**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN MÔN**

**KỸ THUẬT LẬP TRÌNH**

**ĐỀ TÀI: TÌM HIỂU NGÔN NGỮ TRÌNH C VÀ**

**ỨNG DỤNG VÀO BÀI TOÁN QUẢN LÍ SINH VIÊN**

Giảng viên hướng dẫn: TRẦN PHONG NHÃ

Sinh viên thực hiện: TRẦN TIẾN HỢP

VÕ HOÀNG PHÚC

ĐỖ NHẬT THANH

Lớp : CQ.CNTT

Khoá :K65

Tp. Hồ Chí Minh, năm 2025

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN MÔN**

**KỸ THUẬT LẬP TRÌNH**

**ĐỀ TÀI: TÌM HIỂU NGÔN NGỮ LẬP TRÌNH C VÀ**

**ỨNG DỤNG VÀO BÀI TOÁN QUẢN LÍ SINH VIÊN**

Giảng viên hướng dẫn: TRẦN PHONG NHÃ

Sinh viên thực hiện: TRẦN TIẾN HỢP

VÕ HOÀNG PHÚC

ĐỖ NHẬT THANH

Lớp : CQ.CNTT

Khoá :K65

Tp. Hồ Chí Minh, năm 2025

# **LỜI CẢM ƠN**

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

Em xin chân thành cảm ơn quý thầy cô đã giúp đỡ tạo điều kiện để em hoàn thành báo cáo bài tập lớn đề tài “Tìm hiểu về ngôn ngữ lập trình C và ứng dụng vào bài toán quản lí sinh viên”. Đặc biệt em xin cảm ơn thầy Trần Phong Nhã đã nhiệt tình giúp đỡ, hướng dẫn cho em kiến thức, định hướng và kỹ năng để có thể hoàn thành bài báo cáo này.

Tuy đã cố gắng trong quá trình nghiên cứu tìm hiểu tuy nhiên do kiến thức còn hạn chế nên vẫn còn tồn tại nhiều thiếu sót. Vì vậy em rất mong nhận được sự đóng góp ý kiến của Quý thầy cô bộ môn để đề tài của em có thể hoàn thiện hơn.

Lời sau cùng, em xin gửi lời chúc tới Quý Thầy Cô Bộ môn Công nghệ thông tin và hơn hết là thầy Trần Phong Nhã có thật nhiều sức khỏe, có nhiều thành công trong công việc. Em xin chân thành cảm ơn!

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

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

# **MỤC LỤC**

[LỜI CẢM ƠN i](#_Toc198935296)

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN ii](#_Toc198935297)

[MỤC LỤC iii](#_Toc198935298)

[BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ vi](#_Toc198935299)

[CHƯƠNG 1. MỞ ĐẦU 1](#_Toc198935300)

[1.1 Tông quan đề tài 1](#_Toc198935301)

[1.2 Mục tiêu nghiên cứu 1](#_Toc198935302)

[1.3 Phạm vi và đối tượng nghiên cứu 1](#_Toc198935303)

[1.4 Phương pháp thực hiện 1](#_Toc198935304)

[1.5 Cấu trúc báo cáo 1](#_Toc198935305)

[CHƯƠNG 2. CỞ SỞ LÝ THUYẾT 2](#_Toc198935306)

[2.1. Hàm 2](#_Toc198935307)

[2.1.1. Khái niệm 2](#_Toc198935308)

[2.1.2. Khai báo hàm 2](#_Toc198935309)

[2.1.3. Cài đặt hàm 2](#_Toc198935310)

[2.2. Con Trỏ 3](#_Toc198935311)

[2.2.1. Khái niệm. 3](#_Toc198935312)

[2.2.2. Khai báo con trỏ. 3](#_Toc198935313)

[2.2.3. Khởi tạo con trỏ. 4](#_Toc198935314)

[2.2.4. Giải tham chiếu con trỏ. 4](#_Toc198935315)

[2.2.5. Con trỏ và mảng. 5](#_Toc198935316)

[2.2.6. Số học con trỏ. 8](#_Toc198935317)

[2.3. Con trỏ hàm. 9](#_Toc198935318)

[2.3.1. Khái niêm và cú pháp. 9](#_Toc198935319)

[2.3.2. Truyền đối số cho hàm 9](#_Toc198935320)

[2.3.2. Ưu điểm của truyền con trỏ vào hàm 10](#_Toc198935321)

[2.3.3. Các tình huống sử dụng con trỏ trong truyền tham số. 10](#_Toc198935322)

[2.4. Mảng con trỏ. 11](#_Toc198935323)

[2.5. Con trỏ tới con trỏ. 12](#_Toc198935324)

[2.5. Ưu và nhược điểm của con trỏ. 12](#_Toc198935326)

[2.6. CẤP PHÁT ĐỘNG 13](#_Toc198935327)

[2.6.1 Con trỏ vô kiểu. 13](#_Toc198935328)

[2.6.2 Phân vùng bộ nhớ. 14](#_Toc198935329)

[2.3.3. Cấp phát động. 16](#_Toc198935331)

[2.7. STRUCT 17](#_Toc198935332)

[2.7.1 Khái niệm và cú pháp 17](#_Toc198935333)

[2.7.2 khai báo biến có kiểu struct. 18](#_Toc198935334)

[2.7.3 Truy xuất thuộc tính trong struct. 18](#_Toc198935335)

[2.7.4 Kích thước của 1 struct 19](#_Toc198935336)

[2.8. LINKLIST 21](#_Toc198935339)

[2.8.1 Khái niệm 21](#_Toc198935340)

[2.8.2 Một số áp dụng Linked List trong chương trình. 22](#_Toc198935342)

[Chương 3: CHƯƠNG TRÌNH 25](#_Toc198935345)

[3.1 TỔNG QUAN ỨNG DỤNG. 25](#_Toc198935346)

[3.1.1. Mục tiêu 25](#_Toc198935347)

[3.2.2. Cấu trúc dữ liệu 25](#_Toc198935348)

[3.3.3. Các thao tác chính 25](#_Toc198935349)

[3.3.4. Quản lý bộ nhớ 25](#_Toc198935350)

[3.3.5. Giao diện người dùng 26](#_Toc198935351)

[3.2. THIẾT KẾ ỨNG DỤNG 26](#_Toc198935353)

[3.2.1. Cấu trúc chương trình 26](#_Toc198935354)

[3.2.2. Các hàm chính 28](#_Toc198935358)

[3.3.3. Các tính năng nâng cao 29](#_Toc198935359)

[3.2.THỰC NGHIỆM 29](#_Toc198935360)

[Hình 3.9 Ảnh demo chức năng xóa 1 sinh viên và kết quả 31](#_Toc198935365)

[Hình 3.10 Ảnh demo menu con tìm kiếm và kết quả 3 chức năng tìm kiếm 32](#_Toc198935366)

[Hình 3.12 Ảnh demo chức năng thống kê và kết quả 33](#_Toc198935368)

[Hình 3.13 Ảnh demo chức năng sao lưu file và kết quả 33](#_Toc198935369)

[CHƯƠNG IV: KẾT QUẢ VÀ KIẾN NGHỊ 34](#_Toc198935370)

[4.1. Kết quả đạt được 34](#_Toc198935371)

[4.2. Kiến nghị 34](#_Toc198935372)

[TÀI LIỆU THAM KHẢO 35](#_Toc198935373)

# **BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ**

[Hình 2.1 Ảnh minh họa con trỏ trỏ con trỏ 12](#_Toc197518275)

[Hình 2.2 Ảnh memory layout 15](#_Toc197518276)

[Hình 2.3. Cấu trúc bộ nhớ của một struct có kích thước 24 byte với padding. 19](#_Toc197518277)

[Hình 2.4 Cấu trúc bộ nhớ của một struct có kích thước 16 byte với padding. 20](#_Toc197518278)

[Hình 2.5. Minh họa danh sách liên kết đơn với ba nút và con trỏ head 21](#_Toc197518279)

[Hình 2.6 Ảnh code hàm xóa sinh viên 22](#_Toc197518280)

[Hình 2.7 Ảnh code hàm them một sinh viên 23](#_Toc197518281)

[Hình 3.1Ảnh minh họa các chức năng 26](#_Toc197518282)

