**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**HỌC PHẦN: KĨ THUẬT LẬP TRÌNH**

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

**Sinh viên thực hiện:  TRẦN TUẤN ANH**

**NGUYỄN HỮU BẢO KHIÊM**

**NGUYỄN PHÚC SANG**

**LÊ MINH CHÍ**

**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**

**HỌC PHẦN: KĨ THUẬT LẬP TRÌNH**

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

**Sinh viên thực hiện:  TRẦN TUẤN ANH**

**NGUYỄN HỮU BẢO KHIÊM**

**NGUYỄN PHÚC SANG**

**LÊ MINH CHÍ**

**Lớp:    CQ.65. CNTT**

**Khoá:   K65**

**Tp. Hồ Chí Minh, năm 2025**

# **MỤC LỤC**

[**MỤC LỤC** i](#_Toc198497173)

[**LỜI CẢM ƠN** iii](#_Toc198497174)

[**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN** iv](#_Toc198497175)

[**A.LÝ THUYẾT** 1](#_Toc198497176)

[**1. HÀM** 1](#_Toc198497177)

[**1.1 Khái niệm:** 1](#_Toc198497178)

[**1.2 Ví dụ** 2](#_Toc198497179)

[**2. CON TRỎ** 3](#_Toc198497180)

[**2.1 Khái niệm:** 3](#_Toc198497181)

[**2.2 Ví dụ:** 4](#_Toc198497182)

[**3. CON TRỎ MẢNG** 5](#_Toc198497183)

[**3.1 Khái niệm:** 5](#_Toc198497184)

[**3.2 Ví dụ** 5](#_Toc198497185)

[**4.MẢNG CON TRỎ** 6](#_Toc198497186)

[**4.1 Khái niệm:** 6](#_Toc198497187)

[**4.2 Ví dụ:** 7](#_Toc198497188)

[**5. CON TRỎ HÀM** 8](#_Toc198497189)

[**5.1 Khái niệm:** 8](#_Toc198497190)

[**5.2 Ví dụ** 9](#_Toc198497191)

[**6. CẤP PHÁT ĐỘNG** 9](#_Toc198497192)

[**6.1 Khái niệm:** 9](#_Toc198497193)

[**6.2 Lợi ích khi sử dụng cấp phát động:** 10](#_Toc198497194)

[**6.3. So sánh cấp phát động với mảng tĩnh** 11](#_Toc198497195)

[**6.4 Ví dụ:** 12](#_Toc198497196)

[**7. XỬ LÍ TỆP** 13](#_Toc198497197)

[**7.1 Khái niệm:** 13](#_Toc198497198)

[**7.2 Vì sao cần xử lí tệp:** 14](#_Toc198497199)

[**7.3 Ví dụ:** 14](#_Toc198497200)

[**8. KIỂU CẤU TRÚC** 16](#_Toc198497201)

[**8.1 Khái niệm:** 16](#_Toc198497202)

[**8.2 Ví dụ:** 17](#_Toc198497203)

[**9. DANH SÁCH LIÊN KẾT** 18](#_Toc198497204)

[**9.1 Khái niệm** 18](#_Toc198497205)

[**9.2 Lợi ích khi sử dụng danh sách liên kết:** 18](#_Toc198497206)

[**9.3 So sánh danh sách liên kết với mảng:** 19](#_Toc198497207)

[**9.4 Ví dụ:** 20](#_Toc198497208)

[**B. ỨNG DỤNG** 23](#_Toc198497209)

[**1. CÁC TÍNH NĂNG** 23](#_Toc198497210)

[**2. MỤC ĐÍCH CỦA ỨNG DỤNG** 24](#_Toc198497211)

[**TÀI LIỆU THAM KHẢO** 35](#_Toc198497212)

# **LỜI CẢM ƠN**

Chào thầy và các bạn sinh viên thân mến!

Chúng em xin gửi lời cảm ơn chân thành nhất đến tất cả những người đã đóng góp và hỗ trợ cho bài báo cáo này. Sự giúp đỡ, động viên của thầy và các bạn đã đóng vai trò quan trọng trong quá trình chuẩn bị và hoàn thiện bài báo cáo này.

Đầu tiên, chúng em muốn bày tỏ lòng biết ơn sâu sắc đến các giảng viên và người hướng dẫn đã cung cấp cho em những kiến thức và hướng dẫn quý báu. Nhờ sự chỉ dạy của thầy, em đã có cơ hội nắm vững kiến thức và kỹ năng cần thiết để thực hiện và hoàn thành bài báo cáo này.

Không thể không nói đến sự hỗ trợ từ phía gia đình và bạn bè. Những lời khích lệ, sự động viên và tình yêu thương của gia đình và bạn bè đã cho em động lực cho em trong suốt quá trình nghiên cứu và viết báo cáo. Em chân thành cảm ơn những người thân yêu này vì sự ủng hộ và tin tưởng không ngừng nghỉ của họ.

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

Một lần nữa, em xin chân thành cảm ơn tất cả mọi người vì sự đóng góp và hỗ trợ của thầy cô và các bạn. Sự giúp đỡ của mọi người đã góp phần quan trọng vào thành công của bài báo cáo này.

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

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

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

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

# **A.LÝ THUYẾT**

## **1. HÀM**

### **1.1 Khái niệm:**

Hàm trong lập trình là một khối mã được đặt tên, dùng để thực hiện một nhiệm vụ cụ thể. Việc sử dụng hàm giúp chia nhỏ chương trình thành các phần dễ quản lý, tăng tính tái sử dụng, giảm lặp mã và giúp chương trình dễ bảo trì. Có nhiều loại hàm trong các ngôn ngữ lập trình khác nhau, chúng thường được chia thành 2 loại chính: hàm được xây dựng sẵn và hàm do người dùng tự định nghĩa.

* Hàm được xây dựng sẵn: là những hàm đã được ngôn ngữ lập trình cung cấp sẵn trong các thư viện chuẩn, giúp cho lập trình viên thực hiện các công việc phổ biến như xử lí chuỗi, số học cấp phát bộ nhớ.
* Hàm do người dùng tự định nghĩa: là những hàm do lập trình viên tự tạo ra để thực hiện một công việc cụ thể nào đó, nhằm chia nhỏ chương trình, tăng tính tái sử dụng.

