Value)**

* Khi gọi hàm, các đối số được truyền bằng giá trị, tức là bản sao của biến được gửi vào hàm.
* Mọi thay đổi trong hàm chỉ ảnh hưởng đến bản sao, không làm thay đổi biến gốc ngoài hàm.
* Đây là cách truyền tham số mặc định trong ngôn ngữ C.

Ví dụ:

void tangGiaTri(int x) {

x = x + 1; // Chỉ thay đổi biến cục bộ x trong hàm

}

int main() {

int a = 10;

tangGiaTri(a);

printf("%d", a); // In ra 10, không thay đổi

}

### **Phạm vi biến (Scope)**

* Biến cục bộ:
  + Được khai báo trong hàm, chỉ có hiệu lực trong hàm đó.
  + Khi hàm kết thúc, biến bị hủy.
* Biến toàn cục:
  + Được khai báo bên ngoài hàm, có hiệu lực trong toàn bộ chương trình (các hàm đều truy cập được).
* Lưu ý: Nên hạn chế dùng biến toàn cục để tránh khó kiểm soát và lỗi chương trình.

### **Lợi ích của hàm**

* Giúp phân chia chương trình thành các phần nhỏ dễ kiểm soát.
* Tăng khả năng tái sử dụng mã nguồn.
* Dễ dàng bảo trì, sửa lỗi.
* Tăng tính rõ ràng, dễ đọc của chương trình.
* Giúp thực hiện các thuật toán phức tạp bằng cách tách nhỏ thành các hàm đơn giản hơn.

### **Ví dụ minh họa**

#include <stdio.h>

int tinhTong(int a, int b) {// Khai báo và định nghĩa hàm

return a + b;

}

int main() {

int x = 5, y = 7;

int tong = tinhTong(x, y); // Gọi hàm

printf("Tổng = %d\n", tong);

return 0;

}

**CHƯƠNG 2: CON TRỎ**

1. **Khái niệm Con trỏ**

* Con trỏ là biến đặc biệt dùng để lưu địa chỉ bộ nhớ của một biến khác.
* Thay vì lưu giá trị, con trỏ lưu địa chỉ nơi giá trị đó được lưu trong RAM.
* Việc sử dụng con trỏ giúp thao tác trực tiếp với vùng nhớ, tối ưu bộ nhớ và hiệu năng chương trình.

1. **Khai báo con trỏ**

* Cú pháp:

<kiểu dữ liệu> \*<tên con trỏ>;

* Ví dụ:

int \*p; // p là con trỏ trỏ tới kiểu int

1. **Toán tử liên quan đến con trỏ**

* Toán tử & (address-of): Lấy địa chỉ bộ nhớ của biến.  
  Ví dụ: &a trả về địa chỉ của biến a.
* Toán tử \* (dereference): Truy cập giá trị tại địa chỉ mà con trỏ đang trỏ tới.  
  Ví dụ: \*p trả về giá trị biến được con trỏ p trỏ đến.

1. **Ví dụ sử dụng con trỏ**

int a = 10;

int \*p = &a; // p lưu địa chỉ của a

printf("Gia tri cua a: %d\n", a); // 10

printf("Dia chi cua a: %p\n", &a); // Địa chỉ của a

printf("Gia tri tai dia chi p: %d\n", \*p); // 1

1. **Truyền tham chiếu bằng con trỏ (Call by Reference)**

* Khi truyền tham số vào hàm bằng con trỏ, ta truyền địa chỉ của biến.
* Trong hàm, con trỏ có thể truy cập và thay đổi trực tiếp giá trị biến gốc thông qua địa chỉ đó.
* Giúp thay đổi giá trị biến bên ngoài hàm.

1. **Ví dụ truyền tham chiếu**

void tangGiaTri(int \*x) {

\*x = \*x + 1; // Thay đổi giá trị biến gốc thông qua con trỏ

}

int main() {

int a = 10;

tangGiaTri(&a); // Truyền địa chỉ biến a

printf("%d", a); // Kết quả: 11

}

1. **So sánh truyền tham trị và tham chiếu**

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Truyền tham trị | Truyền tham chiếu (bằng con trỏ) |
| Đối số truyền vào | Bản sao giá trị | Địa chỉ biến gốc |
| Thay đổi biến gốc? | Không | Có |
| Cách truyền | Truyền trực tiếp biến | Truyền địa chỉ (con trỏ) |
| Mục đích | An toàn, không thay đổi dữ liệu | Muốn thay đổi dữ liệu ngoài hàm |

1. **Ứng dụng con trỏ trong C**

* Quản lý bộ nhớ động (malloc, free).
* Xây dựng cấu trúc dữ liệu như danh sách liên kết, cây, đồ thị.
* Truyền tham số lớn hiệu quả (truyền con trỏ thay vì bản sao dữ liệu lớn).
* Truy cập trực tiếp tới vùng nhớ phần cứng (thường dùng trong lập trình hệ thống).

1. **Lưu ý khi dùng con trỏ**

* Phải luôn khởi tạo con trỏ trước khi dùng (gán địa chỉ hợp lệ).
* Tránh trỏ tới vùng nhớ không hợp lệ (NULL, vùng nhớ đã giải phóng).
* Để tránh lỗi truy cập bộ nhớ, cần cẩn thận khi dereference con trỏ.
* Sử dụng hợp lý để tránh rò rỉ bộ nhớ hoặc lỗi chương trình.

1. **Ví dụ minh họa**

#include <stdio.h>

void doiCho(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 5, y = 10;

printf("Trước khi đổi: x = %d, y = %d\n", x, y);

doiCho(&x, &y);

printf("Sau khi đổi: x = %d, y = %d\n", x, y);

return 0;

}

**CHƯƠNG 3: CON TRỎ MẢNG (ARRAY POINTER)**

* 1. **Khái niệm**

**Con trỏ mảng (array pointer)** là con trỏ trỏ đến **toàn bộ một mảng** (một block bộ nhớ liên tiếp), chứ **không phải** từng phần tử như con trỏ thường.

