**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 C VÀ**

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

Giảng viên hướng dẫn: TRẦN PHONG NHÃ

Sinh viên thực hiện: LÊ NHỰT TRƯỜNG

PHẠM QUỐC KHÁNH

Lớp : CQ.CNTT

Khoá :K.65

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 C VÀ**

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

Giảng viên hướng dẫn: TRẦN PHONG NHÃ

Sinh viên thực hiện: LÊ NHỰT TRƯỜNG

PHẠM QUỐC KHÁNH

Lớp : CQ.CNTT

Khoá :K.65

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](#_Toc197462029)

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN ii](#_Toc197462030)

[MỤC LỤC iii](#_Toc197462031)

[DANH MỤC CHỮ VIẾT TẮT v](#_Toc197462032)

[BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ (size 15, bold) v](#_Toc197462033)

**PHẦN A. LÝ THUYẾT**

[CHƯƠNG 1. MỞ ĐẦU 1](#_Toc197462034)

[1.1 Tông quan đề tài. 1](#_Toc197462035)

[1.2 Mục tiêu nghiên cứu 1](#_Toc197462036)

[1.3 Phạm vi và đối tượng nghiên cứu 1](#_Toc197462037)

[1.4 Phương pháp thực hiện 1](#_Toc197462038)

[1.5 Cấu trúc báo cáo 1](#_Toc197462039)

[CHƯƠNG 2. CỞ SỞ LÝ THUYẾT 2](#_Toc197462040)

2.1 Hàm

[2.2 Con Trỏ 2](#_Toc197462041)

[**2.1.1 Khái niệm. 2**](#_Toc197462042)

[**2.1.2 Khai báo con trỏ. 2**](#_Toc197462043)

[**2.1.3 Khởi tạo con trỏ. 2**](#_Toc197462044)

[**2.1.4 Giải tham chiếu con trỏ. 2**](#_Toc197462045)

[**2.1.5 Con trỏ và mảng. 3**](#_Toc197462046)

[**2.1.5 Số học con trỏ. 5**](#_Toc197462047)

2.3 Con trỏ mảng

[2.4 Con trỏ hàm. 6](#_Toc197462048)

[**2.4.1 Khái niêm và cú pháp. 6**](#_Toc197462049)

[**2.4.2 Truyền đối số cho hàm 7**](#_Toc197462050)

[**2.4.3 Ưu điểm của truyền con trỏ vào hàm 7**](#_Toc197462051)

[**2.4.4 Các tình huống sử dụng con trỏ trong truyền tham số. 8**](#_Toc197462052)

[2.5 Mảng con trỏ. 8](#_Toc197462053)

[2.6. CẤP PHÁT ĐỘNG 11](#_Toc197462056)

[**2.6.3.1. Hàm malloc(). 13**](#_Toc197462060)

[**2.6.3.2. Hàm calloc(). 13**](#_Toc197462061)

[**2.6.3.3 Hàm realloc() 13**](#_Toc197462062)

[**2.6.3.4. Hàm free() 14**](#_Toc197462063)

[2.7 Xử lý tệp 15](#_Toc197462064)

2.8 Kiểu cấu trúc

[2.9 LINKLIST 18](#_Toc197462069)

[**2.9.1 Khái niệm 18**](#_Toc197462070)

[**2.9.2.Một số ví dụ danh sách liên kết. 18**](#_Toc197462071)

[Chương 3: CHƯƠNG TRÌNH 21](#_Toc197462074)

[3.1 TỔNG QUAN ỨNG DỤNG. 21](#_Toc197462075)

[**3.1.1. Mục tiêu 21**](#_Toc197462076)

[**3.2.2. Cấu trúc dữ liệu 21**](#_Toc197462077)

[**3.3.3. Các thao tác chính 21**](#_Toc197462078)

[**3.3.4. Quản lý bộ nhớ 22**](#_Toc197462079)

[**3.3.5. Giao diện người dùng 22**](#_Toc197462080)

[3.2. THIẾT KẾ ỨNG DỤNG 23](#_Toc197462081)

[**3.2.1. Cấu trúc chương trình 23**](#_Toc197462082)

[**3.2.2. Các hàm chính 24**](#_Toc197462083)

[**3.3.3. Các tính năng nâng cao 25**](#_Toc197462084)

[3.2.THỰC NGHIỆM 25](#_Toc197462085)

[CHƯƠNG 4: KẾT QUẢ VÀ KIẾN NGHỊ 26](#_Toc197462086)

[4.1. Kết quả đạt được 26](#_Toc197462087)

[4.2. Kiến nghị 26](#_Toc197462088)

**PHẦN B. THỰC HÀNH**

# **BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ**

[Hình 2.1 Ảnh minh họa con trỏ trỏ con trỏ 10](#_Toc197518275)

[Hình 2.2 Ảnh memory layout 13](#_Toc197518276)

[Hình 2.3. Cấu trúc bộ nhớ của một struct có kích thước 24 byte với padding. 17](#_Toc197518277)

[Hình 2.4 Cấu trúc bộ nhớ của một struct có kích thước 16 byte với padding. 18](#_Toc197518278)

[Hình 2.5. Minh họa danh sách liên kết đơn với ba nút và con trỏ head 19](#_Toc197518279)

[Hình 2.6 Ảnh code hàm xóa sinh viên 20](#_Toc197518280)

[Hình 2.7 Ảnh code hàm them một sinh viên 21](#_Toc197518281)

[Hình 3.1Ảnh minh họa các chức năng 24](#_Toc197518282)

[Hình 3.3 Ảnh code file lib.cpp 25](#_Toc197518283)

[Hình 3.2 Ảnh code file lib.h 25](#_Toc197518284)

[Hình 3. 4 Ảnh code file main.cpp 26](#_Toc197518285)

[Hình 3.5Ảnh demo giao diện khi chạy chương trình 27](#_Toc197518286)

[Hình3. 6 Ảnh demo chức năng in danh sách 28](#_Toc197518287)

[Hình 3.7 Ảnh demo chức năng thêm sinh viên và kết quả 28](#_Toc197518288)

[Hình 3.8 Ảnh demo chức năng sửa thông tin và kết quả 29](#_Toc197518289)

[Hình 3.9 Ảnh demo chức năng xóa 1 sinh viên và kết quả 29](#_Toc197518290)

[Hình 3.10 Ảnh demo menu con tìm kiếm và kết quả 3 chức năng tìm kiếm 30](#_Toc197518291)

[Hình 3.11 Ảnh demo chức menu con sắp xếp và kêt quả của từng chức năng sắp xếp 30](#_Toc197518292)

[Hình 3.12 Ảnh demo chức năng thống kê và kết quả 31](#_Toc197518293)

[Hình 3.13 Ảnh demo chức năng sao lưu file và kết quả 31](#_Toc197518294)

# **PHẦN A. LÝ THUYẾT**

# **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 dãy các lệnh nhằm thực hiện một công việc nào đó, thường được sử dụng nhiều lần

Ví dụ:

Việc tính sin, cos, tan,… trong toán học

Xây dựng các hàm tính sin, cos,…

Một chương trình C là 1 dãy các hàm, trong đó có một hàm chính, được đặt tên là **main()**

Ví dụ :

#include <stdio.h>

float max2so(float a, float b); /\* Nguyên mẫu của hàm \*/

main() /\* bắt đầu hàm chính \*/

{

float x, y;

printf(“Nhập vào 2 số: ”);

scanf(“%f%f”, &x, &y);

printf(“Giá trị lớn nhất của %f và %f là %f\n”, x, y, max2so(x, y));

} /\* kết thúc hàm main \*/

/\* định nghĩa hàm max2so \*/

float max2so(float a, float b){

float max; /\* khai báo biến cục bộ \*/

max = a > b ? a : b;

return max;

}

**2.1.2 Định nghĩa hàm**

Cú pháp

kiểu\_dữ\_liệu\_trả\_về tên\_hàm ([khai\_báo\_các\_tham\_số]){

các khai báo dùng riêng bên trong hàm nếu có;

các lệnh bên trong hàm;

[return giá\_trị\_trả\_về;]

}

Định nghĩa hàm có thể đặt trước hoặc sau hàm main()

Nếu định nghĩa hàm đặt sau hàm main thì phải khai báo nguyên mẫu hàm ở đầu chương trình.

