**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: ThS. TRẦN PHONG NHÃ**

**Sinh viên thực hiện:  LÊ THỊ THU HUYỀN**

**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: ThS. TRẦN PHONG NHÃ**

**Sinh viên thực hiện:  LÊ THỊ THU HUYỀN**

**Lớp:    CQ.65. CNTT**

**Khoá:   K65**

**Tp. Hồ Chí Minh, năm 2025**

# MỤC LỤC

[MỤC LỤC i](#_Toc199001188)

[LỜI CẢM ƠN ii](#_Toc199001189)

[NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN iii](#_Toc199001190)

[A. LÝ THUYẾT 3](#_Toc199001191)

[**1.HÀM:** 3](#_Toc199001192)

[**2.CON TRỎ:** 5](#_Toc199001193)

[**3.CON TRỎ MẢNG:** 9](#_Toc199001194)

[**4.MẢNG CON TRỎ:** 11](#_Toc199001195)

[**5.CON TRỎ HÀM:** 12](#_Toc199001196)

[**6.CẤP PHÁT ĐỘNG:** 18](#_Toc199001197)

[**7.XỬ LÍ TỆP:** 21](#_Toc199001198)

[**8.KIỂU CẤU TRÚC:** 25](#_Toc199001199)

[**9.DANH SÁCH LIÊN KẾT:** 29](#_Toc199001200)

[B. ỨNG DỤNG 31](#_Toc199001201)

[KẾT LUẬN 43](#_Toc199001202)

[TÀI LIỆU THAM KHẢO 44](#_Toc199001203)

# **LỜI CẢM ƠN**

Chào cô và các bạn sinh viên thân mến!

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

Đầu tiên, 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 tận tâm của cô, em đã có cơ hội nắm vững kiến thức và kỹ năng cần thiết để thực hiện bài báo cáo này.

Không thể không nhắc 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 truyền sức mạnh và 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, 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 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 quý vị 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!

Trân trọng,

*Lê Thị Thu Huyền*

# 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

## **1.HÀM:**

**1.1: Khái niệm:**

-Một hàm trong C là một tập hợp các câu lệnh khi được gọi sẽ thực hiện một số tác vụ cụ thể. Đây là khối xây dựng cơ bản của một chương trình C cung cấp tính mô-đun và khả năng tái sử dụng mã. Các câu lệnh lập trình của một hàm được đặt trong dấu ngoặc nhọn { } , có một số ý nghĩa nhất định và thực hiện một số thao tác nhất định. Chúng cũng được gọi là chương trình con hoặc thủ tục trong các ngôn ngữ khác.

-Cú pháp của hàm trong C có thể được chia thành 3 khía cạnh:

* Khai báo hàm
* Định nghĩa hàm
* Gọi hàm
  + 1. **Khai báo hàm**

-Trong khai báo hàm, chúng ta phải cung cấp tên hàm, kiểu trả về và số lượng và kiểu tham số của hàm. Khai báo hàm cho trình biên dịch biết rằng có một hàm với tên đã cho được định nghĩa ở đâu đó trong chương trình.

-Cú pháp:

return\_type tên\_hàm ( tham số\_1, tham số\_2 );

-Tên tham số không bắt buộc khi khai báo hàm. Chúng ta cũng có thể khai báo hàm mà không cần sử dụng tên của biến dữ liệu.

* + 1. **Định nghĩa hàm**

-Định nghĩa hàm bao gồm các câu lệnh thực tế được thực thi khi hàm được gọi (tức là khi điều khiển chương trình đến hàm).

* + 1. **Gọi hàm**

-Lệnh gọi hàm là một câu lệnh hướng dẫn trình biên dịch thực thi hàm. Chúng ta sử dụng tên hàm và các tham số trong lệnh gọi hàm.

-Gọi hàm là cần thiết để đưa điều khiển chương trình đến định nghĩa hàm. Nếu không gọi, các câu lệnh hàm sẽ không được thực thi.

-Ví dụ:

#include <stdio.h>

int tong(int a, int b) {

return a + b; }

int main(){

int cong= tong(7, 20);

printf("Tong la: %d", cong);

return 0;

}

* + 1. **Kiểu trả về của hàm**

-Kiểu trả về của hàm cho biết loại giá trị nào được trả về sau khi tất cả các hàm được thực thi. Khi chúng ta không muốn trả về giá trị, chúng ta có thể sử dụng kiểu dữ liệu

**Lưu ý:** Chỉ có thể trả về một giá trị từ hàm C. Để trả về nhiều giá trị, chúng ta phải sử dụng con trỏ hoặc cấu trúc.

* + 1. **Tham số**

- 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.

* 1. **Phân loại:**

-Có hai loại hàm trong C:

* Các hàm thư viện
* Hàm do người dùng xác định
  + 1. **Các hàm thư viện**

-Một [hàm thư viện](https://www.geeksforgeeks.org/c-library-functions/) cũng được gọi là "hàm tích hợp" . Một gói trình biên dịch đã tồn tại chứa các hàm này, mỗi hàm có một ý nghĩa cụ thể và được bao gồm trong gói. Các hàm tích hợp có lợi thế là có thể sử dụng trực tiếp mà không cần định nghĩa

Ví dụ: pow(), sqrt(), strcmp(), ...

#include <stdio.h>

#include <math.h>

int main(){

mu;

so=82;

mu = sqrt(so);

printf("can bac 2 cua so la %.2lf = %.2lf", so, mu);

return 0;

}

* + 1. **Hàm do người dùng xác định**

Các hàm mà lập trình viên tạo ra được gọi là hàm do người dùng định nghĩa hoặc “hàm được thiết kế riêng” . Các hàm do người dùng định nghĩa có thể được cải thiện và sửa đổi theo nhu cầu của lập trình viên. Bất cứ khi nào chúng ta viết một hàm cụ thể theo từng trường hợp và không được định nghĩa trong bất kỳ tệp tiêu đề nào, chúng ta cần khai báo và định nghĩa các hàm của riêng mình theo cú pháp.

## **2.CON TRỎ:**

* 1. **Khái niệm:**

Con **trỏ** là bộ nhớ **địa chỉ lưu** trữ biến của một biến khác. Thay vì giữ giá trị trực tiếp, con trỏ có địa chỉ nơi giá trị được lưu trữ trong bộ nhớ. Điều này cho phép chúng tôi vận hành dữ liệu được lưu trữ tại một bộ nhớ cụ thể mà không thực sự sử dụng biến thể của nó. Đây là xương sống của cấp độ bộ nhớ cấp thấp trong C.

