**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO TIỂU LUẬN**

**HỌC PHẦN: KĨ THUẬT LẬP TRÌNH**

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

**Sinh viên thực hiện:  LƯU PHƯỚC ĐẠI**

**Lớp:    CQ.65. CNTT**

**Khoá:   K65**

**Tp. Hồ Chí Minh, năm 2025**

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO TIỂU LUẬN**

**HỌC PHẦN: KĨ THUẬT LẬP TRÌNH**

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

**Sinh viên thực hiện:  LƯU PHƯỚC ĐẠI**

**Lớp:    CQ.65. CNTT**

**Khoá:   K65**

**Tp. Hồ Chí Minh, năm 2025**

# MỤC LỤC

**MỤC LỤC**  [**i**](#_Toc198987764)

[**LỜI CẢM ƠN ii**](#_Toc198987765)

[**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN iii**](#_Toc198987766)

[**A.LÝ THUYẾT 1**](#_Toc198987767)

[**1.HÀM: 1**](#_Toc198987768)

[1.1.KHÁI NIỆM 1](#_Toc198987769)

[1.2.Ví dụ: 3](#_Toc198987770)

[**2.CON TRỎ: 4**](#_Toc198987771)

[2.1.Khái niệm. 4](#_Toc198987772)

[2.2. Ví dụ. 4](#_Toc198987773)

[**3.CON TRỎ MẢNG: 6**](#_Toc198987774)

[3.1.Khái niệm. 6](#_Toc198987775)

[3.2. Ví dụ. 6](#_Toc198987776)

[**4.MẢNG CON TRỎ. 9**](#_Toc198987777)

[4.1.Khái niệm. 9](#_Toc198987778)

[4.2.Ví dụ. 9](#_Toc198987779)

[**5.CON TRỎ HÀM: 11**](#_Toc198987780)

[5.1.Khái niệm. 11](#_Toc198987781)

[5.2.Lợi ích. 11](#_Toc198987782)

[5.3.Ví dụ. 12](#_Toc198987783)

[**6.CẤP PHÁT ĐỘNG 14**](#_Toc198987784)

[6.1.Khái niệm. 14](#_Toc198987785)

[6.2.Lợi ích. 14](#_Toc198987786)

[6.3. So sánh với mạng tĩnh. 15](#_Toc198987787)

[6.3.Ví dụ. 16](#_Toc198987788)

[**7.XỬ LÍ TỆP: 18**](#_Toc198987789)

[7.1.Khái niệm. 18](#_Toc198987790)

[7.2.Tại sao cần xử lý tệp. 18](#_Toc198987791)

[7.3. Ví dụ. 19](#_Toc198987792)

[**8.KIỂU CẤU TRÚC. 21**](#_Toc198987793)

[8.1.Khái niệm. 21](#_Toc198987794)

[8.2.Lợi ích. 21](#_Toc198987795)

[8.3.Ví dụ. 22](#_Toc198987796)

[**9.DANH SÁCH LIÊN KẾT. 25**](#_Toc198987797)

[9.1.Khái niệm. 25](#_Toc198987798)

[9.2.Lợi ích. 25](#_Toc198987799)

[9.3.So sánh danh sách liên kết với mảng. 26](#_Toc198987800)

[9.4.Ví dụ. 27](#_Toc198987801)

[**B.ỨNG DỤNG 30**](#_Toc198987802)

[**KẾT LUẬN 41**](#_Toc198987803)

[**TÀI LIỆU THAM KHẢO 42**](#_Toc198987804)

# **LỜI CẢM ƠN**

Chào thầy và các bạn sinh viên thân mến!

Trước tiên, em xin gửi lời cảm ơn chân thành đến Thầy Trần Phong Nhã – người đã tận tình giảng dạy, truyền đạt kiến thức và định hướng cho chúng em trong suốt quá trình học tập môn *Kỹ thuật lập trình*. Thầy không chỉ cung cấp cho chúng em những kiến thức chuyên môn quý báu mà còn giúp em hiểu rõ hơn về tư duy logic, phương pháp giải quyết vấn đề và cách tiếp cận một cách khoa học trong lập trình.

Em cũng xin chân thành cảm ơn các bạn trong lớp đã luôn hỗ trợ, chia sẻ và cùng nhau trao đổi, học hỏi trong suốt thời gian học tập và thực hiện các bài tập, dự án môn học. Sự hợp tác và tinh thần làm việc nhóm của các bạn đã góp phần tạo nên một môi trường học tập tích cực và hiệu quả.

Nhờ sự hướng dẫn tận tình của Thầy cùng sự giúp đỡ nhiệt tình từ các bạn, em đã có cơ hội rèn luyện kỹ năng lập trình, nâng cao tư duy kỹ thuật và hoàn thành tốt môn học này.

Em xin chân thành cảm ơn!

Trân trọng,

*Lưu Phước Đại*

# NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
..........................................................................................................................................  
.........................................................................................................................................  
..........................................................................................................................................  
....................................................................................................................................................................................................................................................................................  
..........................................................................................................................................

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

# 

# LÝ THUYẾT

## **HÀM:**

### **KHÁI NIỆM**

Hàm trong lập trình là một khối mã được đặt tên và thực hiện một tác vụ cụ thể. Nó là một phần quan trọng trong việc tổ chức mã nguồn và tái sử dụng code. Có nhiều loại hàm trong các ngôn ngữ lập trình khác nhau, nhưng chúng thường chia thành hai loại chính: hàm được xây dựng sẵn và hàm do người dùng tự định nghĩa.

* Hàm được xây dựng sẵn: Là những hàm có sẵn trong ngôn ngữ lập trình, được cung cấp bởi hệ thống hoặc thư viện tiêu chuẩn. Ví dụ, trong code C, **printf()** là một hàm được xây dựng sẵn để in ra màn hình.
* Hàm do người dùng tự định nghĩa: Là những hàm mà người dùng tự viết để thực hiện một tác vụ cụ thể mà họ cần. Khi tạo hàm, người dùng định nghĩa một khối Code có thể được gọi và thực thi bất kỳ lúc nào trong chương trình.

\*Các khái niệm cơ bản về hàm trong lập trình gồm:

- Tham số (Parameters): Là các giá trị được truyền vào hàm khi gọi nó. Những giá trị này có thể được sử dụng bên trong hàm để thực hiện một số công việc cụ thể.

- Giá trị trả về (Return Value): Là giá trị mà hàm trả về sau khi thực thi xong. Điều này cho phép hàm truyền thông tin hoặc kết quả của nó trở lại cho phần của chương trình gọi hàm.

- Gọi hàm (Calling a Function): Là việc sử dụng tên của hàm cùng với các tham số (nếu có) để thực thi hàm đó. Khi một hàm được gọi, quá trình thực hiện sẽ chuyển tới nội dung của hàm và thực thi nó.

- Phạm vi biến (Variable Scope): Định nghĩa xem một biến có thể truy cập được từ đâu trong chương trình. Điều này có thể liên quan đến biến cục bộ (local variables), chỉ có thể truy cập được từ bên trong hàm mà nó được khai báo, và biến toàn cục (global variables), có thể truy cập được từ bất kỳ nơi nào trong chương trình.

Việc sử dụng hàm giúp rất nhiều trong việc tổ chức code, làm cho code dễ đọc hơn, giảm sự lặp lại và tăng khả năng tái sử dụng.

### **Ví dụ:**

* Đoạn code không sử dụng hàm:

#include “stdio.h”

int main(){

int i, n;

long long ketqua = 1;

for (i = 2; i <= n; i++) {

ketqua \*= i;

}

printf("Giai thua cua %d la: %lld\n", n, ketqua);

}

return 0;

* Đoạn code có sử dụng hàm:

#include “stdio.h”

// Hàm tính giai thừa

long long giaithua(int n) {

long long ketqua = 1;

for (int i = 2; i <= n; i++) {

ketqua \*= i;

}

return ketqua;

}

int main(){

int n;

printf(“nhap n: “);

sanf(“%d”, &n);

printf(“%d! = %d”, n, giaithua(n));

return 0;

}

Vd 1 ta chỉ có thể sử dụng 1 lần điều này không hiểu quả đối với các bài tập lớn, và bạn thấy ví dụ 2 ta tạo nó và chỉ cần gọi là có thể sử dụng rất tiện và không rối.