Nên định nghĩa hàm sau hàm main và khai báo nguyên mẫu hàm

Kiểu dữ liệu trả về của hàm và kiểu dữ liệu tham số là kiểu dữ liệu chuẩn hoặc do người lập trình định nghĩa

Tên hàm và tên tham số đặt theo quy tắc tên biến

Câu lệnh return là tùy chọn

Nếu hàm không trả về giá trị, thì không cần có lệnh return

Nếu hàm trả về giá trị thì bắt buộc phải có lệnh return, trong trường

hợp này giá trị trả về phải có cùng kiểu với kiểu dữ liệu trả về của hàm

Nếu hàm không trả về giá trị thì khai báo kiểu trả về của hàm là void

Nếu hàm không có tham số hình thức có thể sử dụng từ khóa void, hoặc không khai báo gì cả.

Lưu ý:

Không cho phép định nghĩa một hàm bên trong hàm khác

Các tham số hình thức và các biến định nghĩa bên trong hàm (biến

cục bộ) chỉ được sử dụng bên trong hàm đó

**2.1.3 Lời gọi hàm.**

Hàm được sử dụng thông qua lời gọi hàm

Cú pháp: tên\_hàm ([danh sách các tham số thực]);

Cần phân biệt

Tham số hình thức hay đối: xuất hiện trong định nghĩa hàm

Tham số thực: xuất hiện trong lời gọi hàm

Ví dụ

max2so(12, 341);

Lưu ý

Số tham số thực phải bằng số tham số hình thức

Kiểu các tham số thực phải phù hợp với kiểu của các tham số hình thức

Ví dụ: viết hàm tính n!

#include <stdio.h>

long giai\_thua(int n); /\* nguyên mẫu hàm \*/

main(){

int n;

long gt;

printf("\n n = "); scanf("%d", &n);

gt = giai\_thua(n); /\* gọi hàm tính giai thừa \*/

printf("\n n! = %ld\n", gt); /\* In ra kết quả \*/

}

long giai\_thua(int n) {

int i;

long gt = 1;

if (n < 0) gt = 0;

else

for (i=2; i <=n; i++) gt = gt \* i;

return (gt);

}

**2.1.4 Biến toàn cục, biến cục bộ.**

Biến toàn cục: được khai báo bên ngoài thân hàm,thường ở đầu chương trình

Biến cục bộ: được khai báo bên trong thân hàm

Phạm vi hoạt động

Biến toàn cục được sử dụng kể từ vị trí khai báo đến cuối chương trình

Biến cục bộ chỉ được sử dụng bên trong hàm đó

Thời gian sống

Biến toàn cục kết thúc thời gian sống khi chương trình kết thúc

Sau khi hàm kết thúc hoạt động thì các tham số hình thức và ác biến cục bộ cũng kết thúc thời gian sống của chúng

Ví dụ

#include <stdio.h>

int i; /\* Biến toàn cục \*/

void vi\_du(void);

main(){

for (i=1; i <=5; i++)

vi\_du();

}

void vi\_du(void){

int m = 3; /\* Biến cục bộ \*/

m++;

printf(" %d %d\n", i, m);

}

Lưu ý :

Biến toàn cục được sử dụng trong khắp chương trình

Việc thay đổi tùy tiện giá trị của biến toàn cục sẽ rất

khó kiểm soát chương trình

Dễ sinh lỗi

Hạn chế sử dụng biến toàn cục

Ví dụ về phạm vi hoạt động biến

#include <stdio.h>

int a = 5;

/\* Biến toàn cục \*/

void ham\_vi\_du(){

int cuc\_bo = 1;

/\* Biến cục bộ \*/

int a = 10;

/\* Biến cục bộ \*/

printf(“Biến cục bộ trong hàm ví dụ:\n cuc\_bo = %d\n a = %d\n”,

cuc\_bo, a);

}

main(){

int cuc\_bo = 100;

/\* Biến cục bộ \*/

ham\_vi\_du();

/\* Gọi hàm ví dụ \*/

printf(“Biến cục bộ trong hàm main:\n cuc\_bo = %d\n”, cuc\_bo);

printf(“Biến toàn cục:\n a = %d\n”, a);

}

Biến cục bộ được chia làm hai loại

Biến cục bộ động

Biến được cấp phát bộ nhớ tự động mỗi khi có lời gọi hàm

Biến cục bộ động không lưu giữ giá trị mỗi khi hàm kết thúc (tức bị giải phóng khỏi bộ nhớ)

Biến cục bộ tĩnh

Biến cục tĩnh được khai báo bên trong thân hàm nhưng vẫn

tồn tại ngay cả khi hàm đã kết thúc hoạt động

Biến cục bộ tĩnh được khai báo với từ khóa static

Ví dụ

#include <stdio.h>

void vi\_du(void);

main(){

int n;

for (n=1; n<=5; n++)

vi\_du();

}

void vi\_du(void){

static int i;

i++;

printf(" Goi lan thu %d\n", i);

}

Sự giống nhau giữa biến cục bộ tĩnh và biến toàn cục

Cùng đều tồn tại trong suốt thời gian chương trình hoạt động

Sự khác nhau giữa biến cục bộ tĩnh và biến toàn cục

Biến toàn cục được sử dụng kể từ vị trí nó khai báo đến cuối chương

trình

Biến cục bộ tĩnh chỉ được sử dụng trong thân hàm nó được khai báo

**2.2 Con trỏ**

**2.2.1 Giới thiệu về con trỏ**

Con trỏ là một biến đặc biệt dùng để lưu trữ địa chỉ của một biến khác trong bộ nhớ máy tính. Thay vì chứa giá trị trực tiếp, con trỏ "trỏ tới" vị trí bộ nhớ nơi giá trị đó được lưu trữ.

**Mối quan hệ với kiểu dữ liệu:** Mỗi con trỏ được liên kết với một kiểu dữ liệu cụ thể. Ví dụ, một con trỏ kiểu int \* chỉ có thể lưu trữ địa chỉ của các biến kiểu int, float \* lưu trữ địa chỉ của biến kiểu float, và cứ như vậy. Điều này giúp trình biên dịch kiểm tra tính hợp lệ của các thao tác trên con trỏ.

**Khai báo con trỏ:** Để khai báo một biến con trỏ, ta sử dụng cú pháp sau:

type \*tên\_con\_trỏ;

Trong đó:

type là kiểu dữ liệu của biến mà con trỏ sẽ trỏ tới (ví dụ: int, float, char, double).

\* là toán tử khai báo con trỏ.

tên\_con\_trỏ là tên của biến con trỏ.