* 1. **Phân biệt 3 khái niệm gần nhau:**

|  |  |
| --- | --- |
| **Biểu thức** | **Ý nghĩa** |
| int \*p; | Con trỏ trỏ đến 1 biến int hoặc 1 phần tử int trong mảng |
| int a[5]; | Mảng 5 phần tử kiểu int |
| int (\*p)[5]; | Con trỏ trỏ đến **nguyên mảng** gồm 5 phần tử int |

* Dấu ngoặc () là bắt buộc để phân biệt:

int \*p[5]; là **mảng gồm 5 con trỏ**,  
int (\*p)[5]; là **con trỏ trỏ đến mảng 5 phần tử**.

* 1. **Khai báo con trỏ mảng**

int (\*p)[5]; // p là con trỏ trỏ đến mảng 5 phần tử kiểu int

* 1. **Gán con trỏ mảng**

int a[5] = {1, 2, 3, 4, 5};

int (\*p)[5]; // p trỏ tới mảng gồm 5 int

p = &a; // hoặc p = &a[0]; sai! (vì kiểu không khớp)

* 1. **Truy cập phần tử thông qua con trỏ mảng**

(\*p)[0] == a[0]

(\*p)[1] == a[1]

Hoặc dùng vòng lặp:

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

printf("%d ", (\*p)[i]);

}

* 1. **Tại sao dùng con trỏ mảng?**
* Khi muốn truyền **toàn bộ mảng** vào hàm, đặc biệt với mảng 2 chiều.
* Khi muốn quản lý mảng động nhiều chiều.
* Khi cần kiểm soát chính xác kích thước mảng.
  1. **Ví dụ: truyền mảng vào hàm dùng con trỏ mảng**

#include <stdio.h>

void inMang(int (\*p)[5]) {

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

printf("%d ", (\*p)[i]);}

}

int main() {

int a[5] = {10, 20, 30, 40, 50};

inMang(&a); // truyền địa chỉ nguyên mảng

return 0;}

* 1. **Con trỏ tới mảng 2 chiều**

int a[3][4];

int (\*p)[4]; // con trỏ trỏ đến mảng 1 chiều 4 phần tử

p = a; // p trỏ tới hàng đầu tiên (a[0])

Truy cập như sau:

p[0][0] == a[0][0]

p[1][2] == a[1][2]

* 1. **Duyệt toàn bộ mảng 2 chiều bằng con trỏ mảng**

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 4; j++) {

printf("%d ", p[i][j]); // tương đương a[i][j]

}

printf("\n");

}

* 1. **Lưu ý**
* Phải khai báo đúng kiểu con trỏ (int (\*p)[n]) với n là số cột.
* Khi truyền mảng nhiều chiều vào hàm, phải biết số cột để dùng được con trỏ mảng.
  1. **So sánh với các loại con trỏ/mảng khác**

|  |  |
| --- | --- |
| **Kiểu khai báo** | **Ý nghĩa** |
| int \*p; | Con trỏ tới int |
| int a[5]; | Mảng gồm 5 int |
| int \*a[5]; | Mảng gồm 5 con trỏ tới int |
| int (\*p)[5]; | Con trỏ tới mảng gồm 5 int |
| int \*\*p; | Con trỏ tới con trỏ tới int (mảng 2 chiều động) |

* 1. **Ví dụ minh họa**

#include <stdio.h>

int main() {

int a[5] = {1, 2, 3, 4, 5};

int (\*p)[5]; // con trỏ p trỏ đến một mảng gồm 5 phần tử kiểu int

p = &a; // gán địa chỉ của mảng a cho con trỏ p

printf("Các phần tử của mảng a thông qua con trỏ p:\n");

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

printf("%d ", (\*p)[i]); // truy xuất phần tử qua con trỏ mảng

}

return 0;

}

**CHƯƠNG 4: MẢNG CON TRỎ (Pointer Array)**

* 1. **Khái niệm mảng con trỏ**
* **Mảng con trỏ** là một mảng mà mỗi phần tử trong mảng là một con trỏ.
* Thường dùng để lưu trữ danh sách địa chỉ của các biến, chuỗi, hoặc mảng khác.
* Là kỹ thuật rất quan trọng trong xử lý chuỗi, quản lý bộ nhớ động, và cấu trúc dữ liệu.
  1. **Cú pháp khai báo**

<kiểu dữ liệu> \*<tên\_mảng>[]; // hoặc: <kiểu dữ liệu>\* <tên\_mảng>[số\_phần\_tử];

Ví dụ:

char \*dsChuoi[3]; // mảng 3 con trỏ trỏ tới kiểu char (chuỗi)

int \*dsSo[5]; // mảng 5 con trỏ trỏ tới số nguyên

* 1. **Minh họa cơ bản với chuỗi**

#include <stdio.h>

int main() {

char \*tenSV[] = {"An", "Binh", "Cuong"};

for (int i = 0; i < 3; i++) {

printf("Sinh vien %d: %s\n", i + 1, tenSV[i]);

}

return 0;

}

Giải thích:

* tenSV là mảng 3 con trỏ, mỗi con trỏ trỏ tới một chuỗi ký tự.
* Dễ dàng truy cập và xử lý từng chuỗi bằng tenSV[i].
  1. **So sánh mảng chuỗi tĩnh và mảng con trỏ**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Mảng chuỗi tĩnh** | **Mảng con trỏ** |
| Cấu trúc | char ten[3][20]; | char \*ten[] = {"A", "B", "C"}; |
| Kích thước mỗi chuỗi | Cố định (mỗi chuỗi dài 20 ký tự) | Thay đổi được tùy độ dài chuỗi |
| Linh hoạt | Kém linh hoạt (tốn bộ nhớ) | Linh hoạt hơn |
| Khả năng cấp phát động | Không | Có thể kết hợp với malloc, strdup |

* 1. **Truy cập dữ liệu qua mảng con trỏ**

Bạn có thể dùng:

ds[i] // Trỏ tới phần tử i

\*(ds + i) // Trỏ tương tự (toán tử con trỏ)

\*ds[i] // Giá trị tại địa chỉ con trỏ ds[i]

* 1. **Cấp phát động mảng con trỏ**

Mảng con trỏ thường kết hợp với malloc để cấp phát động:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

char \*ds[3];

for (int i = 0; i < 3; i++) {

ds[i] = (char \*)malloc(20 \* sizeof(char));

sprintf(ds[i], "Chuoi %d", i + 1);}

for (int i = 0; i < 3; i++) {

printf("%s\n", ds[i]);

free(ds[i]);}

return 0;