[Hình 3.3 Ảnh code file lib.cpp 27](#_Toc197518283)

[Hình 3.2 Ảnh code file lib.h 27](#_Toc197518284)

[Hình 3. 4 Ảnh code file main.cpp 28](#_Toc197518285)

[Hình 3.5Ảnh demo giao diện khi chạy chương trình 29](#_Toc197518286)

[Hình3. 6 Ảnh demo chức năng in danh sách 30](#_Toc197518287)

[Hình 3.7 Ảnh demo chức năng thêm sinh viên và kết quả 30](#_Toc197518288)

[Hình 3.8 Ảnh demo chức năng sửa thông tin và kết quả 31](#_Toc197518289)

[Hình 3.9 Ảnh demo chức năng xóa 1 sinh viên và kết quả 31](#_Toc197518290)

[Hình 3.10 Ảnh demo menu con tìm kiếm và kết quả 3 chức năng tìm kiếm 32](#_Toc197518291)

[Hình 3.11 Ảnh demo chức menu con sắp xếp và kêt quả của từng chức năng sắp xếp 32](#_Toc197518292)

[Hình 3.12 Ảnh demo chức năng thống kê và kết quả 33](#_Toc197518293)

[Hình 3.13 Ảnh demo chức năng sao lưu file và kết quả 33](#_Toc197518294)

# **CHƯƠNG 1. MỞ ĐẦU**

## **1.1 Tông quan đề tài**

Trong lĩnh vực lập trình hệ thống và phát triển phần mềm hiệu năng cao, ngôn ngữ lập trình C giữ một vai trò quan trọng nhờ khả năng thao tác trực tiếp trên bộ nhớ và tối ưu tài nguyên. Một trong những khái niệm cốt lõi tạo nên sức mạnh của ngôn ngữ C chính là con trỏ. Việc hiểu rõ và sử dụng thành thạo con trỏ không chỉ giúp lập trình viên xử lý dữ liệu linh hoạt, mà còn là nền tảng để tiếp cận các cấu trúc dữ liệu như mảng động, danh sách liên kết, cây, đồ thị và các kỹ thuật quản lý bộ nhớ.

## **1.2 Mục tiêu nghiên cứu**

Đề tài này nhằm mục tiêu:

Trình bày có hệ thống các kiến thức lý thuyết về con trỏ, bao gồm: khai báo, khởi tạo, số học con trỏ, con trỏ hàm, con trỏ mảng, con trỏ tới con trỏ, và cấp phát động.

Phân tích mối quan hệ giữa con trỏ và các cấu trúc dữ liệu như mảng, struct, và danh sách liên kết.

Áp dụng kiến thức về con trỏ để xây dựng một số hàm quản lý dữ liệu trong chương trình C, như thêm, xóa sinh viên từ danh sách liên kết.

## **1.3 Phạm vi và đối tượng nghiên cứu**

Đối tượng nghiên cứu chính của đề tài là con trỏ trong ngôn ngữ lập trình C, cụ thể là các kỹ thuật xử lý con trỏ và ứng dụng của chúng trong cấu trúc dữ liệu động. Phạm vi của báo cáo không bao gồm các thư viện ngoài chuẩn hoặc ngôn ngữ lập trình khác.

## **1.4 Phương pháp thực hiện**

Báo cáo được xây dựng dựa trên việc:

Tổng hợp và hệ thống hóa kiến thức từ sách giáo khoa, tài liệu kỹ thuật, và các nguồn học thuật về ngôn ngữ C.

Viết mã nguồn minh họa để chứng minh các khái niệm lý thuyết.

Phân tích, đánh giá hiệu quả và đặc điểm của các đoạn chương trình.

## **1.5 Cấu trúc báo cáo**

Chương 1: Mở đầu.

Chương 2: Cơ sở lý thuyết.

Chương 3: Chương trình.

Chương 4: Kết quả và kiến nghị.

**CHƯƠNG 2. CỞ SỞ LÝ THUYẾT**

**2.1. Hàm**

**2.1.1. Khái niệm**

Hàm (function) là một các khối lệnh có nhiệm vụ thực hiện một chức năng nào đó. Một hàm trong C được hiểu theo nghĩa là một “Routine” hoặc “subprogram".

Hàm là một đơn vị độc lập trong C: Không được xây dựng hàm bên trong 1 hàm khác và mỗi hàm có thể có các biến, hằng, mảng riêng.

Một chương trình viết bằng C gồm 1 hoặc nhiều hàm, trong đó có 1 hàm chính là hàm “main()”.

Hàm có thể có giá trị trả về (kết quả của hàm) hoặc không có giá trị trả về (chỉ đơn thuần thực hiện 1 công việc nào đó).

Hàm có thể có hoặc không có tham số.

**2.1.2. Khai báo hàm**

Nguyên mẫu hàm (prototype của hàm)

* Prototype hàm chỉ rõ các đặc điểm chính
* Tên của hàm
* Số lượng và kiểu của từng tham số hàm sẽ nhận + Giá trị trả về sau khi hàm kết húc.
* Phải khai báo prototype của hàm trước khi sử dụng hàm -> thường khai báo nguyên mẫu ở đầu chương trình.

Prototype hàm không cho thấy hàm sẽ làm những gì.

Công thức khai báo:

Kiểu\_hàm Tên\_hàm (Kiểu\_tham\_số\_1, Kiểu\_tham\_số\_2, ...);

**2.1.3. Cài đặt hàm**

Xác định chính xác những lệnh mà hàm phải thực hiện.

Thường được cài đặt ở cuối chương trình hoặc đặt trong 1 file thư viện riêng

Cách cài đặt:

Kiểu\_hàm Tên\_hàm (Kiểu\_1 Tên\_tham\_số\_1,

Kiểu\_2 Tên\_tham\_số\_2,...)

{

- Khai báo biến, hằng cục bộ trong hàm

- Các lệnh hàm sẽ thực hiện

return <kết quả của hàm>; }

**2.1.4. Quy tắc hoạt động của hàm**

Lời gọi hàm có dạng tổng quát như sau: Tên\_hàm ( [danh sách tham số thực] ).

Số lượng tham số thực trong lời gọi hàm phải bằng số lượng tham số hình thức (trong khai báo hàm).

Kiểu của các tham số thực phải tương ứng với kiểu của tham số hình thức.

Khi gặp 1 lời gọi hàm tại 1 vị trí nào đó trong chương trình, máy sẽ dời vị trí đó chuyển đến thực hiện các lệnh của hàm được gọi.

Thứ tự thực hiện khi có 1 lời gọi hàm:

* Cấp phát bộ nhớ cho các biến cục bộ.
* Gán giá trị của tham số thực sự cho tham số hình thức.
* Thực hiện các lệnh trong thân của hàm
* Gặp lệnh return hoặc dấu } kết thúc hàm thì xóa vùng nhớ đã cấp cho các biến cục bộ và rời khỏi hàm -> trở về vị trí đã dừng sau lời gọi hàm.

Nếu thoát khỏi hàm từ câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

**2.2. Con Trỏ**

**2.2.1. Khái niệm.**

Con trỏ là một biến lưu trữ địa chỉ bộ nhớ của một biến khác. Thay vì giữ một giá trị trực tiếp, nó giữ địa chỉ nơi giá trị được lưu trữ trong bộ nhớ. Có 2 toán tử quan trọng mà chúng ta sẽ sử dụng trong các khái niệm về con trỏ:

Toán tử & − Còn lại được gọi là "Địa chỉ Toán tử". Toán tử này được sử dụng để tham chiếu, tức là lấy địa chỉ của một biến hiện có (sử dụng & ) để đặt một con trỏ biến.

Toán tử \* − Còn được gọi là "toán tử hủy tham chiếu". Hủy tham chiếu một con trỏ được thực hiện bằng toán tử \* để lấy giá trị từ bộ nhớ địa chỉ mà con trỏ tới.

**2.2.2. Khai báo con trỏ.**

Cú pháp : type \*var-name;

**“type”** là kiểu cơ sở của con trỏ. Đây phải là kiểu dữ liệu C hợp lệ . “**var-name”** là tên của biến con trỏ. Dấu hoa thị “ **\* ”** dùng để khai báo con trỏ tương tự như dấu hoa thị dùng để nhân. Tuy nhiên, trong câu lệnh này, dấu hoa thị được sử dụng để chỉ định biến là một con trỏ.