* 1. **Khai báo một con trỏ**

Con trỏ được khai báo bằng cách chỉ định tên và kiểu của nó, giống như khai báo biến đơn giản nhưng có thêm dấu **hoa thị ( )** trước con trỏ tên.**\***

-Cách khai báo con trỏ: data\_type\* name

-Ở đây,  data\_type  định nghĩa kiểu dữ liệu mà con trỏ đến. Một con trỏ số nguyên có thể chỉ hiển thị một số nguyên

Ví dụ: int \*ptr;

-Trong lệnh trên, con trỏ  ptr  có thể lưu trữ địa chỉ của một số nguyên. Nó được phát âm là con trỏ tới số nguyên.

* 1. **Khởi tạo con trỏ**

-Khởi tạo con trỏ có nghĩa là chỉ định một số địa chỉ cho con trỏ biến. Trong C, [địa chỉ toán tử (&)](https://www-geeksforgeeks-org.translate.goog/address-operator-in-c/?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=sc" \t "_blank) được sử dụng để lấy địa chỉ bộ nhớ của bất kỳ biến nào. Sau đó, bộ nhớ địa chỉ này được lưu trữ trong một con trỏ biến.

Ví dụ:

int var = 10;

int \*ptr = &var;

-Trong lệnh trên, con trỏ  ptr  lưu trữ địa chỉ của biến  x  được xác định bằng địa chỉ toán tử  (&).

* 1. **Hủy tham chiếu một con trỏ**

Truy cập trực tiếp vào con trỏ sẽ chỉ cung cấp cho chúng ta địa chỉ được lưu trữ trong con trỏ.

-Ví dụ:

*#include <stdio.h>*

*int main() {*

*int var = 10;*

*int\* ptr = &var;*

*printf("%d", ptr);*

*return 0;*

*}*

* 1. **Các loại con trỏ đặc biệt**
     1. **Con trỏ NULL**

Con [trỏ NULL](https://www.geeksforgeeks.org/few-bytes-on-null-pointer-in-c/) là những con trỏ không trỏ đến bất kỳ vị trí bộ nhớ nào. Chúng có thể được tạo bằng cách gán giá trị NULL cho con trỏ. Một con trỏ của bất kỳ kiểu nào cũng có thể được gán giá trị NULL.

Con trỏ NULL thường được sử dụng để biểu diễn sự vắng mặt của bất kỳ địa chỉ nào. Điều này cho phép chúng ta kiểm tra xem con trỏ có trỏ đến bất kỳ vị trí bộ nhớ hợp lệ nào không bằng cách kiểm tra xem nó có bằng NULL không.

Ví dụ: int\* ptr= NULL;

* + 1. **Con trỏ Void**

-Con [trỏ void](https://www.geeksforgeeks.org/void-pointer-c-cpp/) trong C là con trỏ có kiểu void . Điều đó có nghĩa là chúng không có bất kỳ kiểu dữ liệu liên quan nào. Chúng cũng được gọi là con trỏ chung vì chúng có thể trỏ đến bất kỳ kiểu nào và có thể được ép kiểu thành bất kỳ kiểu nào.

* + 1. **Con trỏ hoang dã**

Con [trỏ hoang dã](https://www.geeksforgeeks.org/what-are-wild-pointers-how-can-we-avoid/) là con trỏ chưa được khởi tạo bằng thứ gì đó. Các loại con trỏ C này có thể gây ra sự cố trong chương trình của chúng ta và cuối cùng có thể khiến chúng bị sập. Nếu các giá trị được cập nhật bằng con trỏ hoang dã, chúng có thể gây ra tình trạng hủy dữ liệu hoặc hỏng dữ liệu.

* + 1. **Con trỏ lơ lửng**

Con trỏ trỏ đến vị trí bộ nhớ đã bị xóa (hoặc giải phóng) được gọi là [con trỏ lơ lửng](https://www.geeksforgeeks.org/dangling-void-null-wild-pointers/) . Tình huống như vậy có thể dẫn đến hành vi không mong muốn trong chương trình và cũng là nguồn gây ra lỗi trong các chương trình C.

Ví dụ;

#include <stdlib.h>

int main(){

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

free(ptr); // sau khi gọi free thì \*ptr trở thành con trỏ lơ lửng

printf(" giai phong bo nho");

ptr = NULL; // xóa con trỏ lơ lửng

return 0;

}

## **3.CON TRỎ MẢNG:**

* 1. **Khái niệm:**

-Con trỏ đến một mảng là con trỏ trỏ đến toàn bộ mảng thay vì phần tử đầu tiên của mảng. Nó coi toàn bộ mảng là một đơn vị duy nhất thay vì là một tập hợp các phần tử đã cho.

-Ví dụ :

#include<stdio.h>

int main(){

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

int \* ptr = array;

printf ( "%p\n" ,ptr);

Return 0 ;

}

-Trong chương trình trên, chúng ta có một con trỏ ptr trỏ đến phần tử thứ 0 của mảng. Tương tự, chúng ta cũng có thể khai báo một con trỏ có thể trỏ đến toàn bộ mảng thay vì chỉ một phần tử của mảng. Con trỏ này hữu ích khi nói về mảng đa chiều.

-Cú pháp của con trỏ mảng: type(\*ptr)[size];

-Trong đó:

* type: Kiểu dữ liệu mà mảng lưu trữ.
* ptr: Tên của biến con trỏ.
* size: Kích thước của mảng mà con trỏ sẽ trỏ tới.

Ví dụ: int (\*ptr)[10];

-Ở đây ptr là con trỏ trỏ đến một mảng gồm 10 số nguyên. Vì subscript có độ ưu tiên cao hơn indirection nên cần phải bao gồm toán tử indirection và tên con trỏ trong dấu ngoặc đơn.

## **4.MẢNG CON TRỎ:**

* 1. **Khái niệm:**

-Trong C, mảng con trỏ là một tập hợp đồng nhất các biến con trỏ được lập chỉ mục là tham chiếu đến một vị trí bộ nhớ. Nó thường được sử dụng trong Lập trình C khi chúng ta muốn trỏ đến nhiều vị trí bộ nhớ có kiểu dữ liệu tương tự trong chương trình C của mình. Chúng ta có thể truy cập dữ liệu bằng cách hủy tham chiếu con trỏ trỏ đến nó.