## **CON TRỎ:**

### **Khái niệm.**

Con trỏ là một khái niệm quan trọng trong lập trình, đặc biệt là trong ngôn ngữ C và C++. Một con trỏ là một biến chứa địa chỉ của một biến khác trong bộ nhớ. Điều này cho phép bạn thực hiện các thao tác trên bộ nhớ và truy cập trực tiếp vào dữ liệu được lưu trữ trong đó.

Đây là một số khái niệm cơ bản về con trỏ:

1. Địa chỉ (Address): Mỗi biến trong bộ nhớ có một địa chỉ duy nhất để xác định vị trí của nó trong bộ nhớ.

2. Con trỏ (Pointer): Là một biến chứa địa chỉ của một biến khác.

3. Toán tử con trỏ (**&** và **\***):

- Toán tử **&**: Trả về địa chỉ của một biến.

- Toán tử **\***: Trả về giá trị của biến được trỏ tới bởi một con trỏ.

4. Khai báo và sử dụng con trỏ:

- Để khai báo một con trỏ, bạn sử dụng dấu **\*** trước tên biến.

- Để gán địa chỉ của một biến cho một con trỏ, bạn sử dụng toán tử **&**.

- Để truy cập giá trị của biến được trỏ tới bởi một con trỏ, bạn sử dụng toán tử **\***.

5. Dùng con trỏ để thực hiện các thao tác trên bộ nhớ:

- Bằng cách sử dụng con trỏ, bạn có thể thực hiện các thao tác như cấp phát bộ nhớ động, truy cập mảng, và chuyển đổi dữ liệu.

### **2.2. Ví dụ.**

Khi không dùng con trọ :

#include “stdio.h”

void hoanDoi(int a, int b) {

int temp = a;

a = b;

b = temp;

}

int main() {

int x = 5, y = 10;

hoanDoi(x, y);

printf("Gia tri sau khi khong dung con tro:\n");

printf("x = %d, y = %d\n", x, y); // Kết quả: x = 5, y = 10 (không đổi)

return 0;

}

Ví dụ trên không hoán vị được giá trị gốc và ví dụ dưới nó hoán vị giá trị gốc. Ta thấy chỉ con trỏ mới cho phép thay đổi giá trị **gốc** từ bên trong hàm. Không cần trả về giá trị, tiết kiệm bộ nhớ và công sức xử lý.

Khi dùng con trọ:

#include “stdio.h”

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 5, y = 10;

hoanDoi(&x, &y);

printf("Gia tri sau khi dung con tro:\n");

printf("x = %d, y = %d\n", x, y); // Kết quả: x = 10, y = 5 (đã hoán đổi)

return 0;

}

## **CON TRỎ MẢNG:**

### **Khái niệm.**

Con trỏ mảng là khái niệm liên quan đến việc sử dụng con trỏ để tham chiếu đến các phần tử của mảng trong lập trình.

Trong ngôn ngữ lập trình C và C++, một mảng thực chất là một con trỏ. Khi khai báo một mảng là cũng đang tạo một con trỏ trỏ tới địa chỉ của phần tử đầu tiên trong mảng. Cụ thể, tên của mảng là một con trỏ không thể thay đổi (const pointer) trỏ tới phần tử đầu tiên của mảng.

Một số khái niệm quan trọng liên quan đến con trỏ mảng:

**1. Tham chiếu đến phần tử của mảng:** Bạn có thể sử dụng con trỏ để truy cập đến các phần tử của mảng bằng cách sử dụng toán tử chỉ mục **[]** hoặc toán tử pointer **\***.

**2. Duyệt mảng:** Bạn có thể sử dụng con trỏ để duyệt qua tất cả các phần tử của mảng bằng cách di chuyển con trỏ từ phần tử đầu tiên đến phần tử cuối cùng của mảng.

**3. Tính toán địa chỉ:** Bạn có thể sử dụng con trỏ để tính toán địa chỉ của các phần tử của mảng và truy cập trực tiếp vào bộ nhớ.

**4. Truyền mảng vào hàm:** Bạn có thể truyền mảng vào hàm bằng cách sử dụng con trỏ, cho phép hàm thực hiện các thao tác trên mảng mà không cần phải sao chép toàn bộ mảng.

### **3.2. Ví dụ.**

#include “stdio.h”

int main() {

*// Khai báo mảng và con trỏ*

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

int \*ptr;

*// Gán địa chỉ của phần tử đầu tiên của mảng cho con trỏ*

ptr = arr;

*// Truy cập và in ra các phần tử của mảng bằng cách sử dụng con trỏ*

printf("Cac phan tu cua mang:\n");

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

printf("Phan tu thu %d: %d\n", i + 1, \*(ptr + i));

}

return 0;

}

**Trong ví dụ này:**

- Chúng ta khai báo một mảng **arr** có 5 phần tử và một con trỏ **ptr** kiểu **int**.

- Chúng ta gán địa chỉ của phần tử đầu tiên của mảng cho con trỏ **ptr** bằng cách sử dụng tên mảng (mảng là một con trỏ tới phần tử đầu tiên của nó).

- Sau đó, chúng ta sử dụng con trỏ để truy cập và in ra các phần tử của mảng bằng cách sử dụng toán tử **\*** để lấy giá trị của phần tử mà con trỏ đang trỏ đến.

## **MẢNG CON TRỎ.**

### **Khái niệm.**

Mảng con trỏ là một mảng trong đó mỗi phần tử không phải là một giá trị trực tiếp mà là một con trỏ. Mỗi con trỏ này thường trỏ đến một vùng nhớ khác, nơi chứa giá trị thực sự của các phần tử.

Điều quan trọng cần lưu ý là mảng con trỏ không giống như mảng thông thường. Mỗi phần tử của mảng con trỏ có thể trỏ đến một vùng nhớ khác nhau trong bộ nhớ, do đó chúng ta có thể sử dụng mảng con trỏ để lưu trữ địa chỉ của các biến khác nhau.

Cú pháp để khai báo một mảng con trỏ trong ngôn ngữ lập trình C là:

**kiểu\_dữ\_liệu \*tên\_mảng\_con\_trỏ[kích\_thước\_mảng];**

Trong đó:

- **kiểu\_dữ\_liệu**: là kiểu dữ liệu của các phần tử mà mỗi phần tử của mảng con trỏ sẽ trỏ đến.

- **tên\_mảng\_con\_trỏ**: là tên của mảng con trỏ.

- **kích\_thước\_mảng**: là số lượng phần tử trong mảng con trỏ.

Ví dụ:

**int \*ptrArray[5]**;

Trong ví dụ này:

- **int** là kiểu dữ liệu của các phần tử mà mỗi phần tử của mảng con trỏ **ptrArray** sẽ trỏ đến.

- **ptrArray** là tên của mảng con trỏ.

- **5** là số lượng phần tử trong mảng con trỏ **ptrArray**.

Mảng con trỏ này sẽ chứa 5 con trỏ, mỗi con trỏ có thể trỏ đến một vùng nhớ khác nhau trong bộ nhớ, nơi lưu trữ các giá trị kiểu `int`.

### **Ví dụ.**

#include “stdio.h”

int main() {

// Khai báo 3 biến nguyên

int a = 10, b = 20, c = 30;

// Khai báo mảng con trỏ chứa địa chỉ của các biến

int \*arr[3]; // Mảng có 3 phần tử, mỗi phần tử là một con trỏ kiểu int

// Gán địa chỉ của các biến vào mảng con trỏ

arr[0] = &a; // arr[0] chứa địa chỉ của a

arr[1] = &b; // arr[1] chứa địa chỉ của b

arr[2] = &c; // arr[2] chứa địa chỉ của c

// In giá trị của các biến thông qua mảng con trỏ

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

printf("Giá trị tại arr[%d]: %d\n", i, \*arr[i]);

}

// Thay đổi giá trị của biến a thông qua con trỏ

\*arr[0] = 100;

// In lại giá trị của a để kiểm tra

printf("Giá trị mới của a: %d\n", a);

return 0;

}

**Chương trình sau khi chạy:**

**Giá trị tại arr[0]: 10**

**Giá trị tại arr[1]: 20**

**Giá trị tại arr[2]: 30**

**Giá trị mới của a: 100**

**Giải thích:**

* + int \*arr[3]; tạo một mảng có 3 phần tử, mỗi phần tử là một con trỏ kiểu int\* (trỏ đến một biến kiểu int).
  + Các con trỏ trong mảng (arr[0], arr[1], arr[2]) được gán địa chỉ của các biến a, b, c.