Ví dụ:

char \*ch ;

double \*dp;

float \*fp;

int \*ip;

Chú ý:

Mỗi con trỏ chỉ trỏ tới một kiểu dữ liệu cụ thể.

Ngoại lệ duy nhất là void \* → có thể trỏ tới bất kỳ kiểu nào, không thể truy xuất trực tiếp (không thể dùng \*vp với void \*vp).

Ví dụ:

void \*vp;

int a = 5;

vp = &a;

printf("%d\n", \*vp); // Lỗi! Vì không biết vp trỏ tới kiểu gì

printf("%d\n", \*(int \*)vp); // Đúng! Đã ép kiểu về int\*

**2.2.3. Khởi tạo con trỏ.**

Khởi tạo con trỏ là quá trình chúng ta gán một số giá trị ban đầu cho biến con trỏ. Chúng ta sử dụng toán tử (&) addressof để lấy địa chỉ bộ nhớ của một biến và sau đó lưu trữ nó trong biến con trỏ.

Cú pháp: pointer\_variable = &variable;

Ví dụ:

int x = 10;

int \*ptr = &x;

Ở đây, x là một biến số nguyên, ptr là một con trỏ số nguyên. Con trỏ ptr đang được khởi tạo bằng x.

**2.2.4. Giải tham chiếu con trỏ.**

Dùng để truy cập giá trị tại địa chỉ mà con trỏ trỏ đến bằng cách dùng toán tử \*.

Ví dụ:

int x = 1, y = 2, z[10];

int \*ip;

ip = &x;

y = \*ip;

\*ip = 0;

ip = &z[0];

Dòng

int x = 1, y = 2, z[10] ;

Khai báo: x = 1 y = 2 z là mảng 10 phần tử kiểu int

Dòng

int \*ip;

Khai báo con trỏ ip có thể trỏ đến một biến kiểu int

Dòng

ip = &x;

Gán địa chỉ của x cho ip. Bây giờ ip trỏ đến x.

y = \*ip;

Lấy giá trị tại địa chỉ ip trỏ đến (giá trị của x)

Gán vào y → y = 1

\*ip = 0;

Gán 0 vào nơi ip trỏ tới → tức là x = 0

ip = &z[0];

ip giờ trỏ tới phần tử đầu tiên của mảng z

Từ đây, bạn có thể truy cập mảng bằng con trỏ:

Ví dụ: \*(ip + 1) là z[1], \*(ip + 2) là z[2], ...

**2.2.5. Con trỏ và mảng.**

Trong C, có một mối quan hệ chặt chẽ giữa con trỏ và mảng, đủ mạnh để con trỏ và mảng nên được thảo luận cùng nhau. Mọi thao tác có thể thực hiện với việc chỉ số mảng cũng có thể thực hiện với con trỏ. Phiên bản con trỏ thường nhanh hơn nhưng, ít nhất đối với những người chưa quen, có thể sẽ hơi khó hiểu.

Khai báo:

int a[10];

A white rectangular object with black lines

AI-generated content may be incorrect.định nghĩa một mảng có kích thước 10, tức là một khối gồm 10 đối tượng liên tiếp có tên là a[0], a[1], ..., a[9].

Cách ghi chú a[i] ám chỉ phần tử thứ i của mảng. Nếu pa là một con trỏ đến kiểu số nguyên, khai báo như sau:

int \*pa;

thì câu lệnh:

pa = &a[0];

gán pa trỏ tới phần tử thứ 0 của mảng a, nghĩa là pa chứa địa chỉ của a[0].

A line of squares with a white background

AI-generated content may be incorrect.

Bây giờ, câu lệnh:

x = \*pa;

sao chép giá trị của a[0] vào x.

Nếu pa trỏ đến một phần tử cụ thể trong mảng, thì theo định nghĩa, pa+1 trỏ đến phần tử tiếp theo, pa+i trỏ đến i phần tử sau pa, và pa-i trỏ đến i phần tử trước. Do đó, nếu pa trỏ tới a[0],

\*(pa+1)

chỉ đến giá trị của a[1], pa+i là địa chỉ của a[i], và \*(pa+i) là giá trị của a[i].

A line with a curved arrow

AI-generated content may be incorrect. Theo định nghĩa, giá trị của một biến hoặc biểu thức có kiểu mảng là địa chỉ của phần tử đầu tiên trong mảng. Do đó, sau khi gán:

pa = &a[0];

thì pa và a có giá trị giống nhau. Vì tên của một mảng là một từ đồng nghĩa với địa chỉ của phần tử đầu tiên, câu lệnh pa = &a[0] cũng có thể viết thành:

pa = a;

Trong C, biểu thức a[i] có thể thay thế bằng \*(a + i), vì C tự động chuyển đổi chúng thành nhau. Tương tự, &a[i] và a + i đều là địa chỉ của phần tử thứ i trong mảng a. Nếu pa là con trỏ, thì pa[i] tương đương với \*(pa + i).

Tuy nhiên, có sự khác biệt giữa tên mảng và con trỏ: con trỏ là một biến, nên pa = a và pa++ hợp lệ. Trong khi đó, tên mảng không phải là một biến, nên các cấu trúc như a = pa và a++ là không hợp lệ

Khi một tên mảng được truyền vào một hàm, điều được truyền đi là địa chỉ của phần tử đầu tiên. Trong hàm được gọi, tham số này là một biến cục bộ, do đó tham số kiểu tên mảng là một con trỏ, tức là một biến chứa một địa chỉ.

Ví dụ:

Hàm strlen tính chiều dài của một chuỗi:

int strlen(char \*s) {

int n;

for (n = 0; \*s != '\0'; s++)

n++;

return n;

}

Vì s là một con trỏ, việc tăng giá trị của nó là hoàn toàn hợp lệ; s++ không ảnh hưởng đến chuỗi ký tự trong hàm gọi strlen, mà chỉ tăng con trỏ của hàm strlen. Điều này có nghĩa là các lời gọi như:

strlen("hello, world"); /\* chuỗi hằng \*/

strlen(array); /\* mảng ký tự array[100]; \*/

strlen(ptr); /\* con trỏ ký tự ptr; \*/

đều hợp lệ.

Về tham số chính thức trong định nghĩa hàm:

char s[];

và

char \*s;

là tương đương; chúng ta thường sử dụng cái sau vì nó chỉ rõ hơn rằng biến là một con trỏ. Khi một tên mảng được truyền vào hàm, hàm có thể thuận tiện cho rằng nó nhận được một mảng hoặc một con trỏ và thao tác với nó tương ứng. Nó thậm chí có thể sử dụng cả hai cách ghi chú nếu cần thiết và rõ ràng.

Có thể truyền một phần của mảng vào hàm, bằng cách truyền một con trỏ đến phần đầu của mảng con. Ví dụ, nếu a là một mảng, thì:

f( &a[2])

và

f(a+2)

đều truyền vào hàm f địa chỉ của mảng con bắt đầu từ a[2]. Trong hàm f, khai báo tham số có thể là:

f(int arr[]) { ... }

hoặc

f(int \*arr) { ... }

Vì vậy, đối với hàm f, việc tham số này tham chiếu một phần của mảng lớn hơn là không quan trọng.

**2.2.6. Số học con trỏ.**

Số [học con trỏ](https://www.geeksforgeeks.org/pointer-arithmetics-in-c-with-examples/) đề cập đến các phép tính số học hợp lệ hoặc hợp lệ có thể được thực hiện trên một con trỏ. Nó hơi khác so với các phép tính mà chúng ta thường sử dụng cho các phép tính toán học vì chỉ có một tập hợp các phép tính giới hạn có thể được thực hiện trên các con trỏ. Các phép tính này bao gồm:

Tăng/Giảm 1tức là nó sẽ di chuyển sang phần tử kế tiếp/trước đó của kiểu mà nó trỏ đến.

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

int \*p = arr;

p++; // trỏ tới arr[1]

printf("%d\n", \*p); // Kết quả: 20

p--; // quay lại arr[0]

printf("%d\n", \*p); // Kết quả: 10

Phép cộng/trừ số nguyên n với một con trỏ tương đương với việc dịch n phần tử.

