**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN**

**HỌC PHẦN: KĨ THUẬT LẬP TRÌNH**

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

**Sinh viên thực hiện:  BÙI HỮU HÙNG**

**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 BÀI TẬP LỚN**

**HỌC PHẦN: KĨ THUẬT LẬP TRÌNH**

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

**Sinh viên thực hiện:  BÙI HỮU HÙNG**

**Lớp:    CQ.65.CNTT**

**Khoá:   K65**

**Tp. Hồ Chí Minh, năm 2025**

# MỤC LỤC

[MỤC LỤC i](#_Toc198802038)

[LỜI CẢM ƠN iii](#_Toc198802039)

[NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN iv](#_Toc198802040)

[A. LÝ THUYẾT 1](#_Toc198802041)

[**1.** **HÀM:** 1](#_Toc198802042)

[**1.1.** **Khái niệm:** 1](#_Toc198802043)

[**1.2.** **Ví dụ:** 3](#_Toc198802044)

[**2.** **CON TRỎ:** 5](#_Toc198802045)

[**2.1.** **Khái niệm:** 5](#_Toc198802046)

[**2.2.** **Ví dụ:** 5](#_Toc198802047)

[**3.** **CON TRỎ MẢNG:** 7](#_Toc198802048)

[**3.1.** **Khái niệm:** 7](#_Toc198802049)

[**3.2.** **Ví dụ:** 7](#_Toc198802050)

[**4.** **MẢNG CON TRỎ:** 9](#_Toc198802051)

[**4.1.** **Khái niệm:** 9](#_Toc198802052)

[**4.2.** **Ví dụ:** 9](#_Toc198802053)

[**5.** **CON TRỎ HÀM:** 11](#_Toc198802054)

[**5.1.** **Khái niệm:** 11](#_Toc198802055)

[**5.2.** **Lợi ích khi sử dụng hàm con trỏ:** 11](#_Toc198802056)

[**5.3.** **Ví dụ:** 12](#_Toc198802057)

[**6.** **CẤP PHÁT ĐỘNG:** 14](#_Toc198802058)

[**6.1.** **Khái niệm:** 14](#_Toc198802059)

[**6.2.** **Lợi ích khi sử dụng cấp phát động:** 14](#_Toc198802060)

[**6.3.** **So sánh với mảng tĩnh:** 15](#_Toc198802061)

[**6.4.** **Ví dụ:** 16](#_Toc198802062)

[**7.** **XỬ LÍ TỆP:** 18](#_Toc198802063)

[**7.1.** **Khái niệm:** 18](#_Toc198802064)

[**7.2.** **Tại sao cần xử lí tệp:** 18](#_Toc198802065)

[**7.3.** **Ví dụ:** 19](#_Toc198802066)

[**8. DANH SÁCH LIÊN KẾT:** 21](#_Toc198802067)

[**8.1.** **Khái niệm:** 21](#_Toc198802068)

[**8.2.** **Lợi ích khi sử dụng danh sách liên kết:** 21](#_Toc198802069)

[**8.3.** **So sánh danh sách liên kết với mảng:** 22](#_Toc198802070)

[**8.4.** **Ví dụ:** 23](#_Toc198802071)

[B. ỨNG DỤNG 26](#_Toc198802072)

[KẾT LUẬN 32](#_Toc198802073)

[TÀI LIỆU THAM KHẢO 33](#_Toc198802074)

# **LỜI CẢM ƠN**

Kính chào Thầy Nhã và các bạn sinh viên thân mến!

Em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến Thầy Nhã – người đã tận tâm truyền đạt không chỉ kiến thức chuyên môn quý báu mà còn là nguồn cảm hứng mạnh mẽ giúp em thêm yêu và hiểu sâu hơn về lĩnh vực Công nghệ Thông tin.