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

* Tham số(Parameters): là những giá trị hoặc biến mà hàm sử dụng để nhận dữ liệu đầu vào khi được gọi. Tham số được khai báo trong phần định nghĩa của hàm và có thể có nhiều tham số (hoặc không có tham số).
* Phạm vi biến (Variable Scope): là phạm vi mà trong đó một biến có thể được truy cập (được nhìn thấy và sử dụng) trong chương trình. Điều này có thể liên quan đến biến cục bộ, chỉ có thể truy cập từ bên trong hàm mà nó được khai báo, và biến toàn cục có thể truy cập được bất kì nơi nào trong chương trình.
* Giá trị trả về(Return value): là giá trị mà một hàm trả lại cho nơi gọi nó sau khi thực thi xong. Trong C hàm có thể trả về một giá trị thông qua từ khóa “return”. Giá trị này thường dùng để tiếp tục xử lý hoặc lưu trữ ở nơi gọi hàm.
* Gọi hàm(Calling a Function): là việc sử dụng tên của hàm với các tham số để thực thi hàm đó. Khi một hàm được gọi quá trình thực thi sẽ chuyển tới hàm đó.
* Việc sử dụng hàm sẽ giúp ta rất nhiều trong việc code, làm cho code đơn giản, dễ quản lý và bảo trì, giảm sự lặp lại và tăng khả năng tái sử dụng.

**1.2 Ví dụ**

**-** Đoạn code không sử dụng hàm:

#include <stdio.h>

**int** main() {

**int** a,b;

**int** sum = 0;

printf(**"Nhap hai so a va b: "**);

scanf(**"%d%d"**, &a, &b);

printf(**"Tong hai so la: %d\n"**, sum);

**return** 0;

}

**-** Đoạn code sử dụng hàm:

#include <stdio.h>

**int** tinhTong(**int** x, int y) {

**return** x + y;

}

**int** main() {

**int** a, b;

**int** sum = 0;

printf(**"Nhap so thu nhat: "**);

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

printf(**"Nhap so thu hai: "**);

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

sum = tinhTong(a,b);

printf(**"Tong hai so la: %d\n"**, sum);

**return** 0;

}

* Ở đoạn code đầu tiên, phép tính tổng được thực hiện trực tiếp trong main(). Trong khi đó đoạn code thứ hai sử dụng hàm tinhTong() giúp giúp rõ ràng, dễ tái sử dụng và quản lí hơn. Bằng cách sử dụng hàm giúp chương trình và mã nguồn trở nên sạch sẽ.

## **2. CON TRỎ**

### **2.1 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 đặc biệt lưu trữ địa chỉ ô nhớ của biến khác. Thay vì lưu giá trị trực tiếp, con trỏ lưu vị trí trong bộ nhớ (địa chỉ) nơi giá trị đó được lưu.

Dưới đây là một số khái niệm cơ bản vềc con trỏ:

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

+Con trỏ(Pointer): là một biến đặc biệt chứa địa chỉ của một biến khác.

+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.

+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 con trỏ bạn sử dụng toán tử &.
  + Để truy cập giá trị của biến mà con trỏ trỏ tới, bạn sử dụng toán tử \*.

+Dùng con trỏ dệ 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 động bộ nhớ, truy cập mảng và chuyển đổi dữ liệu.

### **2.2 Ví dụ:**

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

#include <stdio.h>

**int** main() {

**int** a, b;

printf(**"Nhap hai so a va b: "**);

scanf(**"%d%d"**, &a, &b);

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

**int** temp = a;

a = b;

b = temp;

printf(**"Sau khi hoán đổi: a = %d, b = %d\n"**, a, b);

**return** 0;

}

Khi không dùng con trỏ, ta không thể thay đổi giá trị gốc của biến trong hàm, gây bất tiện khi hoán đổi, xử lý mảng hay dữ liệu lớn. Việc truyền bản sao còn làm tốn bộ nhớ và giảm hiệu năng. Con trỏ giúp giải quyết những hạn chế này, tăng hiệu quả và tính linh hoạt.

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

#include <stdio.h>

**void** swap(**int** \*x, **int** \*y) {

**int** temp = \*x;

\*x = \*y;

\*y = temp;

}

**int** main() {

**int** a = 15, b = 11;

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

swap(&a, &b);

printf(**"Sau khi hoán đổi: a = %d, b = %d\n"**, a, b);

**return** 0;

}

## **3. CON TRỎ MẢNG**

### **3.1 Khái niệm:**

Con trỏ mảng là khái niệm liên quan đến việc sử dụng con trỏ để tham chiến đến các phần tử trong mảng lập trình. Trong ngôn ngữ C và C++ một mảng thực chất là một con trỏ. Khi khai báo một mảng là cũng đang tạo nên một con trỏ trỏ tới địa chỉ của phần tử đầu tiên trong mảng.

Một số khái niệm liên quan đến con trỏ mảng:

+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\*.

+Duyệt mảng: là quá trình sử dụng con trỏ duyệt qua tất cả các phần tử của mảng bằng cách di chuyển con trỏ từ phần đầu đến phần tử cuối của mảng.

+Tính toán địa chỉ: là quá trình xác định vị trí ô nhớ của một phần từ trong mảng bằng cách sử dụng địa chỉ và chỉ số của phần tử đó.

+Truyền mảng vào hàm: bạn có thể truyền mảng vào hàm thông qua con trỏ, giúp hàm thao tác trực tiếp trên mảng gốc mà không cần phải sao chép toàn bộ dữ liệu.

### **3.2 Ví dụ**

#include <stdio.h>

**int** main() {

**int** arr[5] = {5, 19, 27, 46, 80};

**int** (\*ptr)[5];

ptr = &arr;

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;

}

Ở ví dụ này:

* Chúng ta khai bảo một mảng arr có 5 phần tử kiểu int và một con trỏ mảng ptr kiểu int (\*ptr)[5], trỏ đến một mảng có 5 phần tử.
* Chúng ta gán địa chỉ của toàn bộ mảng arr cho con trỏ ptr bằng cách sử dụng ptr = &arr
* Dùng con trỏ để truy cập và in các phần tử của mảng bằng cách sử dụng (\*ptr)[i], trong đó (\*ptr) con trỏ để lấy mảng, và [i] để truy cập phần tử thứ i trong mảng.

## **4.MẢNG CON TRỎ**

### **4.1 Khái niệm:**

Mảng con trỏ là một mảng chứa các con trỏ, mỗi phần tử của mảng là một con trỏ trỏ đến giá trị (hoặc một địa chỉ) kiểu dữ liệu cụ thể. Điều này khác với mảng thông thường, nơi mỗi phần tử trong mảng là một giá trị dữ liệu.