-Cú pháp: kiểu\_con\_trỏ \*tên\_mảng [kích\_thước\_mảng];

-Trong đó:

* kiểu\_con\_trỏ: Kiểu dữ liệu mà con trỏ đang trỏ tới.
* tên\_mảng: Tên của mảng con trỏ.
* kích\_thước\_mảng:  Kích thước của mảng con trỏ.

-Lưu ý: Điều quan trọng là phải ghi nhớ thứ tự ưu tiên của toán tử và tính kết hợp trong mảng các khai báo con trỏ có kiểu khác nhau vì một thay đổi duy nhất sẽ có nghĩa là toàn bộ điều khác nhau. Ví dụ, bao gồm \*array\_name trong dấu ngoặc đơn sẽ có nghĩa là array\_name là một con trỏ đến một mảng

-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, 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.

## **5.CON TRỎ HÀM:**

* 1. **Khái niệm:**

-Trong C, **con trỏ hàm** là một kiểu con trỏ lưu trữ địa chỉ của hàm, cho phép các hàm được truyền dưới dạng đối số và được gọi một cách động. Nó hữu ích trong các kỹ thuật như hàm gọi lại, chương trình điều khiển sự kiện và đa hình (một khái niệm trong đó hàm hoặc toán tử hoạt động khác nhau dựa trên ngữ cảnh).

-Con trỏ hàm được khai báo theo chữ ký của hàm mà chúng sẽ trỏ tới.

-Cú pháp chung của [khai báo con trỏ hàm](https://www.geeksforgeeks.org/how-to-declare-a-pointer-to-a-function/" \t "_blank) : return\_type (\*pointer\_name)(parameter\_types);

* return\_type: Kiểu giá trị mà hàm trả về.
* parameter\_types: Kiểu tham số mà hàm sử dụng.
* pointer\_name: Tên của con trỏ hàm.

-Dấu ngoặc đơn quanh pointer\_name là bắt buộc, nếu không, nó sẽ được coi là một khai báo hàm có kiểu trả về là return\_type\* và tên là pointer\_name.

-Kiểu của hàm được quyết định bởi kiểu trả về, số lượng và kiểu tham số. Vì vậy, [con trỏ](https://www.geeksforgeeks.org/c-pointers/) hàm phải được khai báo theo cách khớp với chữ ký của hàm mà sau đó nó trỏ tới.

- Ví dụ, trong đoạn mã trên, con trỏ hàm được khai báo là:

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

* 1. **Khởi tạo**

-Sau đó, con trỏ hàm được khởi tạo bằng cách gán địa chỉ của hàm.

pointer\_name = &function\_name

-Chúng ta cũng có thể bỏ qua địa chỉ của toán tử vì tên hàm tự nó hoạt động giống như một con trỏ hàm hằng số.

pointer\_name = function\_name;

-Bắt buộc phải gán hàm có chữ ký tương tự như được chỉ định trong khai báo con trỏ. Nếu không, trình biên dịch có thể hiển thị lỗi không khớp kiểu.

* 1. **Thuộc tính của con trỏ hàm**

-Con trỏ hàm trỏ đến mã thay vì dữ liệu nên có một số hạn chế đối với con trỏ hàm so với các con trỏ khác. Sau đây là một số thuộc tính quan trọng của con trỏ hàm:

* Trỏ tới địa chỉ bộ nhớ của một hàm trong đoạn mã.
* Yêu cầu chữ ký hàm chính xác (kiểu trả về và danh sách tham số).
* Có thể trỏ tới các chức năng khác nhau có chữ ký trùng khớp.
* Không thể thực hiện các phép tính số học như tăng hoặc giảm.
* Hỗ trợ chức năng giống mảng cho các bảng con trỏ hàm.
  1. **Ví dụ**

**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;

}

#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.

## **6.CẤP PHÁT ĐỘNG:**

* 1. **Khái niệm:**

Trong C, một biến được định nghĩa trong một hàm được lưu trữ trong bộ nhớ ngăn xếp. Yêu cầu của bộ nhớ này là nó cần biết kích thước của dữ liệu vào bộ nhớ tại thời điểm biên dịch (trước khi chương trình chạy). Ngoài ra, sau khi được định nghĩa, chúng ta không thể thay đổi kích thước hoặc xóa hoàn toàn bộ nhớ. Để giải quyết vấn đề này, C cung cấp một tính năng gọi là **cấp phát động.** Tính năng này cho phép bạn phân bổ bộ nhớ khi chạy, giúp chương trình của bạn có khả năng xử lý dữ liệu có kích thước khác nhau. Tài nguyên động được lưu trữ trong bộ nhớ [**heap**](https://www.geeksforgeeks.org/memory-layout-of-c-program/)thay vì stack.

* 1. **Phân loại** 
     1. **Malloc**

-Hàm **malloc()** (viết tắt của **memory allocation )** được sử dụng để phân bổ một khối bộ nhớ liền kề trên heap khi chạy. Bộ nhớ được malloc() phân bổ chưa được khởi tạo, nghĩa là nó chứa các giá trị rác.

-Cú pháp: malloc(size);

-Trong đó **size** là số byte cần phân bổ.

-Hàm này trả về một con trỏ void đến bộ nhớ được phân bổ cần được chuyển đổi thành con trỏ có kiểu bắt buộc để có thể sử dụng được. Nếu phân bổ không thành công, nó trả về con trỏ NULL

-Ví dụ: Giả sử chúng ta muốn tạo một mảng để lưu trữ 5 số nguyên. Vì kích thước của int là 4 byte, chúng ta cần 5 \* 4 byte = 20 byte bộ nhớ.

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr = (int \*)malloc(sizeof(int) \* 5); // sử dụng sizeof để lưu trữ 5 số nguyên

// không có bộ nhớ khả dụng, malloc sẽ thất bại và trả về NULL, nên cần kiểm tra lỗi bằng cách so sánh ptr với NULL

if (ptr == NULL) {

printf("that bai");

exit(0);

}

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

ptr[i] = i + 1;

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

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

return 0;

}

* + 1. **Calloc**

-Hàm calloc() (viết tắt của contiguous allocationtục ) tương tự như malloc(), nhưng nó khởi tạo bộ nhớ được phân bổ thành số không. Nó được sử dụng khi bạn cần bộ nhớ có giá trị mặc định là số không.

-Cú pháp: calloc(n, size);

-Trong đó **n** là số phần tử và **size** là kích thước của mỗi phần tử tính theo byte.