}

Giải thích:

* Cấp phát vùng nhớ riêng cho từng chuỗi.
* Tránh giới hạn độ dài chuỗi như mảng tĩnh.
  1. **Mảng con trỏ hai chiều (Mảng con trỏ trỏ tới mảng)**

Ta có thể tạo mảng con trỏ trỏ tới mảng số:

int a[] = {1, 2, 3};

int b[] = {4, 5};

int c[] = {6, 7, 8, 9};

int \*ds[] = {a, b, c};

Kết hợp thêm mảng lưu kích thước từng dãy:

int kichThuoc[] = {3, 2, 4};

for (int i = 0; i < 3; i++) {

for (int j = 0; j < kichThuoc[i]; j++) {

printf("%d ", ds[i][j]);

}

printf("\n");

}

* 1. **Lưu ý khi dùng mảng con trỏ**
* Cẩn thận cấp phát và giải phóng bộ nhớ nếu dùng malloc.
* Tránh để con trỏ trỏ tới vùng nhớ bị giải phóng (dangling pointer).
* Đảm bảo không vượt quá giới hạn phần tử khi truy cập.
  1. **Ứng dụng thực tế**
* Quản lý danh sách tên sinh viên, từ điển, dòng văn bản...
* Làm tham số cho hàm xử lý nhiều chuỗi.
* Làm danh sách các mảng động trong thuật toán.
* Tạo cấu trúc dữ liệu như danh sách các hàng, cột, hoặc ma trận bất đối xứng.
  1. **Ví dụ minh họa**

#include <stdio.h>

int main() {

const char \*dsTen[] = {

"An",

"Bình",

"Chi",

"Dũng"

};

int n = sizeof(dsTen) / sizeof(dsTen[0]);

printf("Danh sách tên:\n");

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

printf("%s\n", dsTen[i]);

}

return 0;

}

**CHƯƠNG 5: CON TRỎ HÀM (FUNCTION POINTER)**

1. **Khái niệm**

**Con trỏ hàm** là con trỏ trỏ đến **địa chỉ của một hàm**, cho phép:

* Gọi hàm qua con trỏ.
* Truyền hàm như đối số cho hàm khác.
* Cài đặt callback, menu, thuật toán chọn hàm động...

1. **Cú pháp khai báo**

kiểu\_trả\_về (\*tên\_con\_trỏ)(danh\_sách\_tham\_số);

Ví dụ:

int (\*p)(int, int); // p là con trỏ trỏ đến hàm nhận 2 int, trả về int

1. **Gán con trỏ cho hàm**

int tong(int a, int b) {

return a + b;}

int (\*p)(int, int); // khai báo con trỏ

p = tong; // hoặc p = &tong;

1. **Gọi hàm qua con trỏ**

int kq = p(2, 3); // tương đương tong(2, 3)

1. **Truyền hàm qua con trỏ hàm**

int tong(int a, int b) { return a + b; }

int hieu(int a, int b) { return a - b; }

int tinhToan (int x, int y, int (\*ham)(int, int)){

return ham(x, y); // gọi hàm qua con trỏ

}

int main () {

printf("Tong: %d\n", tinhToan(5, 3, tong));

printf("Hieu: %d\n", tinhToan(5, 3, hieu));

return 0;}

1. **Mảng con trỏ hàm (Menu động)**

#include <stdio.h>

int tong(int a, int b) { return a + b; }

int hieu(int a, int b) { return a - b; }

int tich(int a, int b) { return a \* b; }

int thuong(int a, int b) { return a / b; }

int main() {

int (\*menu[])(int, int) = {tong, hieu, tich, thuong};

int chon = 2; // ví dụ chọn phép nhân

int kq = menu[chon](6, 4);

printf("Ket qua: %d\n", kq);

return 0;

}

1. **Con trỏ tới hàm không tham số**

void xinChao() {

printf("Xin chao!\n");

}

void (\*p)(); // khai báo con trỏ tới hàm không tham số

p = xinChao;

p(); // gọi hàm

1. **Lưu ý**

* Khi khai báo con trỏ hàm, phải viết đầy đủ kiểu tham số và kiểu trả về.
* Có thể dùng typedef để gọn hơn.

1. **Dùng typedef với con trỏ hàm**

typedef int (\*HAMSO)(int, int);

int tong(int a, int b) { return a + b; }

int main() {

HAMSO p = tong;

printf("Ket qua: %d\n", p(3, 4));

return 0;

}

1. **Ứng dụng thực tế**

* Viết chương trình menu động.
* Cài đặt callback.
* Chọn thuật toán phù hợp (ví dụ trong sắp xếp, tìm kiếm...).
* Kỹ thuật hướng đối tượng cơ bản trong C.

1. **Ví dụ minh họa**

#include <stdio.h>

// Hàm cộng hai số

int cong(int a, int b) {

return a + b;

}

int main() {

// Khai báo con trỏ hàm

int (\*ptrHam)(int, int);

// Gán địa chỉ hàm 'cong' cho con trỏ

ptrHam = cong;

// Gọi hàm thông qua con trỏ

int kq = ptrHam(5, 3);

printf("Tổng = %d\n", kq); // Kết quả: 8

return 0;

}

**CHƯƠNG 6: CẤP PHÁT ĐỘNG**

1. **Khái niệm**

**Cấp phát động** là quá trình **cấp phát bộ nhớ trong lúc chương trình đang chạy**, giúp:

* Tạo biến, mảng có kích thước linh hoạt.
* Giải phóng bộ nhớ sau khi dùng.
* Quản lý dữ liệu lớn, số lượng không cố định.