**Ví dụ khai báo:**

int x, y, \*px, \*c; // Khai báo hai biến int (x, y) và hai con trỏ int (\*px, \*c)float \*t, \*d; // Khai báo hai con trỏ float (\*t, \*d)

**Gán địa chỉ cho con trỏ:** Để gán địa chỉ của một biến cho con trỏ, ta sử dụng toán tử & (toán tử lấy địa chỉ).

int bien\_so = 10;int \*ptr = &bien\_so; // ptr bây giờ chứa địa chỉ của bien\_so

**Ví dụ cụ thể:**

int y = 20;int x = 5;int \*c = &y; // Con trỏ c chứa địa chỉ của biến yint \*px = &x; // Con trỏ px chứa địa chỉ của biến x

**Lưu ý về kiểu dữ liệu:** Con trỏ phải có kiểu tương ứng với kiểu của biến mà nó trỏ tới. Việc gán địa chỉ của một biến kiểu này cho con trỏ kiểu khác sẽ gây ra lỗi biên dịch hoặc hành vi không xác định. Ví dụ:

float bien\_f = 3.14;float \*ptr\_f = &bien\_f; // Đúng// float \*ptr\_f\_sai = &y; // Sai: cố gắng gán địa chỉ int cho con trỏ float

**"Con trỏ trỏ tới biến":** Khi một con trỏ chứa địa chỉ của một biến, ta nói rằng con trỏ đó "trỏ tới" biến đó. Trong ví dụ trên, c trỏ tới y, và px trỏ tới x.

**2.2.2 Quy tắc sử dụng con trỏ trong biểu thức.**

Ta có thể sử dụng tên con trỏ hoặc dạng khai báo của nó (khi thêm toán tử \*) trong các biểu thức.

**Sử dụng tên con trỏ:** Bản thân tên con trỏ là một biến, và giá trị của nó là địa chỉ bộ nhớ mà nó đang giữ.

Khi tên con trỏ xuất hiện trong một biểu thức, giá trị địa chỉ của nó sẽ được sử dụng.

Khi tên con trỏ đứng ở bên trái của toán tử gán (=), giá trị ở bên phải (phải là một địa chỉ) sẽ được gán cho con trỏ, làm thay đổi địa chỉ mà nó đang trỏ tới.

<!-- end list -->

float a = 1.5;float \*p = &a;float \*q;

q = p; // Bây giờ q cũng chứa địa chỉ của a, cả p và q đều trỏ tới a// p = &some\_other\_float\_variable; // p bây giờ trỏ tới một biến float khác

**Dịch chuyển con trỏ (trong mảng):** Nếu một con trỏ trỏ tới một phần tử của mảng, bạn có thể sử dụng các toán tử số học (++, --, +, -) để di chuyển con trỏ đến các phần tử lân cận trong bộ nhớ. Ví dụ, nếu p trỏ tới a[i], thì ++p sẽ làm cho p trỏ tới a[i+1] (với điều kiện p là con trỏ của kiểu dữ liệu có kích thước xác định).

**Sử dụng dạng khai báo của con trỏ (**\*tên\_con\_trỏ**):** Khi thêm toán tử \* trước tên con trỏ, ta đang truy cập vào **giá trị** được lưu trữ tại địa chỉ mà con trỏ đang giữ. Toán tử \* được gọi là **toán tử giải tham chiếu (dereference operator)**.

**Nguyên lý tương đương:** Nếu con trỏ px trỏ tới biến x, thì x và \*px là hoàn toàn tương đương trong mọi ngữ cảnh. Cả hai đều đại diện cho giá trị của biến x.

<!-- end list -->

float x = 10.0;float y;float z = 5.0;float \*px = &x;float \*py = &y;

y = 3 \* x + z; // Tương đương với: y = 3 \* 10.0 + 5.0; (y = 35.0)

\*py = 3 \* x + z; // Tương đương với: y = 3 \* 10.0 + 5.0; (y = 35.0)

\*py = 3 \* (\*px) + z; // Tương đương với: y = 3 \* 10.0 + 5.0; (y = 35.0)

**Thay đổi giá trị thông qua con trỏ:** Vì \*px tương đương với x, ta có thể sử dụng nó ở bên trái của toán tử gán để thay đổi giá trị của biến mà con trỏ trỏ tới.

<!-- end list -->

int num = 100;int \*ptr\_num = &num;printf("Giá trị ban đầu của num: %d\n", num); // In ra 100

\*ptr\_num = 200; // Thay đổi giá trị của num thành 200 thông qua con trỏprintf("Giá trị sau khi thay đổi: %d\n", num); // In ra 200

**2.2.3 Quy tắc về kiểu giá trị trong khai báo.**

Khi bạn khai báo nhiều biến, mảng và con trỏ trên cùng một dòng với một kiểu dữ liệu chung, thì mọi thành phần trong khai báo đó đều có kiểu giá trị đã chỉ định.

**Ví dụ:**

float a, b[7], \*px;

Trong khai báo này:

a là một biến kiểu float. Khi a xuất hiện trong biểu thức, nó sẽ mang giá trị kiểu float.

b là một mảng gồm 7 phần tử kiểu float. Khi b[i] (với i từ 0 đến 6) xuất hiện trong biểu thức, nó sẽ mang giá trị kiểu float.

px là một con trỏ kiểu float. Khi \*px (giải tham chiếu px) xuất hiện trong biểu thức, nó sẽ mang giá trị kiểu float (giá trị của biến mà px đang trỏ tới).

**Quy tắc tổng quát:** Mọi định danh (biến, phần tử mảng, kết quả giải tham chiếu con trỏ) trong cùng một khai báo đều có cùng một kiểu dữ liệu cơ bản.

**2.2.4 Hàm có đối con trỏ**

Một trong những ứng dụng quan trọng của con trỏ là khi chúng được sử dụng làm đối số (tham số) của hàm.

**Truyền địa chỉ cho hàm:** Nếu một đối số của hàm được khai báo là một con trỏ (ví dụ: int \*ptr), thì khi gọi hàm, bạn phải truyền địa chỉ của một biến có kiểu tương ứng (ví dụ: &bien\_int) làm tham số thực tế.

**Cho phép hàm thay đổi giá trị biến bên ngoài:** Khi một hàm nhận một con trỏ làm đối số, nó có thể truy cập và thay đổi trực tiếp giá trị của biến mà con trỏ đó trỏ tới trong bộ nhớ của nơi gọi hàm. Điều này khác với việc truyền giá trị thông thường, nơi hàm chỉ làm việc với bản sao của biến.

**Ví dụ: Hàm hoán vị (swap) hai số float:**

#include <stdio.h>

void hoan\_vi(float \*px, float \*py) {

float temp;

temp = \*px; // Lưu giá trị tại địa chỉ mà px trỏ tới vào temp

\*px = \*py; // Gán giá trị tại địa chỉ mà py trỏ tới cho địa chỉ mà px trỏ tới

\*py = temp; // Gán giá trị đã lưu trong temp cho địa chỉ mà py trỏ tới

}

int main() {

float a = 7.6;

float b = 13.5;

printf("Trước khi hoán vị: a = %.2f, b = %.2f\n", a, b);

hoan\_vi(&a, &b); // Truyền địa chỉ của a và b cho hàm hoan\_vi

printf("Sau khi hoán vị: a = %.2f, b = %.2f\n", a, b);

return 0;

}

Trong ví dụ này, hàm hoan\_vi nhận địa chỉ của hai biến float (px và py). Bằng cách sử dụng toán tử giải tham chiếu (\*), hàm có thể truy cập và thay đổi giá trị của các biến a và b trong hàm main

**2.2.5 Khi nao sử dụng đối con trỏ**

Trong thiết kế hàm, ta có thể phân loại đối số thành hai loại chính:

**Đối vào (Input arguments):** Các đối số này được sử dụng để truyền dữ liệu vào hàm. Hàm sử dụng các giá trị này để thực hiện các tính toán hoặc thao tác. Thông thường, đối vào có thể được truyền theo giá trị.