Đầu tiên, nhờ sự chỉ dẫn tận tâm, kiến thức chuyên môn sâu rộng cùng những góp ý quý báu của Thầy, em đã có thể hoàn thành bài báo cáo một cách hiệu quả và có ý nghĩa hơn. Sự hỗ trợ và động viên từ Thầy,Gia Đình và bạn bè chính là động lực lớn giúp em vượt qua những khó khăn trong quá trình nghiên cứu và thực hành. Em xin chân thành cảm ơn Thầy Nhã – những người thân yêu – gia đình và người thân – vì sự ủng hộ, động viên và niềm tin không ngừng nghỉ mà họ đã dành cho em trong suốt quá trình học tập và thực hiện báo cáo này.

Cuối cùng, em xin gửi lời cảm ơn đến tất cả những người đã đọc báo cáo này và quan tâm đến công trình của em. Sự quan tâm và phản hồi của Thầy và các bạn là nguồn động lực lớn để em tiếp tục phát triển và nỗ lực hơn nữa trong tương lai.

Xin cảm ơn và chúc mọi người một ngày thật hạnh phúc và tràn đầy niềm vui!

Trân trọng,

*Bùi Hữu Hùng*

# NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

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

|  |
| --- |
| **Tp. Hồ Chí Minh, ngày tháng năm 2025 Giảng viên hướng dẫn**  **ThS. TRẦN PHONG NHÃ** |

# 

# LÝ THUYẾT

## **HÀM:**

* 1. **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.

* 1. **Ví dụ:**
* Đoạn code không sử dụng hàm:

#include <stdio.h>

int main() {

int n;

int sum = 0;

printf("Nhap mot so nguyen duong: ");

scanf("%d", &n);

*// Tinh tong cac so tu 1 den n*

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

sum += i;

}

printf("Tong cac so tu 1 den %d la: %d**\n**", n, sum);

**return** 0;

}

* Đoạn code có sử dụng hàm:

#include <stdio.h>

*// Ham de tinh tong cac so tu 1 den n*

int sumUpToN(int n) {

int sum = 0;

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

sum += i;

}

**return** sum;

}

int main() {

int n;

printf("Nhap mot so nguyen duong: ");

scanf("%d", &n);

*// Goi ham sumUpToN de tinh tong*

int result = sumUpToN(n);

printf("Tong cac so tu 1 den %d la: %d**\n**", n, result);

**return** 0;

}

Trong đoạn code đầu tiên, tổng của các số từ 1 đến n được tính trực tiếp trong hàm **main()**. Trong khi đó, trong đoạn code thứ hai, chúng ta đã định nghĩa một hàm **sumUpToN()** để tính tổng này. Bằng cách này, chúng ta có thể tái sử dụng hàm **sumUpToN()** ở bất kỳ đâu trong chương trình và giúp mã nguồn trở nên sạch sẽ và dễ quản lý hơn

## **CON TRỎ:**

* 1. **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.

* 1. **Ví dụ:**

Đoạn Code thực hiện chức năng đổi vị trí 2 số nguyên không sử dụng con trỏ:

#include <stdio.h>

*// Ham de tinh tong cac so tu 1 den n*

int sumUpToN(int n) {

int sum = 0;

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

sum += i;

}

**return** sum;

}

int main() {

int n;

printf("Nhap mot so nguyen duong: ");

scanf("%d", &n);

*// Goi ham sumUpToN de tinh tong*

int result = sumUpToN(n);

printf("Tong cac so tu 1 den %d la: %d**\n**", n, result);

**return** 0;

}

Khi không sử dụng con trỏ, ta đang truyền giá trị của các biến đó vào trong hàm. Trong hàm, các tham số được khai báo như là các biến mới (cục bộ) và giá trị của các tham số này được sao chép từ các biến gọi hàm. Khi ta thay đổi giá trị của các tham số trong hàm, ta thực sự chỉ thay đổi giá trị của các biến cục bộ này, không ảnh hưởng đến các biến ban đầu trong phạm vi gọi hàm (trong hàm **main()** trong trường hợp này).

Đoạn Code thực hiện chức năng đổi vị trí 2 số nguyên sử dụng con trỏ:

#include <stdio.h>

*// Ham swap su dung con tro*

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 5, y = 10;

printf("Truoc khi hoan doi: x = %d, y = %d**\n**", x, y);

*// Goi ham swap va truyen dia chi cua x va y*

swap(&x, &y);

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

**return** 0;

}

## **CON TRỎ MẢNG:**

* 1. **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.