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

int \*p = arr;

p = p + 2; // hoặc p += 2;

printf("%d\n", \*p); // Kết quả: 30

p = p - 1;

printf("%d\n", \*p); // Kết quả: 20

Trừ hai con trỏ cùng kiểu trong một mảng để biết khoảng cách (số phần tử):

int arr[] = {5, 10, 15, 20};

int \*p1 = &arr[1];

int \*p2 = &arr[3];

int diff = p2 - p1;

printf("Khoảng cách: %d\n", diff); // Kết quả: 2

p2 - p1 cho ra số phần tử cách nhau giữa 2 con trỏ (nếu trỏ cùng mảng).

So sánh/Gán hai con trỏ cùng loại: có thể so sánh bằng ==, !=, >, <, v.v. nếu hai con trỏ trỏ cùng vùng nhớ.

int x = 10, y = 20;

int \*p1 = &x;

int \*p2 = &y;

if (p1 != p2) {

printf("Hai con trỏ khác nhau\n");

}

p2 = p1; // gán địa chỉ p1 cho p2

if (p1 == p2) {

printf("Hai con trỏ giống nhau\n");

}

**2.3. Con trỏ hàm.**

**2.3.1. Khái niêm và cú pháp.**

Con trỏ hàm là một loại con trỏ trong ngôn ngữ lập trình C, dùng để lưu trữ địa chỉ của một hàm. Thông qua con trỏ hàm, người lập trình có thể gọi gián tiếp một hàm, truyền hàm như một đối số cho hàm khác, hoặc xây dựng các cấu trúc chương trình động linh hoạt (ví dụ: bảng chọn chức năng, callback...).

Cú pháp khai báo:

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

Ví dụ:

int (\*p)(int, int);

//con trỏ hàm p trỏ đến một hàm nhận hai tham số int và trả về int.

Để gán địa chỉ của một hàm cụ thể vào con trỏ, ta sử dụng:

p = ten\_ham;

Và để gọi hàm thông qua con trỏ, cú pháp:

p(a, b);

Ví dụ :

int main() {

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

p = add; // gán địa chỉ hàm add

int result = p(3, 4); // gọi hàm qua con trỏ

printf("Kết quả: %d\n", result); // In ra: Kết quả: 7

return 0;

}

**2.3.2. Truyền đối số cho hàm**

Hàm trong C có thể được gọi theo hai cách: gọi theo giá trị (Call by Value): Truyền bản sao giá trị vào hàm. Mọi thay đổi trong hàm không ảnh hưởng tới biến gốc. Gọi theo tham chiếu (Call by Reference): Truyền địa chỉ (con trỏ) vào hàm. Mọi thay đổi trong hàm sẽ ảnh hưởng trực tiếp tới biến gốc.

Cú pháp gọi hàm theo tham chiếu:

type function\_name(type \*var1, type \*var2, ...);

Ví dụ:

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

Khi gọi hàm:

swap(&x, &y);

ta truyền địa chỉ của x và y cho hàm swap.

**2.3.2. Ưu điểm của truyền con trỏ vào hàm**

Cho phép thay đổi biến gốc từ trong hàm: khi truyền con trỏ, các thay đổi trong hàm ảnh hưởng trực tiếp đến vùng nhớ chứa giá trị gốc. Điều này giúp vượt qua giới hạn của "call by value".

Trả về nhiều giá trị từ một hàm: C thông thường chỉ cho phép hàm trả về một giá trị. Tuy nhiên, nếu ta truyền con trỏ vào hàm, ta có thể thay đổi nhiều biến từ bên trong hàm tương đương với việc trả về nhiều giá trị. Ngoài ra, ta cũng có thể trả về con trỏ tới mảng hoặc struct, giúp truyền lại nhiều dữ liệu.

**2.3.3. Các tình huống sử dụng con trỏ trong truyền tham số.**

Truyền con trỏ đến biến int: dùng để thay đổi trực tiếp giá trị biến từ hàm gọi.

void change(int \*p) {

\*p = 100;

}

int x = 10;

change(&x);

// x bây giờ là 100

Truyền con trỏ vào mảng:Vì mảng trong C mặc định là con trỏ, bạn có thể thao tác trực tiếp trên mảng đã truyền.

void updateArray(int \*arr, int size) {

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

arr[i] += 1;

}

}

Truyền con trỏ đến struct:Cho phép sửa đổi trực tiếp các trường bên trong một biến kiểu struct.

struct Point {

int x, y;

};

void move(struct Point \*p) {

p->x += 1;

p->y += 1;

}

**2.4. Mảng con trỏ.**

Giống như một mảng số nguyên chứa nhiều biến kiểu int, một mảng con trỏ chứa nhiều biến kiểu con trỏ. Mỗi phần tử trong mảng là một con trỏ, và mỗi con trỏ này trỏ đến một địa chỉ bộ nhớ nào đó.

Tên của một mảng trong C có thể được dùng như một con trỏ, vì nó chính là địa chỉ của phần tử đầu tiên trong mảng. Nếu ta lưu địa chỉ của mảng này vào một con trỏ khác, ta có thể thao tác với mảng bằng cách sử dụng phép toán trên con trỏ (pointer arithmetic).

Để khai báo một mảng con trỏ trong ngôn ngữ C, ta khai báo gần giống như khai báo một biến con trỏ bình thường, nhưng có thêm cặp dấu ngoặc vuông để thể hiện đó là một mảng. Cụ thể, ta dùng kiểu dữ liệu, theo sau là dấu sao và tên biến, rồi đến dấu ngoặc vuông chứa kích thước mảng.

Ví dụ:

char \*arr[5];

Trong mảng này, mỗi phần tử arr[i] là một con trỏ đến một chuỗi ký tự (string).

Ví dụ cụ thể bên dưới cho thấy cách sử dụng mảng con trỏ để trả về tên tháng:

char \*month\_name(int n)

{

static char \*name[] = {

"Illegal month",

"January", "February", "March",

"April", "May", "June",

"July", "August", "September",

"October", "November", "December"

};

return (n < 1 || n > 12) ? name[0] : name[n]; }

Trong đoạn mã trên, biến name là một mảng các con trỏ đến kiểu char. Mỗi phần tử trong mảng trỏ đến một chuỗi tương ứng với tên của một tháng. Nếu n nằm ngoài phạm vi từ 1 đến 12, hàm sẽ trả về chuỗi "Illegal month".Trình biên dịch sẽ tự động xác định kích thước của mảng name dựa trên số lượng chuỗi trong danh sách khởi tạo.

**A diagram of a number

AI-generated content may be incorrect.2.5. Con trỏ tới con trỏ.**

Hình 2.1Ảnh minh họa con trỏ trỏ con trỏ

Chúng ta có thể có một biến con trỏ lưu trữ địa chỉ của chính con trỏ khác.Trong hình trên, "a" là một biến "int" bình thường, có con trỏ là "x". Đổi lại, biến này lưu trữ địa chỉ của "x".Lưu ý rằng "y" được khai báo là "int \*\*" để chỉ ra rằng nó là con trỏ đến một biến con trỏ khác. Rõ ràng, "y" sẽ trả về địa chỉ của "x" và "\*y" là giá trị trong "x" (là địa chỉ của "a").Để lấy giá trị của "a" từ "y", chúng ta cần sử dụng biểu thức "\*\*y". Thông thường, "y" sẽ được gọi là con trỏ đến một con trỏ .

Ví dụ:

#include <stdio.h>

int main() {

int x = 10;

int \*p = &x; // con trỏ p trỏ đến biến x

int \*\*pp = &p; // con trỏ pp trỏ đến con trỏ p

printf("Giá trị của x: %d\n", x); // 10

printf("Giá trị thông qua \*p: %d\n", \*p); // 10

printf("Giá trị thông qua \*\*pp: %d\n", \*\*pp); // 10

return 0;

}

**2.5. Ưu và nhược điểm của con trỏ.**

Ưu điểm:

* Sau đây là những ưu điểm chính của con trỏ trong C:
* Con trỏ được sử dụng để phân bổ và hủy phân bổ bộ nhớ động.
* Một mảng hoặc một cấu trúc có thể được truy cập hiệu quả bằng con trỏ.
* Con trỏ hữu ích trong việc truy cập các vị trí bộ nhớ.
* Con trỏ được sử dụng để tạo thành các cấu trúc dữ liệu phức tạp như linked list, đồ thị, …
* Con trỏ làm giảm độ dài của chương trình và thời gian thực hiện của nó.