Lưu ý: mảng thông thường lưu trữ các giá trị cùng kiểu dữ liệu liên tiếp trong bộ nhớ và có thể truy cập trực tiếp qua chỉ số. Mảng con trỏ chứa các con trỏ mỗi con trỏ trỏ đến một địa chỉ khác nhau, và cần sử dụng \* để truy cập giá trị. Mảng con trỏ thường dùng khi làm việc với địa chỉ bộ nhớ hoặc dữ liệu động.

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.

### **4.2 Ví dụ:**

#include <stdio.h>

**int** main() {

**int** a = 10, b = 40, c = 100;

**int** \*arr[3];

arr[0] = &a;

arr[1] = &b;

arr[2] = &c;

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

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

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

}

**return** 0;

}

**Giải thích:**

* Ta đã khai báo mảnng con trỏ: **int \*arr[3]** là một mảng chứa 3 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, 40 và 100.
* 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 đều trỏ dến một biến khác nhau trong bộ nhớ, cho phép chúng ta lưu trữ địa chỉ của các biến khác nhau trong một mảng.

## **5. CON TRỎ HÀM**

### **5.1 Khái niệm:**

Con trỏ hàm (Function Pointer): là một con trỏ trong C dùng để lưu trữ địa chỉ của một hàm thay vì lưu trữ dữ liệu thông thường như các kiểu dữ liệu khác. Nó cho phép bạn gọi hàm thông qua con trỏ, mang lại sự linh hoạt khi xử lý các hàm động hoặc cần gọi các hàm khác nhau trong các tình huống cụ thể.

Trong ngôn ngữ lập trình 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 \*.

Và lợi ích khi sử dụng hàm con trỏ trong lập trình:

* Tính linh hoạt: con trỏ hàm cho phép chọn hàm cần gọi trong thời gian chạy, giúp chương trình linh hoạt hơn thay vì phải xác định trước hàm cụ thể.
* Tái sử dụng mã: có thể tái sử dụng lại các hàm mà không cần phải viết lại mã, chỉ cần thay đổi hàm được gọi thông qua con trỏ hàm.
* Xử lí sự kiện và callback: trong lập trình sự kiện (như trong các ứng dụng GUI), con trỏ hàm thường được sử dụng để đăng kí các hàm callback, được gọi khi xảy ra một sự kiện cụ thể
* Quản lý tài nguyên: con trỏ hàm có thể được sử dụng để gọi các hàm quản lý tài nguyên (như mở/đóng tệp, cấp phát bộ nhớ), giúp mã dễ dàng quản lý và mở rộng hơn.
* Sử dụng hàm con trỏ mang lại rất nhiều lợi ích như linh hoạt, mở rộng và tái sử dụng mã. Nó cho phép thay đổi hành vi của chương trình mà không làm phức tạp hóa mã nguồn và rất hữu ích trong các ứng dụng lớn hoặc phức tạp.

### **5.2 Ví dụ**

#include <stdio.h>

**int** (\*tichPtr)(**int**, **int**);

**int** tich(**int** x, **int** y) {

**return** x \* y;

}

**int** main() {

tichPtr = tich;

**int** kqua = tichPtr(15, 7);

printf(**"Tich cua 15 va 7 la: %d\n"**, kqua);

**return** 0;

}

**Giải thích:**

* Đầu tiên, chúng ta khai báo một con trỏ hàm có tên là tichPtr. Kiểu của con trỏ hàm này là int (\*)(int, int), nghĩa là một con trỏ trỏ đến hàm trả về kiểu int và nhận hai tham số kiểu int.
* Tiếp theo, chúng ta định nghĩa hàm tich với chức năng tính tích của hai số nguyên và trả về kết quả.
* Trong hàm main, chúng ta gán con trỏ hàm tichPtr cho hàm tich, tức là tichPtr giờ sẽ trỏ đến hàm tich. Khi sử dụng tichPtr, thực chất chúng ta đang gọi hàm tich.
* Sau đó, chúng ta gọi hàm thông qua con trỏ hàm tichPtr với hai tham số là 15 và 7. Kết quả trả về (là tích của hai số này) được lưu vào biến kqua.

## **6. CẤP PHÁT ĐỘNG**

### **6.1 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 hàm quan trọng được sử dụng trong cấp phát động:

* **malloc** (memory allocation): là hàm cấp phát một khối bộ nhớ có kích thước xác định (tính bằng byte). Hàm trả về một con trỏ trỏ đến vùng nhớ vừa được cấp phát. Tuy nhiên, các byte trong vùng nhớ này không được khởi tạo.
* **calloc** (contiguous allocation): là hàm cấp phát bộ nhớ cho một mảng gồm nhiều phần tử, mỗi phần tử có kích thước cố định. Không giống như malloc, calloc sẽ khởi tạo tất cả các byte trong vùng nhớ được cấp phát về giá trị 0.
* **realloc** (reallocation): là hàm thay đổi kích thước của một vùng nhớ đã được cấp phát trước đó. Nếu mở rộng, phần bổ sung có nội dung không xác định. Hàm trả về con trỏ trỏ đến vùng nhớ mới (có thể khác vùng cũ).
* **free**: là hàm Giải phóng vùng nhớ đã được cấp phát động trước đó, giúp tránh rò rỉ bộ nhớ và giải phóng 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 hoặc đồ thị. Nhờ đó, chương trình có thể xử lý dữ liệu có kích thước thay đổi một cách hiệu quả. Tuy nhiên, việc quản lý bộ nhớ động khá phức tạp và nếu không cẩn thận, có thể dẫn đến các lỗi như rò rỉ bộ nhớ hoặc truy cập sai vùng nhớ. Do đó, lập trình viên cần sử dụng cấp phát động một cách có kiểm soát và luôn đảm bảo giải phóng bộ nhớ sau khi sử dụng.

### **6.2 Lợi ích khi sử dụng cấp phát động:**

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

* Tính linh hoạt: cấp phát động cho phép tạo các cấu trúc dữ liệu có kích thước thay đổi trong quá trình thực thi. Điều này giúp chương trình dễ thích ứng với nhiều tình huống và dữ liệu khác nhau.
* Tiết kiệm bộ nhớ: bạn chỉ cấp phát bộ nhớ khi thực sự cần thiết và có thể giải phóng khi không còn dùng, giúp tránh lãng phí tài nguyên hệ thống.
* Hỗ trợ cấu trúc dữ liệu linh hoạt: việc triển khai các cấu trúc dữ liệu như mảng động, danh sách liên kết, cây nhị phân và đồ thị trở nên dễ dàng hơn, đồng thời cho phép thao tác chèn, xóa, cập nhật linh hoạt.
* Quản lý bộ nhớ hiệu quả: cấp phát động giúp kiểm soát việc sử dụng bộ nhớ tốt hơn. Nếu được quản lý đúng cách, nó giúp giảm thiểu rò rỉ bộ nhớ và lỗi truy cập sai vùng nhớ.
* Hỗ trợ các cấu trúc dữ liệu phức tạp: các cấu trúc như bảng băm, hàng đợi ưu tiên, bộ đệm,... đều cần sự linh hoạt của cấp phát động để vận hành hiệu quả trong các ứng dụng lớn và phức tạp.