1. **Thư viện cần dùng**

#include <stdlib.h> // chứa malloc, calloc, realloc, free

1. **Các hàm cấp phát động**

|  |  |
| --- | --- |
| **Hàm** | **Chức năng** |
| malloc(size\_t size) | Cấp phát size byte. Dữ liệu rác. |
| calloc(n, size) | Cấp phát n \* size byte. Gán 0 cho toàn bộ. |
| realloc(ptr, newsize) | Thay đổi kích thước vùng nhớ đã cấp. |
| free(ptr) | Giải phóng vùng nhớ. |

1. **Cấp phát động biến đơn**

int \*p;

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

\*p = 5;

printf("%d", \*p);

free(p); // giải phóng

1. **Cấp phát động mảng 1 chiều**

int \*a;

int n;

printf("Nhap n: ");

scanf("%d", &n);

a = (int\*) malloc(n \* sizeof(int));

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

printf("a[%d] = ", i);

scanf("%d", &a[i]);

}

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

printf("%d ", a[i]);

free(a);

1. **Cấp phát động mảng 2 chiều //Cấp phát mảng int a[m][n]**

int \*\*a;

int m, n;

scanf("%d%d", &m, &n);

a = (int\*\*) malloc(m \* sizeof(int\*));

for(int i = 0; i < m; i++)

a[i] = (int\*) malloc(n \* sizeof(int));

// Nhập mảng

for(int i = 0; i < m; i++)

for(int j = 0; j < n; j++)

scanf("%d", &a[i][j]);

// In mảng

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

for(int j = 0; j < n; j++)

printf("%d ", a[i][j]);

printf("\n");

}

for(int i = 0; i < m; i++)// Giải phóng

free(a[i]);

free(a);

1. **Cấp phát với calloc**

int \*a = (int\*) calloc(5, sizeof(int)); // cấp phát 5 int và gán 0

1. **Thay đổi kích thước với realloc**

int \*a = (int\*) malloc(3 \* sizeof(int)); // cấp phát 3 phần tử

a = (int\*) realloc(a, 5 \* sizeof(int)); // mở rộng thành 5 phần tử

1. **Giải phóng bộ nhớ**

free(p); // giải phóng biến đơn

free(a); // mảng 1 chiều

for(...) // mảng 2 chiều cần giải phóng từng dòng

free(a[i]);

free(a);

1. **Lưu ý quan trọng**

* Luôn kiểm tra con trỏ sau cấp phát:
* if(p == NULL) {
* printf("Cap phat that bai!");
* exit(1);
* }
* Sau free, không nên dùng lại con trỏ trừ khi cấp phát lại.
* Tránh rò rỉ bộ nhớ (không free vùng nhớ không dùng nữa).

1. **Ví dụ minh họa**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n;

printf("Nhập số phần tử: ");

scanf("%d", &n);

// Cấp phát động mảng n phần tử kiểu int

int \*arr = (int \*)malloc(n \* sizeof(int));

if (arr == NULL) {

printf("Cấp phát bộ nhớ thất bại!\n");

return 1;

}

// Nhập dữ liệu cho mảng

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

printf("arr[%d] = ", i);

scanf("%d", &arr[i]);

}

// In mảng

printf("Mảng vừa nhập: ");

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

printf("%d ", arr[i]);

}

printf("\n");

// Giải phóng bộ nhớ cấp phát động

free(arr);

return 0;

}

**CHƯƠNG 7: XỬ LÝ TỆP (FILE)**

* + 1. **Tệp và lợi ích của tệp**
* Trong ngôn ngữ C, **tệp (file)** là phương tiện lưu trữ dữ liệu ngoài bộ nhớ chính, giúp dữ liệu **tồn tại sau khi chương trình kết thúc**. Theo định nghĩa kinh điển của Kernighan & Ritchie, tệp là *“một chuỗi các byte được lưu trên thiết bị ngoại vi và được truy cập qua con trỏ FILE \*, đại diện cho một luồng dữ liệu (stream)”*.
* Việc sử dụng tệp mang lại lợi ích lớn như:
* Dữ liệu không bị mất sau khi chương trình kết thúc.
* Hỗ trợ ghi log, lưu cấu hình, hoặc quản lý thông tin người dùng lâu dài.
* Tách biệt giữa xử lý trong bộ nhớ và lưu trữ trên ổ đĩa.

Tệp được quản lý thông qua con trỏ FILE\* trong thư viện <stdio.h>. Mọi thao tác như mở, đọc, ghi, đóng tệp đều thông qua con trỏ này.

* + 1. **Chế độ mở tệp**
* Khi mở tệp bằng fopen(), cần chỉ định chế độ (mode) để xác định **cách thức truy cập**: đọc, ghi, hay cập nhật.

|  |  |
| --- | --- |
| **Chế độ** | **Mô tả** |
| r | Mở để đọc, tệp phải tồn tại. |
| w | Tạo mới để ghi. Nếu tệp tồn tại, nội dung bị xóa. |
| a | Mở để ghi tiếp vào cuối. Tạo mới nếu chưa có. |
| r+ | Mở để đọc/ghi. Tệp phải tồn tại. |
| w+ | Tạo mới để đọc/ghi. Nếu tệp tồn tại, nội dung bị xóa. |
| a+ | Mở để đọc/ghi, chỉ ghi vào cuối. |
| rb, wb, ab, rb+, wb+, ab+ | Tương tự các chế độ trên nhưng dùng cho **tệp nhị phân**. |

* + 1. **Thao tác trên tệp**
  1. **Mở tệp – fopen()**

Cú pháp: FILE \*f = fopen("tenfile.txt", "chedo");

* Nếu mở thành công: f != NULL
* Nếu thất bại: f == NULL → nên kiểm tra trước khi sử dụng

**Ví dụ:**

FILE \*f = fopen("data.txt", "r");

if (f == NULL) {

printf("Không thể mở tệp.\n");

return 1;

}

* 1. **Ghi dữ liệu vào tệp**

Các hàm ghi:

|  |  |
| --- | --- |
| **Hàm** | **Chức năng** |
| fprintf | Ghi dữ liệu định dạng như printf |
| fputs | Ghi một chuỗi |
| fputc | Ghi một ký tự |

**Ví dụ:**

FILE \*f = fopen("output.txt", "w");

fprintf(f, "Ho ten: %s\n", "Nguyen Van A");

fputs("Lop: 12A1\n", f);

fputc('X', f);

fclose(f);

* 1. **Đọc dữ liệu từ tệp**

Các hàm đọc:

|  |  |
| --- | --- |
| **Hàm** | **Chức năng** |
| fscanf | Đọc định dạng như scanf |
| fgets | Đọc một dòng (kết thúc bởi \n) |
| fgetc | Đọc một ký tự |

**Ví dụ:**

char ten[50], lop[20];

int tuoi;

FILE \*f = fopen("output.txt", "r");

fscanf(f, "Ho ten: %s", ten);

fgets(lop, 20, f);

char c = fgetc(f);

printf("Ten: %s\n", ten);

printf("Lop: %s", lop);

printf("Ky tu doc tiep: %c\n", c);

fclose(f);

* 1. **Đọc/Ghi tệp nhị phân**

Khi cần ghi/đọc khối dữ liệu như mảng, struct, dùng fwrite, fread:

int a[5] = {1, 2, 3, 4, 5};

FILE \*f = fopen("data.bin", "wb");

fwrite(a, sizeof(int), 5, f);

fclose(f);

int b[5]; // Đọc lại

f = fopen("data.bin", "rb");

fread(b, sizeof(int), 5, f);

fclose(f);

* 1. **Đóng tệp – fclose()**

Sau khi kết thúc thao tác, **phải đóng tệp** để ghi dữ liệu còn lại và giải phóng tài nguyên.