Nhược điểm:

* Con trỏ dễ bị lỗi và có những nhược điểm sau:
* Có thể xảy ra lỗi bộ nhớ nếu cung cấp giá trị không chính xác cho con trỏ.
* Con trỏ có vẻ hơi phức tạp để hiểu.
* Con trỏ là nguyên nhân chính gây ra [rò rỉ bộ nhớ trong](https://www.geeksforgeeks.org/what-is-memory-leak-how-can-we-avoid/) C.
* Con trỏ chậm hơn so với biến trong C.
* Con trỏ chưa được khởi tạo có thể gây ra lỗi phân đoạn.

**2.6. CẤP PHÁT ĐỘNG**

**2.6.1 Con trỏ vô kiểu.**

Con trỏ void trong C là một kiểu con trỏ không liên kết với bất kỳ kiểu dữ liệu nào. Con trỏ void có thể giữ địa chỉ của bất kỳ kiểu nào và có thể được ép kiểu thành bất kỳ kiểu nào. Chúng cũng được gọi là con trỏ mục đích chung hoặc con trỏ chung.

Trong lập trình C, hàm [malloc()](https://www.tutorialspoint.com/c_standard_library/c_function_malloc.htm) và [calloc()](https://www.tutorialspoint.com/c_standard_library/c_function_calloc.htm) trả về " void \* " hoặc con trỏ chung.

Cú pháp:

void \*ptr;

Trong chương trình ví dụ sau, chúng ta đã khai báo một mảng các con trỏ void và lưu trữ trong đó các con trỏ tới các biến có kiểu dữ liệu khác nhau (int, float và char \*) trong mỗi chỉ số của nó.

void \*arr[3];

int a = 100;

float b = 20.5;

char \*c = "Hello";

arr[0] = &a;

arr[1] = &b;

arr[2] = &c;

printf("Integer: %d\n", \*((int \*)arr[0]));

printf("Float: %f\n", \*((float \*)arr[1]));

printf("String: %s\n", \*((char \*\*)arr[2]));

Khi bạn chạy mã này, nó sẽ tạo ra kết quả sau:

Integer: 100

Float: 20.500000

String: Hello

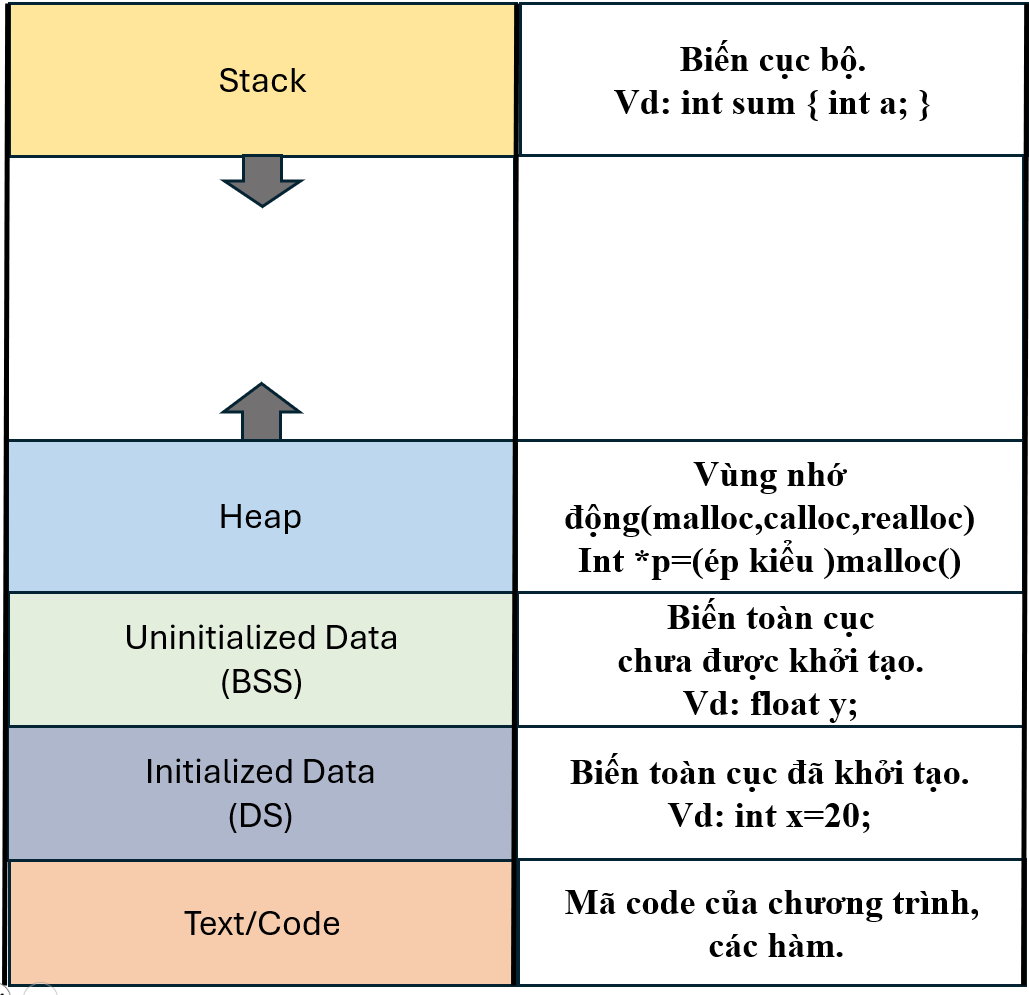
Hàm malloc() có sẵn dưới dạng hàm thư viện trong tệp tiêu đề [stdlib.h](https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm) . Hàm này phân bổ động một khối bộ nhớ trong thời gian chạy của chương trình. Khai báo [biến thông](https://www.tutorialspoint.com/cprogramming/c_variables.htm) thường khiến bộ nhớ được phân bổ tại thời điểm biên dịch.

void \*malloc(size\_t size);

Con trỏ void được sử dụng để triển khai các hàm chung. Các hàm phân bổ động malloc() và calloc() trả về kiểu " void \*" và tính năng này cho phép các hàm này được sử dụng để phân bổ bộ nhớ của bất kỳ kiểu dữ liệu nào.

**2.6.2 Phân vùng bộ nhớ.**

Khi một chương trình C được thực thi, hệ điều hành sẽ cấp phát một vùng bộ nhớ riêng cho chương trình đó. Bộ nhớ này được chia thành nhiều phân vùng khác nhau, mỗi vùng phục vụ cho một mục đích nhất định, bao gồm: Stack, Heap, vùng dữ liệu khởi tạo (Initialized Data), vùng dữ liệu chưa khởi tạo (BSS), và vùng mã lệnh (Text Segment). Trong đó, **Stack** và **Heap** là hai vùng bộ nhớ cực kỳ quan trọng, ảnh hưởng trực tiếp đến quá trình quản lý biến, cấp phát bộ nhớ động, và hiệu suất chương trình.



Hình 2.2 Ảnh memory layout

**3.2.1. Stack.**

Stack là nơi lưu trữ:Các biến cục bộ (local variables),thông tin về lời gọi hàm (function call),

con trỏ khi được khai báo mà chưa dùng malloc hay calloc. Mỗi khi một hàm được gọi, một vùng nhớ tương ứng sẽ được đẩy vào Stack; khi hàm kết thúc, vùng nhớ đó sẽ được tự động giải phóng. Stack hoạt động theo nguyên lý LIFO (Last In, First Out) – vào sau ra trước.

Ví dụ:

int main() {

int \*p;

}

Con trỏ p nằm ở stack, nhưng nó chưa trỏ đến vùng nhớ hợp lệ nào nếu chưa được cấp phát.

Nếu bạn in giá trị \*p lúc này, sẽ lỗi do chưa có vùng nhớ trỏ tới.

**3.2.2. Heap.**

Khi bạn dùng malloc, calloc, realloc → hệ điều hành sẽ cấp phát một vùng nhớ mới trên Heap. Bộ nhớ này tồn tại cho đến khi lập trình viên chủ động gọi hàm free() để giải phóng nó.