**Cơ chế hoạt động**

* **Khai báo và gán giá trị**:
  + int a = 10, b = 20, c = 30; tạo 3 biến nguyên và lưu giá trị của chúng trong bộ nhớ.
  + int \*arr[3]; tạo một mảng 3 phần tử, mỗi phần tử là một con trỏ chưa được khởi tạo.
  + arr[0] = &a; arr[1] = &b; arr[2] = &c; gán địa chỉ của các biến a, b, c vào các phần tử của mảng con trỏ.
  + Lúc này, arr[i] chứa địa chỉ của biến tương ứng, và \*arr[i] (dereference) truy xuất giá trị tại địa chỉ đó.
* **Truy cập giá trị**:
  + Vòng lặp for duyệt qua mảng con trỏ và sử dụng \*arr[i] để lấy giá trị của biến mà con trỏ trỏ tới (10, 20, 30).
* **Thay đổi giá trị**:
  + \*arr[0] = 100; thay đổi giá trị tại địa chỉ mà arr[0] trỏ tới (tức là biến a), làm cho a từ 10 thành 100.

## **CON TRỎ HÀM:**

### **Khái niệm.**

Con trỏ hàm là một khái niệm trong lập trình được sử dụng để tham chiếu đến một hàm cụ thể trong mã chương trình. Thay vì sử dụng tên hàm để gọi, bạn có thể sử dụng con trỏ hàm để trỏ đến hàm mà bạn muốn thực thi. Điều này cho phép bạn truyền hàm như một đối số cho một hàm khác, lưu trữ hàm trong một cấu trúc dữ liệu, hoặc gán một hàm cho một biến, giúp tăng tính linh hoạt và tái sử dụng trong mã của bạn.

Trong một số ngôn ngữ lập trình như C và C++, con trỏ hàm được khai báo bằng cách chỉ ra kiểu dữ liệu của hàm mục tiêu cùng với tên của con trỏ và dấu sao (\*). Ví dụ:

**int (\*funcPtr)(int, int);**

Đây là một con trỏ hàm có kiểu trả về là int và nhận hai tham số kiểu int. Để gán một hàm cụ thể cho con trỏ này, bạn cần trỏ đến hàm đó:

int sum(int a, int b) {

return a + b;

}

int main() {

int (\*funcPtr)(int, int); *// Khai báo con trỏ hàm*

funcPtr = sum; *// Gán con trỏ hàm vào hàm sum*

int result = funcPtr(3, 5); *// Sử dụng con trỏ hàm để gọi hàm sum*

printf("%d\n", result); *// In kết quả: 8*

return 0;

}

### **Lợi ích.**

Việc sử dụng hàm con trỏ mang lại một số lợi ích và tác dụng quan trọng trong lập trình, bao gồm:

1. Linhh động và tái sử dụng mã: Bằng cách sử dụng con trỏ hàm, bạn có thể truyền hàm như một đối số cho một hàm khác. Điều này giúp tăng tính linh hoạt của mã của bạn và làm cho mã trở nên dễ dàng tái sử dụng.

2. Tách biệt logic: Việc sử dụng con trỏ hàm cho phép bạn tách biệt logic thực thi của hàm ra khỏi việc quyết định loại hàm nào sẽ được sử dụng. Điều này giúp tạo ra các hàm chức năng mà không phụ thuộc vào các chi tiết cụ thể của logic thực thi.

3. Phát triển các mẫu thiết kế phần mềm: Hàm con trỏ là một phần quan trọng của nhiều mẫu thiết kế phần mềm như Chiến lược (Strategy), Trình tự (Command), và Trình xử lý sự kiện (Event Handler). Chúng cho phép bạn định nghĩa các chiến lược hoặc hành động một cách linh hoạt và thay đổi chúng tại thời điểm chạy.

4. Xử lý sự kiện và callback: Trong lập trình sự kiện (như trong các ứng dụng GUI), con trỏ hàm thường được sử dụng để đăng ký các hàm callback, được gọi khi xảy ra một sự kiện cụ thể.

5. Optimization: Trong một số trường hợp, sử dụng hàm con trỏ có thể giúp tối ưu hóa mã của bạn. Chẳng hạn, khi bạn cần truy cập vào các hàm không biết trước tại thời điểm biên dịch, việc sử dụng con trỏ hàm có thể làm cho mã của bạn linh hoạt và ít phụ thuộc hơn vào cấu trúc cụ thể của chương trình.

Tóm lại, việc sử dụng hàm con trỏ mang lại tính linh hoạt, tái sử dụng và giúp bạn thiết kế mã của mình một cách cấu trúc và dễ bảo trì hơn.

### **Ví dụ.**

#include “stdio.h”

*// Khai báo một hàm con trỏ có kiểu trả về là int và nhận hai tham số kiểu int*

int (\*sumPtr)(int, int);

*// Hàm tính tổng của hai số*

int sum(int a, int b) {

return a + b;

}

int main() {

*// Gán con trỏ hàm vào hàm sum*

sumPtr = sum;

*// Sử dụng con trỏ hàm để gọi hàm sum*

int result = sumPtr(3, 5);

*// In kết quả: 8*

printf("Tong cua 3 va 5 la: %d\n", result);

return 0;

}

**Trong ví dụ này:**

1. Đầu tiên, chúng ta khai báo một con trỏ hàm có tên là **sumPtr**. Kiểu của con trỏ hàm này phải trùng với kiểu của hàm mà nó sẽ trỏ đến, nên ở đây kiểu của **sumPtr** là **int (\*)(int, int)**, tức là một con trỏ hàm trả về kiểu **int** và nhận hai tham số kiểu **int**.

2. Tiếp theo, chúng ta định nghĩa hàm **sum** để tính tổng của hai số.

3. Trong hàm **main**, chúng ta gán con trỏ hàm **sumPtr** vào hàm **sum**. Điều này có nghĩa là **sumPtr** sẽ trỏ đến hàm **sum**, và khi chúng ta sử dụng **sumPtr**, nó sẽ thực hiện các câu lệnh trong hàm **sum**.

4. Sau đó, chúng ta sử dụng con trỏ hàm **sumPtr** để gọi hàm **sum** với hai số 3 và 5 làm tham số. Kết quả sẽ được lưu vào biến **result**.

5. Cuối cùng, chúng ta in ra kết quả. Trong trường hợp này, kết quả sẽ là 8, tức là tổng của 3 và 5.

## **CẤP PHÁT ĐỘNG**

### **Khái niệm.**

Cấp phát động là quá trình trong lập trình khi bạn cần cấp phát bộ nhớ trong quá trình thực thi của chương trình, thay vì cấp phát bộ nhớ tĩnh trong quá trình biên dịch. Quá trình này cho phép bạn tạo ra các cấu trúc dữ liệu có kích thước có thể thay đổi hoặc không biết trước tại thời điểm biên dịch.

Các từ khóa quan trọng trong cấp phát động:

- malloc: Hàm này cấp phát một lượng bộ nhớ cụ thể và trả về một con trỏ tới vùng nhớ được cấp phát.

- calloc: Tương tự như malloc, nhưng calloc cấp phát một số lượng các đối tượng và khởi tạo tất cả các byte của bộ nhớ được cấp phát thành giá trị 0.

- realloc: Hàm này thay đổi kích thước của vùng nhớ đã được cấp phát trước đó. Nó có thể làm tăng hoặc giảm kích thước của vùng nhớ và trả về một con trỏ mới.

- free: Hàm này được sử dụng để giải phóng bộ nhớ đã được cấp phát trước đó, giúp trả lại tài nguyên cho hệ thống.

Việc sử dụng cấp phát động cho phép tạo ra các cấu trúc dữ liệu linh hoạt như mảng động, danh sách liên kết, cây nhị phân, và đồ thị. Tuy nhiên, cần lưu ý rằng việc quản lý bộ nhớ động có thể phức tạp và có thể dẫn đến các vấn đề như rò rỉ bộ nhớ và xung đột bộ nhớ, do đó, việc sử dụng cấp phát động cần được thực hiện một cách cẩn thận.