FILE \*f = fopen("file.txt", "w");

if (f != NULL) {

fprintf(f, "Xin chao!\n");

fclose(f);

}

1. **Ví dụ minh họa**

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*fp;

// Ghi dữ liệu văn bản vào file bằng fprintf và fputc

fp = fopen("sinhvien.txt", "w");

if (fp == NULL) {

     printf("Không thể mở tệp để ghi.\n");

     return 1;

}

fprintf(fp, "Nguyen Van A %d\n", 20); // Ghi có định dạng

fputc('H', fp); // Ghi một ký tự

fputs("ello, world!\n", fp); // Ghi một chuỗi

fclose(fp);

// Đọc lại dữ liệu từ file bằng fscanf, fgetc và fgets

char ten[50], chuoi[100];

int tuoi;

fp = fopen("sinhvien.txt", "r");

if (fp == NULL) {

     printf("Không thể mở tệp để đọc.\n");

    return 1;

}

fscanf(fp, "%s %s %d", ten, chuoi, &tuoi); // Đọc chuỗi có định dạng

char c = fgetc(fp); // Đọc 1 ký tự tiếp theo

fgets(chuoi, 100, fp); // Đọc cả dòng còn lại

printf("Tên: %s %s, Tuổi: %d\n", ten, chuoi, tuoi);

printf("Ký tự kế tiếp sau dòng đầu: %c\n", c);

printf("Phần còn lại của dòng: %s\n", chuoi);

fclose(fp);

return 0;

}

**CHƯƠNG 8: KIỂU CẤU TRÚC (STRUCT)**

1. **Giới thiệu**

* struct là kiểu dữ liệu cho phép gom nhiều biến (khác tên, khác kiểu) dưới một tên chung.
* Khác mảng (cùng kiểu), cấu trúc có thể chứa nhiều kiểu như: int, float, char, mảng, con trỏ, hoặc lồng struct.
* Dùng để mô hình hóa dữ liệu thực tế: sinh viên, nhân viên, sản phẩm,...
* Giúp mã rõ ràng, dễ tổ chức, bảo trì – rất hữu ích trong chương trình lớn.

1. **Định nghĩa cấu trúc**

* **Cú pháp:**

struct TênStruct {

kiểu\_dữ\_liệu thành\_viên1;

kiểu\_dữ\_liệu thành\_viên2;

...

};

Ví dụ:

struct SinhVien {

char hoTen[20];

int mssv;

float GPA;

};

* Trong C để mở rộng khả năng biểu diễn còn cung cấp từ khóa typedef, cho phép định nghĩa các tên gọi thay thế cho các kiểu dữ liệu, bao gồm cả cấu trúc. Việc sử dụng typedef làm cho mã nguồn trở nên ngắn gọn, dễ hiểu và dễ bảo trì hơn, nhất là khi làm việc với các cấu trúc có độ phức tạp cao.
* **Cú pháp**

typedef struct {

       kiểu dữ liệu thành viên;

}name-struct;

typedef struct {

char hoTen[20];

int mssv;

float GPA;

} SinhVien;

→ Giúp khai báo ngắn gọn hơn: SinhVien sv;

1. **Khai báo biến**

* Sau khi định nghĩa struct, ta phải khai báo biến cụ thể mới cấp phát bộ nhớ.
* Có thể khai báo cùng lúc nhiều biến:

struct SinhVien sv, svs[20], \*svPtr;

* Hoặc khai báo luôn khi định nghĩa:

struct SinhVien {

char hoTen[20];

int mssv;

float GPA;

} sv, svs[20], \*svPtr;

1. **Toán tử . và ->**

* Toán tử .: dùng khi có biến cấu trúc trực tiếp
* printf("%s", sv.hoTen);
* Toán tử ->: dùng khi có con trỏ trỏ đến cấu trúc
* printf("%s", svPtr->hoTen); // tương đương (\*svPtr).hoTen

1. **Ví dụ minh họa: Quản lý sinh viên**

#include <stdio.h>

#include <string.h>

struct SinhVien {

char hoTen[20];

int mssv;

float GPA;

};

void nhapDanhSach(struct SinhVien ds[], int n);

void hienThiDanhSach(struct SinhVien ds[], int n);

void timGPA\_Max(struct SinhVien ds[], int n);

int main() {

int n;

printf("Nhập số lượng sinh viên: ");

scanf("%d", &n);

struct SinhVien ds[100];

nhapDanhSach(ds, n);

hienThiDanhSach(ds, n);

timGPA\_Max(ds, n);

return 0;

}

void nhapDanhSach(struct SinhVien ds[], int n) {

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

getchar();

printf("\nSV %d - Họ tên: ", i + 1);

fgets(ds[i].hoTen, sizeof(ds[i].hoTen), stdin);

ds[i].hoTen[strcspn(ds[i].hoTen, "\n")] = '\0';

printf("MSSV: "); scanf("%d", &ds[i].mssv);

printf("GPA: "); scanf("%f", &ds[i].GPA);

}

}

void hienThiDanhSach(struct SinhVien ds[], int n) {

printf("\nDanh sách sinh viên:\n");

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

printf ("SV %d | Tên: %s | MSSV: %d | GPA: %.2f\n", i + 1, ds[i].hoTen, ds[i].mssv, ds[i].GPA);