* 1. **Ví dụ:**

#include <stdio.h>

**int** main() {

*// Khai bao mang va con tro*

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

**int** \*ptr;

*// Gan dia chi cua phan tu dau tien cua mang cho con tro*

ptr = a;

*// Truy cap va in ra cac phan tu cua mang bang cach su dung con tro*

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 **a** 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:**

* 1. **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`.

* 1. **Ví dụ:**

#include <stdio.h>

**int** main() {

*// Khai bao mang va con tro*

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

**int** \*ptr;

*// Gan dia chi cua phan tu dau tien cua mang cho con tro*

ptr = a;

*// Truy cap va in ra các phan tu cua mang bang cach su dung con tro*

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;

}

**Giải thích:**

- Trong ví dụ này, ta đã khai báo một mảng con trỏ **ptrArray** có 3 phần tử. Mỗi phần tử của mảng này là một con trỏ kiểu **int**.

- Sau đó, chúng ta đã khai báo và khởi tạo các biến **a**, **b**, và **c** lần lượt với các giá trị là 10, 20 và 30.

- Tiếp theo, chúng ta gán địa chỉ của các biến này cho các phần tử của mảng con trỏ **ptrArray** bằng cách sử dụng toán tử **&**.

- Cuối cùng, chúng ta sử dụng toán tử dereference **\*** để truy cập và in ra giá trị của các phần tử của mảng con trỏ.

Trong ví dụ này, mỗi phần tử của mảng con trỏ **ptrArray** đều trỏ đến một biến khác nhau trong bộ nhớ, cho phép chúng ta lưu trữ địa chỉ của các biến khác nhau trong một mảng.

## **CON TRỎ HÀM:**

* 1. **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 bao con tro ham*

funcPtr = sum; *// Gan con tro ham vao ham sum*

**int** result = funcPtr(3, 5); *// Su dung con tro ham de goi ham sum*

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

**return** 0;

}

* 1. **Lợi ích khi sử dụng hàm con trỏ:**

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.

* 1. **Ví dụ:**

#include <stdio.h>

*// Khai bao mot ham con tro co kieu tra ve la int va nhan hai tham so kieu int*

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

*// Ham tinh tong cua hai so*

**int** sum(**int** a, **int** b) {

**return** a + b;

}

**int** main() {

*// Gan con tro ham vao ham sum*

sumPtr = sum;

*// Su dung con tro ham de goi ham sum*

**int** result = sumPtr(3, 5);

*// In ket qua: 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:**

* 1. **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.

* 1. **Lợi ích khi sử dụng cấp phát động:**

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.

* 1. **So sánh với mảng 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.

* 1. **Ví dụ:**

#include <stdio.h>

#include <stdlib.h> *// Thu vien de su dung cac ham cap phat dong*

**int** main() {

**int** n; *// So luong phan tu cua mang, nhap tu nguoi dung*

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

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

*// Cap phat dong mot mang có n phan tu kieu int*

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

*// Kiem tra xem cap phat da thanh cong chua*

**if** (a == NULL) {

printf(**"Khong du bo nho!"**);

**return** 1; *// Ket thuc chuong trinh voi ma loi*

}

*// Nhap cac phan tu cua mang tu nguoi dung*

printf(**"Nhap cac phan tu cua mang:\n"**);

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

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

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

}

*// In cac phan tu cua mang*

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

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

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

}

*// Giai phong bo nho da cap phat dong*

free(a);

**return** 0;

}

Trong ví dụ này:

1. Người dùng nhập số lượng phần tử của mảng từ bàn phím.

2. Sử dụng hàm **malloc** để cấp phát động một mảng có số lượng phần tử là **n**.

3. Kiểm tra xem việc cấp phát đã thành công hay không. Nếu **malloc** trả về **NULL**, điều này có thể chỉ ra rằng không đủ bộ nhớ trống còn.