### **Lợi ích.**

Việc sử dụng cấp phát động mang lại một số lợi ích quan trọng trong lập trình, bao gồm:

**1. Tính linh hoạt:** Cấp phát động cho phép bạn tạo ra các cấu trúc dữ liệu có kích thước có thể thay đổi trong quá trình thực thi của chương trình. Điều này làm cho mã của bạn trở nên linh hoạt hơn và có thể xử lý các tình huống đa dạng.

**2. Tiết kiệm bộ nhớ:** Trong một số trường hợp, cấp phát động có thể giúp tiết kiệm bộ nhớ bằng cách chỉ cấp phát bộ nhớ khi cần và giải phóng nó khi không cần thiết nữa. Điều này giúp tránh lãng phí tài nguyên.

**3. Hỗ trợ cấu trúc dữ liệu linh hoạt:** Cấp phát động làm cho việc triển khai các cấu trúc dữ liệu động như danh sách liên kết, cây nhị phân, và đồ thị trở nên dễ dàng hơn. Bạn có thể thay đổi kích thước của cấu trúc dữ liệu và thực hiện các thao tác chèn, xóa và sắp xếp một cách linh hoạt.

**4. Quản lý bộ nhớ hiệu quả:** Cấp phát động cho phép bạn quản lý bộ nhớ của chương trình một cách hiệu quả hơn. Bạn có thể chỉ cấp phát bộ nhớ khi cần và giải phóng nó khi không cần thiết nữa, giúp tránh các vấn đề như rò rỉ bộ nhớ và xung đột bộ nhớ.

**5. Hỗ trợ các cấu trúc dữ liệu phức tạp:** Cấp phát động làm cho việc triển khai các cấu trúc dữ liệu phức tạp như bảng băm, hàng đợi ưu tiên và vùng nhớ đệm trở nên dễ dàng hơn. Điều này giúp tạo ra các ứng dụng phần mềm phức tạp và hiệu quả hơn.

Tóm lại, việc sử dụng cấp phát động mang lại tính linh hoạt, tiết kiệm bộ nhớ, hỗ trợ cấu trúc dữ liệu linh hoạt, quản lý bộ nhớ hiệu quả và hỗ trợ các cấu trúc dữ liệu phức tạp trong lập trình.

### **6.3. So sánh với mạng tĩnh.**

**1. Kích thước:**

- Mảng tĩnh: Kích thước của mảng tĩnh được xác định tại thời điểm biên dịch và không thể thay đổi trong quá trình thực thi của chương trình.

- Cấp phát động: Kích thước của vùng nhớ được cấp phát động có thể thay đổi trong quá trình thực thi của chương trình.

**2. Quản lý bộ nhớ:**

- Mảng tĩnh: Bộ nhớ cho mảng tĩnh được quản lý tự động, do đó không cần phải giải phóng bộ nhớ thủ công.

- Cấp phát động: Bạn phải tự quản lý việc cấp phát và giải phóng bộ nhớ khi sử dụng cấp phát động.

**3. Tính linh hoạt:**

- Mảng tĩnh: Mảng tĩnh không linh hoạt về kích thước và không thể thay đổi trong quá trình thực thi của chương trình.

- Cấp phát động: Cấp phát động cho phép bạn tạo ra các cấu trúc dữ liệu có kích thước linh hoạt và có thể thay đổi tùy ý trong quá trình thực thi của chương trình.

**4. Hiệu suất:**

- Mảng tĩnh: Truy cập vào các phần tử của mảng tĩnh thường nhanh hơn do vị trí của các phần tử được biết trước và không thay đổi.

- Cấp phát động: Truy cập vào các phần tử của cấu trúc dữ liệu cấp phát động thường có hiệu suất kém hơn một chút do cần thêm thời gian để truy cập thông qua con trỏ.

**5. Điều kiện sử dụng:**

- Mảng tĩnh: Thích hợp cho các trường hợp mà kích thước của dữ liệu là cố định và không thay đổi.

- Cấp phát động: Thích hợp cho các trường hợp mà kích thước của dữ liệu có thể thay đổi hoặc không biết trước.

Tóm lại, cả mảng tĩnh và cấp phát động đều có những ưu điểm và nhược điểm riêng của chúng và thích hợp cho các tình huống sử dụng khác nhau trong lập trình. Lựa chọn giữa chúng phụ thuộc vào yêu cầu cụ thể của dự án và tính linh hoạt cần thiết.

### **Ví dụ.**

#include “stdio.h”

#include “stdlib.h”

int main() {

int n;

int \*arr; // Con trỏ để lưu địa chỉ của mảng động

// Bước 1: Nhập số lượng phần tử

printf("Nhap so luong phan tu cua mang: ");

scanf("%d", &n);

// Bước 2: Cấp phát động bộ nhớ cho mảng

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

// Bước 3: Kiểm tra xem cấp phát có thành công không

if (arr == NULL) {

printf("Cap phat bo nho that bai!\n");

return 1;

}

// Bước 4: Nhập giá trị cho mảng

printf("Nhap %d phan tu:\n", n);

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

printf("Phan tu thu %d: ", i + 1);

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

}

// Bước 5: In mảng để kiểm tra

printf("Mang vua nhap la:\n");

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

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

}

printf("\n");

// Bước 6: Giải phóng bộ nhớ

free(arr);

arr = NULL;

return 0;

}

**GIẢ SỬ NGƯỜI DÙNG NHẬP:**

Nhap so luong phan tu cua mang: 3

Nhap 3 phan tu:

Phan tu thu 1: 10

Phan tu thu 2: 20

Phan tu thu 3: 30

**Mang vua nhap la:**

10 20 30

**Trong ví dụ này:**

**Cấp phát bộ nhớ động bằng malloc**:

* Dòng arr = (int \*)malloc(n \* sizeof(int)); yêu cầu hệ thống cấp phát một vùng nhớ đủ lớn để chứa n số nguyên (int).
* sizeof(int) trả về kích thước của kiểu int (thường là 4 byte trên nhiều hệ thống).
* n \* sizeof(int) tính tổng số byte cần thiết.
* malloc trả về địa chỉ của vùng nhớ được cấp phát, được lưu vào con trỏ arr. Ép kiểu (int \*) để đảm bảo con trỏ tương thích với kiểu int.

**Kiểm tra cấp phát thành công**:

* Nếu bộ nhớ không đủ, malloc trả về NULL. Kiểm tra này giúp tránh lỗi khi truy cập vùng nhớ không hợp lệ.
* Nếu thất bại, chương trình in thông báo lỗi và dừng lại.

**Giải phóng bộ nhớ bằng free**:

* Sau khi sử dụng xong, free(arr) trả lại vùng nhớ cho hệ thống để tránh rò rỉ bộ nhớ (memory leak).
* Đặt arr = NULL để tránh tình trạng con trỏ "treo" (dangling pointer), tức là con trỏ vẫn trỏ đến vùng nhớ đã được giải phóng.

## **XỬ LÍ TỆP:**

### **Khái niệm.**

Xử lý tệp là quá trình trong lập trình máy tính mà bạn tương tác với các tệp và thư mục trên hệ thống tệp của máy tính. Quá trình này có thể bao gồm việc đọc từ tệp, ghi vào tệp, thay đổi tệp, xóa tệp, hoặc thậm chí là tạo mới tệp. Xử lý tệp là một phần quan trọng trong việc làm việc với dữ liệu trong các ứng dụng và hệ thống.

Các hoạt động cơ bản trong xử lý tệp bao gồm:

**1. Mở tệp:** Đây là bước đầu tiên trong xử lý tệp, trong đó bạn mở một tệp để đọc hoặc ghi dữ liệu vào đó.

**2. Đọc từ tệp:** Bạn có thể đọc dữ liệu từ tệp đã mở và sử dụng nó trong ứng dụng của mình. Điều này thường được thực hiện theo dạng các thao tác đọc tuần tự hoặc đọc tại một vị trí cụ thể trong tệp.

**3. Ghi vào tệp:** Bạn có thể ghi dữ liệu vào tệp đã mở. Điều này có thể là việc ghi dữ liệu mới hoặc ghi đè lên dữ liệu đã tồn tại trong tệp.