Hàm này cũng trả về một con trỏ void đến bộ nhớ được phân bổ được chuyển đổi thành con trỏ có kiểu bắt buộc để có thể sử dụng được. Nếu phân bổ không thành công, nó trả về con trỏ NULL.

-Ví dụ:

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr = (int \*)calloc(5, sizeof(int))

if (ptr == NULL) {

printf("that bai");

exit(0);

}

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

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

return 0;

}

* + 1. **Free**

Bộ nhớ được phân bổ bằng các hàm malloc() và calloc() không được giải phóng riêng. Hàm [free()](https://www.geeksforgeeks.org/free-function-in-c-library-with-examples/) được sử dụng để giải phóng bộ nhớ được phân bổ động trở lại hệ điều hành. Điều cần thiết là giải phóng bộ nhớ không còn cần thiết để tránh rò rỉ bộ nhớ**.**

**-**Cú pháp: free(ptr);

-Trong đó ptr là con trỏ tới bộ nhớ được phân bổ.

Sau khi giải phóng một khối bộ nhớ, con trỏ trở nên không hợp lệ và không còn trỏ đến vị trí bộ nhớ hợp lệ nữa.

-Ví dụ:

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr = (int \*)calloc(sizeof(int), 5);

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

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

return 0;

}

* + 1. **Realloc**

-**[Hàm realloc()](https://www.geeksforgeeks.org/g-fact-66/" \t "_blank)** được sử dụng để thay đổi kích thước khối bộ nhớ đã được phân bổ trước đó. Nó cho phép bạn thay đổi kích thước của phân bổ bộ nhớ hiện có mà không cần giải phóng bộ nhớ cũ và phân bổ khối mới.

-Cú pháp: realloc(ptr, new\_size);

**-Trong đó, ptr là** con trỏ tới khối bộ nhớ được phân bổ trước đó và **new\_size** là kích thước được phân bổ lại mà khối bộ nhớ phải có tính bằng byte.

-Hàm này trả về một con trỏ đến bộ nhớ mới được phân bổ hoặc NULL nếu việc phân bổ lại không thành công. Nếu không thành công, khối bộ nhớ gốc vẫn không thay đổi.

**-**Ví dụ:Giả sử ban đầu chúng ta phân bổ bộ nhớ cho 5 số nguyên nhưng sau đó cần mở rộng mảng để chứa 10 số nguyên. Chúng ta có thể sử dụng realloc() để thay đổi kích thước khối bộ nhớ:

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr = (int \*)malloc(5 \* sizeof(int));

ptr = (int \*)realloc(ptr, 10 \* sizeof(int));

if (ptr == NULL) {

printf("that bai");

exit(0);

}

return 0;

}

* 1. **Các vấn đề liên quan đến phân bổ bộ nhớ động**

-Mặc dù phân bổ bộ nhớ động hữu ích, nhưng nó cũng dễ xảy ra lỗi đòi hỏi phải xử lý cẩn thận để tránh sử dụng bộ nhớ quá mức hoặc thậm chí là sập hệ thống. Một số lỗi phổ biến được đưa ra dưới đây:

* [Rò rỉ bộ nhớ](https://www.geeksforgeeks.org/what-is-memory-leak-how-can-we-avoid/) : Không giải phóng được bộ nhớ được phân bổ động sẽ dẫn đến rò rỉ bộ nhớ, làm cạn kiệt tài nguyên hệ thống.
* [Con trỏ lơ lửng](https://www.geeksforgeeks.org/dangling-void-null-wild-pointers/) : Sử dụng con trỏ sau khi giải phóng bộ nhớ có thể gây ra hành vi không xác định hoặc sự cố.
* [Phân mảnh](https://www.geeksforgeeks.org/what-is-fragmentation-in-operating-system/) : Việc phân bổ và hủy phân bổ lặp đi lặp lại có thể phân mảnh bộ nhớ, khiến không gian heap sử dụng không hiệu quả.
* Lỗi phân bổ : Nếu phân bổ bộ nhớ không thành công, chương trình có thể bị sập trừ khi lỗi được xử lý đúng cách.

## **7.XỬ LÍ TỆP:**

* 1. **Khái niệm:**

Xử lý tệp trong C là quá trình chúng ta tạo, mở, đọc, ghi và đóng các hoạt động trên một tệp. Ngôn ngữ C cung cấp các hàm khác nhau như fopen() , fwrite() , fread() , fseek() , fprintf() , v.v. để thực hiện nhập, xuất và nhiều hoạt động tệp C khác nhau trong chương trình của chúng ta.

Cho đến nay, các hoạt động trong chương trình C được thực hiện trên một dấu nhắc/thiết bị đầu cuối trong đó dữ liệu chỉ được lưu trữ trong bộ nhớ tạm thời (RAM). Dữ liệu này bị xóa khi chương trình được đóng. Nhưng trong ngành công nghiệp phần mềm, hầu hết các chương trình được viết để lưu trữ thông tin được lấy từ chương trình. Việc sử dụng xử lý tệp chính xác là những gì tình huống này đòi hỏi.

Xử lý tệp cho phép chúng ta đọc và ghi dữ liệu vào các tệp được lưu trữ trong bộ nhớ thứ cấp như ổ cứng từ chương trình C của chúng ta.

* 1. **Cách hoạt động:**

Ngôn ngữ C cung cấp các thao tác khác nhau sau đây mà chúng ta có thể thực hiện trên một tệp từ chương trình C của mình:

1. Tạo một tập tin mới.
2. Mở một tập tin hiện có.
3. Đọc từ tập tin.
4. Ghi vào tập tin.
5. Di chuyển đến một vị trí cụ thể trong tệp.
6. Đóng một tập tin.
   1. **Các thành phần trong xử lí tệp C**
      1. **Tệp**

Tệp là một vật chứa dữ liệu. Nó có thể được phân loại thành hai loại dựa trên cách tệp lưu trữ dữ liệu. Chúng như sau:

Tệp văn bản : Tệp văn bản chứa dữ liệu dưới dạng ký tự ASCII và thường được dùng để lưu trữ luồng ký tự.

* + Mỗi dòng trong tệp văn bản kết thúc bằng ký tự xuống dòng ('\n').
  + Nó có thể được đọc hoặc viết bằng bất kỳ trình soạn thảo văn bản nào.
  + Chúng thường được lưu trữ dưới dạng tệp có phần mở rộng là .txt .
  + Tệp văn bản cũng có thể được sử dụng để lưu trữ mã nguồn.