4. Nhập các phần tử của mảng từ người dùng.

5. In các phần tử của mảng ra màn hình.

6. Sử dụng hàm **free** để giải phóng bộ nhớ đã cấp phát động khi không cần thiết nữa.

Ví dụ này minh họa việc sử dụng cấp phát động để tạo ra một mảng động và giải phóng bộ nhớ sau khi đã sử dụng xong.

## **XỬ LÍ TỆP:**

* 1. **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.

* 1. **Tại sao cần xử lí 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.

* 1. **Ví dụ:**

#include <stdio.h>

**int** main() {

*// Mo tep de doc*

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

*// Kiem tra xem viec mo tep co thanh cong hay khong*

**if** (file == NULL) {

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

**return** 1; }

*// Doc va hien thi noi dung cua tep*

**char** line[100]; *// Buffer de luu du lieu tu tep*

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

printf(**"%s"**, line);

}

*// Dong tep sau khi da doc xong*

fclose(file);

**return** 0;

Kết quả khi chạy chương trình này sẽ là hiển thị nội dung của tệp "input.txt" lên màn hình:

**Hello, world!**

**This is a text file.**

Trong ví dụ này:

- Chúng ta sử dụng hàm **fopen** để mở tệp **"input.txt"** để đọc (**"r"** là chế độ đọc).

- Sử dụng một vòng lặp để đọc từng dòng của tệp bằng cách sử dụng hàm **fgets**.

- Hiển thị từng dòng được đọc lên màn hình bằng cách sử dụng hàm **printf**.

- Cuối cùng, chúng ta sử dụng hàm **fclose** để đóng tệp sau khi đã đọc xong.

## **8. DANH SÁCH LIÊN KẾT:**

* 1. **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.

* 1. **Lợi ích khi sử dụng danh sách liên kết:**

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 tài nguyên 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.

* 1. **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.

* 1. **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);

*// Them phan tu vao cuoi danh sach ngay sau khi them vao dau danh sach*

**if** (l.pTail == NULL)

l.pTail = new\_node;

}

}

*// Xuat danh sach lien ket*

**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 cần thiết:**

1. **Nhập sinh viên**
2. **Hiển thị sinh viên**
3. **Tính điểm trung bình lớp**
4. **Lọc sinh viên giỏi (>=8.0)**
5. **Xóa sinh viên theo mã số sinh viên**
6. **Ghi học sinh giỏi vào file**

**Mục đích của bài toán:**

* Bài toán nhằm xây dựng một chương trình quản lý sinh viên phục vụ cho việc nhập, lưu trữ, xử lý và hiển thị thông tin sinh viên một cách hiệu quả và có hệ thống. Thông tin quản lý bao gồm: mã sinh viên, họ tên, giới tính, năm sinh, điểm các môn học (Toán, Lý, Hóa) và điểm trung bình.
* Để hiện thực hóa bài toán, em sử dụng danh sách liên kết đơn để lưu trữ danh sách sinh viên. Việc sử dụng danh sách liên kết giúp:
  + Quản lý dữ liệu động (không giới hạn số lượng sinh viên).
  + Linh hoạt trong việc thêm hoặc xóa sinh viên.
  + Tối ưu hóa bộ nhớ sử dụng nhờ cấp phát động.
* Ngoài ra, chương trình còn cung cấp các chức năng tiện ích như: • Tính điểm trung bình của cả lớp.
* Lọc danh sách sinh viên giỏi (có điểm trung bình ≥ 8.0).
* Ghi danh sách sinh viên giỏi ra file để lưu trữ.
* Xóa sinh viên theo mã số sinh viên.
* Thông qua bài toán, em rèn luyện được kỹ năng lập trình sử dụng cấu trúc dữ liệu động, tổ chức chương trình theo hướng mô-đun, và biết cách kết hợp giữa xử lý dữ liệu và thao tác file trong ngôn ngữ lập trình C.