### **6.3. So sánh cấp phát động với mảng tĩnh**

1. Kích thước:
   * Mảng tĩnh: Kích thước được xác định cố định tại thời điểm biên dịch, không thể thay đổi khi chương trình đang chạy.
   * Cấp phát động: Kích thước được xác định tại thời điểm thực thi và có thể thay đổi linh hoạt (ví dụ bằng realloc).
2. Quản lý bộ nhớ:
   * Mảng tĩnh: Bộ nhớ được cấp phát và quản lý tự động; không cần giải phóng thủ công.
   * Cấp phát động: Lập trình viên phải tự cấp phát (malloc, calloc, realloc) và giải phóng bộ nhớ (free) để tránh rò rỉ bộ nhớ.
3. Tính linh hoạt:
   * Mảng tĩnh: Thiếu linh hoạt vì không thể mở rộng hoặc thu nhỏ kích thước sau khi đã khai báo.
   * Cấp phát động: Cho phép thay đổi kích thước, dễ dàng tạo ra các cấu trúc dữ liệu động như danh sách liên kết, cây, mảng động,...
4. Hiệu suất:
   * Mảng tĩnh: Truy cập nhanh hơn do các phần tử liên tiếp trong bộ nhớ và địa chỉ đã biết trước.
   * Cấp phát động: Truy cập chậm hơn một chút vì phải thông qua con trỏ, và vị trí bộ nhớ có thể phân tán.
5. Điều kiện sử dụng:
   * Mảng tĩnh: Thích hợp khi biết trước kích thước dữ liệu và không cần thay đổi trong quá trình chạy.
   * Cấp phát động: Phù hợp khi kích thước dữ liệu có thể thay đổi hoặc không thể xác định trước.

### **6.4 Ví dụ:**

#include <stdio.h>

#include <stdlib.h>

**int** main() {

**int** n;

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

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

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

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

printf(**"Loi cap phat bo nho!"**);

**return** 1;

}

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

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

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

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

}

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

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

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

}

free(a);

**return** 0;

}

**Giải thích:**

* + Người dùng nhập số lượng phần tử của mảng từ bàn phím.
  + Dùng hàm **malloc** để cấp phát động mảng.
  + Kiểm tra việc cấp phát động có thành công hay không. Nếu **malloc** trả về **NULL** điều này có thể chỉ rằng không đủ bộ nhớ trống.
  + Nhập các phần tử của mảng.
  + In các phần tử ra mảng
  + Sử dụng hàm free để giải phóng bộ nhớ.

## **7. XỬ LÍ TỆP**

### **7.1 Khái niệm:**

Xử lí tệp là quá trình trong lập trình mà bạn làm việc với các tệp và thư mục trên hệ thống tệp của máy tính. Các thao tác này có thể bao gồm mở tệp, đọc hoặc ghi dữ liệu vào tệp, chỉnh sửa nội dung tệp, xóa tệp hoặc tạo tệp mới. Xử lý tệp rất quan trọng trong các ứng dụng yêu cầu lưu trữ và truy xuất dữ liệu.

Các hoạt động cơ bản trong xử lí tệp:

* **Mở tệp** (Open File): là bước đầu tiên khi làm việc với tệp. Bạn cần chỉ định chế độ mở tệp (đọc, ghi, thêm mới, v.v.) để hệ thống biết bạn định làm gì với tệp đó.
* **Đọc từ tệp** (Read File): sau khi tệp được mở, bạn có thể đọc dữ liệu từ tệp vào bộ nhớ. Việc đọc có thể thực hiện theo cách tuần tự hoặc chọn vị trí cụ thể trong tệp để đọc dữ liệu cần thiết.
* **Ghi vào tệp** (Write File): bạn có thể ghi dữ liệu vào tệp đã mở. Điều này có thể là thêm dữ liệu vào cuối tệp hoặc ghi đè lên dữ liệu đã tồn tại, tùy thuộc vào chế độ mở tệp.
* **Đóng tệp** (Close File): sau khi hoàn tất các thao tác với tệp, bạn cần đóng tệp lại. Đóng tệp sẽ giải phóng tài nguyên hệ thống, đảm bảo rằng tệp không còn bị khóa và các thay đổi đã được lưu vào tệp.
* **Xử lý thư mục** (Directory Operations): ngoài việc thao tác với các tệp, bạn cũng có thể làm việc với thư mục trên hệ thống tệp, như tạo mới, di chuyển, xóa thư mục, hoặc liệt kê các tệp trong thư mục đó.

### **7.2 Vì sao cần xử lí tệp:**

* Lưu trữ và truy xuất dữ liệu: xử lý tệp giúp bạn lưu trữ dữ liệu ngoài bộ nhớ RAM, giúp tiết kiệm không gian bộ nhớ khi làm việc với lượng dữ liệu lớn. Tệp có thể lưu trữ dữ liệu dài hạn, ngay cả khi chương trình kết thúc hoặc hệ thống tắt, dữ liệu vẫn được bảo vệ và có thể được truy xuất lại sau.
* Quản lý dữ liệu lâu dài: **k**hi bạn cần lưu trữ dữ liệu từ các lần chạy chương trình, như dữ liệu người dùng, kết quả tính toán, hoặc nhật ký hoạt động, xử lý tệp cho phép lưu trữ dữ liệu lâu dài mà không bị mất khi chương trình kết thúc.
* Đảm bảo tính bền vững và khả năng phục hồi: việc lưu trữ dữ liệu vào tệp giúp đảm bảo rằng các thông tin quan trọng không bị mất khi chương trình hoặc hệ thống gặp sự cố. Ví dụ, các ứng dụng có thể lưu trữ trạng thái của hệ thống hoặc thông tin phiên làm việc vào tệp, giúp phục hồi lại đúng vị trí khi khởi động lại.
* 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.