Tệp nhị phân : Tệp nhị phân chứa dữ liệu ở dạng nhị phân (tức là 0 và 1) thay vì ký tự ASCII. Chúng chứa dữ liệu được lưu trữ theo cách tương tự như cách lưu trữ trong bộ nhớ chính.

* + Các tập tin nhị phân chỉ có thể được tạo từ bên trong một chương trình và nội dung của chúng chỉ có thể được đọc bởi một chương trình.
  + An toàn hơn vì chúng không dễ đọc được.
  + Chúng thường được lưu trữ với phần mở rộng tệp là .bin .
    1. **Con trỏ tệp**

Con trỏ tệp là tham chiếu đến một vị trí cụ thể trong tệp đã mở. Nó được sử dụng trong xử lý tệp để thực hiện tất cả các thao tác tệp như đọc, ghi, đóng,...

* 1. **Chức năng của xử lí tệp**
     1. **Mở một tệp tin trong C**

-Để mở một tệp trong C, hàm [fopen()](https://www.geeksforgeeks.org/c-fopen-function-with-examples/)được sử dụng với tên tệp hoặc đường dẫn tệp cùng với các chế độ truy cập cần thiết .

Cú pháp: FILE\* fopen(\*file\_name, \*access\_mode);

Các tham số

* file\_name: tên của tệp khi có trong cùng thư mục với tệp nguồn. Nếu không, đường dẫn đầy đủ.
* access\_mode : Chỉ định thao tác nào sẽ được sử dụng để mở tệp.

-Một số chế độ mở tệp tin:

* (r): mở file ra để đọc, file này phải tồn tại trước.
* (w): mở file ra để ghi, file sẽ được tạo nếu chưa tồn tại, ngược lại sẽ xóa nội dung của file.
* (a): mở file ra để ghi, file sẽ được tạo nếu chưa tồn tại, ngược lại sẽ thêm nội dung vào cuối file. (r+): mở file ra để đọc và ghi, file này phải tồn tại trước. (w+): mở file ra để ghi và đọc, file sẽ được tạo nếu chưa tồn tại, ngược lại sẽ xóa nội dung của file.
* (a+) mở file ra để ghi và đọc, file sẽ được tạo nếu chưa tồn tại, ngược lại sẽ thêm nội dung vào cuối file.

**-**Ví dụ:

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE\* fptr;

fptr = fopen("filename.txt", "r"); //Mở tệp để đọc

if (fptr == NULL) {

printf(" khong the mo tep");

}

return 0;

}

* + 1. **Tạo một tệp tin**

Hàm fopen() không chỉ có thể mở một file mà còn có thể tạo một file nếu file đó chưa tồn tại. Để làm được điều đó, chúng ta phải sử dụng các chế độ cho phép tạo file nếu không tìm thấy như w, w+, wb, wb+, a, a+, ab và ab+ .

-Ví dụ:

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE\* fptr;

fptr = fopen("file.txt", "w");

if (fptr == NULL)

printf("Mo tep khong thanh cong.");

else

printf("tao tep thanh cong.");

return 0;

}

* + 1. **Ghi vào một tệp tin**

Các hoạt động ghi tệp có thể được thực hiện bởi các hàm fprintf() và fputs(). Lập trình C cũng cung cấp một số hàm khác có thể được sử dụng để ghi dữ liệu vào tệp như:

* fprintf(): Tương tự như printf(), hàm này sử dụng chuỗi định dạng và danh sách đối số biến để in đầu ra ra tệp.
* [fputs()](https://www.geeksforgeeks.org/how-to-write-in-a-file-using-fputs-in-c/): In toàn bộ dòng trong tệp và ký tự xuống dòng ở cuối.
* fputc(): In một ký tự đơn vào tệp.
* fputw(): In một số vào tập tin.
* fwrite(): Hàm này ghi số byte được chỉ định vào tệp nhị phân

-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;

* + 1. **Đóng một tập tin**

Hàm fclose() được sử dụng để đóng tệp. Sau các thao tác tệp thành công, bạn phải luôn đóng tệp để xóa tệp khỏi bộ nhớ.

Cú pháp: fclose(file\_pointer);

* + 1. **Di chuyển con trỏ tệp**

Con trỏ tệp thường trỏ đến vị trí theo chế độ hoặc thao tác đọc/ghi cuối cùng. Chúng ta có thể di chuyển con trỏ này theo cách thủ công đến bất kỳ vị trí nào trong tệp bằng hàm [fseek()](https://www.geeksforgeeks.org/fseek-in-c-with-example/" \t "_blank) .

Cú pháp:fseek(fptr, offset, pos);

Trong đó, *pos* là vị trí mà offset được tính và *offset* là số vị trí dịch chuyển từ pos (có thể là số âm hoặc số dương).

## **8.KIỂU CẤU TRÚC:**

* 1. **Khái niệm:**

Trong C, một cấu trúc là một kiểu dữ liệu do người dùng định nghĩa có thể được sử dụng để nhóm các mục có thể có các kiểu khác nhau thành một kiểu duy nhất. Từ khóa struct được sử dụng để định nghĩa một cấu trúc. Các mục trong cấu trúc được gọi là thành viên của nó và chúng có thể thuộc bất kỳ kiểu dữ liệu hợp lệ nào**.**

Không giống như [mảng](https://www.w3schools.com/c/c_arrays.php) , cấu trúc có thể chứa nhiều kiểu dữ liệu khác nhau ( int, float, char, ...).

Cú pháp khai báo :

struct ten\_struct{

data\_type1 data\_field1;

data\_type2 data\_field2;

....

data\_typen data\_fieldn;

};

Trong struct bạn sẽ liệt kê các trường thông tin mà bạn cần lưu cho cấu trúc bạn đang xây dựng kèm theo kiểu dữ liệu của trường thông tin đó.

Ví dụ:

#include <stdio.h>

struct A {

int x;

};

int main() {

struct A a;

a.x = 11;

printf("%d", a.x);

return 0;

}

Trong ví dụ này, một cấu trúc A được định nghĩa để chứa một thành viên số nguyên x . Một biến a có kiểu struct A được tạo ra và thành viên x của nó được khởi tạo thành 11 bằng cách truy cập nó bằng toán tử dot. Sau đó, giá trị của ax được in ra bảng điều khiển.

* 1. **Tạo Biến Cấu Trúc**

Sau khi định nghĩa cấu trúc, chúng ta phải tạo biến của cấu trúc đó để sử dụng nó. Nó tương tự như bất kỳ loại khai báo biến nào khác:

Cú pháp: struct strcuture\_name var;

Chúng ta cũng có thể khai báo các biến cấu trúc bằng định nghĩa cấu trúc.