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

*//Cau truc du lieu*

**typedef** **struct** {

char maSV[20];

char hoTen[50];

char gioiTinh[10];

int namSinh;

float toan, ly, hoa;

float diemTB;

} SinhVien;

*//Cau truc danh sach lien ket*

**typedef** **struct** Node {

SinhVien sv;

**struct** Node\* next;

} Node;

**typedef** **struct** {

Node\* head;

} DanhSach;

*//Ham nhap sinh vien*

void nhapSV(SinhVien\* sv) {

printf("Nhap ma SV: ");

scanf("%s", sv->maSV);

printf("Nhap ho ten: ");

fflush(stdin);

fgets(sv->hoTen, **sizeof**(sv->hoTen), stdin);

sv->hoTen[strcspn(sv->hoTen, "**\n**")] = '\0';

printf("Nhap gioi tinh: ");

scanf("%s", sv->gioiTinh);

printf("Nhap nam sinh: ");

scanf("%d", &sv->namSinh);

printf("Nhap diem Toan, Ly, Hoa: ");

scanf("%f%f%f", &sv->toan, &sv->ly, &sv->hoa);

sv->diemTB = (sv->toan + sv->ly + sv->hoa) / 3;

}

*//Cap phat bo nho*

Node\* taoNode(SinhVien sv) {

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

p->sv = sv;

p->next = NULL;

**return** p;

}

*//Ham them sinh vien*

void themSV(DanhSach\* ds, SinhVien sv) {

Node\* p = taoNode(sv);

**if** (ds->head == NULL) {

ds->head = p;

} **else** {

Node\* temp = ds->head;

**while** (temp->next != NULL)

temp = temp->next;

temp->next = p;

}

}

*//Ham hien thi sinh vien*

void hienThi(DanhSach ds) {

Node\* p = ds.head;

printf("**\n**%-10s %-10s %-10s %-10s %-6s %-6s %-6s %-6s**\n**",

"MaSV", "HoTen", "GioiTinh", "NamSinh", "Toan", "Ly", "Hoa","DTB");

**while** (p != NULL) {

SinhVien sv = p->sv;

printf("%-10s %-10s %-10s %-10d %-6.2f %-6.2f %-6.2f %-6.2f**\n**",

sv.maSV, sv.hoTen, sv.gioiTinh, sv.namSinh,

sv.toan, sv.ly, sv.hoa, sv.diemTB);

p = p->next;

}

}

*//Ham tinh diem trung binh*

float tinhDTB(DanhSach ds) {

Node\* p = ds.head;

int count = 0;

float tong = 0;

**while** (p != NULL) {

tong += p->sv.diemTB;

count++;

p = p->next;

}

**return** count == 0 ? 0 : tong / count;

}

*//Ham loc sinh vien gioi*

DanhSach locSinhVienGioi(DanhSach ds) {

DanhSach gioi;

gioi.head = NULL;

Node\* p = ds.head;

**while** (p != NULL) {

**if** (p->sv.diemTB >= 8.0)

themSV(&gioi, p->sv);

p = p->next;

}

**return** gioi;

}

*//Ham xoa sinh vien*

int xoaSinhVien(DanhSach\* ds, **const** char\* maSV) {

Node\* p = ds->head;

Node\* prev = NULL;

**while** (p != NULL) {

**if** (strcmp(p->sv.maSV, maSV) == 0) {

**if** (prev == NULL) {

ds->head = p->next;

} **else** {

prev->next = p->next;

}

free(p);

**return** 1;

}

prev = p;

p = p->next;

}

**return** 0;

}

*//Ham ghi sinh vien gioi ra file*

void ghiSinhVienGioiRaFile(DanhSach ds, **const** char\* filename) {

FILE\* f = fopen(filename, "w");

**if** (f == NULL) {

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

**return**;

}

Node\* p = ds.head;

fprintf(f, "%-10s %-10s %-10s %-10s %-6s %-6s %-6s %-6s**\n**",

"MaSV", "HoTen", "GioiTinh", "NamSinh", "Toan", "Ly", "Hoa","DTB");

**while** (p != NULL) {

**if** (p->sv.diemTB >= 8.0) {

SinhVien sv = p->sv;

fprintf(f, "%-10s %-10s %-10s %-10d %-6.2f %-6.2f %-6.2f %6.2f**\n**",

sv.maSV, sv.hoTen, sv.gioiTinh, sv.namSinh,

sv.toan, sv.ly, sv.hoa, sv.diemTB);