**Đối ra (Output arguments):** Các đối số này được sử dụng để truyền kết quả ra khỏi hàm. Hàm sẽ gán các giá trị kết quả cho các biến được chỉ định bởi đối số này. **Đối ra thường phải là con trỏ.**

**Ví dụ: Hàm giải phương trình bậc hai:**

#include <stdio.h>#include <math.h>

int ptb2(float a, float b, float c, float \*x1, float \*x2) {

float delta;

if (a == 0) return 0; // a = 0: không phải phương trình bậc hai

delta = b \* b - 4 \* a \* c;

if (delta < 0) return -1; // Delta < 0: vô nghiệm

\*x1 = (-b - sqrt(delta)) / (2 \* a); // Gán nghiệm thứ nhất thông qua con trỏ

\*x2 = (-b + sqrt(delta)) / (2 \* a); // Gán nghiệm thứ hai thông qua con trỏ

return 1; // Delta >= 0: có nghiệm

}

int main() {

float a, b, c, x1, x2;

int s;

printf("Nhap a, b, c: ");

scanf("%f %f %f", &a, &b, &c);

s = ptb2(a, b, c, &x1, &x2);

if (s == 0) {

printf("a = 0\n");

} else if (s == -1) {

printf("Delta < 0: Phuong trinh vo nghiem\n");

} else {

printf("x1 = %.2f, x2 = %.2f\n", x1, x2);

}

return 0;

}

Trong hàm ptb2, a, b, c là các đối vào (truyền giá trị). x1 và x2 là các đối ra (truyền địa chỉ) để hàm có thể trả về hai nghiệm của phương trình. Giá trị trả về của hàm (s) cho biết trạng thái giải phương trình.

**2.3 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.

**2.3.1 Con trỏ mảng 1 chiều**

Mảng và địa chỉ

Toán tử & dùng để lấy địa chỉ một biến

Toán tử & cũng được dùng để lấy địa chỉ của một phần tử mảng

Các phần tử trong mảng được bố trí các ô nhớ liên tiếp nhau trên bộ nhớ

=> Nếu biết được địa chỉ phần tử thứ i sẽ xác định được địa chỉ phần tử

thứ i+1

Địa chỉ phần tử đầu tiên là địa chỉ của mảng

Tên mảng mang địa chỉ của mảng đó

Truy cập các phần tử mảng theo dạng con trỏ

Ta có các quy tắc sau:

&Tên\_mảng[0] ~ Tên\_mảng

&Tên\_mảng[vị\_trí] ~ Tên\_mảng + vị\_trí

Tên\_mảng[vị\_trí] ~ \*(Tên\_mảng + vị\_trí)

Ví dụ:

float a[100];

float \*pa;

Các cách viết sau là tương đương:

a ⇔ &a[0]

a + i ⇔ &a[i]

\*(a + i) ⇔ a[i]

Các phép gán hợp lệ

pa = a;

pa = &a[0];

Mảng là tham số của hàm

Khi sử dụng mảng là tham số của hàm, ta có thể khai báo, chẳng hạn:

int a[]

Hoặc

int \*a

Như thế, hai cách khai báo sau là tương đương:

f(int a[]) { … }

f(int \*a) { … }

Khi sử dụng, có thể gọi:

f(a); Hoặc f(&a[0]);

Ví dụ: Hàm đọc vào một mảng số nguyên n phần tử

void nhap\_mang(int \*x, int n){

int i;

/\* Đọc các giá trị mảng \*/

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

printf(" x[%d]= ", i+1);

scanf("%d", x+i); // x+i~ &x[i]

}

}

Ví dụ: Hàm in ra mảng số nguyên n phần tử

void xuat\_mang(int \*x, int n){

int i;

/\* In các giá trị mảng \*/

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

printf("%5d", \*(x+i));// \*(x+i) ~ x[i]

}

**2.3.2 Con trỏ mảng nhiều chiều.**

Ví dụ, khai báo mảng hai chiều

float a[2][3]; /\* là mảng số thực có 2 hàng, 3 cột \*/

Truy cập các phần tử của mảng: a[i][j] (i<=1, j<=2)

Ví dụ khác: float arr[3][4][5];

Lưu ý: C quy ước mảng 2 chiều là mảng 1 chiều mà các phần tử của nó lại là 1 mảng 1 chiều (hay là mảng của mảng)

Tên mảng a biểu thị địa chỉ đầu tiên của mảng tức a[0,0], khi đó:

a trỏ tới đầu hàng thứ nhất (tức a[0][0])

a+1 trỏ tới đầu hàng thứ hai (tức a[1][0])

…

Phép toán lấy địa chỉ & không dùng được với các phần tử của mảng nhiều chiều (đối với mảng số nguyên 2 chiều có thể

dùng &a[i][j]).

Ví dụ: Hàm nhập mảng 2 chiều sử dụng phép lấy địa chỉ &

void nhap\_ma\_tran(float a[][maxCot], int n, int m){

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

for (int j=0; j < m; ++j){

printf(“ a[%d][%d]: “, i+1, j+1);

scanf(“%f", &a[i][j]);

}

}

}

(\*) Lưu ý: Không được dùng phương pháp này

Để lần lượt duyệt trên các phần tử của mảng 2 chiều ta có thể sử dụng con trỏ theo cách sau:

float a[2][3], \*pa;

pa = (float\*) a; // ép kiểu

Khi đó,

pa trỏ tới a[0][0]

pa+1 trỏ tới a[0][1]

pa+2 trỏ tới a[0][2]

pa+3 trỏ tới a[1][0]

pa+4 trỏ tới a[1][1]

pa+5 trỏ tới a[1][2]

Chú ý: Ta không thể gán trực tiếp pa=a;

Vì kiểu của pa và a là khác nhau: pa là con trỏ float, còn a là địa chỉ kiểu float[3]

Ví dụ: Hàm nhập mảng 2 chiều sử dụng biến trung gian

void nhap\_ma\_tran(float a[][maxCot], int n, int m){

int i, j;

float x;

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

for (j=0; j < m; ++j){

printf(“ a[i][j]: “, i+1, j+1);

scanf("%f", &x);

a[i][j] = x;

}

}

}

Ví dụ: Tính tổng 2 mảng số nguyên

void tinh\_tong(float a[][Max\_Cot], float b[][Max\_Cot],

float c[][Max\_Cot], int n, int m){

int i, j;

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

for (j=0; j < m; ++j)

c[i][j]= a[i][j] + b[i][j];

}

**2.4. Con trỏ hàm.**

**2.4.1 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;

}

**2.4.2. 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.

**2.4.3. 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.

**2.4.5. 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;

}

**2.5. 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.2.5.1. 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;

**2.6. CẤP PHÁT ĐỘNG**

Biến dùng để lưu trữ dữ liệu trong chương trình. Biến cần được khai báo trước

khi sử dụng. Thực chất của việc khai báo biến là xin cấp phát một vùng bộ nhớ

để lưu trữ dữ liệu. Việc khai báo biến đã đề cập trong các chương trước chính là

việc cấp phát tĩnh bộ nhớ. Tuy nhiên, không phải khi nào số lượng và kích thước

của các biến cũng xác định được ngay khi biên dịch, chẳng hạn nó có thể phụ

thuộc vào các thông tin người dùng cung cấp khi thực hiện chương trình. Trong

trường hợp này, bộ nhớ cần được 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 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 a, \*pa;

pa = (int\*)malloc(sizeof(int)); /\*cấp phát vùng nhớ 2 bytes\*/

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

**2.6.3 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 a, \*pb;

pb = (int\*)calloc(10, sizeof(int)); /\*cấp phát vùng nhớ 20 bytes\*/

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

**2.6.4 Hàm realloc()**

Ý nghĩa: cấp phát lại 1 vùng nhớ cho con trỏ pa quản lý có kích thước mới là size, nội dung của vùng nhớ trước đó vẫn tồn tại.