Cú pháp: struct tên\_cấu\_trúc {   
...   
}var1, var2....;

* 1. **Các hoạt động cơ bản của cấu trúc**

Sau đây là các thao tác cơ bản thường được sử dụng trên các cấu trúc:

* + 1. **Các thành viên cấu trúc truy cập**

Để truy cập hoặc sửa đổi các thành viên của một cấu trúc, chúng ta sử dụng [toán tử dấu chấm (.)](https://www.geeksforgeeks.org/dot-operator-in-c/) . Điều này áp dụng khi chúng ta sử dụng trực tiếp các biến cấu trúc.

tên\_cấu\_trúc . thành\_viên1;   
tên\_cấu\_trúc . thành\_viên2;

Ví dụ:

struct thisinh { char SBD[15];

float toan,ly; };

//Khai bao bien cau truc struct thisinh ts;

//Nhap du lieu cho thi sinh

printf("So bao danh: “);

scanf(“%s”,&ts.SBD);

printf(“Diem Toan: “);

scanf(“%f”,&ts.toan);

printf(“Diem Ly: “);

scanf(“%f”,&ts.ly);

Trong trường hợp chúng ta có con trỏ tới cấu trúc, chúng ta cũng có thể sử dụng toán tử mũi tên để truy cập các thành viên.

structure\_ptr -> thành viên1   
structure\_ptr -> thành viên2

* + 1. **Khởi tạo các thành viên cấu trúc**

Các thành viên cấu trúc không thể được khởi tạo bằng khai báo.

Ví dụ, chương trình C sau đây không biên dịch được.

struct structure\_name {   
data\_type1 member1 = value1; // không thể khởi tạo các thành viên ở đây   
data\_type2 member2 = value2; // không thể khởi tạo các thành viên ở đây   
...   
};

Lý do cho lỗi trên là khi một kiểu dữ liệu được khai báo, không có bộ nhớ nào được phân bổ cho nó. Bộ nhớ chỉ được phân bổ khi các biến được tạo. Vì vậy, không có không gian để lưu trữ.

Chúng ta có thể khởi tạo các thành viên cấu trúc theo 4 cách như sau:

* + - Khởi tạo mặc định

Theo mặc định, các thành viên cấu trúc không được tự động khởi tạo thành 0 hoặc NULL. Các thành viên cấu trúc chưa được khởi tạo sẽ chứa các giá trị rác. Tuy nhiên, khi một biến cấu trúc được khai báo bằng một trình khởi tạo, tất cả các thành viên không được khởi tạo rõ ràng đều được khởi tạo bằng 0.

struct structure\_name = {0}; // Cả x và y đều được khởi tạo thành 0

* Khởi tạo sử dụng toán tử gán

struct structure\_name str;   
str.member1 = giá trị1;   
*....*

Lưu ý: Chúng ta không thể khởi tạo mảng hoặc chuỗi bằng toán tử gán sau khi khai báo biến.

* Khởi tạo bằng cách sử dụng Initializer List

struct structure\_name str = {giá trị1, giá trị2, giá trị3 ....};

Trong kiểu khởi tạo này, các giá trị được gán theo thứ tự tuần tự khi chúng được khai báo trong mẫu cấu trúc.

* Khởi tạo bằng cách sử dụng Danh sách khởi tạo được chỉ định

Khởi tạo được chỉ định cho phép các thành viên cấu trúc được khởi tạo theo bất kỳ thứ tự nào.

struct structure\_name str = { .member1 = giá trị1, .member2 = giá trị2, .member3 = giá trị3 };

* + 1. **Phép gán cấu trúc**

Ta có thể gán một biến cấu trúc cho một biến cấu trúc cùng kiểu.

Ví dụ:

//Khai bao kieu cau truc

struct thisinh { char SBD[15];

float toan,ly,hoa; };

//Khai bao bien cau truc

struct thisinh ts1={"NNHA23456",7,8,9};

struct thisinh ts2;

ts2=ts1

* + 1. **Mảng cấu trúc**

Sau khi khai báo kiểu cấu trúc thì tên kiểu cấu trúc được dùng như các kiểu dữ liệu khác. Chẳng hạn, dùng cấu trúc làm kiểu phần tử của mảng.

Ví dụ:

//Khai bao kieu cau truc

struct thisinh { char SBD[15];

float toan,ly,hoa; };

//Khai bao bien cau truc

struct thisinh ds[100];

strcpy(ds[0].SBD,"NNHA23456");

ds[0].toan=8;

ds[0].ly=8;

ds[0].hoa=9;

## **9.DANH SÁCH LIÊN KẾT:**

**9.1 Khái niệm:**

Danh sách liên kết là một cấu trúc dữ liệu tuyến tính trong đó mỗi phần tử (gọi là nút) được kết nối với phần tử tiếp theo bằng con trỏ. Không giống như mảng, các phần tử của danh sách liên kết được lưu trữ ở các vị trí bộ nhớ ngẫu nhiên.

Danh sách liên kết là một chuỗi các nút trong đó mỗi nút được tạo thành từ hai phần:

* Dữ liệu : Giá trị được lưu trữ trong nút.
* Con trỏ : Tham chiếu đến nút tiếp theo trong chuỗi. (Có thể có nhiều con trỏ cho các loại danh sách liên kết khác nhau.)

Không giống như [mảng](https://www.geeksforgeeks.org/c-arrays/" \t "_blank) , danh sách liên kết không lưu trữ các nút ở các vị trí bộ nhớ liền kề. Thay vào đó, mỗi nút chứa con trỏ đến nút tiếp theo, tạo thành cấu trúc giống như chuỗi và để truy cập bất kỳ phần tử nào (nút), trước tiên chúng ta cần duyệt tuần tự tất cả các nút trước nó.

9.2 Các loại danh sách liên kết:

9.2.1 Danh sách liên kết đơn

Danh sách liên kết hoặc [danh sách liên kết đơn](https://www.geeksforgeeks.org/c-program-to-implement-singly-linked-list/" \t "_blank) là một cấu trúc dữ liệu tuyến tính được tạo thành từ một nhóm các nút trong đó mỗi nút có hai phần: dữ liệu và con trỏ đến nút tiếp theo. Con trỏ của nút cuối cùng (còn được gọi là đuôi) trỏ đến NULL để chỉ ra kết thúc của danh sách liên kết.