}

p = p->next;

}

fclose(f);

printf("Da ghi danh sach sinh vien gioi vao file %s**\n**", filename);

}

void giaiPhong(DanhSach\* ds) {

Node\* p = ds->head;

**while** (p != NULL) {

Node\* temp = p;

p = p->next;

free(temp);

}

ds->head = NULL;

}

*//Ham chuc nang chinh*

int main() {

DanhSach ds;

ds.head = NULL;

int choice;

**do** {

printf("**\n**================== MENU ==================**\n**");

printf("1. Nhap danh sach sinh vien**\n**");

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

printf("3. Tinh diem TB lop**\n**");

printf("4. Loc sinh vien gioi (DTB >= 8.0)**\n**");

printf("5. Xoa sinh vien theo ma SV**\n**");

printf("6. Ghi danh sach sinh vien gioi vao file**\n**");

printf("7. Thoat**\n**");

printf("Chon chuc nang: ");

scanf("%d", &choice);

**switch** (choice) {

**case** 1: {

int n;

printf("Nhap so sinh vien: ");

scanf("%d", &n);

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

printf("**\n**-- Sinh vien thu %d --**\n**", i + 1);

SinhVien sv;

nhapSV(&sv);

themSV(&ds, sv);

}

**break**;

}

**case** 2:

hienThi(ds);

**break**;

**case** 3:

printf("Diem trung binh ca lop: %.2f**\n**", tinhDTB(ds));

**break**;

**case** 4: {

DanhSach gioi = locSinhVienGioi(ds);

printf("Danh sach sinh vien gioi:");

hienThi(gioi);

giaiPhong(&gioi);

**break**;

}

**case** 5: {

char maXoa[20];

printf("Nhap ma sinh vien can xoa: ");

scanf("%s", maXoa);

**if** (xoaSinhVien(&ds, maXoa)) {

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

} **else** {

printf("Khong tim thay sinh vien co ma %s.**\n**", maXoa);

}

**break**;

}

**case** 6: {

ghiSinhVienGioiRaFile(ds, "sinhvien\_gioi.txt");

**break**;

}

**case** 7:

printf("Ket thuc chuong trinh.**\n**");

**break**;

default:

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

}

} **while** (choice != 7);

giaiPhong(&ds);

**return** 0;

}

# KẾT LUẬN

Ứng dụng quản lý sinh viên là một công cụ hữu ích giúp các cơ sở giáo dục, giảng viên và cán bộ quản lý có thể theo dõi, cập nhật và xử lý thông tin sinh viên một cách hiệu quả, nhanh chóng và chính xác. Thay vì phải xử lý dữ liệu thủ công, việc áp dụng phần mềm quản lý sinh viên giúp tiết kiệm thời gian, giảm thiểu sai sót và tăng tính chuyên nghiệp trong công tác quản lý.

Thông qua ứng dụng, các chức năng như quản lý hồ sơ sinh viên, điểm số, lớp học, thông báo, và các thông tin liên quan được tích hợp và trình bày rõ ràng. Ngoài ra, sinh viên cũng có thể dễ dàng tra cứu thông tin cá nhân và kết quả học tập, từ đó tăng cường sự minh bạch và tương tác giữa nhà trường và người học.

Bên cạnh đó, em mong được nhận những nhận xét từ thầy để phần nào cải thiện khả năng lập trình của mình. Cũng như có thể nâng cấp được bài toán đã trình bày trên từ đó mà tối ưu nhất cách thực thi về việc quản lí sinh viên.

# TÀI LIỆU THAM KHẢO

[1]. “C Tutorial – Learning C Programming”,[Online]. Available: [C Tutorial (w3schools.com)](https://www.w3schools.com/c/index.php). [Accessed 22/05/2025].

[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 22/05/2025].

[3]. “GitHub”, [Online]. Available: [GitHub](https://github.com/) (<https://github.com>). [Accessed 23/05/2025].