### **7.3 Ví dụ:**

#include <stdio.h>

**int** main() {

**FILE** \*file;

**char data[] = "Hello World!";**

file = fopen(**"example.txt","w"**);

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

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

**return** 1;

}

fprintf(file, **%s\n**, data);

fprintf(**"Du lieu duoc ghi vao tep"**);

fclose(file);

file = fopen(**"example.txt","r"**);

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

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

**return** 1;

**char** name[200];

printf(**"Du lieu trong tep:\n"**);

**while** (fgets(name, **sizeof**(name), file != NULL)) {

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

}

fclose(file);

**return** 0;

}

Kết quả khi chương trình chạy xong sẽ hiển thị nội dung của tệp “example.txt” lên màn hình là:

**Hello World!**

**Giải thích:**

Mở tệp để ghi:

* fopen**("example.txt", "w"):** mở tệp với chế độ ghi. Nếu tệp chưa tồn tại, nó sẽ được tạo ra.
* Nếu không mở được tệp, chương trình sẽ in thông báo lỗi và kết thúc.

Ghi vào tệp:

* **fprintf(file, "%s\n", data);** : ghi chuỗi data vào tệp.

Đóng tệp sau khi ghi:

* **fclose(file);** : đóng tệp sau khi hoàn tất ghi dữ liệu.

Mở tệp để đọc:

* **fopen("example.txt", "r");** : mở tệp ở chế độ đọc.
* Nếu không mở được tệp, chương trình sẽ in thông báo lỗi và kết thúc.

Đọc tệp và hiển thị dữ liệu:

* **fgets(buffer, sizeof(buffer), file);** : đọc dữ liệu từ tệp vào mảng buffer.
* Sau đó, chương trình in nội dung của tệp ra màn hình.

Đóng tệp sau khi đọc:

* **fclose(file);** : đóng tệp sau khi hoàn tất đọc dữ liệu

## **8. KIỂU CẤU TRÚC**

### **8.1 Khái niệm:**

Kiểu cấu trúc (struct) là một kiểu dữ liệu do người dùng tự định nghĩa trong ngôn ngữ C, cho phép gom nhóm nhiều biến (gọi là thành phần hoặc trường) có thể thuộc nhiều kiểu dữ liệu khác nhau vào một đơn vị chung, được gọi là biến cấu trúc.

Cách khai báo kiểu cấu trúc:

Cú pháp

struct TênCấuTrúc {

Kiểu1 thànhPhần1;

Kiểu2 thànhPhần2;

...

KiểuN thànhPhầnN;

};

### **8.2 Ví dụ:**

#include <stdio.h>

**struct** sinhVien {

**char** ten[50];

**int** tuoi;

**float** diem;

**int** main() {

**struct** sinhVien sv;

printf(**"Nhap ten sinh vien: "**);

fgets(sv.ten, **sizeof**(sv.ten), **stdin**);

printf(**"Nhap tuoi: "**);

scanf(**"%d",** &sv.tuoi);

printf(**"Nhap diem: "**);

scanf(**"%f",** &sv.diem);

printf(**"\n---Thong tin sinh vien---\n"**);

printf(**"Ten: %s",** sv.ten);

printf(**"Tuoi: %d\n",** sv.tuoi);

printf(**"Diem: %.2f\n",** sv.diem);

**return** 0;

}

**Giải thích:**

* Đầu tiên, ta khai báo kiểu **struct SinhVien** với các trường dữ liệu: ten, tuoi, và diem.
* Sau đó, ta khai báo một biến thuộc kiểu struct (ở đây là sv), nơi sẽ lưu trữ thông tin của một sinh viên.
* Chương trình yêu cầu người dùng nhập thông tin cho các trường này (tên, tuổi, điểm).
* Sau khi nhập xong, chương trình sẽ in ra thông tin sinh viên đã nhập.

## **9. DANH SÁCH LIÊN KẾT**

### **9.1 Khái niệm**

Danh sách liên kết là một cấu trúc dữ liệu trong lập trình, được sử dụng để lưu trữ và quản lý một tập hợp các phần tử (hoặc nút). Mỗi nút trong danh sách chứa dữ liệu và một con trỏ trỏ đến nút tiếp theo. Điều này cho phép danh sách liên kết có khả năng thay đổi kích thước linh hoạt, vì các nút có thể được tạo ra và xóa bỏ mà không cần thay đổi kích thước của cấu trúc dữ liệu.

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

* Dữ liệu (Data): 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.
* Con trỏ tiếp theo: là một con trỏ trong mỗi nút, trỏ đến nút tiếp theo trong danh sách liên kết. Điều này tạo thành một chuỗi các nút, mỗi nút chỉ biết đến nút tiếp theo của nó, hình thành một danh sách liên kết.

Có hai loại danh sách liên kết chính:

* Danh sách liên kết đơn: Mỗi nút trong danh sách chỉ trỏ đến nút tiếp theo, tạo thành một chuỗi các nút mà mỗi nút chỉ biết đến nút kế tiếp.
* Danh sách liên kết đôi: Mỗi nút không chỉ chứa một con trỏ đến nút tiếp theo mà còn chứa một con trỏ đến nút trước đó, giúp duyệt danh sách linh hoạt theo cả hai chiều (từ trước ra sau và ngược lại).

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

### **9.2 Lợi ích khi sử dụng danh sách liên kết:**

* Dễ dàng thay đổi kích thước: danh sách liên kết có thể dễ dàng thay đổi kích thước mà không cần phải di chuyển dữ liệu hoặc cấp phát lại bộ nhớ, như với mảng. Bạn có thể thêm hoặc xóa các phần tử bất kỳ mà không ảnh hưởng đến các phần tử khác trong danh sách.
* Khả năng thêm xóa linh hoạt: các thao tác thêm và xóa phần tử (ở đầu, giữa hoặc cuối danh sách) rất nhanh và hiệu quả, chỉ cần thay đổi con trỏ của các nút liên quan mà không cần phải thay đổi các phần tử khác. Điều này giúp tiết kiệm thời gian khi xử lý các thao tác này.
* Quản lý bộ nhớ linh hoạt: vì danh sách liên kết là cấu trúc dữ liệu động, bạn không cần phải biết trước kích thước của nó. Điều này rất hữu ích trong các ứng dụng mà bạn không thể xác định số lượng phần tử từ đầu, như trong các chương trình cần xử lý một lượng dữ liệu thay đổi liên tục.
* Cấu trúc dữ liệu linh hoạt: danh sách liên kết có thể sử dụng để xây dựng các cấu trúc dữ liệu phức tạp hơn như ngăn xếp (stack), hàng đợi (queue), và thậm chí là các đồ thị hoặc cây, giúp linh hoạt trong việc xử lý các bài toán phức tạp.
* 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.

### **9.3 So sánh danh sách liên kết với mảng:**

Kích thước

* Danh sách liên kết: Kích thước động, có thể thay đổi trong suốt quá trình thực thi.
* Mảng: Kích thước cố định, không thể thay đổi sau khi khai báo.