**4. Đóng tệp:** Sau khi bạn đã hoàn thành công việc của mình với một tệp, bạn cần đóng nó. Điều này giải phóng tài nguyên hệ thống và đảm bảo rằng không có ai khác có thể truy cập vào tệp đó trong khi bạn không sử dụng.

**5. Xử lý thư mục:** Ngoài việc xử lý các tệp, bạn cũng có thể cần làm việc với các thư mục trong hệ thống tệp. Điều này có thể bao gồm việc tạo mới, di chuyển, xóa thư mục và thậm chí là liệt kê các tệp trong một thư mục.

### **Tại sao cần xử lý tệp.**

Xử lý tệp là một khía cạnh quan trọng của lập trình vì nó cho phép chương trình tương tác với dữ liệu được lưu trữ trong các tệp và thư mục trên hệ thống tệp của máy tính. Đây là một số lý do tại sao ta cần xử lý tệp:

**1. Lưu trữ và truy cập dữ liệu:** Tệp là nơi lưu trữ dữ liệu trong lập trình. Bằng cách sử dụng tệp, chương trình có thể lưu trữ dữ liệu vào bộ nhớ không thể thay đổi và truy cập nó lại sau này.

**2. Dữ liệu lớn:** Trong các ứng dụng thực tế, dữ liệu thường lớn và không thể lưu trữ hoặc xử lý trong bộ nhớ RAM. Sử dụng tệp cho phép lưu trữ và xử lý các tập tin dữ liệu lớn mà không cần phải tải toàn bộ dữ liệu vào bộ nhớ.

**3. Giao tiếp với người dùng:** Các chương trình thường cần đọc dữ liệu từ người dùng hoặc ghi dữ liệu vào tệp để lưu trữ thông tin nhập liệu hoặc kết quả của chương trình.

**4. Đồng bộ hóa và chia sẻ dữ liệu:** Trong các ứng dụng đa tiến trình hoặc đa luồng, việc sử dụng tệp để lưu trữ và truy cập dữ liệu giữa các tiến trình hoặc luồng là cần thiết để đồng bộ hóa và chia sẻ dữ liệu.

**5. Bảo trì và quản lý dữ liệu:** Sử dụng tệp cho phép dễ dàng sao lưu, di chuyển, xóa và quản lý các tập tin và thư mục, giúp cho việc bảo trì và quản lý dữ liệu trở nên dễ dàng hơn.

**6. Tích hợp với hệ thống và ứng dụng khác:** Sử dụng tệp cho phép các chương trình tương tác với các tệp và thư mục được tạo bởi các ứng dụng và hệ thống khác, cho phép tích hợp và tương tác giữa các ứng dụng khác nhau.

### **7.3. Ví dụ.**

#include “stdio.h”

int main() {

*// Mở tệp để đọc*

FILE \*file = fopen("input.txt", "r");

*// Kiểm tra xem việc mở tệp có thành công hay không*

if (file == NULL) {

printf("Khong the mo file.\n");

return 1; }

*// Đọc và hiển thị nội dung của tệp*

char line[100]; *// Buffer để lưu dữ liệu từ tệp*

while (fgets(line, sizeof(line), file)) {

printf("%s", line);

}

*// Đóng tệp sau khi đã đọc xong*

fclose(file);

return 0;

Chương trình đọc nội dung từ tệp văn bản có tên "input.txt" và in từng dòng ra màn hình. Nếu tệp không mở được, nó báo lỗi "Không thể mở file" và thoát.

1. **Mở tệp**: Dùng fopen("input.txt", "r") để mở tệp "input.txt" ở chế độ chỉ đọc.
2. **Kiểm tra lỗi**: Nếu tệp không mở được (ví dụ: không tồn tại), in thông báo lỗi và thoát.
3. **Đọc từng dòng**:
   * Tạo một "hộp chứa" (line) để lưu tối đa 100 ký tự mỗi dòng.
   * Dùng fgets để đọc từng dòng từ tệp và lưu vào line.
   * In dòng đó ra màn hình bằng printf.
4. **Đóng tệp**: Sau khi đọc xong, đóng tệp bằng fclose để giải phóng tài nguyên.
5. **Kết thúc**: Thoát chương trình với mã 0 .

## **KIỂU CẤU TRÚC.**

### **Khái niệm.**

Cấu trúc (structure) trong C là một **kiểu dữ liệu tổng hợp** (composite data type) do người dùng định nghĩa, cho phép nhóm nhiều biến có kiểu dữ liệu khác nhau (hoặc cùng kiểu) vào một thực thể duy nhất. Điều này giúp tổ chức dữ liệu phức tạp một cách logic và dễ quản lý, đặc biệt khi làm việc với các tập hợp thông tin liên quan, chẳng hạn như thông tin về một sinh viên, một nhân viên, hoặc một bản ghi trong cơ sở dữ liệu.

Cấu trúc được định nghĩa bằng từ khóa struct và bao gồm các **thành viên** (members), mỗi thành viên là một biến với kiểu dữ liệu cụ thể. Cấu trúc không chỉ đơn thuần là một tập hợp dữ liệu mà còn cung cấp một cách tiếp cận có tổ chức để xử lý thông tin trong lập trình.

### **Lợi ích.**

**1. Tổ chức dữ liệu một cách logic:**

* Cấu trúc cho phép nhóm các biến liên quan (có thể thuộc các kiểu dữ liệu khác nhau) thành một thực thể duy nhất, giúp biểu diễn các đối tượng thực tế một cách trực quan.
* Ví dụ: Thay vì sử dụng các biến riêng lẻ như char hoTen[50], int maSV, float diemTB, bạn có thể nhóm chúng vào một cấu trúc SinhVien, giúp mã dễ đọc và quản lý hơn.

**2. Tăng khả năng tái sử dụng mã:**

* Một cấu trúc được định nghĩa có thể được sử dụng để tạo nhiều biến hoặc mảng, giúp tái sử dụng định dạng dữ liệu ở nhiều nơi trong chương trình.
* Ví dụ: Định nghĩa một cấu trúc SinhVien có thể được sử dụng để lưu trữ thông tin của hàng trăm sinh viên trong một mảng SinhVien danhSachSV[100].

**3. Hỗ trợ quản lý dữ liệu phức tạp:**

* Cấu trúc cho phép lồng nhau (nested structures) và tự tham chiếu (self-referential structures), giúp biểu diễn các mối quan hệ phức tạp như danh sách liên kết, cây, hoặc đồ thị.

**4. Tăng hiệu quả khi truyền dữ liệu:**

* Khi làm việc với hàm, thay vì truyền nhiều biến riêng lẻ, bạn có thể truyền một cấu trúc (hoặc con trỏ tới cấu trúc) để giảm số lượng tham số và tăng hiệu suất (đặc biệt khi truyền qua con trỏ).

**5. Hỗ trợ làm việc với file và giao tiếp hệ thống:**

* Cấu trúc rất hữu ích khi đọc/ghi dữ liệu từ file nhị phân hoặc xử lý các giao thức mạng, vì chúng tổ chức dữ liệu một cách gọn gàng và dễ dàng ánh xạ vào các định dạng dữ liệu chuẩn.

**6. Tối ưu hóa bộ nhớ với bit-field:**

* Cấu trúc hỗ trợ **bit-field**, cho phép lưu trữ dữ liệu ở mức bit, tiết kiệm bộ nhớ trong các ứng dụng yêu cầu tối ưu hóa tài nguyên, như lập trình nhúng hoặc hệ thống.

**7. Tăng tính dễ bảo trì và mở rộng**

* Cấu trúc giúp mã nguồn dễ hiểu và dễ bảo trì hơn, vì các thành viên được nhóm lại theo ngữ nghĩa. Nếu cần thêm thành viên mới, bạn chỉ cần cập nhật định nghĩa cấu trúc mà không ảnh hưởng đến logic chính của chương trình.
* **Ví dụ**: Thêm một thành viên char lop[10] vào cấu trúc SinhVien không yêu cầu thay đổi các hàm đã viết nếu chúng chỉ truy cập các thành viên hiện có.