Ví dụ:

struct Node {

int data;

struct Node\* next;

}

**9.2.2 Danh sách liên kết kép**

Danh [sách liên kết đôi](https://www.geeksforgeeks.org/doubly-linked-list-in-c/" \t "_blank) phức tạp hơn một chút so với danh sách liên kết đơn. Trong đó, mỗi nút chứa ba phần: dữ liệu, một con trỏ đến nút tiếp theo và một con trỏ bổ sung trỏ đến nút trước đó. Điều này cho phép duyệt theo cả hai hướng, làm cho nó linh hoạt hơn danh sách liên kết đơn.

Ví dụ:

struct Node {

int data;

struct Node\* next;

struct Node\* prev;

}

**9.2.3 Danh sách liên kết tròn**

Danh [sách liên kết vòng tròn](https://www.geeksforgeeks.org/c-program-to-implement-circular-linked-list/" \t "_blank) là một biến thể của danh sách liên kết đơn trong đó nút cuối cùng trỏ ngược lại nút đầu tiên, tạo thành một vòng tròn. Điều này có nghĩa là không có NULL ở cuối và danh sách có thể được duyệt theo cách vòng tròn.

Ví dụ:

struct Node {

int data;

struct Node\* next;

}

# ỨNG DỤNG

**DANH SÁCH QUẢN LÍ SINH VIÊN**

**1. Cấu trúc dữ liệu:**

- Sử dụng danh sách liên kết đơn cho danh sách sinh viên (SinhVien) và danh sách môn học (MonHoc) của mỗi sinh viên.

- Mỗi sinh viên lưu thông tin MSSV, họ tên, lớp và con trỏ tới danh sách môn học.

- Mỗi môn học lưu tên môn, điểm và con trỏ tới môn học tiếp theo.

**2. Các chức năng chính:**

- Thêm sinh viên: Sử dụng cấp phát động (malloc) để tạo node sinh viên mới và thêm vào đầu danh sách.

- Thêm môn học: Thêm môn học vào danh sách môn học của sinh viên được chọn.

- Hiển thị danh sách: Duyệt qua danh sách sinh viên và môn học bằng con trỏ, hiển thị thông tin và điểm trung bình.

- Tính điểm trung bình và xếp loại: Tính điểm trung bình từ danh sách môn học và xếp loại theo yêu cầu (Giỏi, Khá, Trung bình, Yếu).

- Tìm kiếm: Tìm sinh viên theo MSSV hoặc tên (sử dụng strstr để tìm chuỗi con trong tên).

- Thống kê đậu/rớt: Đếm số sinh viên có điểm trung bình >= 5 (đậu) và < 5 (rớt).

- Lưu/đọc file: Lưu dữ liệu vào file dạng text, sử dụng dấu phân cách ---- giữa các sinh viên. Đọc file và tái tạo danh sách liên kết.

- Xuất báo cáo: Ghi thông tin chi tiết của tất cả sinh viên ra file baocao.txt.

- Giải phóng bộ nhớ: Giải phóng tất cả bộ nhớ đã cấp phát khi kết thúc chương trình hoặc khi đọc lại từ file.

**3. Kỹ thuật sử dụng:**

- Cấp phát động: Sử dụng malloc để cấp phát bộ nhớ cho sinh viên và môn học.

- Con trỏ: Sử dụng con trỏ để duyệt danh sách liên kết và quản lý danh sách môn học của từng sinh viên.

- Danh sách liên kết: Danh sách sinh viên và danh sách môn học đều là danh sách liên kết đơn.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Khai bao cau truc mon hoc

typedef struct MonHoc {

char tenMon[50];

float diem;

struct MonHoc\* next;

} MonHoc;

// Khai bao cau truc sinh vien

typedef struct SinhVien {

char maSo[20];

char hoTen[50];

char lop[20];

MonHoc\* dsMonHoc;

struct SinhVien\* next;

} SinhVien;

// Tao mon hoc moi

MonHoc\* taoMonHoc(char ten[], float diem) {

MonHoc\* mon = (MonHoc\*)malloc(sizeof(MonHoc));

strcpy(mon->tenMon, ten);

mon->diem = diem;

mon->next = NULL;

return mon;

}

// Them mon hoc vao danh sach cua sinh vien

void themMonHoc(SinhVien\* sv, MonHoc\* mon) {

if (sv->dsMonHoc == NULL) {

sv->dsMonHoc = mon;

} else {

MonHoc\* temp = sv->dsMonHoc;

while (temp->next != NULL)

temp = temp->next;

temp->next = mon;

}

}

// Tao sinh vien moi

SinhVien\* taoSinhVien(char maSo[], char hoTen[], char lop[]) {

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

strcpy(sv->maSo, maSo);

strcpy(sv->hoTen, hoTen);

strcpy(sv->lop, lop);

sv->dsMonHoc = NULL;

sv->next = NULL;

return sv;

}

// Them sinh vien vao danh sach

void themSinhVien(SinhVien\*\* head, SinhVien\* sv) {

if (\*head == NULL) {

\*head = sv;

} else {

SinhVien\* temp = \*head;

while (temp->next != NULL)

temp = temp->next;

temp->next = sv;

}

}

// Hien thi danh sach sinh vien

void hienThiDanhSach(SinhVien\* head) {

SinhVien\* temp = head;

while (temp != NULL) {

printf("Ma so: %s, Ho ten: %s, Lop: %s\n", temp->maSo, temp->hoTen, temp->lop);

temp = temp->next;

}

}

// Tim sinh vien theo ma so

SinhVien\* timSinhVienTheoMa(SinhVien\* head, char maSo[]) {

SinhVien\* temp = head;

while (temp != NULL) {

if (strcmp(temp->maSo, maSo) == 0)

return temp;

temp = temp->next;

}

return NULL;

}

// Tinh diem trung binh cua sinh vien

float tinhDiemTB(SinhVien\* sv) {

MonHoc\* mon = sv->dsMonHoc;

if (mon == NULL) return 0;

float tong = 0;

int dem = 0;

while (mon != NULL) {

tong += mon->diem;

dem++;

mon = mon->next;

}

return (dem == 0) ? 0 : tong / dem;

}

// Xep loai theo diem

const char\* xepLoai(float diemTB) {

if (diemTB >= 8.0) return "Gioi";

if (diemTB >= 6.5) return "Kha";

if (diemTB >= 5.0) return "Trung binh";

return "Yeu";