Thêm và xóa phần tử

* Danh sách liên kết: Thêm và xóa phần tử nhanh chóng, không cần di chuyển phần tử khác.
* Mảng: Thêm và xóa phần tử phức tạp, cần di chuyển các phần tử còn lại.

Truy cập phần tử

* Danh sách liên kết: Truy cập phần tử chậm hơn, phải duyệt qua các nút (O(n)).
* Mảng: Truy cập phần tử nhanh chóng, O(1), nhờ bộ nhớ liên tiếp.

Linh hoạt về kích thước

* Danh sách liên kết: Linh hoạt, có thể thêm hoặc bớt phần tử bất kỳ lúc nào.
* Mảng: Không linh hoạt, kích thước đã được xác định và không thể thay đổi.

Quản lý bộ nhớ

* Danh sách liên kết: Tiết kiệm bộ nhớ vì bộ nhớ được cấp phát động cho từng phần tử.
* Mảng: Bộ nhớ phải cấp phát một lần cho toàn bộ mảng, có thể gây lãng phí nếu số phần tử không đủ lớn.

### **9.4 Ví dụ:**

#include <stdio.h>

#include <stdlib.h>

**struct** Node {

**int** value;

**struct** Node\* next;

};

**struct** linkedList {

**struct** Node\* head;

**struct** Node\* tail;

};

**void** intiList(**struct** linkedList \*list) {

list->head = NULL;

list->tail = NULL;

**struct** Node\* cretaNode(**int** val) {

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

**if** (node == NULL) {

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

**return** NULL;

}

node->value = val;

node->next = NULL;

**return** node;

}

**void** addtail(**struct** linkedlList \*list, **struct** Node\* newNode {

**if** (list->head == NULL)

list->head = list->tail = newNode;

**else** {

list->tail->next = newNode;

list->tail = newNode;

}

}

**void** printList(**struct** linkedlList \*list) {

**struct** Node \*current = list.head;

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

printf(**"%d"**,current->value);

current = current->next;

}

}

**void** freeList(**struct** linkedlList \*list) {

**struct** Node \*current = list->head;

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

**struct** Node \*temp = curent;

current = current-next;

**free**(temp);

}

List->head = list->tail = NULL

}

**int** main() {

**struct** linkedList myList;

initList(&myList);

**int** n;

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

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

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

**int** x;

printf(**"Nhap gia tri: "**);

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

**struct** Node \*node = createNode(x);

**if** (node != NULL) {

addTail(&myList, node);

}

}

printf(**"Danh sach vua nhap: "**);

printList(myList);

freeList(&myList);

**return** 0;

}

**Đoạn code trên thực hiện các chức năng sau:**

1. Định nghĩa cấu trúc Node – Đại diện cho một phần tử trong danh sách liên kết, gồm hai thành phần:
   * data: chứa giá trị dữ liệu của nút.
   * next: là một con trỏ trỏ đến nút tiếp theo trong danh sách.
2. Định nghĩa cấu trúc linkedList – Đại diện cho toàn bộ danh sách liên kết, gồm:
   * head: trỏ đến nút đầu tiên trong danh sách.
   * tail: trỏ đến nút cuối cùng trong danh sách.
3. Hàm initList() – Khởi tạo một danh sách rỗng bằng cách gán cả head và tail là NULL.
4. Hàm createNode(int value) – Cấp phát động một nút mới có giá trị value. Nếu cấp phát thành công, trả về con trỏ đến nút mới; nếu không, thông báo lỗi bộ nhớ.
5. Hàm addtaill() – Thêm một nút mới vào cuối danh sách:
   * Nếu danh sách đang rỗng, nút đó trở thành cả head và tail.
   * Nếu danh sách đã có phần tử, nút mới được nối vào sau tail và cập nhật lại tail.
6. Hàm printList() – Duyệt qua toàn bộ danh sách từ head đến NULL và in ra dữ liệu của từng nút.
7. Trong main():
   * Khởi tạo danh sách rỗng bằng initList().
   * Nhập số lượng phần tử từ người dùng, sau đó nhập từng giá trị.
   * Với mỗi giá trị nhập vào:

+Tạo nút mới bằng createNode().

+Nếu tạo thành công, thêm nút vào cuối danh sách bằng addtail ().

* + Sau khi nhập xong, hiển thị toàn bộ danh sách ra màn hình bằng printList().

# **B. ỨNG DỤNG**

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

## **1. CÁC TÍNH NĂNG**

- Xuất bảng các sinh viên theo MSSV ra màn hình

- Xuất bảng các sinh viên theo điểm tổng kết giảm dần ra màn hình (xếp hạng sinh viên)

- Xuất bảng các sinh viên theo lớp ra màn hình

- Tìm kiếm sinh viên theo MSSV

- Xuất danh sách sinh viên đạt và học lại

- Thêm sinh viên mới

- Xóa sinh viên

- Hiển thị lại danh sách sinh viên sau khi thay đổi

## **2. MỤC ĐÍCH CỦA ỨNG DỤNG**

Ứng dụng quản lý sinh viên được tạo ra nhằm giúp lưu trữ, theo dõi và xử lý thông tin của sinh viên một cách thuận tiện và nhanh chóng. Thay vì ghi chép bằng tay hoặc quản lý thủ công, ứng dụng này cho phép thực hiện các công việc như:

* Nhập và lưu thông tin sinh viên.
* Xem danh sách sinh viên một cách rõ ràng.
* Tìm kiếm, sắp xếp hoặc cập nhật thông tin khi cần thiết.

Mục đích chính là hỗ trợ việc quản lý thông tin sinh viên hiệu quả hơn, giảm thiểu sai sót khi xử lý dữ liệu, tiết kiệm thời gian cho người quản lý và đảm bảo thông tin được tổ chức một cách khoa học, dễ tra cứu khi cần thiết

**3. SOURCE CODE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

// Dinh nghia cau truc sinh vien dung DSLK don

// Ap dung: Cau truc, DSLK don, Con tro

**typedef** **struct** sinhvien{

**long** **long** MSSV;

**char** HoTen[50];

**float** DiemTK;

**char** Lop[50];

**struct** sinhvien \*next;

} SV;

// Ham themSinhVien: Them sinh vien moi vao dau danh sach

// Ap dung: Cau truc, Con tro, DSLK don, Ham

**void** themSinhVien(SV\*\* head) {

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

**if** (sv == NULL) {

printf(**"Khong du bo nho de them sinh vien.\n"**);