}

}

void timGPA\_Max(struct SinhVien ds[], int n) {

if (n <= 0) return;

int viTri = 0;

for (int i = 1; i < n; i++) {

if (ds[i].GPA > ds[viTri].GPA) viTri = i;

}

printf("\nSV có GPA cao nhất:\n");

printf("Tên: %s | MSSV: %d | GPA: %.2f\n",ds[viTri].hoTen, ds[viTri].mssv, ds[viTri].GPA);

}

**CHƯƠNG 9: DANH SÁCH LIÊN KẾT**

1. **Cấu trúc tự tham chiếu**

* Trong ngôn ngữ C, cấu trúc có thể chứa con trỏ trỏ đến chính nó. Đây được gọi là **cấu trúc tự tham chiếu**. Cấu trúc này là nền tảng để xây dựng các kiểu dữ liệu động như danh sách liên kết, cây, hàng đợi,...

struct node {

int data;

struct node \*next;

};

* data: dữ liệu lưu trong node.
* next: trỏ đến node kế tiếp trong danh sách.
* Khi liên kết nhiều node lại với nhau bằng con trỏ next, ta tạo thành **danh sách liên kết**.

1. **Danh sách liên kết đơn (Single Linked List)**
   1. **Khái niệm**

Là tập hợp các node được liên kết tuyến tính theo một chiều từ đầu đến cuối. Mỗi node chứa:

* Một trường dữ liệu.
* Một con trỏ trỏ đến node kế tiếp.

Node cuối cùng có con trỏ next = NULL để đánh dấu kết thúc danh sách.

* 1. **Cấu trúc**

Ví dụ xây dựng danh sách sinh viên:

struct SinhVien {

int mssv;

char hoTen[50];

float diem;

struct SinhVien \*next;

};

Danh sách được quản lý bằng một con trỏ đầu danh sách:

struct SinhVien \*head = NULL;

* 1. **Ưu điểm**
* Không cần xác định trước số phần tử.
* Thêm/xóa phần tử dễ dàng (không phải dịch chuyển phần tử như mảng).
* Quản lý linh hoạt khi dữ liệu thay đổi liên tục.
  1. **Nhược điểm**
* Truy cập chậm hơn mảng (phải duyệt tuần tự từ đầu).
* Quản lý con trỏ phức tạp (nếu sai dễ gây lỗi hoặc rò rỉ bộ nhớ).

1. **Các thao tác cơ bản trên danh sách liên kết đơn**
   1. **Khởi tạo danh sách**

SinhVien \*head = NULL;

* 1. **Thêm phần tử vào cuối**

void themCuoi(SinhVien \*\*head, int mssv, char \*hoTen, float diem) {

SinhVien \*moi = (SinhVien\*)malloc(sizeof(SinhVien));

moi->mssv = mssv;

strcpy(moi->hoTen, hoTen);

moi->diem = diem;

moi->next = NULL;

if (\*head == NULL) {

\*head = moi;

} else {

SinhVien \*p = \*head;

while (p->next != NULL) p = p->next;

p->next = moi;

}

}

* 1. **Duyệt và in danh sách**

void inDanhSach(SinhVien \*head) {

SinhVien \*p = head;

while (p != NULL) {

printf("MSSV: %d | Họ tên: %s | Điểm: %.2f\n", p->mssv, p->hoTen, p->diem);

p = p->next;

}

}

* 1. **Tìm kiếm theo MSSV**

SinhVien \*timTheoMSSV(SinhVien \*head, int mssv) {

while (head != NULL) {

if (head->mssv == mssv)

return head;

head = head->next;

}return NULL;

}

* 1. **Xóa một phần tử theo MSSV**

void xoaTheoMSSV(SinhVien \*\*head, int mssv) {

SinhVien \*p = \*head, \*truoc = NULL;

while (p != NULL && p->mssv != mssv) {

truoc = p;

p = p->next;}

if (p == NULL) return;

if (truoc == NULL) \*head = p->next;

else truoc->next = p->next;

free(p);

}

* 1. **Giải phóng toàn bộ danh sách**

void giaiPhong(SinhVien \*head) {

SinhVien \*p;

while (head != NULL) {

p = head;

head = head->next;

free(p);

}

}

1. **Danh sách liên kết đôi (Double Linked List)**
   1. **Khái niệm**

Mỗi node chứa 2 con trỏ:

* next: trỏ đến node sau.
* prev: trỏ đến node trước.
  1. **Cấu trúc**

struct node {

int data;

struct node \*next;

struct node \*prev;

};

* 1. **Ưu điểm**
* Duyệt 2 chiều thuận lợi.
* Xóa/thêm linh hoạt cả ở đầu và cuối.
  1. **Nhược điểm**
* Mỗi node chiếm nhiều bộ nhớ hơn.
* Việc quản lý con trỏ phức tạp hơn danh sách đơn.

1. **So sánh: Mảng – DSLK đơn – DSLK đôi**

|  |  |  |  |
| --- | --- | --- | --- |
| **Đặc điểm** | **Mảng** | **DSLK đơn** | **DSLK đôi** |
| Bộ nhớ | Cố định, liên tục | Động, rời rạc | Động, rời rạc |
| Truy cập phần tử | Trực tiếp (O(1)) | Tuần tự (O(n)) | Tuần tự (O(n)) |
| Thêm/Xóa | Chậm (dịch chuyển) | Nhanh hơn mảng | Linh hoạt hơn nữa |
| Duyệt 2 chiều | Không | Không | Có |

1. **Ứng dụng**

* Quản lý danh sách sinh viên, sách, đơn hàng...
* Cấu trúc dữ liệu nâng cao như stack, queue, cây nhị phân...
* Quản lý bộ nhớ, undo/redo, hệ điều hành...