Lưu ý: Kết quả trả về của hàm là địa chỉ đầu tiên của vùng nhớ mới, địa chỉ này có thể khác với địa chỉ cấp ở phát lần đầu.

Cú pháp:

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

Ví dụ:

int a, \*pa;

pa = (int\*)malloc(sizeof(int)); /\*cấp phát vùng nhớ 2 bytes\*/

pa = realloc(pa, 6); /\*cấp phát lại vùng nhớ có kích thước 6 bytes\*/

**2.6.5 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 a, \*pa, \*pb;

pa = (int\*)malloc(sizeof(int)); /\*cấp phát vùng nhớ 2 bytes\*/

pb = (int\*)calloc(10, sizeof(int)); /\*cấp phát vùng nhớ 20 bytes\*/

free(pa); // giải phóng vùng nhớ quản lý bởi con trỏ pa

free(pb); // giải phóng vùng nhớ quản lý bởi con trỏ pb

**2.7 Xử lý tệp**

Các thao tác tệp trong C được thực hiện nhờ các hàm thư viện. Các hàm này được chia thành hai nhóm: cấp 1 và cấp 2. Mỗi hàm (dù cấp 1 hay cấp 2) đều có thể truy xuất theo cả hai kiểu nhị phân và văn bản.

Các hàm cấp 1 (còn gọi là các hàm nhập/xuất hệ thống) có các đặc trưng cơ bản là:

+ Không có dịch vụ nhập xuất riêng cho từng kiểu dữ liệu mà chỉ có dịch vụ đọc ghi một dẫy các byte.

+ Mỗi tệp có một thẻ (handle) và các hàm cấp 1 làm việc với tệp thông qua số hiệu tệp.

Các hàm cấp 2 được xây dựng từ các hàm cấp 1 nên dễ sử dụng và có nhiều khả năng hơn:

+ Có dịch vụ truy xuất cho từng kiểu dữ liệu.

+ C tự động cung cấp một vùng đệm. Mỗi lần đọc/ghi thì thường tiến hành trên vùng đệm chứ không hẳn trên tệp. Chẳng hạn khi ghi một số nguyên thì số được đưa vào vùng đệm và khi nào đầy thì vùng đệm mới được đẩy lên đĩa. Khi đọc, thông tin được lấy từ vùng đệm, và chỉ khi vùng đệm đã trống rỗng thì máy mới lấy dữ liệu từ đĩa chứa vào vùng đệm. Việc sử dụng vùng đệm sẽ giảm số lần nhập xuất trên đĩa và nâng cao tốc độ làm việc. Tuy vậy trong một số trương hợp ta phải nhớ tới tác nghiệp vét vùng đệm để đề phòng mất mát thông tin.

+ Các hàm cấp 2 làm việc với tệp thông qua một biến con trỏ tệp.

Các hàm cấp 1 ở mức độ sâu hơn, gần hệ điều hành hơn nên tốc độ truy nhập sẽ nhanh hơn. Tuy vậy, các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1, nên trong chương trình C các hàm cấp 2 được ưa chuộng hơn. Các hàm cấp 2 sẽ là nội dung của chương này.

**2.7.1 Kiểu nhập xuất nhị phân văn bản.**

Một tệp tin (dù nó được xây dựng bằng cách nào) đơn giản chỉ là một dẫy các byte (có giá trị từ 0 đến 255) ghi trên đĩa. Số byte của dẫy chính là độ dài của tệp.

**2.7.1a Kiểu nhị phân**

Trong quá trình nhập xuất dữ liệu không bị biến đổi. Dữ liệu ghi trên tệp theo các byte nhị phân như trong bộ nhớ.

Trong khi đọc nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác không.

Các hàm chỉ dùng cho tệp nhị phân

 +putw: dùng để ghi một số nguyên (2 byte) lên tệp.

 +getw: dùng để đọc một số nguyên (2 byte) từ tệp.

 +fwrite: dùng để ghi một số mẫu tin lên tệp.

 +fread: dùng để đọc một số mẫu tin từ tệp.

**2.7.1b Kiểu văn bản**

Kiểu nhập xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng (mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.

Khi ghi, một ký tự LF (mã 10) được chuyển thành hai ký tự CR (mã 13) và LF. Khi đọc, hai ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF.

Trong khi đọc nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (số -1) và hàm feof(fp) cho giá trị khác không (số 1). Điều này cốt để phù hợp với một số hệ soạn thảo (như C, Pascal). Tệp soạn thảo trong các hệ này kết thúc bởi mã 26.

Các hàm chỉ dùng cho tệp văn bản

 +fprintf: dùng để ghi dữ liệu theo khuôn dạng lên tệp.

 +fscanf: dùng để đọc dữ liệu từ tệp theo khuôn dạng.

 +fputs: dùng để ghi một chuỗi ký tự lên têp.

 +fgets: dùng để đọc một dẫy ký tự từ tệp.

Ví dụ:

Chương trình sau tạo 2 tệp có tên là vb và np. Trong chương trình dùng các

hàm:

+ fopen để mở tệp,

+ fputc để ghi một ký tự lên tệp,

+ fclose để đóng tệp.

#include &lt;stdio.h&gt;

main()

{

/\* Khai báo biến con trỏ tệp \*/

FILE \*fvb, \*fnp;

/\* Mở tệp vb để ghi theo kiểu văn bản

gán con trỏ tệp cho biến fvb \*/

fvb=fopen(&quot;vb&quot;,&quot;wt&quot;);

/\* Mở tệp np để ghi theo kiểu nhị phân gán con trỏ tệp cho biến fnp \*/

fnp=fopen(&quot;np&quot;,&quot;wb&quot;);

/\* Ghi các ký tự lên tệp fvb \*/

fputc(&#39;A&#39;,fvb);

fputc(26,fvb);

fputc(10,fvb);

fputc(&#39;B&#39;,fvb);

/\* Ghi các ký tự lên tệp fnp \*/

fputc(&#39;A&#39;,fnp);

fputc(26,fnp);

fputc(10,fnp);

fputc(&#39;B&#39;,fnp);

fclose(fvb); /\* Đóng tệp fvb \*/

fclose(fnp); /\* Đóng tệp fnp \*/

}

Kết quả:

+ Tệp vb có các ký tự ứng với các mã: 65 26 13 10 66

+ Tệp np có các ký tự ứng với các mã: 65 26 10 66

Chú ý: Nếu dùng kiểu văn bản để đọc tệp vb hay tệp np, thì ta chỉ nhận được một ký tự đầu (mã 65) vì khi gặp ký tự thứ hai (mã 26) thì ta nhận được mã kết thúc EOF.

**2.7.2 CÁC THAO TÁC TRÊN TỆP TIN**

**2.7.2a Các hàm dùng chung cho cả hai loại tệp**

Hàm fopen

+ Dạng hàm:

FILE \*fopen(const char \*tên\_tệp, const char \*kiểu);

+ Công dụng: Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu

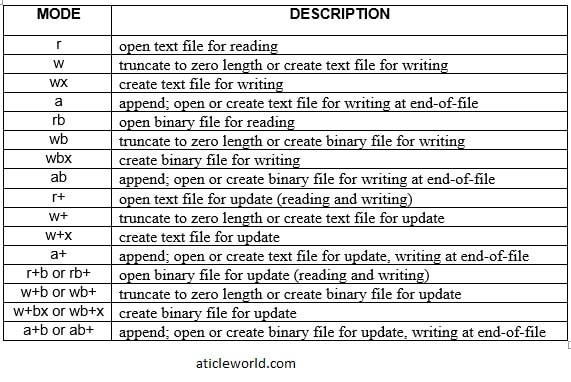
FILE ứng với tệp vừa mở. Các hàm cấp 2 sẽ làm việc với tệp thông qua con trỏ

này. Nếu có lỗi hàm trả về giá trị NULL.