**return**;

}

printf(**"Nhap MSSV: "**);

scanf("%lld", &sv->MSSV); getchar();

printf(**"Nhap ho ten: "**);

fgets(sv->HoTen, sizeof(sv->HoTen), stdin);

printf(**"Nhap diem tong ket: "**);

scanf("%f", &sv->DiemTK); getchar();

printf(**"Nhap lop: "**);

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

sv->next = \*head;

\*head = sv;

}

// Ham xuatDSSV: Xuat thong tin sinh vien ra man hinh va file (neu co ten file)

// Ap dung: Ham, Ghi file, Con tro

**void** xuatDSSV(SV\* head, **const** **char**\* tenFile) {

**FILE** \*f = NULL;

**if** (tenFile != NULL) f = fopen(tenFile, "w");

**int** stt = 1;

printf(**"\n%-5s\t%-15s\t%-30s\t%-10s\t%-10s\n", "STT", "MSSV", "Ho ten", "Diem", "Lop"**);

**if** (f) fprintf(f, **"%-5s\t%-15s\t%-30s\t%-10s\t%-10s\n", "STT", "MSSV", "Ho ten", "Diem", "Lop"**);

**while** (head) {

printf(**"%-5d\t%-15lld\t%-30s\t%-10.2f\t%-10s\n"**, stt, head->MSSV, head->HoTen, head->DiemTK, head->Lop);

**if** (f) fprintf(f, **"%-5d\t%-15lld\t%-30s\t%-10.2f\t%-10s\n"**, stt, head->MSSV, head->HoTen, head->DiemTK, head->Lop);

head = head->next;

stt++;

}

**if** (f) fclose(f);

**if** (tenFile != NULL) printf(**"Danh sach da duoc ghi vao file %s\n"**, tenFile);

}

// Ham sxDSSVTheoDiem: Sap xep giam dan theo diem TK bang cach doi du lieu

// Ap dung: Sap xep, DSLK don, Con tro

**void** sxDSSVTheoDiem(SV\*\* head) {

**for** (SV\* i = \*head; i != NULL; i = i->next) {

**for** (SV\* j = i->next; j != NULL; j = j->next) {

**if** (i->DiemTK < j->DiemTK) {

**long** **long** tempMSSV = i->MSSV;

i->MSSV = j->MSSV;

j->MSSV = tempMSSV;

**float** tempDiem = i->DiemTK;

i->DiemTK = j->DiemTK;

j->DiemTK = tempDiem;

**char** tempHoTen[50], tempLop[50];

strcpy(tempHoTen, i->HoTen);

strcpy(i->HoTen, j->HoTen);

strcpy(j->HoTen, tempHoTen);

strcpy(tempLop, i->Lop);

strcpy(i->Lop, j->Lop);

strcpy(j->Lop, tempLop);

}

}

}

}

// Ham sxDSSVTheoMSSV: Sap xep tang dan theo MSSV bang cach doi du lieu

// Ap dung: Sap xep, DSLK don, Con tro

**void** sxDSSVTheoMSSV(SV\*\* head) {

**for** (SV\* i = \*head; i != NULL; i = i->next) {

**for** (SV\* j = i->next; j != NULL; j = j->next) {

**if** (i->MSSV > j->MSSV) {

**long** **long** tempMSSV = i->MSSV;

i->MSSV = j->MSSV;

j->MSSV = tempMSSV;

**float** tempDiem = i->DiemTK;

i->DiemTK = j->DiemTK;

j->DiemTK = tempDiem;

**char** tempHoTen[50], tempLop[50];

strcpy(tempHoTen, i->HoTen);

strcpy(i->HoTen, j->HoTen);

strcpy(j->HoTen, tempHoTen);

strcpy(tempLop, i->Lop);

strcpy(i->Lop, j->Lop);

strcpy(j->Lop, tempLop);

}

}

}

}

// Ham daXuatLop: Kiem tra 1 lop da duoc xuat chua

// Ap dung: Mang, Ham

**int** daXuatLop(**char** daXuat[][50], **int** dem, **char**\* lop) {

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

**if** (strcmp(daXuat[i], lop) == 0) **return** 1;

}

**return** 0;

}

// Ham xuatTheoLop: Xuat danh sach theo tung lop, tranh trung lap lop

// Ap dung: Ham, Mang, Ghi file, Con tro

**void** xuatTheoLop(SV \*head, **const** **char** \*tenFile) {

**char** daXuat[100][50];

**int** dem = 0;

**FILE** \*f = NULL;

**if** (tenFile != NULL) f = fopen(tenFile, "w");

**for** (SV \*i = head; i != NULL; i = i->next) {

**if** (!daXuatLop(daXuat, dem, i->Lop)) {

printf(**"\nDanh sach sinh vien lop %s: \n"**, i->Lop);

printf(**"%-5s %-15s %-30s %-10s %-10s\n"**, **"STT"**, **"MSSV"**, **"Ho ten"**, **"Diem"**, **"Lop"**);

**if** (f) fprintf(f, "\n===== Lop %s =====\n", i->Lop);

**if** (f) fprintf(f, **"%-5s %-15s %-30s %-10s %-10s\n"**, "STT", "MSSV", **"Ho ten"**, **"Diem"**, **"Lop"**);

**int** stt = 1;

**for** (SV \*j = head; j != NULL; j = j->next) {

**if** (strcmp(i->Lop, j->Lop) == 0) {

printf(**"%-5d %-15lld %-30s %-10.2f %-10s\n"**, stt, j->MSSV, j->HoTen, j->DiemTK, j->Lop);

**if** (f) fprintf(f, **"%-5d %-15lld %-30s %-10.2f %-10s\n"**, stt, j->MSSV, j->HoTen, j->DiemTK, j->Lop);

stt++;

}

}

strcpy(daXuat[dem++], i->Lop);

}

}

**if** (f) {

fclose(f);

printf(**"Danh sach theo lop da duoc ghi vao file %s\n"**, tenFile);

}

}

// Ham File\_Dau: Ghi danh sach SV dat (DiemTK >= 5) vao file

// Ap dung: Ghi file, Ham

**void** File\_Dau(SV\* head, **const** **char** \*filename) {

sxDSSVTheoDiem(&head);

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

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

printf(**"Khong the mo tep %s\n"**, filename);

**return**;

}

**int** count = 0, stt = 1;

**while** (head) {

**if** (head->DiemTK >= 5.0) {

printf(**"%-5d %-15lld %-30s %-10.2f\n"**, stt, head->MSSV, head->HoTen, head->DiemTK);

fprintf(file, **"%-5d %-15lld %-30s %-10.2f\n"**, stt, head->MSSV, head->HoTen, head->DiemTK);

count++; stt++;