### **Ví dụ.**

#include “stdio.h”

#include “string.h”

// Định nghĩa cấu trúc SinhVien

struct SinhVien {

char ten[50]; // Tên sinh viên

int maSV; // Mã số sinh viên

float diemTB; // Điểm trung bình

};

int main() {

// Khai báo một biến kiểu struct SinhVien

struct SinhVien sv1;

// Gán giá trị cho các thành phần của cấu trúc

strcpy(sv1.ten, "Nguyen Van A"); // Gán tên

sv1.maSV = 12345; // Gán mã số

sv1.diemTB = 8.5; // Gán điểm trung bình

// In thông tin sinh viên

printf("Thông tin sinh viên:\n");

printf("Tên: %s\n", sv1.ten);

printf("Mã số: %d\n", sv1.maSV);

printf("Điểm trung bình: %.1f\n", sv1.diemTB);

// Khai báo một mảng các struct SinhVien

struct SinhVien danhSachSV[2];

strcpy(danhSachSV[0].ten, "Tran Thi B");

danhSachSV[0].maSV = 67890;

danhSachSV[0].diemTB = 7.8;

strcpy(danhSachSV[1].ten, "Le Van C");

danhSachSV[1].maSV = 54321;

danhSachSV[1].diemTB = 9.0;

// In danh sách sinh viên

printf("\nDanh sách sinh viên:\n");

for (int i = 0; i < 2; i++) {

printf("Sinh viên %d:\n", i + 1);

printf("Tên: %s\n", danhSachSV[i].ten);

printf("Mã số: %d\n", danhSachSV[i].maSV);

printf("Điểm trung bình: %.1f\n\n", danhSachSV[i].diemTB);

}

return 0;

}

**Đoạn code trên thực hiện:**

**1. Định nghĩa cấu trúc (struct):**

* struct SinhVien là một kiểu dữ liệu do người dùng định nghĩa, dùng để gộp nhiều loại dữ liệu (tên, mã số, điểm) thành một đơn vị.
* Các thành phần (fields) của cấu trúc: ten (chuỗi), maSV (số nguyên), diemTB (số thực).

**2. Khai báo biến kiểu struct:**

* struct SinhVien sv1; tạo một biến sv1 thuộc kiểu struct SinhVien.
* Có thể truy cập các thành phần của cấu trúc bằng dấu chấm (.), ví dụ: sv1.ten, sv1.maSV.

**3. Gán giá trị:**

* Sử dụng strcpy để gán chuỗi cho ten vì đây là mảng ký tự.
* Gán trực tiếp giá trị cho maSV và diemTB.

**4. Mảng của struct:**

* struct SinhVien danhSachSV[2]; tạo một mảng chứa 2 sinh viên.
* Truy cập từng sinh viên trong mảng bằng chỉ **số, ví dụ: danhSachSV[0].ten.**

**5. In thông tin:**

* Dùng printf để in thông tin của từng sinh viên, sử dụng các định dạng phù hợp (%s cho chuỗi, %d cho số nguyên, %.1f cho số thực).

# DANH SÁCH LIÊN KẾT.

### **Khái niệm.**

Danh sách liên kết là một cấu trúc dữ liệu trong lập trình máy tính, được sử dụng để lưu trữ và quản lý một tập hợp các phần tử dữ liệu. Trong danh sách liên kết, mỗi phần tử được gọi là "nút" và bao gồm dữ liệu của nút đó cùng một con trỏ chỉ đến nút tiếp theo trong danh sách.

Mỗi nút trong danh sách liên kết chứa hai phần chính:

**1. Dữ liệu:** Thông tin được lưu trữ trong nút, có thể là bất kỳ kiểu dữ liệu nào, chẳng hạn như số nguyên, số thực, ký tự, hoặc thậm chí là một cấu trúc phức tạp hơn.

**2. Con trỏ tiếp theo:** Một con trỏ chỉ đến nút tiếp theo trong danh sách liên kết. Điều này tạo ra một chuỗi các nút mà mỗi nút chỉ biết đến nút tiếp theo của nó, tạo thành một danh sách.

**Có hai loại danh sách liên kết chính:**

**1. Danh sách liên kết đơn:** Mỗi nút chỉ trỏ đến nút tiếp theo trong danh sách.

**2. Danh sách liên kết đôi:** Mỗi nút chứa một con trỏ không chỉ đến nút tiếp theo, mà còn chỉ đến nút trước đó trong danh sách, tạo thành một danh sách có thể được duyệt cả từ phía trước và từ phía sau.

Các thao tác phổ biến trên danh sách liên kết bao gồm: thêm một nút mới vào đầu hoặc cuối danh sách, xóa một nút, tìm kiếm một giá trị trong danh sách, và duyệt qua các nút của danh sách để thực hiện các thao tác khác nhau.

Danh sách liên kết thường được sử dụng trong các ứng dụng cần thêm/xóa dữ liệu một cách linh hoạt hoặc trong các trường hợp mà kích thước của dữ liệu không biết trước.

### **Lợi ích.**

**Sử dụng danh sách liên kết mang lại nhiều lợi ích quan trọng trong lập trình, bao gồm:**

**1. Khả năng thêm/xóa linh hoạt:** Danh sách liên kết cho phép thêm và xóa các phần tử một cách linh hoạt mà không cần phải di chuyển hoặc sao chép các phần tử khác như trong mảng. Điều này làm cho việc thêm/xóa các phần tử trở nên đơn giản và hiệu quả hơn.

**2. Khả năng mở rộng động:** Danh sách liên kết có thể mở rộng động, tức là bạn có thể thêm bất kỳ số lượng phần tử nào vào danh sách mà không gặp phải giới hạn của kích thước tĩnh như trong mảng.

**3. Duyệt dữ liệu dễ dàng:** Duyệt qua các phần tử trong danh sách liên kết là dễ dàng, vì mỗi phần tử chỉ cần trỏ đến phần tử tiếp theo. Điều này giúp thực hiện các thao tác duyệt và truy cập dữ liệu một cách dễ dàng và hiệu quả.

**4. Khả năng chia sẻ tài nguyên:** Danh sách liên kết cho phép chia sẻ tàinguyên giữa các phần tử một cách dễ dàng. Ví dụ, nếu nhiều danh sách chia sẻ các nút có cùng dữ liệu, không cần phải lưu trữ nhiều bản sao của dữ liệu đó.

**5. Xử lý dữ liệu không đồng nhất:** Danh sách liên kết cho phép bạn lưu trữ dữ liệu không đồng nhất, tức là các phần tử trong danh sách có thể có các kiểu dữ liệu khác nhau. Điều này làm cho việc lưu trữ và quản lý các dữ liệu có cấu trúc phức tạp trở nên dễ dàng hơn.

**6. Điều chỉnh kích thước một cách linh hoạt:** Bạn có thể dễ dàng điều chỉnh kích thước của danh sách liên kết, không như mảng có kích thước cố định. Điều này giúp tiết kiệm bộ nhớ và làm cho việc quản lý dữ liệu trở nên linh hoạt hơn.

### **So sánh danh sách liên kết với mảng.**

So sánh giữa danh sách liên kết và mảng dựa trên một số yếu tố quan trọng:

**1. Khả năng thêm/xóa phần tử:**

- Danh sách liên kết: Thêm và xóa phần tử là linh hoạt và hiệu quả, vì bạn chỉ cần điều chỉnh con trỏ giữa các nút. Việc này có độ phức tạp thời gian O(1).

- Mảng: Thêm và xóa phần tử trong mảng có thể đòi hỏi di chuyển và sao chép các phần tử, đặc biệt là nếu phần tử được thêm hoặc xóa ở giữa mảng. Điều này có độ phức tạp thời gian là O(n) trong trường hợp tệ nhất.

**2. Truy cập phần tử:**

- Danh sách liên kết: Truy cập vào phần tử ở vị trí bất kỳ trong danh sách có độ phức tạp thời gian là O(n) do bạn phải duyệt từ đầu danh sách.

- Mảng: Truy cập vào phần tử trong mảng có độ phức tạp thời gian là O(1) do chỉ cần sử dụng chỉ mục để truy cập.

**3. Sử dụng bộ nhớ:**

- Danh sách liên kết: Danh sách liên kết sử dụng một lượng bộ nhớ linh hoạt hơn, vì các nút có thể được cấp phát động.

- Mảng: Mảng yêu cầu một lượng bộ nhớ liên tục và có kích thước cố định, do đó có thể gặp phải vấn đề về bộ nhớ khi kích thước mảng cần phải được thay đổi.