1. **Ví dụ minh họa**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct SinhVien {

    int mssv;

    char hoTen[50];

    float GPA;

    struct SinhVien \*next;

} SinhVien;

SinhVien\* taoSinhVien(int mssv, const char \*hoTen, float GPA);

void themSV(SinhVien \*\*head, int mssv, const char \*hoTen, float GPA);

void nhapDanhSach(SinhVien \*\*head);

void inDanhSach(SinhVien \*head;

void giaiPhong(SinhVien \*head);

int main() {

    SinhVien \*ds = NULL;

    nhapDanhSach(&ds);

    inDanhSach(ds);

    giaiPhong(ds);

    return 0;

}

// Tạo node mới

SinhVien\* taoSinhVien(int mssv, const char \*hoTen, float GPA) {

    SinhVien \*sv = (SinhVien\*)malloc(sizeof(SinhVien));

    sv->mssv = mssv;

    strcpy(sv->hoTen, hoTen);

    sv->GPA = GPA;

    sv->next = NULL;

    return sv;

}

// Thêm sinh viên vào cuối danh sách

void themSV(SinhVien \*\*head, int mssv, const char \*hoTen, float GPA) {

    SinhVien \*moi = taoSinhVien(mssv, hoTen, GPA);

    if (\*head == NULL) {

        \*head = moi;

    } else {

        SinhVien \*p = \*head;

        while (p->next != NULL)

            p = p->next;

        p->next = moi;

    }

}

// Nhập danh sách sinh viên

void nhapDanhSach(SinhVien \*\*head) {

    int n;

    printf("Nhập số lượng sinh viên: ");

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {

        int mssv;

        char ten[50];

        float gpa;

        printf("\nSinh viên %d\n", i + 1);

        printf("MSSV: ");

        scanf("%d", &mssv);

        getchar();

        printf("Họ tên: ");

        fgets(ten, sizeof(ten), stdin);

        ten[strcspn(ten, "\n")] = '\0';

        printf("GPA: ");

        scanf("%f", &gpa);

        themSV(head, mssv, ten, gpa);}

}

// In danh sách sinh viên

void inDanhSach(SinhVien \*head) {

    printf("\nDanh sách sinh viên\n");

    while (head != NULL) {

        printf("MSSV: %d | Họ tên: %s | GPA: %.2f\n",

                head->mssv, head->hoTen, head->GPA);

        head = head->next;}

}

// Giải phóng danh sách

void giaiPhong(SinhVien \*head) {

    SinhVien \*temp;

    while (head != NULL) {

        temp = head;

        head = head->next;

        free(temp);

    }

}

# **PHẦN B: ỨNG DỤNG**

### **1. Mục Đích**

Mục đích của bài toán là xây dựng một phần mềm quản lý thư viện có khả năng hỗ trợ lưu trữ, cập nhật và theo dõi thông tin mượn trả sách một cách tự động, chính xác và hiệu quả. Phần mềm nhằm tối ưu hóa quá trình vận hành thư viện, giúp giảm tải công việc nhưng mà vẫn đảm bảo xử lý đúng quy định về mượn sách, trả sách, phạt quá hạn, cũng như quản lý thẻ độc giả.

### **2. Đặt Vấn Đề**

Trong thực tế, các thư viện thường phải quản lý một lượng lớn sách và thông tin độc giả. Nếu thực hiện quản lý theo cách thủ công, việc theo dõi các hoạt động mượn trả, kiểm tra tình trạng thẻ độc giả, xử lý các trường hợp vi phạm như quá hạn, mất sách... sẽ rất dễ gây ra nhầm lẫn, thất thoát hoặc quá tải cho người quản lý.

Không chỉ vậy, việc mượn sách bị ràng buộc bởi nhiều quy định như: mỗi sách chỉ được mượn tối đa trong 91 ngày (nếu quá hạn sẽ bị phạt tiền 5.000 đồng/ngày); nếu thẻ hết hạn thì phải trả sách hoặc gia hạn; nếu mất sách thì bị phạt gấp đôi giá trị...

Do đó, cần thiết phải có một hệ thống phần mềm giúp tổ chức dữ liệu một cách rõ ràng, dễ truy xuất, đồng thời lưu trữ thông tin bền vững qua file để tránh mất mát. Việc giải quyết bài toán này sẽ giúp tăng hiệu quả quản lý, hạn chế rủi ro và nâng cao chất lượng phục vụ trong hoạt động thư viện.

### **3. Cấu Trúc Chương Trình**

**3.1 Tổ chức thư mục và file**

* **main.c**

Gồm hàm main() khai báo biến, gọi hàm menu, thực hiện giải phóng bộ nhớ cuối chương trình.

* **menu.c / menu.h**

Hiển thị menu chính và gọi các nhóm chức năng như quản lý sách, độc giả, phiếu, thống kê

* **quan\_ly\_doc\_gia.c / quan\_ly\_doc\_gia.h**

Định nghĩa cấu trúc docGia, cùng các hàm nhập, thêm, xóa, sửa, ghi file và xuất danh sách độc giả.

* **quan\_ly\_sach.c / quan\_ly\_sach.h**

Xử lý các thao tác liên quan đến sách như nhập, tìm kiếm, thêm mới, lưu file, xuất danh sách...

* **quan\_ly\_phieu\_muon\_sach.c / quan\_ly\_phieu\_muon\_sach.h**

Quản lý phiếu mượn sách: tạo phiếu, lưu thông tin, cập nhật số lượng sách, kiểm tra hạn sử dụng thẻ.

* **quan\_ly\_phieu\_tra\_sach.c / quan\_ly\_phieu\_tra\_sach.h**

Quản lý phiếu trả sách: xử lý trả sách, tính số ngày trễ, phạt, cập nhật dữ liệu và lưu file.

* **date.c / date.h**

Chứa các hàm xử lý ngày tháng: kiểm tra ngày hợp lệ, tính số ngày trong tháng, năm nhuận, so sánh ngày,...