Ví dụ:

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

Con trỏ p vẫn nằm trên stack (vì nó là biến cục bộ). Nhưng p trỏ tới một vùng nhớ trên Heap (5 ô nhớ kiểu int). Khi cấp phát xong, bạn có thể gán giá trị: p[0] = 10; , p[1] = 20;,...Quan trọng: Bộ nhớ trên Heap sẽ không tự động giải phóng. Bạn cần dùng free(p) khi không dùng nữa.

**2.3.3. Cấp phát động.**

Trong C, để sử dụng bộ nhớ động (dynamic memory), ta sử dụng các hàm trong thư viện <stdlib.h> như: malloc(), calloc(), realloc(), và free(). Các hàm này cấp phát vùng nhớ trên heap và trả về con trỏ trỏ đến vùng nhớ đó. Con trỏ thường được khai báo trong stack, còn vùng nhớ được cấp phát trên heap.

**2.3.3.1. Hàm malloc().**

Cấp phát một khối bộ nhớ liên tiếp có kích thước size byte trong vùng heap. Có giá trị khởi tạo không xác định (giá trị rác).

Cú pháp:

void\* malloc(size\_t size);

Ví dụ:

int \*p;

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

p là con trỏ khai báo trong stack, còn vùng nhớ được malloc() cấp phát nằm trên heap.

**2.3.3.2. Hàm calloc().**

Cấp phát một vùng nhớ liên tiếp gồm count phần tử, mỗi phần tử có kích thước size byte. Tất cả các byte đều được khởi tạo bằng 0.

Cú pháp:

void\* calloc(size\_t count, size\_t size);

Ví dụ:

int \*p;

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

Cấp phát vùng nhớ đủ chứa 5 số nguyên (int), và tất cả đều được khởi tạo là 0.

**2.3.3.3. Hàm realloc()**

Thay đổi kích thước vùng nhớ đã được cấp phát từ trước. Cách hoạt động nếu con trỏ là NULL: realloc() hoạt động như malloc().Nếu con trỏ không phải NULL: hệ thống sẽ cố gắng mở rộng hoặc thu hẹp vùng nhớ hiện tại.

Cú pháp:

void\* realloc(void \*ptr, size\_t new\_size);

Ví dụ:

int \*p = NULL;

p = (int \*)realloc(p, 5 \* sizeof(int));

Con trỏ p đang là NULL, nên realloc() hoạt động giống như malloc() để cấp phát mới.

**2.3.3.4. Hàm free()**

Giải phóng vùng nhớ đã được cấp phát bằng malloc(), calloc() hoặc realloc().Lưu ý: Sau khi gọi free(), con trỏ vẫn còn tồn tại nhưng không còn trỏ đến vùng nhớ hợp lệ nữa (trỏ rác).

Cú pháp:

free(ptr);

Ví dụ:

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

free(p);

Giải phóng vùng nhớ mà p đang trỏ đến.

**2.7. STRUCT**

**2.7.1 Khái niệm và cú pháp**

Trong ngôn ngữ lập trình C, struct (cấu trúc) là kiểu dữ liệu do người dùng tự định nghĩa, cho phép gom nhiều kiểu dữ liệu khác nhau vào một thực thể duy nhất. Việc khai báo và sử dụng struct giúp tổ chức dữ liệu một cách rõ ràng và khoa học hơn.

struct NhanVien{

int id;

char hoten[30];

} ;

Trong đó “ struct ” là từ khóa để khai báo kiểu dữ liệu struct. “ NhanVien” là tên của struct.

“ int id; ” , “ char hoten; ” được gọi là các thược tính của struct.

**2.7.2 khai báo biến có kiểu struct.**

Khi đã có định nghĩa kiểu struct, ta có thể khai báo biến theo nhiều cách:

Struct nhanvien nv1,nv2;

Struct nhanvien nv1[30]; // mảng

Struct nhanvien \*nv2; // con trỏ

Ta cũng có thể vừa định nghĩa vừa khai báo biến :

Ví dụ: Struct nhanvien{

Int id;

Char hoten[30];

} nv ; // nv ở đây là khai báo biến struct

**2.7.3 Truy xuất thuộc tính trong struct.**

Với biến thông thường, sử dụng dấu “ . ” để truy xuất:

nv.id;  
nv.hoTen;

Với con trỏ đến struct, sử dụng toán tử -> (sau khi đã cấp phát bộ nhớ):

nv = (struct NhanVien\*)malloc(sizeof(struct NhanVien));  
nv->id;  
nv->hoTen;

Cấu trúc cũng có thể chứa các cấu trúc khác, gọi là struct lồng nhau.

Ví dụ:

struct Ngay {

int ngay;

int thang;

int nam;

};

struct SinhVien {

char maSV[10];

char hoTen[30];

struct Ngay ngaySinh; // Struct Ngay nằm trong struct SinhVien

};

Truy xuất thuộc tính struct lồng struct:

sv.ngaySinh.ngay;  
sv.ngaySinh.thang;  
sv.ngaySinh.nam;

**2.7.4 Kích thước của 1 struct**

Xét hai struct có cùng trường dữ liệu nhưng thứ tự khác nhau:

|  |  |
| --- | --- |
| Struct 1{  int id; //4 byte   char \* ptr; //8 byte  char name[2]; //2byte  } employee1; // tổng 24 byte | Struct 2{  char \* ptr; //8 byte  int id; //4 byte  char name[2]; //2byte  } employee2; //16 byte |

Kích thước của struct sẽ được căn chỉnh theo các phần tử có kích thước lớn nhất (kích thước của struct phải là bội số của kích thước lớn nhất).Giữa các phần tử của struct có thể được thêm những padding để đảm bảo rằng mỗi phần tử được đặt tại địa chỉ phù hợp với kích thước của nó

Ví dụ:

Char 1 byte đặt ở : 0x00; 0x01; 0x02;…

Int 4byte đặt ở: 0x00;0x04;0x08,…

Double (8byte) đặt ở: 0x00;0x08;0x16,…

A screen shot of a chart

AI-generated content may be incorrect.Struct 1 – Gây lãng phí bộ nhớ (24 byte)

Hình 2.3. Cấu trúc bộ nhớ của một struct có kích thước 24 byte với padding.

Trường id (4 byte) được đặt tại địa chỉ 0x01. Để đảm bảo ptr (8 byte) bắt đầu tại địa chỉ chia hết cho 8, bộ dịch chèn thêm 4 byte padding từ địa chỉ 0x04 đến 0x08. Sau đó, ptr chiếm từ địa chỉ 0x08 đến 0x016. Trường name[2] chiếm 2 byte tiếp theo (0x17, 0x18). Cuối cùng, 6 byte padding được thêm để làm tròn tổng thể thành 24 byte.

A screenshot of a computer

AI-generated content may be incorrect.Struct 2 – Kích thước 16 byte (tối ưu hơn)

Hình 2.4 Cấu trúc bộ nhớ của một struct có kích thước 16 byte với padding.

Con trỏ *ptr* chiếm 8 byte từ địa chỉ 0x00 đến 0x07. Trường *id* bắt đầu tại địa chỉ 0x08 (đúng chuẩn). Trường *name*[2] bắt đầu tại địa chỉ 0x013. Trình biên dịch chỉ cần thêm 2 byte *padding* ở cuối để làm tròn thành 16 byte.

**2.4.4 các hàm hay sử dụng với struct**

* “ *fgets(bien\_chuoi, kich\_thuoc, stdin)* ” : dùng để nhập chuỗi từ bàn phím, thay cho scanf("%s", ...) khi cần đọc cả câu có dấu cách. “ fgets ” giữ lại ký tự \n nếu người dùng nhấn Enter, nên có thể cần xử lý thêm để loại bỏ.
* “ *fflus(stdin)* ” : dùng để xóa bộ đệm nhập trước khi sử dụng fgets() hoặc getchar() sau scanf(), nhằm tránh việc các ký tự thừa (\n) gây lỗi.
* “ *getchar* ”: dùng để nhận một ký tự từ bàn phím.