**4. Thao tác duyệt phần tử:**

- Danh sách liên kết: Thao tác duyệt danh sách liên kết từ đầu đến cuối có thể đòi hỏi độ phức tạp thời gian là O(n).

- Mảng: Thao tác duyệt mảng từ đầu đến cuối có độ phức tạp thời gian là O(n).

**5. Dữ liệu không đồng nhất:**

- Danh sách liên kết: Cho phép lưu trữ các loại dữ liệu không đồng nhất trong các nút khác nhau.

- Mảng: Cần phải lưu trữ các phần tử cùng loại dữ liệu trong mảng.

Tóm lại, danh sách liên kết thích hợp cho các tình huống cần thêm/xóa phần tử thường xuyên và cần linh hoạt về kích thước và cấu trúc dữ liệu, trong khi mảng thích hợp cho việc truy cập phần tử nhanh chóng và yêu cầu một lượng bộ nhớ liên tục.

### **Ví dụ.**

#include “stdio.h”

#include “stdlib.h”

struct Node {

int data;

struct Node\* pNext;

};

struct List {

struct Node\* pHead;

struct Node\* pTail;

};

struct Node\* getNode(int x) {

struct Node\* p = (struct Node\*)malloc(sizeof(struct Node));

if (p == NULL) {

printf("Not enough space of memory!");

return NULL;

}

p->data = x;

p->pNext = NULL;

return p;

}

void addHead(struct List\* l, struct Node\* new\_node) {

if (l->pHead == NULL)

l->pHead = l->pTail = new\_node;

else {

new\_node->pNext = l->pHead;

l->pHead = new\_node;

}

}

int main() {

struct List l;

int n;

scanf("%d", &n);

while (n--) {

int x;

scanf("%d", &x);

struct Node\* new\_node = getNode(x);

if (new\_node != NULL) {

addHead(&l, new\_node);

*// Thêm phần tử vào cuối danh sách ngay sau khi thêm vào đầu danh sách*

if (l.pTail == NULL)

l.pTail = new\_node;

}

}

*// Xuất danh sách liên kết*

struct Node\* temp = l.pHead;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->pNext;

}

return 0;

}

Đoạn code trên thực hiện:

1. Định nghĩa cấu trúc **Node**, đại diện cho một nút trong danh sách liên kết, với hai trường: **data** (chứa dữ liệu của nút) và **pNext** (con trỏ chỉ đến nút tiếp theo trong danh sách).

2. Định nghĩa cấu trúc **List**, chứa hai con trỏ **pHead** và **pTail**, lần lượt chỉ đến đầu và cuối của danh sách liên kết.

3. Xây dựng hàm **getNode(int x)** để tạo một nút mới với dữ liệu **x**. Hàm này sử dụng **malloc** để cấp phát bộ nhớ cho nút mới và trả về con trỏ tới nút đó.

4. Xây dựng hàm **addHead(struct List\* l, struct Node\* new\_node)** để thêm một nút mới vào đầu của danh sách liên kết. Nếu danh sách đang rỗng, nút mới được thêm vào cũng là đuôi của danh sách. Nếu không, nút mới được thêm vào đầu danh sách và cập nhật con trỏ đầu danh sách.

5. Trong hàm **main()**, nhập số lượng phần tử cần thêm vào danh sách từ người dùng. Sau đó, trong mỗi vòng lặp, nhập giá trị của một phần tử và tạo một nút mới chứa giá trị đó. Nếu nút mới được tạo thành công, nó được thêm vào đầu danh sách và kiểm tra nếu danh sách rỗng, nút mới cũng được coi là đuôi của danh sách.

6. Cuối cùng, danh sách liên kết được xuất ra màn hình bằng cách duyệt từ đầu đến cuối và in ra giá trị của mỗi nút.

# ỨNG DỤNG

**XÂY DỰNG ỨNG DỤNG CHO VIỆC QUẢN LÍ SINH VIÊN.**

**Các tính năng:**

**1. Thêm Sinh Viên.**

**2. Hiện thị danh sách.**

**3. Tìm kiếm sinh viên.**

**4. Sửa thông tin sinh viên.**

**5. Xóa sinh viên.**

**6. Sắp xếp danh sách (tăng theo điểm).**

**7. Sắp xếp danh sách (giảm theo điểm).**

**8. Lưu danh sách vào file.**

**9. Đọc danh sách từ file.**

**10. Thống kê điểm sinh viên.**

**Mục đích:**

**1. Lưu trữ và tổ chức thông tin sinh viên một cách có hệ thống**

* Mỗi sinh viên có thông tin như: mã số, họ tên, điểm trung bình.
* Việc lưu trữ có tổ chức giúp dễ dàng tìm kiếm, hiển thị, thống kê, sắp xếp hoặc cập nhật thông tin khi cần.

**2. Tự động hóa công việc quản lý**

* Giảm bớt công việc thủ công như ghi chép, tính toán, tìm kiếm hồ sơ.
* Tăng tốc độ, độ chính xác và hiệu quả trong việc theo dõi sinh viên.

**3. Hỗ trợ việc phân tích và ra quyết định**

* Dễ dàng thống kê sinh viên giỏi, sinh viên cần hỗ trợ học tập.
* Hỗ trợ giáo viên, nhà trường trong việc đánh giá và lập kế hoạch giảng dạy.

**Đoạn Code:**

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

// Cấu trúc sinh viên

typedef struct Student {

    char id[10];

    char name[50];

    float gpa;

    struct Student\* next; // Con trỏ cho danh sách liên kết

} Student;

// Cấu trúc danh sách liên kết

typedef struct {

    Student\* head;

    int size;

} StudentList;

// Hàm khởi tạo danh sách

StudentList\* initList() {

    StudentList\* list = (StudentList\*)malloc(sizeof(StudentList));

    list->head = NULL;

    list->size = 0;

    return list;

}

// Hàm thêm sinh viên

void addStudent(StudentList\* list) {

    Student\* newStudent = (Student\*)malloc(sizeof(Student));

    printf("\nNhap ma so sinh vien: ");

    scanf("%s", newStudent->id);

    printf("Nhap ho ten: ");

    getchar(); // Xóa buffer

    fgets(newStudent->name, 50, stdin);

    newStudent->name[strcspn(newStudent->name, "\n")] = 0; // Xóa ký tự newline

    printf("Nhap diem trung binh: ");

    scanf("%f", &newStudent->gpa);

    newStudent->next = NULL;

    if (list->head == NULL) {

        list->head = newStudent;

    } else {

        newStudent->next = list->head;

        list->head = newStudent;

    }

    list->size++;

    printf("Them sinh vien thanh cong!\n");

}

// Hàm hiển thị danh sách sinh viên

void displayList(StudentList\* list) {

    if (list->size == 0) {

        printf("\nDanh sach rong!\n");

        return;

    }

    printf("\nDanh sach sinh vien:\n");

    printf("%-10s %-30s %-10s\n", "Ma SV", "Ho ten", "Diem TB");

    printf("----------------------------------------\n");

    Student\* current = list->head;

    while (current != NULL) {

        printf("%-10s %-30s %-10.2f\n", current->id, current->name, current->gpa);

        current = current->next;

    }

}

// Hàm tìm kiếm sinh viên theo mã

Student\* findStudent(StudentList\* list, char\* id) {

    Student\* current = list->head;

    while (current != NULL) {

        if (strcmp(current->id, id) == 0) {

            return current;

        }

        current = current->next;

    }

    return NULL;

}

// Hàm xóa sinh viên

void deleteStudent(StudentList\* list) {

    char id[10];

    printf("\nNhap ma so sinh vien can xoa: ");

    scanf("%s", id);

    Student\* current = list->head;

    Student\* prev = NULL;

    while (current != NULL && strcmp(current->id, id) != 0) {

        prev = current;

        current = current->next;

    }

    if (current == NULL) {

        printf("Khong tim thay sinh vien!\n");

        return;

    }

    if (prev == NULL) {

        list->head = current->next;

    } else {

        prev->next = current->next;

    }

    free(current);

    list->size--;

    printf("Xoa sinh vien thanh cong!\n");

}

// Hàm sửa thông tin sinh viên

void editStudent(StudentList\* list) {

    char id[10];

    printf("\nNhap ma so sinh vien can sua: ");

    scanf("%s", id);