}

head = head->next;

}

**if** (count == 0) {

fprintf(file, **"Khong co sinh vien nao dau.\n"**);

}

fclose(file);

printf(**"Co %d sinh vien dau. Da in danh sach vao tep %s\n"**, count, filename);

}

// Ham File\_HocLai: Ghi danh sach SV rot (DiemTK < 5) vao file

// Ap dung: Ghi file, Ham

**void** File\_HocLai(SV\* head, **const** **char** \*filename) {

sxDSSVTheoDiem(&head);

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

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

printf(**"Khong the mo tep %s\n"**, filename);

**return**;

}

**int** count = 0, stt = 1;

**while** (head) {

**if** (head->DiemTK < 5.0) {

printf(**"%-5d %-15lld %-30s %-10.2f\n"**, stt, head->MSSV, head->HoTen, head->DiemTK);

fprintf(file, **"%-5d %-15lld %-30s %-10.2f\n"**, stt, head->MSSV, head->HoTen, head->DiemTK);

count++; stt++;

}

head = head->next;

}

**if** (count == 0) {

fprintf(file, **"Khong co sinh vien nao hoc lai.\n"**);

}

fclose(file);

printf(**"Co %d sinh vien hoc lai. Da in danh sach vao tep %s\n"**, count, filename);

}

// Ham timSV: Tim sinh vien theo MSSV

// Ap dung: Ham, DSLK don

**void** timSV(SV\*head){

**long** **long** n;

printf(**"Vui long nhap vao ma so sinh vien ban can tim : "**);

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

**while**(head){

**if**(head->MSSV==n){

printf(**"Sinh vien ban can tim la :\n"**);

printf(**"MSSV : %lld\ nHo ten : %s\n Diem TK : %.2f\n Lop : %s\n"**,head->MSSV,head->HoTen,head->DiemTK,head->Lop);

**return**;

}

head=head->next;

}

printf(**"Khong tim thay\n"**);

}

// Ham xoaSinhVien: Xoa sinh vien theo MSSV

// Ap dung: 4. Ham, 3. DSLK don

**void** xoaSinhVien(SV\*\* head) {

**long** **long** MSSV;

printf(**"Nhap MSSV sinh vien can xoa: "**);

scanf(**"%lld"**, &MSSV);

SV \*temp = \*head, \*prev = NULL;

**if** (temp != NULL && temp->MSSV == MSSV) {

\*head = temp->next;

free(temp);

printf(**"Sinh vien da duoc xoa.\n"**);

**return**;

}

**while** (temp != NULL && temp->MSSV != MSSV) {

prev = temp;

temp = temp->next;

}

**if** (temp == NULL) {

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

**return**;

}

prev->next = temp->next;

free(temp);

printf(**"Sinh vien da duoc xoa\n"**);

}

// Ham giaiPhongBoNho: Giai phong toan bo danh sach lien ket

// Ap dung: Ham, DSLK don

**void** giaiPhongBoNho(SV \*head){

**while**(head){

SV\* the = head;

head = head->next;

free(the);

}

}

// Ham xuatDanhSach: xuat danh sach sinh vien ra man hinh

**void** xuatDanhSach(SV \*head) {

**if** (head == NULL) {

printf(**"Danh sach rong\n"**);

**return**;

}

printf(**"\n%-5s %-15s %-25s %-10s %-10s\n"**, **"STT"**, **"MSSV"**, **"Ho ten"**, **"Diem"**, **"Lop"**);

**int** stt = 1;

**while** (head) {

printf(**"%-5d %-15lld %-49s %-10.2f %1s\n"**, stt, head->MSSV, head->HoTen, head->DiemTK, head->Lop);

head = head->next;

stt++;

}

}

// Ly thuyet duoc ap dung trong ham main: bien, con tro, ham, dieu kien, vong lap, cau truc, xau ky tu, toan tu, tap tin

**int** main() {

**bool** t = true;

**int** x;

SV \*head = NULL;

printf(**"Nhap so luong sinh vien: "**);

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

getchar();

**while** (x > 100 || x <= 0) {

printf(**"So luong khong hop le. Moi nhap lai: "**);

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

}

**if**(x==1) {

printf(**"Nhap thong tin cua sinh vien: \n"**);

themSinhVien(&head);

} **else** {

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

printf(**"Nhap thong tin cua sinh vien thu %d: \n"**,i+1);

themSinhVien(&head);

}

}

xuatDanhSach(head);

**int** luaChon;

luaChon = 9;

**char** tenFile[100];

**while** (t) {

**if** (luaChon == 9) {

printf(**"\n1. Xuat theo MSSV tang dan\n"**);

printf(**"2. Xuat theo diem TK giam dan\n"**);

printf(**"3. Xuat theo lop\n"**);

printf(**"4. Tim sinh vien theo MSSV\n"**);

printf(**"5. Xuat danh sach sinh vien dat va hoc lai\n"**);

printf(**"6. Them sinh vien\n"**);

printf(**"7. Xoa sinh vien\n"**);

printf(**"8. Hien lai danh sach sinh vien\n"**);

printf(**"9. Hien lai bang chuc nang\n"**);

printf(**"10. Thoat\n"**);

}

printf(**"Chon: "**);

scanf(**"%d"**, &luaChon); getchar();

**switch** (luaChon) {

**case 1:**

sxDSSVTheoMSSV(&head);

xuatDSSV(head, **"DSSV\_MSSV.txt"**);

**break**;

**case 2:**

sxDSSVTheoDiem(&head);

xuatDSSV(head, **"DSSV\_Diem.txt"**);

**break**;

**case 3:**

sxDSSVTheoMSSV(&head);

xuatTheoLop(head, **"DSSV\_Lop.txt"**);

**break**;

**case 4:**

timSV(head);

**break**;

**case 5:**

File\_Dau(head, **"Dat.txt"**);

File\_HocLai(head, **"Khongdat.txt"**);

**break**;

**case 6:**

themSinhVien(&head);

**break**;

**case 7:**

xoaSinhVien(&head);

**break**;

**case 8:**

xuatDanhSach(head);

**case 9:**

**break**;

**case 10:**

t = false;

**break**;

**default:**

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

}

}

giaiPhongBoNho(head);

**return** 0;

}

# **TÀI LIỆU THAM KHẢO**

[1]. <https://blog.28tech.com.vn/c-ham> - Hàm, con trỏ, mảng - truy cập (1/5/2025)

[2]. <https://www.w3schools.com/c/c_structs.php> - Struct - truy cập (10/5/2025)