+ Các đối:

Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

Kiểu có thể có các giá trị sau:



Hình 2.7.2a Các kiểu dữ liệu trong hàm fopen

Chú ý: Trong các kiểu đọc/ghi, cần làm sạch vùng đệm trước khi chuyển từ đọc sang ghi hoặc từ ghi sang đọc. Các hàm fflush và hàm di chuyển đầu từ đều làm được chuyện này.

Hàm fclose

+ Dạng hàm:

int fclose(FILE \*fp);

+ Công dụng: Hàm dùng để đóng tệp. Nội dung đóng tệp gồm: Đẩy dữ liệu còn trong vùng đệm lên đĩa (khi đang ghi), xoá vùng đệm (khi đang đọc) và giải phóng biến fp để nó có thể dùng cho tệp khác. Nếu thành công hàm cho giá trị 0, trái lại hàm cho EOF.

+ Đối: fp là con trỏ tương ứng với tệp cần đóng.

Hàm fcloseall

+ Dạng hàm:

int fcloseall(void);

+ Công dụng: Hàm dùng để đóng tất cả các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đóng được, trái lại hàm cho EOF.

Hàm fflush

+ Dạng hàm:

int fflush(FILE \*fp);

+ Công dụng: Hàm dùng làm sạch vùng đệm của tệp fp. Nếu thành công hàm cho giá trị 0, trái lại hàm cho EOF.

+ Đối: fp là con trỏ tệp.

Hàm fflushall

+ Dạng hàm:

int fflushall(void);

+ Công dụng: Hàm dùng làm sạch vùng đệm của các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đang mở, trái lại hàm cho EOF.

Hàm feof

+ Dạng hàm:

int feof(FILE \*fp);

+ Công dụng: Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm cho giá trị 0.

+ Đối: fp là con trỏ tệp.

**2.7.2b Nhập xuất ký tự**

Các hàm nhập xuất ký tự dùng được cả trong kiểu nhị phân và ký tự, nhưng tác dụng có những điểm khác nhau.

Các hàm putc và fputc.

+ Dạng hàm:

int putc(int ch, FILE \*fp);

int fputc(int ch, FILE \*fp);

+ Công dụng: Hàm ghi lên tệp fp một ký tự có mã bằng:

m= ch % 256

trong đó ch được xem là số nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại hàm cho EOF.

+ Đối: ch là một giá trị nguyên, fp là con trỏ tệp.

Ví dụ câu lệnh:

putc(-1,fp);

sẽ ghi lên tệp fp mã 255, vì dạng không dấu của -1 là 65535.

+ Ghi chú:

- Hai hàm trên có ý nghiã như nhau.

- Trong kiểu văn bản, nếu m=10 thì hàm sẽ ghi lên tệp hai mã 13 và 10.

Các hàm getc và fgetc.

+ Dạng hàm:

int getc(FILE \*fp);

int fgetc(FILE \*fp);

+ Công dụng: Hàm đọc một ký tự từ tệp fp. Nếu thành công hàm cho mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm cho EOF.

+ Đối: fp là con trỏ tệp.

+ Ghi chú:

- Hai hàm trên có ý nghiã như nhau.

- Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10.

- Trong kiểu văn bản, khi gặp mã 26 thì hàm không trả về 26 mà trả về EOF.

Ví dụ: Chương trình sao tệp.

/\* Dùng mode nhị phân và getc, putc \*/

#include &quot;stdio.h&quot;

main()

{

int c;

char t1[14],t2[14];

FILE \*f1,\*f2;

printf(&quot;\n TEP NGUON: &quot;);

gets(t1);

printf(&quot;\n TEP DICH: &quot;);

gets(t2);

f1=fopen(t1,&quot;rb&quot;);

if(f1==NULL)

{

printf(&quot;\n%s khong ton tai&quot;,t1);

getch();

exit(1);

}

f2=fopen(t2,&quot;wb&quot;);

/\* Sao tệp \*/

while ((c=fgetc(f1))!=EOF)

fputc(c,f2);

fclose(f1); fclose(f2);

}

**2.7.3 CÁC HÀM NHẬP XUẤT KIỂU VĂN BẢN**

**2..7.3a Hàm fprintf**

+ Dạng hàm:

int fprintf(FILE \*fp, const char \*dk, ds);

+ Công dụng: Giá trị các đối được ghi lên tệp fp theo khuôn dạng xác định trong chuỗi điều khiển dk. Nếu thành công, hàm trả về một giá trị nguyên bằng số byte ghi lên tệp. Khi có lỗi hàm cho EOF. Hàm làm việc giống như printf.

+ Đối:

fp là con trỏ tệp,

dk chứa địa chỉ của chuỗi điều khiển,

ds là danh sách các đối mà giá trị của chúng cần ghi lên tệp.

Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm printf.

Ví dụ: Xét chương trình:

#include &lt;stdio.h&gt;

main()

{

FILE \*f;

int i;

f=fopen(&quot;text&quot;,&quot;wt&quot;);

fprintf(f, &quot;Cac dong&quot;);

for(i=1; i&lt;=2; ++i)

fprintf(f,&quot;\nDong%2d&quot;, i);

fclose(f);

}

Chương trình trên sẽ tạo ra tệp văn bản text gồm 3 dòng với nội dung như sau:

Cac dong

Dong 1 Dong 2

**2.7.3b Hàm fscanf**

+ Dạng hàm:

int fscanf(FILE \*fp, const char \*dk, ...);

+ Công dụng: Đọc dữ liệu từ tệp fp, biến đổi theo khuôn dạng (đặc tả) trong dk và lưu kết quả vào các đối. Hàm làm việc giống như scanf. Hàm trả về một giá trị bằng số trường được đọc.

+ Đối:

fp là con trỏ tệp,

dk chứa địa chỉ của chuỗi điều khiển,

... là danh sách các đối chứa kết quả đọc được từ tệp.

Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm scanf.

**2.7.3c Hàm fputs**

+ Dạng hàm:

int fputs(const char \*s,FILE \*fp);

+ Công dụng: Ghi chuỗi s lên tệp fp (dấu &#39;\0&#39; không ghi lên tệp). Khi thành công, hàm trả về ký tự cuối cùng được ghi lên tệp. Khi có lỗi hàm cho EOF.

+ Đối:

s là con trỏ trỏ tới địa chỉ đầu của một chuỗi ký tự kết thúc bằng dấu &#39;\0&#39; fp là con trỏ tệp.

**2.7.3d Hàm fgets**

+ Dạng hàm:

char \*fgets(char \*s, int n, FILE \*fp);

+ Công dụng: Đọc một dẫy ký tự từ tệp fp chứa vào vùng nhớ s. Việc đọc kết thúc khi:

- Hoặc đã đọc n-1 ký tự.

- Hoặc gặp dấu xuống dòng (Cặp mã 13 10). Khi đó mã 10 được đưa vào xâu kết quả.

- Hoặc kết thúc tệp.

Xâu kết quả sẽ được bổ sung thêm dấu hiệu kết thúc chuỗi &#39;\0&#39;. Khi thành công, hàm trả địa chỉ vùng nhận kết quả. Khi có lỗi hoặc gặp cuối tệp, hàm cho giá trị NULL.

+ Đối:

s là con trỏ (kiểu char)trỏ tới một vùng nhớ đủ lớn để chứa chuỗi ký tự đọc từ tệp.

n là số nguyên xác định độ dài cực đại của dẫy cần đọc.

fp là con trỏ tệp.

Ví dụ: Chương trình sau sẽ nhập các dòng ký tự từ bàn phím và ghi lên tệp “dong\_vb”.