* **thong\_ke.c / thong\_ke.h**

Xử lý thống kê: tổng sách trong thư viện, sách đang mượn, số độc giả quá hạn

**3.2 Các khai báo kiểu dữ liệu**

* **docGia:** lưu thông tin độc giả (họ tên, mssv, ngày sinh, giới tính, ngày tạo và hết hạng thẻ)
* **sach:** lưu thông tin của sách (mã số sách, tên sách, tác giả, ngày xuất bản, thể loại, giá sách, số lượng)
* **phieuMuonSach:** lưu thông tin của phiếu mượn sách ( mssv của người mượn sách, ngày mượn, ngày trả dự kiến, sóo lượng sách dược mượn)
* **phieuTraSach:** lưu thông tin của phiếu trả sách ( mssv của người trae sách, ngày trả sách thực tế, danh sách và số lượng sách được trả, danh sách và số lượng sách bị mất)
* **date:** lưu ngày thánng năm
  1. **Các hàm quản lý chính**
  2. **Quản lý Độc giả**
* **Các hàm:** taoDG, themDocGia, xoaDG, chinhSuaDG, xuatDSDG, ghiFileDG
* **Dữ liệu lưu trữ:** danh sách liên kết đơn docGia \*headDG.
* **File lưu trữ:** quanLyDocGia.txt.
  1. **Quản lý Sách**
* **Các hàm:** themSach, nhapTTSach, ghiFileSach, xuatDSSach, timKiemISBN, timKiemTS, themSM.
* **Dữ liệu lưu trữ:** mảng động sach \*dsSach.
* **File lưu trữ:** quanLySach.txt.
  1. **Quản lý Phiếu Mượn – Phiếu Trả**
* **Các hàm:** nhapPMS, ghiFilePM, nhapPTS, ghiFilePT, xuatSachMuon, capNhatSLS, capNhapPMS, kiemTraXuPhat.
* **Dữ liệu lưu trữ:** mảng động phieuMuonSach \*pMS, phieuTraSach \*pTS.
* **File lưu trữ:** quanLyPhieuMuon.txt, quanLyPhieuTra.txt.
  1. **Thống kê**
* **Các hàm:** soLuongSachTV, dsSM, soDGTH, dgTreHan.

### **4. Chức Năng**

**4.1  Quản lý độc giả**

* Xem danh sách độc giả
* Thêm độc giả
* Xóa độc giả theo MSSV
* Chỉnh sửa thông tin độc giả

**4.2 Quản lý sách**

* Xem danh sách sách trong thư viện
* Tìm kiếm sách theo ISBN
* Tìm kiếm sách theo tên sách
* Thêm sách mới

**4.3 Lập phiếu mượn và trả sách**

* Lập phiếu mượn sách
* Lập phiếu trả sách
* Cập nhật số lượng sách khi mượn
* Xem danh sách sách đang mượn của một độc giả theo MSSV

**4.4 Các thống kê cơ bản**

* Thống kê số lượng sách trong thư viện
* Thống kê số sách đang được mượn
* Thống kê số lượng độc giả trễ hạn trả sách

### **Kiến Thức Vận Dụng**

* **Hàm:** *nhapDSDG(), ghiFileSach(), kiemTraNgay(), dGTreHan()*, …
* **Con trỏ**: *int \*n, char \*MSSV, docGia \*taoDG(), …*
* **Con trỏ mảng:** *sach \*dsSach, phieuMuonSach \*pMS, phieuTraSach \*pTS, …*
* **Mảng con trỏ:** *char \*\*ISBNm, char \*\*ISBNt*
* **Con trỏ hàm:** *docGia\* taoDG(), char \*\*nhapISBN(), …*
* **Cấp phát động:** sử dụng *malloc(), calloc(), realloc()* và *free()* để cấp phát và giải phóng bộ nhớ: *docGia, sach, phieuMuonSach, ...*
* **Xử lý tệp:** Đọc, ghi dữ liệu vào file với *fopen(), fprintf(), fclose()* cho các file như: *quanLyDocGia.txt, quanLySach.txt,...*
* **Kiểu cấu trúc:** Định nghĩa nhiều *struct* như: *date, docGia, sach, phieuMuonSach, phieuTraSach* để tổ chức dữ liệu.
* **Danh sách liên kết đơn:** *struct docGia* có trường *\*next* và các thao tác: thêm node *themDocGia(),* xóa node *xoaDG(),* duyệt danh sách *headDG = headDG->next,…*
* **Hàm thao tác chuỗi:** *strcpy(), strlen(), strcmp(), strdup()* để xử lý chuỗi: *MSSV, ISBN*, *tenDG, …*
* **Sử dụng con trỏ cấp 2:** *sach \*\*dsSach, docGia \*\*headDG, char \*\*ISBN, ...*
* **Truyền tham chiếu, tham trị:**
* Tham chiếu: *void nhapTTSach(sach \*\*dsSach, int* *\*m).*
* Tham trị: *void xuatDSDG(docGia \*headDG)*

### **6. Hướng Mở Rộng**

**6.1 Bổ sung giao diện người dùng đồ họa (GUI)**

Hiện tại, chương trình sử dụng giao diện dòng lệnh đơn giản, gây khó khăn cho người dùng phổ thông. Trong các hệ thống thư viện thực tế, giao diện đồ họa sẽ giúp thao tác nhanh, dễ hiểu và trực quan hơn.

* 1. **Gợi ý phát triển:**
* Sử dụng các thư viện giao diện như GTK hoặc WinAPI nếu tiếp tục sử dụng ngôn ngữ C.
* Thiết kế giao diện dạng cửa sổ với:
  + Menu chính bằng các nút bấm (quản lý sách, độc giả, phiếu mượn...).
  + Các biểu mẫu (form) nhập liệu trực quan, có kiểm tra đầu vào.
  + Bảng hiển thị danh sách có thể cuộn, sắp xếp và lọc theo tiêu chí.
  1. **Lợi ích:**
* Thân thiện với người dùng không rành kỹ thuật.
* Giảm rủi ro nhập sai dữ liệu, dễ thao tác hơn so với nhập bằng dòng lệnh.

**6.2 Thêm hệ thống đăng nhập và phân quyền người dùng**

Hiện tại chương trình chưa có hệ thống đăng nhập, bất kỳ ai sử dụng đều có thể chỉnh sửa dữ liệu. Trong thực tế, hệ thống thư viện cần kiểm soát quyền truy cập để đảm bảo an toàn thông tin.

1. **Gợi ý phát triển:**

* Thêm bước đăng nhập khi khởi động chương trình: người dùng cần nhập tài khoản và mật khẩu để tiếp tục.

1. **Phân loại người dùng:**

* **Thủ thư (quản trị viên):** có toàn quyền quản lý (thêm, sửa, xoá sách và độc giả, lập thống kê).
* **Độc giả:** chỉ được phép đăng nhập để tra cứu thông tin sách và xem tình trạng mượn/trả của chính mình.

1. **Lợi ích:**

* Giúp bảo vệ dữ liệu thư viện khỏi bị thay đổi ngoài ý muốn.
* Đảm bảo người dùng chỉ được thực hiện những chức năng đúng với quyền hạn của họ.