* “*strcpy(diem1.ten, "Toán")*”: dùng để gán chuỗi vào biến trong struct, đặc biệt là khi không thể gán trực tiếp như với biến nguyên.
* “*strcmp(sv1.hoten, sv2.hoten)* == 0” : so sánh 2 chuỗi ký tự (ví dụ: so sánh tên sinh viên).
* Đặc biệt khi sử dụng *scanf()* để nhập chuỗi hoặc số, cần chú ý: *scanf()* không loại bỏ ký tự newline (\n) sau khi người dùng nhấn Enter. Nếu nhập chuỗi sau *scanf*(), nên dùng *fflush*(*stdin*) hoặc gọi *getchar*() để loại bỏ ký tự \n trong bộ đệm.

**2.8. LINKLIST**

**2.8.1 Khái niệm**

Linked list là một cấu trúc tuần tự bao gồm một chuỗi các item theo thứ tự tuyến tính được liên kết với nhau.Do đó, ta chỉ có thể truy cập tuần tự vào linked list, không thể thực hiện truy cập ngẫu nhiên.

Linked list là cung cấp cho chúng ta một cấu trúc dữ liệu đơn giản và linh hoạt cho các tập hợp động.

* Các phần tử trong linked list được gọi là các node.
* Mỗi node sẽ chứa một key và một con trỏ tới node kế tiếp của nó, được gọi là next
* Thuộc tính tên là head trỏ tới phần tử đầu tiên của linked list
* A green rectangle with black text

  AI-generated content may be incorrect.Phần tử cuối cùng của linked list có tên là tail

Hình 2.5. Minh họa danh sách liên kết đơn với ba nút và con trỏ head

Một số loại linked list có thể kế tới bao gồm:

* Singly linked list: Duyệt qua các phần tử chỉ có thể thực hiện theo chiểu hướng về phía trước.
* Doubly linked list: Duyệt qua các phần tử có thể thực hiện theo cả chiều tiến và lùi. Các node sẽ bao gồm thêm một con trỏ được gọi là pre, trỏ tới node trước đó.
* Circular linked list: Là một doubly linked list đặc biệt, khi mà con trỏ prev của head trỏ tới tail và con trỏ next của tail trỏ tới head.

Các phép toán trên linked list:

* Tim kiếm: tìm kiếm phần tử đầu tiên với key là k trong một linked list được cho trước được thực hiện đơn giản bằng một quá trình duyệt tuần tự và trả về con trỏ trỏ tới phẩn tử đó.
* Thêm :Để thêm một key vào linked list có sẵn, ta có thể thực hiện theo 3 cách: thêm vào đầu list, thêm vào giữa list hoặc thêm vào cuối của list.
* Xóa: Xóa một phần tử x khỏi một linked list cho trước. Ta không thể xóa một node với chỉ một bước. Việc xóa một node có thể thực hiện theo 3 cách: xóa từ đầu danh sách , xóa từ giữa danh sách hoặc xóa từ cuối danh sách

Ưa điểm:

Tiết kiệm bộ nhớ và cấp phát động: Không như array cần 1 lượng chỉ định ô nhớ trên bộ nhớ ngay khi khởi tạo. Linked list chỉ sử dụng bộ nhớ để lưu trữ khi dữ liệu thực sự được lưu vào linked list. Nó còn có thể lưu các phần tử ở bất cứ đâu được phép trên bộ nhớ mà không cần các ô nhớ liền kề nhau như array

**2.8.2 Một số áp dụng Linked List trong chương trình.**

**2.8.2.1 Xóa sinh viên**

Hàm xoaSinhVien thực hiện chức năng **xóa một sinh viên** khỏi danh sách liên kết đơn dựa trên mã số sinh viên (mssv) được cung cấp. Sau khi tìm thấy sinh viên cần xóa, hàm cập nhật các con trỏ để loại bỏ node khỏi danh sách và giải phóng vùng nhớ tương ứng.

Hình 2.6 Ảnh code hàm xóa sinh viên

A screenshot of a computer code

AI-generated content may be incorrect.Quy trình hoạt động

1. Khởi tạo con trỏ duyệt danh sách

* “ p ” bắt đầu trỏ đến phần tử đầu tiên của danh sách (head).
* “ truoc ” dùng để lưu giữ node đứng trước node hiện tại, ban đầu là NULL.

1. Tìm kiếm phần tử có MSSV trùng khớp

* Sử dụng vòng lặp while để duyệt danh sách đến khi tìm thấy node có mssv khớp hoặc đến cuối danh sách.
* Mỗi lần lặp, truoc được cập nhật bằng p, và p chuyển sang node tiếp theo.

1. Xử lý khi không tìm thấy sinh viên

* Nếu duyệt hết danh sách (p == NULL) mà không có MSSV phù hợp, in thông báo và thoát hàm.

1. Xóa node tương ứng

* Nếu node cần xóa là node đầu tiên (truoc == NULL), cập nhật head sang phần tử tiếp theo.
* Nếu node cần xóa không nằm đầu danh sách, liên kết truoc->next tới p->next, tức là bỏ qua node p.

1. Giải phóng bộ nhớ

* Sử dụng free(p) để giải phóng vùng nhớ đã cấp phát cho node cần xóa, tránh rò rỉ bộ nhớ.

**2.8.2.2 Hàm themSinhVien:**

A screenshot of a computer code

AI-generated content may be incorrect.Hàm themSinhVien có nhiệm vụ chèn một sinh viên mới vào danh sách liên kết đơn sao cho danh sách luôn được duy trì theo **thứ tự tăng dần của mã số sinh viên (MSSV)**. Việc duy trì thứ tự này giúp thuận tiện cho việc tìm kiếm, thống kê và hiển thị dữ liệu.

Hình 2.7 Ảnh code hàm them một sinh viên

Quy trình hoạt động

1. Cấp phát bộ nhớ và khởi tạo node mới

* Sử dụng malloc để cấp phát vùng nhớ cho node mới.
* Dữ liệu từ đối tượng svMoi được sao chép vào node mới.
* Trường next của node được khởi tạo là NULL, đánh dấu node hiện chưa liên kết với node nào khác.

1. Trường hợp chèn vào đầu danh sách

* Nếu danh sách đang rỗng (head == NULL), hoặc MSSV của sinh viên mới nhỏ hơn MSSV của phần tử đầu danh sách (strcmp(...) < 0) → node mới sẽ được chèn vào đầu danh sách.
* Con trỏ head sẽ được cập nhật để trỏ đến node mới.

1. Trường hợp chèn vào giữa hoặc cuối danh sách

* Duyệt danh sách từ đầu đến khi tìm được vị trí thích hợp mà MSSV tiếp theo lớn hơn hoặc bằng MSSV mới.
* Node mới được chèn vào sau node p, thông qua việc cập nhật các con trỏ liên kết.

# **Chương 3: CHƯƠNG TRÌNH**

## **3.1 TỔNG QUAN ỨNG DỤNG.**

### **3.1.1. Mục tiêu**

Mục tiêu của đề tài là xây dựng một chương trình quản lý sinh viên đơn giản bằng ngôn ngữ lập trình C, sử dụng danh sách liên kết đơn làm cấu trúc dữ liệu chính. Chương trình hỗ trợ các thao tác quản lý cơ bản như thêm, sửa, xóa, tìm kiếm, sắp xếp và thống kê thông tin sinh viên.

### **3.2.2. Cấu trúc dữ liệu**

Chương trình sử dụng cấu trúc dữ liệu danh sách liên kết đơn (singly linked list), trong đó mỗi node đại diện cho một sinh viên, bao gồm thông tin như họ tên, MSSV, lớp, GPA và con trỏ trỏ tới sinh viên tiếp theo.

Khai báo cấu trúc sinh viên

Vd:

typedef struct SinhVien {

char hoTen[50];

char mssv[20];

char lopHoc[20];

float GPA;

struct SinhVien\* next;

} SinhVien;

### **3.3.3. Các thao tác chính**

Các chức năng cơ bản mà chương trình cung cấp:

Thêm sinh viên mới

Cập nhật thông tin sinh viên theo MSSV

Xóa sinh viên khỏi danh sách

Tìm kiếm sinh viên theo MSSV, tên, hoặc khoảng điểm

Sắp xếp danh sách theo tên hoặc điểm GPA

Thống kê GPA, điểm cao nhất

Lưu và đọc danh sách từ tệp văn bản

### **3.3.4. Quản lý bộ nhớ**

Dữ liệu được quản lý thông qua cấp phát động với malloc và giải phóng bằng free.