/\* ghi các dòng văn bản lên tệp \*/

#include &lt;stdio.h&gt;

#include &lt;conio.h&gt;

main()

{

int i=0;

char d[256];

FILE \*f;

f=fopen(&quot;dong\_vb&quot;,&quot;w&quot;);

clrscr();

while(1)

{

++i;

printf(&quot;\nDong %d: &quot;,i);

gets(d);

if(d[0]==&#39;\0&#39;)

break;

if(i&gt;1)

fputc(10,f);

fputs(d,f);

}

fclose(f);

}

**2.7.4. NHẬP XUẤT NGẪU NHIÊN**

Mỗi tệp khi đang mở có một con trỏ chỉ vị dùng để xác định vị trí đọc ghi trên tệp. Khi mở tệp tin để đọc hay ghi, con trỏ chỉ vị luôn luôn ở đầu tệp tin (byte 0). Nhưng nếu tệp được mở theo mode &quot;a&quot; thì con trỏ chỉ vị ở cuối tệp để ghi thêm dữ liệu vào tệp này. Việc nhập xuất dữ liệu được bắt đầu từ vị trí hiện tại của con trỏ chỉ vị và sau khi hoàn thành thì con trỏ này dịch chuyển một số byte chính bằng số byte đã đọc hay ghi. Như vậy việc nhập xuất được tiến hành tuần tự theo chiều từ đầu đến cuối tệp tin.

Trong mục này sẽ nói cách di chuyển con trỏ chỉ vị đến vị trí mong muốn, nhờ đó ta có thể nhập xuất tại bất kỳ chỗ nào trên tệp tin. Cách làm này gọi là nhập xuất ngẫu nhiên rất tiện lợi cho việc sữa chữa, bổ sung dữ liêu trực tiếp trên tệp.

**2.7.4a Hàm rewind**

+ Dạng hàm:

void rewind(FILE \*fp);

+ Công dụng: Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu tệp.

+ Đối:

fp là con trỏ tệp.

**2.7.4b Hàm fseek**

+ Dạng hàm:

int fseek(FILE \*fp, long sb, int xp);

+ Công dụng: Hàm di chuyển con trỏ chỉ vị của tệp fp từ vị trí xác định bởi xp qua một số byte bằng giá trị tuyệt đối cua sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại sẽ di chuyển về phía đầu tệp.

Khi thành công hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

+ Đối:

fp là con trỏ tệp,

sb là số byte cần di chuyển,

xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đấy

xp có thể nhận các giá trị sau:

xp = SEEK\_SET hay 0: Xuất phát từ đầu tệp.

xp = SEEK\_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

xp = SEEK\_END hay 2: Xuất phát từ cuối tệp.

Chú ý: Không nên dùng fseek trên kiểu văn bản, vì sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

**2.7.4c Hàm ftell**

+ Dạng hàm:

long ftell(FILE \*fp);

+ Công dụng: Khi thành công hàm cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tệp fp). Số thứ tự của byte được tính từ 0. Khi có lỗi hàm trả về -1L.

+ Đối:

fp là con trỏ tệp.

Ví dụ: Chương trình dưới đây dùng các hàm fseek và ftell để xác định độ dài của tệp bất kỳ.

/\* Dung fseek va ftell xac dinh do dai tep tin \*/

#include &quot;stdio.h&quot;

#include &quot;conio.h&quot;

main()

{

FILE \*fp;

long n;

char tep[14];

clrscr();

printf(&quot;\nTen tep: &quot;); gets(tep);

fp=fopen(tep,&quot;rb&quot;);

if(fp==NULL)

{

printf(&quot;Khong ton tai %s&quot;,tep);

exit(1);

}

fseek(fp,0,SEEK\_END);

n=ftell(fp);

fclose(fp);

printf(&quot;\nDo dai cua tep %s la %ld byte&quot;,tep,n);

}

**2.8 STRUCT**

Để lưu trữ và xử lý thông tin trong máy tính ta có các biến và các mảng. Mỗi biến chứa được một giá trị. Mảng có thể xem là tập hợp nhiều biến có cùng một kiểu giá trị

và được biểu thị bằng một tên. Cấu trúc có thể xem như một sự mở rộng của các khái niệm biến và mảng, nó cho phép lưu trữ và xử lý các dạng thông tin phức tạp hơn. Cấu trúc là một tập hợp các biến, các mảng và được biểu thị bởi một tên duy nhất.

**2.8.1 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 Sinhvien{

int mssv;

char hoten[30];

} ;

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

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

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

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

Struct sinhvien sv1,sv2;

Struct sinhvien sv1[30]; // mảng

Struct sinhvien \*sv2; // con trỏ

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

Ví dụ: Struct sinhvien{

Int mssv;

Char hoten[30];

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

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

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

nv.mssv;  
nv.hoTen;

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

sv = (struct sinhvien\*)malloc(sizeof(struct sinhvien));  
nv->mssv;  
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;

**2.8.4 Con trỏ cấu trúc.**

Con trỏ cấu trúc dùng để lưu giữ địa chỉ của biến cấu trúc và

mảng cấu trúc.

Cách khai báo cũng tương tự như các loại con trỏ khác

struct <tên\_cấu\_trúc> \*<tên\_biến\_con\_trỏ>;

Hoặc: <tên\_cấu\_trúc> \*<tên\_biến\_con\_trỏ>;

Ví dụ: Khai báo một biến con trỏ kiểu dia\_chi:

struct dia\_chi \*p; // dia\_chi \*p;

Khi khai báo con trỏ cấu trúc, biến con trỏ chưa có địa chỉ cụ hể, lúc này nó chỉ được cấp phát 2 bytes để lưu giữ địa chỉ nhưng chưa chỉ đến một đối tượng cụ thể nào. Do đó, để thao tác trên con trỏ cấu trúc ta phải:

+ Cấp phát một vùng nhớ cho nó

+ Hoặc cho nó quản lý địa chỉ của một biến cấu trúc hay một mảng cấu trúc nào đó

Ví dụ: Giả sử đã định nghĩa kiểu cấu trúc nhan\_su. Ta khai báo các biến, mảng, con trỏ cấu trúc như sau:

nhan\_su ns1, ns2, \*p, \*q, \*t, ds[10];

Khi đó, các phép gán sau đây là hợp lệ:

p = &ns1;

// gửi địa chỉ biến cấu trúc ns1 vào p

q = &ds[4]; //q=&ds[i]; // gửi địa chỉ phần tử ds[4] vào q

t=ds;

//gửi địa chỉ mảng ds hay ds[0] vào t

TRUY NHẬP THÔNG QUA CON TRỎ:

Có 2 cách truy nhập thành phần cấu trúc bằng con trỏ:

Tên\_con\_trỏ -> tên\_thành\_phần

(\*Tên\_con\_trỏ) . tên\_thành\_phần

Ví dụ: Giả sử đã định nghĩa cấu trúc nhan\_su

nhan\_su ns1, ns2, \*p, \*q, \*t, ds[20];

p= &ns1;

q=&ds[4];

t=ds;

Khi đó:

ns1.hoten ~ p->hoten

ds[4].otai.sonha ~ (\*q).otai.sonha

**2.8.5 Kiểu cấu trúc**

**2.8.5.a Mảng cấu trúc**

Khai báo mảng gồm các phần tử có kiểu cấu trúc

Ví dụ

nhan\_su mang\_nhan\_su[100];

Sử dụng

for (i = 0; i < 100; i++)

puts(mang\_nhan\_su[i].ho\_ten);

Ví dụ :

void nhap(nhan\_su \*ns){

printf("Ho ten: ");

gets(ns->hoten);

}

void xuat(nhan\_su \*ns){

printf("%30s\n", ns->hoten);