}

// Thong ke dau/rot

void thongKeDauRot(SinhVien\* head) {

int dau = 0, rot = 0;

while (head != NULL) {

float dtb = tinhDiemTB(head);

if (dtb >= 5.0)

dau++;

else

rot++;

head = head->next;

}

printf("So sinh vien dau: %d\n", dau);

printf("So sinh vien rot: %d\n", rot);

}

// Giai phong bo nho

void giaiPhong(SinhVien\* head) {

while (head != NULL) {

MonHoc\* m = head->dsMonHoc;

while (m != NULL) {

MonHoc\* tmpM = m;

m = m->next;

free(tmpM);

}

SinhVien\* tmp = head;

head = head->next;

free(tmp);

}

}

// Luu vao file

void luuVaoFile(SinhVien\* head, const char\* tenFile) {

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

if (!f) {

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

return;

}

while (head != NULL) {

fprintf(f, "%s;%s;%s\n", head->maSo, head->hoTen, head->lop);

MonHoc\* m = head->dsMonHoc;

while (m != NULL) {

fprintf(f, "%s,%.2f\n", m->tenMon, m->diem);

m = m->next;

}

fprintf(f, "#\n"); // Danh dau ket thuc mon hoc cua 1 sinh vien

head = head->next;

}

fclose(f);

}

// Doc tu file

SinhVien\* docTuFile(const char\* tenFile) {

FILE\* f = fopen(tenFile, "r");

if (!f) {

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

return NULL;

}

SinhVien\* head = NULL;

char line[100];

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

if (strchr(line, ';')) {

char maSo[20], hoTen[50], lop[20];

sscanf(line, "%[^;];%[^;];%[^\n]", maSo, hoTen, lop);

SinhVien\* sv = taoSinhVien(maSo, hoTen, lop);

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

if (line[0] == '#') break;

char tenMon[50];

float diem;

sscanf(line, "%[^,],%f", tenMon, &diem);

themMonHoc(sv, taoMonHoc(tenMon, diem));

}

themSinhVien(&head, sv);

}

}

fclose(f);

return head;

}

// Xuat bao cao ra file

void xuatBaoCao(SinhVien\* head, const char\* tenFile) {

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

while (head != NULL) {

fprintf(f, "Ho ten: %s - Lop: %s\n", head->hoTen, head->lop);

MonHoc\* m = head->dsMonHoc;

while (m != NULL) {

fprintf(f, " %s: %.2f\n", m->tenMon, m->diem);

m = m->next;

}

float dtb = tinhDiemTB(head);

fprintf(f, " Diem TB: %.2f - Xep loai: %s\n\n", dtb, xepLoai(dtb));

head = head->next;

}

fclose(f);

}

// Ham main - menu chinh

int main() {

SinhVien\* head = NULL;

int choice;

do {

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

printf("1. Them sinh vien\n");

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

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

printf("4. Hien thi chi tiet sinh vien\n");

printf("5. Thong ke dau rot\n");

printf("6. Luu vao file\n");

printf("7. Doc tu file\n");

printf("8. Xuat bao cao\n");

printf("0. Thoat\n");

printf("Nhap lua chon: ");

scanf("%d", &choice);

getchar(); // Xoa '\n'

if (choice == 1) {

char maSo[20], hoTen[50], lop[20];

printf("Nhap ma so: "); fgets(maSo, sizeof(maSo), stdin);

printf("Nhap ho ten: "); fgets(hoTen, sizeof(hoTen), stdin);

printf("Nhap lop: "); fgets(lop, sizeof(lop), stdin);

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

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

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

SinhVien\* sv = taoSinhVien(maSo, hoTen, lop);

int soMon;

printf("Nhap so mon hoc: ");

scanf("%d", &soMon);

getchar();

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

char tenMon[50];

float diem;

printf(" Mon %d - Ten: ", i + 1);

fgets(tenMon, sizeof(tenMon), stdin);

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

printf(" Diem: ");

scanf("%f", &diem);

getchar();

themMonHoc(sv, taoMonHoc(tenMon, diem));

}

themSinhVien(&head, sv);

} else if (choice == 2) {

hienThiDanhSach(head);

} else if (choice == 3) {

char ma[20];

printf("Nhap ma so: ");

fgets(ma, sizeof(ma), stdin);

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

SinhVien\* sv = timSinhVienTheoMa(head, ma);

if (sv)

printf("Tim thay: %s - %s\n", sv->hoTen, sv->lop);

else

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

} else if (choice == 4) {

char ma[20];

printf("Nhap ma so: ");

fgets(ma, sizeof(ma), stdin);

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

SinhVien\* sv = timSinhVienTheoMa(head, ma);

if (sv) {

printf("Ho ten: %s\n", sv->hoTen);

printf("Lop: %s\n", sv->lop);

MonHoc\* m = sv->dsMonHoc;

if (!m) printf("Khong co mon hoc nao.\n");

else {

while (m) {

printf(" %s: %.2f\n", m->tenMon, m->diem);

m = m->next;

}

}

float dtb = tinhDiemTB(sv);

printf("Diem TB: %.2f (%s)\n", dtb, xepLoai(dtb));

} else printf("Khong tim thay!\n");

} else if (choice == 5) {

thongKeDauRot(head);

} else if (choice == 6) {

luuVaoFile(head, "sinhvien.txt");

} else if (choice == 7) {

giaiPhong(head);

head = docTuFile("sinhvien.txt");

printf("Da doc tu file.\n");

} else if (choice == 8) {

xuatBaoCao(head, "baocao.txt");

printf("Da xuat bao cao vao file.\n");

}

} while (choice != 0);

giaiPhong(head);

return 0;

}

# KẾT LUẬN

Thông qua cách xây dựng bài toán trên, cùng với việc kết hợp giữa Mảng và Danh sách Liên kết, cơ bản đã đáp ứng được các điều cần có của việc quản lí danh sách sinh viên. Tuy nhiên, đây chỉ mới là ý tưởng và đang trong quá trình cải thiện, nâng cấp. Không tránh khỏi một số sai sót trong khâu thiết kế chương trình.

Bên cạnh đó, em mong được nhận những nhận xét từ cô 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 của bài toán trong việc quản lí danh sách 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 15/5/2025].

[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 16/04/2024].

[3]. “Stack Overflow”, [Online]. Available: [Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/). [Accessed 16/05/2025].

[4]. “GitHub”, [Online]. Available: [GitHub](https://github.com/) (<https://github.com>). [Accessed 19/05/2025].