Mỗi thao tác thêm sinh viên cấp phát vùng nhớ mới cho một node.

Khi xóa sinh viên hoặc thoát chương trình, vùng nhớ được giải phóng nhằm tránh rò rỉ bộ nhớ (memory leak).

### **3.3.5. Giao diện người dùng**

A screen shot of a computer program

AI-generated content may be incorrect.Giao diện người dùng được cung cấp thông qua menu, cho phép người dùng thực hiện các thao tác có sẵn như:

Hình 3.1Ảnh minh họa các chức năng

## **3.2. THIẾT KẾ ỨNG DỤNG**

### **3.2.1. Cấu trúc chương trình**

Chương trình được viết bằng ngôn ngữ C, biên dịch trên phần mềm Visual Studio 2022.  
Một tệp dữ liệu có tên DANHSACHNHOM9.txt được sử dụng để lưu trữ và nạp thông tin.

Để dễ dàng tổ chức và bảo trì, chương trình được chia thành ba tệp chính:

* + lib.h: Khai báo cấu trúc SinhVien, biến toàn cục head, và các nguyên mẫu của hàm.
  + lib.cpp: Định nghĩa toàn bộ các hàm xử lý như: thêm, xóa, sửa, tìm kiếm, sắp xếp, thống kê, v.v.
  + main.cpp: Quản lý luồng xử lý chính của chương trình, đồng thời hiện thị giao diện menu cho người dùng thao tác.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 3.3 Ảnh code file lib.cpp

Hình 3.2 Ảnh code file lib.h

Hình 3 Ảnh code file lib.cpp

A screenshot of a computer program

AI-generated content may be incorrect.

Hình 3. 4 Ảnh code file main.cpp

### **3.2.2. Các hàm chính**

Các hàm xử lý chính trong chương trình bao gồm:

1. void them\_sinh\_vien(SinhVien sv)
2. void xoa\_sinh\_vien(char mssv[])
3. void sua\_thong\_tin(char mssv[])
4. SinhVien\* tim\_theo\_mssv(char mssv[])
5. void sap\_xep\_theo\_ten(SinhVien\* head)
6. void ghi\_file(const char\* tenFile)
7. voi doc\_file(const char\* tenFile)
8. void thong\_ke(SinhVien\* danh\_sach)

### **3.3.3. Các tính năng nâng cao**

1. Tìm theo khoảng điểm GPA
2. Thống kê GPA cao nhất
3. Tách tên riêng để sắp xếp chính xác hơn
4. Sắp xếp linh hoạt theo tên hoặc GPA

**3.2.THỰC NGHIỆM**

Dưới đây là giao diện khi chạy chương trình. A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 3.5Ảnh demo giao diện khi chạy chương trình

Tính năng 1: In danh sách sinh viên. A black screen with white text

AI-generated content may be incorrect.

Hình3.6 Ảnh demo chức năng in danh sách

Tính năng 2 : Thêm thông tin một sinh viên mới. A screenshot of a video game

AI-generated content may be incorrect.

Hình 3.7 Ảnh demo chức năng thêm sinh viên và kết quả

Tính năng 3 : Sửa thông tin của một sinh viên. A screenshot of a computer

AI-generated content may be incorrect.

Hình 3.8 Ảnh demo chức năng sửa thông tin và kết quả

Tính năng 4 : Xóa 1 học sinh thông qua mã số sinh viên. A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 3.9 Ảnh demo chức năng xóa 1 sinh viên và kết quả

Tính năng 5 : Chương trình hiển thi menu con tim kiếm. A screenshot of a computer menu

AI-generated content may be incorrect.

Hình 3.10 Ảnh demo menu con tìm kiếm và kết quả 3 chức năng tìm kiếm

Tính năng 6: Chương trình hiển thị menu con sắp xếp. A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 3.11 Ảnh demo chức menu con sắp xếp và kêt quả của từng chức năng sắp xếp

Tính năng 7 : Chương trình sẽ thống kế. A screen shot of a computer

AI-generated content may be incorrect.

Hình 3.12 Ảnh demo chức năng thống kê và kết quả

Tính năng 8: Chương trình sẽ cập nhật những gì đã thao tác và lưu vào file ban đầu. A screenshot of a computer program

AI-generated content may be incorrect.

Hình 3.13 Ảnh demo chức năng sao lưu file và kết quả

**CHƯƠNG IV: KẾT QUẢ VÀ KIẾN NGHỊ**

**4.1. Kết quả đạt được**

Sau quá trình nghiên cứu và triển khai, chương trình quản lý sinh viên đã được hoàn thiện với đầy đủ các chức năng cơ bản như thêm mới sinh viên, chỉnh sửa thông tin, xóa sinh viên theo mã số, tìm kiếm theo tên hoặc khoảng điểm, sắp xếp danh sách theo tên hoặc GPA, và thống kê kết quả học tập.

Chương trình hoạt động ổn định, xử lý đúng dữ liệu, hiển thị thông tin rõ ràng và hỗ trợ lưu trữ dữ liệu bằng tệp văn bản. Việc tổ chức mã nguồn thành ba phần riêng biệt gồm lib.h, lib.cpp và main.cpp giúp chương trình dễ bảo trì, có cấu trúc rõ ràng và thuận tiện cho việc mở rộng sau này.

Việc áp dụng danh sách liên kết đơn cho phép xử lý linh hoạt danh sách sinh viên, tránh lãng phí bộ nhớ và giúp người thực hiện củng cố kiến thức về cấp phát động, thao tác con trỏ, cấu trúc dữ liệu và quản lý bộ nhớ.

Thông qua đồ án này, người thực hiện đã nâng cao khả năng lập trình bằng ngôn ngữ C, rèn luyện kỹ năng thiết kế chương trình, tổ chức mã nguồn theo mô-đun và tư duy giải quyết vấn đề.

**4.2. Kiến nghị**

Mặc dù chương trình đã hoàn thành đúng yêu cầu và hoạt động ổn định, vẫn còn một số mặt có thể cải tiến trong tương lai nhằm nâng cao trải nghiệm người dùng và khả năng mở rộng ứng dụng.

Trước hết, nên bổ sung kiểm tra lỗi đầu vào nhằm tránh tình trạng nhập sai định dạng dữ liệu, đặc biệt đối với GPA hoặc chuỗi ký tự chứa ký tự không hợp lệ. Ngoài ra, cần cải thiện giao diện bằng cách làm rõ các hướng dẫn nhập, trình bày thông tin rõ ràng hơn, hoặc có thể phát triển một phiên bản giao diện đồ họa (GUI) để tăng tính trực quan.

Về lưu trữ dữ liệu, chương trình hiện đang sử dụng tệp văn bản đơn giản, do đó có thể nâng cấp bằng cách sử dụng tệp nhị phân hoặc kết nối với cơ sở dữ liệu để đảm bảo an toàn và hiệu quả truy xuất cao hơn.

Cuối cùng, chương trình có thể được phát triển thêm các chức năng nâng cao như lọc sinh viên theo lớp, khóa học, học kỳ, thống kê nâng cao theo nhóm điểm, hoặc kết xuất báo cáo sang định dạng PDF nhằm tăng tính thực tiễn trong môi trường giáo dục hoặc quản trị đào tạo.

**TÀI LIỆU THAM KHẢO**

1. Kernighan, B.W. & Ritchie, D.M. (1988), *The C Programming Language*, 2nd ed., Prentice Hall, Englewood Cliffs, New Jersey.
2. [https://blog.28tech.com.vn](https://blog.28tech.com.vn/c-malloc-calloc-free-va-realloc), “*Hướng dẫn lập trình C cơ bản*, truy cập ngày 07 tháng 5 năm 2025”
3. [https://geeksforgeeks.org](https://www.geeksforgeeks.org/c-pointers/), “*C Programming Language – Basics and Examples*, truy cập ngày 07 tháng 5 năm 2025.”
4. [https://www.tutorialspoint.com](https://www.tutorialspoint.com/cprogramming/c_pointers.htm?utm_source=chatgpt.com), “*C Programming Tutorial*, truy cập ngày 07 tháng 5 năm 2025.”

**A qr code with text

AI-generated content may be incorrect.**