}

**2.8.5.b Hàm có tham số kiểu cấu trúc**

Ví dụ :

void hoanvi(nhan\_su \*p1, nhan\_su \*p2){

nhan\_su tmp;

tmp = \*p1; \*p1 = \*p2; \*p2 = tmp;

}

void sapxep(nhan\_su \*p, int n){

int i, j;

for (i = 0; i < n-1; i++)

for(j = i+1; j < n; j++)

if(strcmp(p[i].hoten, p[j].hoten) > 0)

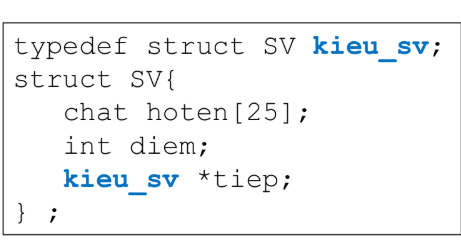
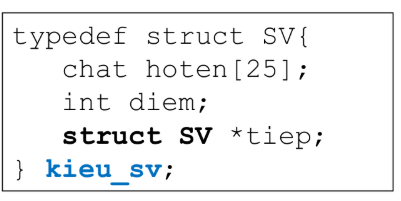
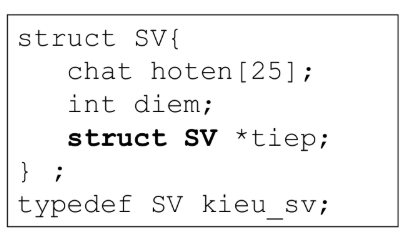
hoanvi(&p[i], &p[j]);

}

**2.8.6 Cấu trúc tự trỏ và danh sách liên kết**

Cấu trúc có ít nhất một thành phần là con trỏ kiểu cấu trúc đang định nghĩa gọi là cấu trúc tự trỏ.

Cấu trúc tự trỏ có thể được định nghĩa bằng các cách sau:



Hình 2.8.6 Cấu trúc tự trỏ.

Cấu trúc tự trỏ được dùng để xây dựng DSLK, đó là một nhóm các cấu trúc có tính chất sau:

Biết địa chỉ cấu trúc đầu đang được lưu trữ trong một con trỏ nào đó

Trong mỗi cấu trúc (trừ cấu trúc cuối) chứa địa chỉ của cấu trúc tiếp theo trong danh sách

Cấu trúc cuối chứa hằng NULL

Danh sách có 3 tính chất trên gọi là danh sách liên kết theo chiều thuận

=> Ta có thể lần lượt truy nhập từ cấu trúc đầu tới cấu trúc cuối theo chiều từ trên xuống dưới.

Ngược lại, ta có danh sách liên kết nghịch, với ds này ta có thể truy nhập từ cấu trúc cuối tới cấu trúc đầu theo chiều từ dưới lên.

Trong thực tế, ta có thể xây dựng danh sách liên kết mà mỗi phần tử chứa 2 địa chỉ: địa chỉ cấu trúc trước và địa chỉ cấu trúc sau => có thể truy nhập theo 2 chiều

**2.9 LINKLIST**

**2.9.1 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
* Phần tử cuối cùng của linked list có tên là tail

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

# **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:

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 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.”

**PHẦN B. THỰC HÀNH**

**TÊN : Hệ thống Quản lý Game**

Dưới đây là code của bài hệ thống quản lý game

A screenshot of a computer

AI-generated content may be incorrect.

Chúng ta tạo một struct có kiểu dữ liệu cấu trúc tên là TroChoi

**A close-up of a computer code

AI-generated content may be incorrect.**

Tạo struct có tên là Node để tạo thành các nút trong một danh sách liên kết, và mỗi nút này sẽ chứa dữ liệu là thông tin của một trò chơi (kiểu TroChoi) và một con trỏ đến nút kế tiếp.

A computer code with text

AI-generated content may be incorrect.

Đây là những hàm để thực hiện những tính năng của bài như thêm , hiển thị , sắp xếp , tìm kiếm( theo mã game, tên game , thể loại , giá ),….

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Đây là hàm main của bài giúp chúng ta xây dựng như 1 chương trình hoàn chỉnh với giao diên menu

A screenshot of a computer program

AI-generated content may be incorrect.

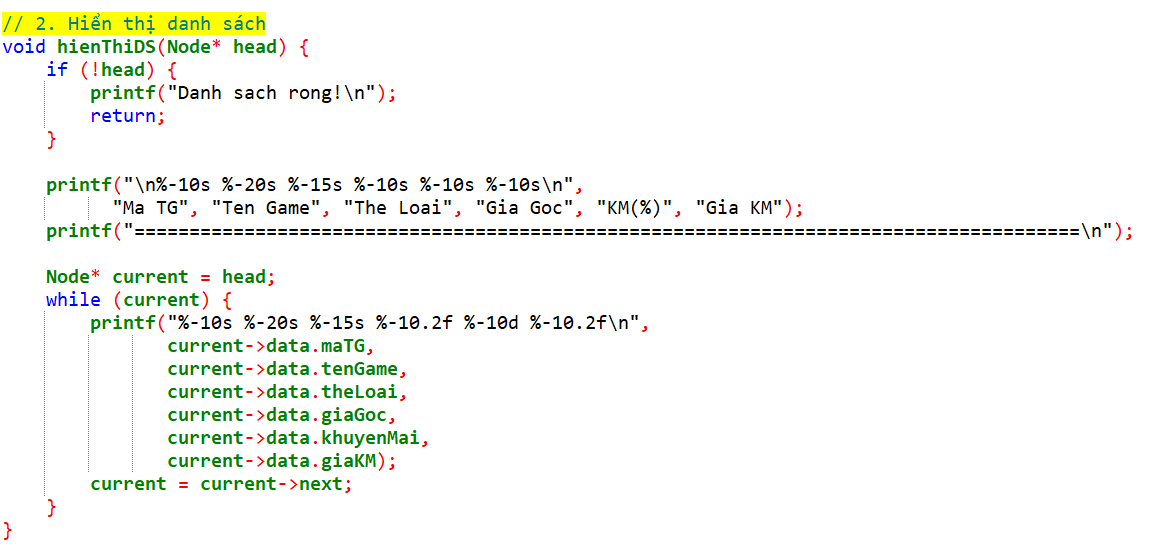
Với giao diện như này

A screenshot of a computer program

AI-generated content may be incorrect.

Đây là hàm thêm 1 game mới vào đầu danh sách của chúng ta

A computer screen shot of a code

AI-generated content may be incorrect.

A white screen with black text

AI-generated content may be incorrect.

Đây là hàm sắp xếp theo giá bán chúng ta dùng thuật mergeSort để xử lý

A computer screen shot of a code

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

AI-generated content may be incorrect.

A computer code with text

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

AI-generated content may be incorrect.

Đây là hàm dùng để tìm kiếm game theo nhiều cách như mã game , tên game , thể loại , giá góc

A computer screen shot of a code

AI-generated content may be incorrect.

Đây là hàm xóa 1 trò chơi ra khỏi danh sách

A screenshot of a computer code

AI-generated content may be incorrect.

Đây là hàm dùng để áp dụng khuyến mãi cho game

A computer code with many different colored text

AI-generated content may be incorrect.

Đây là hàm hủy khuyến mãi

A screen shot of a computer code

AI-generated content may be incorrect.

Hàm này dùng để ghi danh sách vào file

A screenshot of a computer program

AI-generated content may be incorrect.

Hàm này dùng để đọc danh sách từ file

A computer code with text

AI-generated content may be incorrect.

Và cuối cùng là hàm để giải phóng bộ nhớ

A screenshot of a computer

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

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

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

AI-generated content may be incorrect.

Và đây là kết quả của bài ………..

EM XIN CẢM ƠN!!!