    Student\* student = findStudent(list, id);

    if (student == NULL) {

        printf("Khong tim thay sinh vien!\n");

        return;

    }

    printf("Nhap ho ten moi: ");

    getchar();

    fgets(student->name, 50, stdin);

    student->name[strcspn(student->name, "\n")] = 0;

    printf("Nhap diem trung binh moi: ");

    scanf("%f", &student->gpa);

    printf("Cap nhat thong tin thanh cong!\n");

}

// Hàm sắp xếp danh sách theo điểm trung bình

void sortList(StudentList\* list, int ascending) {

    if (list->size <= 1) return;

    Student \*current, \*index;

    char tempId[10], tempName[50];

    float tempGpa;

    for (current = list->head; current != NULL; current = current->next) {

        for (index = current->next; index != NULL; index = index->next) {

            if ((ascending && current->gpa > index->gpa) ||

                (!ascending && current->gpa < index->gpa)) {

                // Hoán đổi thông tin

                strcpy(tempId, current->id);

                strcpy(tempName, current->name);

                tempGpa = current->gpa;

                strcpy(current->id, index->id);

                strcpy(current->name, index->name);

                current->gpa = index->gpa;

                strcpy(index->id, tempId);

                strcpy(index->name, tempName);

                index->gpa = tempGpa;

            }

        }

    }

    printf("Sap xep danh sach thanh cong!\n");

}

// Hàm lưu danh sách vào file

void saveToFile(StudentList\* list) {

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

    if (file == NULL) {

        printf("Khong the mo file!\n");

        return;

    }

    Student\* current = list->head;

    while (current != NULL) {

        fprintf(file, "%s,%s,%.2f\n", current->id, current->name, current->gpa);

        current = current->next;

    }

    fclose(file);

    printf("Luu danh sach vao file thanh cong!\n");

}

// Hàm đọc danh sách từ file

void loadFromFile(StudentList\* list) {

    FILE\* file = fopen("students.txt", "r");

    if (file == NULL) {

        printf("Khong the mo file hoac file khong ton tai!\n");

        return;

    }

    // Xóa danh sách hiện tại

    Student\* current = list->head;

    while (current != NULL) {

        Student\* temp = current;

        current = current->next;

        free(temp);

    }

    list->head = NULL;

    list->size = 0;

    char id[10], name[50];

    float gpa;

    while (fscanf(file, "%[^,],%[^,],%f\n", id, name, &gpa) == 3) {

        Student\* newStudent = (Student\*)malloc(sizeof(Student));

        strcpy(newStudent->id, id);

        strcpy(newStudent->name, name);

        newStudent->gpa = gpa;

        newStudent->next = NULL;

        if (list->head == NULL) {

            list->head = newStudent;

        } else {

            newStudent->next = list->head;

            list->head = newStudent;

        }

        list->size++;

    }

    fclose(file);

    printf("Doc danh sach tu file thanh cong!\n");

}

// Hàm thống kê sinh viên có điểm trên 8.0

void statsAboveEight(StudentList\* list) {

    int count = 0;

    Student\* current = list->head;

    while (current != NULL) {

        if (current->gpa >= 8.0) {

            count++;

        }

        current = current->next;

    }

    printf("\nSo luong sinh vien co diem trung binh >= 8.0: %d\n", count);

}

// Hàm giải phóng danh sách

void freeList(StudentList\* list) {

    Student\* current = list->head;

    while (current != NULL) {

        Student\* temp = current;

        current = current->next;

        free(temp);

    }

    free(list);

}

// Hàm hiển thị menu

void displayMenu() {

    printf("\n=== QUAN LY SINH VIEN ===\n");

    printf("1. Them sinh vien\n");

    printf("2. Hien thi danh sach\n");

    printf("3. Tim kiem sinh vien\n");

    printf("4. Sua thong tin sinh vien\n");

    printf("5. Xoa sinh vien\n");

    printf("6. Sap xep danh sach (tang dan theo diem)\n");

    printf("7. Sap xep danh sach (giam dan theo diem)\n");

    printf("8. Luu danh sach vao file\n");

    printf("9. Doc danh sach tu file\n");

    printf("10. Thong ke sinh vien diem >= 8.0\n");

    printf("0. Thoat\n");

    printf("Nhap lua chon: ");

}

int main() {

    StudentList\* list = initList();

    int choice;

    do {

        displayMenu();

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                addStudent(list);

                break;

            case 2:

                displayList(list);

                break;

            case 3: {

                char id[10];

                printf("\nNhap ma so sinh vien can tim: ");

                scanf("%s", id);

                Student\* student = findStudent(list, id);

                if (student) {

                    printf("%-10s %-30s %-10s\n", "Ma SV", "Ho ten", "Diem TB");

                    printf("----------------------------------------\n");

                    printf("%-10s %-30s %-10.2f\n", student->id, student->name, student->gpa);

                } else {

                    printf("Khong tim thay sinh vien!\n");

                }

                break;

            }

            case 4:

                editStudent(list);

                break;

            case 5:

                deleteStudent(list);

                break;

            case 6:

                sortList(list, 1); // Sắp xếp tăng dần

                displayList(list);

                break;

            case 7:

                sortList(list, 0); // Sắp xếp giảm dần

                displayList(list);

                break;

            case 8:

                saveToFile(list);

                break;

            case 9:

                loadFromFile(list);

                break;

            case 10:

                statsAboveEight(list);

                break;

            case 0:

                printf("Tam biet!\n");

                break;

            default:

                printf("Lua chon khong hop le!\n");

        }

    } while (choice != 0);

    freeList(list);

    return 0;

}

# KẾT LUẬN

Việc xây dựng chương trình quản lý sinh viên bằng ngôn ngữ C như trên không chỉ đơn thuần là một bài tập lập trình, mà còn mang lại nhiều giá trị thiết thực trong cả học tập và ứng dụng thực tế.

Trước hết, chương trình giúp người học rèn luyện kỹ năng lập trình cơ bản như sử dụng cấu trúc dữ liệu (struct), danh sách liên kết, con trỏ, thao tác với file và chuỗi trong ngôn ngữ C. Đây là những kỹ năng quan trọng và nền tảng trong khoa học máy tính, giúp sinh viên phát triển tư duy logic và khả năng tổ chức dữ liệu một cách hiệu quả.

Thứ hai, chương trình này mô phỏng lại một phần nhỏ của hệ thống quản lý sinh viên thực tế đang được sử dụng tại các trường học, trung tâm đào tạo hay phần mềm quản lý nội bộ của doanh nghiệp. Thông qua bài toán này, người học có thể hình dung được cách xây dựng một hệ thống phần mềm từ khâu nhập liệu, xử lý dữ liệu đến lưu trữ lâu dài, từ đó làm nền tảng để phát triển những phần mềm lớn hơn và phức tạp hơn trong tương lai.

Ngoài ra, hệ thống còn góp phần tự động hóa các thao tác quản lý thông tin sinh viên như thêm, sửa, xóa, tìm kiếm và thống kê. Điều này giúp tiết kiệm thời gian, nâng cao độ chính xác và giảm thiểu sai sót so với phương pháp làm việc thủ công truyền thống.

Cuối cùng, việc triển khai một chương trình như vậy giúp sinh viên làm quen với quy trình xây dựng một ứng dụng phần mềm thực tế: từ phân tích yêu cầu, thiết kế dữ liệu, tổ chức giao diện người dùng, đến việc lưu trữ và xử lý dữ liệu. Đây là bước khởi đầu quan trọng trên hành trình trở thành một lập trình viên chuyên nghiệp.

# TÀI LIỆU THAM KHẢO

[1]. “C Tutorial – Learning C Programming”,[Online]. Available: [C Tutorial (w3schools.com)](https://www.w3schools.com/c/index.php). [Accessed 12/05/2025].

[2]. “Lập trình C cơ bản – Giới thiệu ngôn ngữ C”,[Online]. Available: [Lập trình C cơ bản - Giới thiệu ngôn ngữ C (200lab.io)](https://200lab.io/blog/lap-trinh-c-co-ban/). [Accessed 12/05/2025].

[3]. “Stack Overflow”, [Online]. Available: [Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/). [Accessed 12/05/2025].

[4]. “GitHub”, [Online]. Available: [GitHub](https://github.com/) (<https://github.com>). [Accessed 12/05/2025].