**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN HỌC**

**KỸ THUẬT LẬP TRÌNH**

**BÀI TẬP LỚN**

Giảng viên hướng dẫn: ThS. TRẦN PHONG NHÃ

Sinh viên thực hiện: HUỲNH GIA BẢO

Mã sinh viên: 6551071007

Lớp: CQ.CNTT.65

Khóa: 65

Tp. Hồ Chí Minh, tháng 5, 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 với đề tài “**Bài Tập Lớn**”. 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 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!

**MỤC LỤC**

[***A.Lý Thuyết*** 5](#_Toc199017443)

[1. Hàm 5](#_Toc199017444)

[1.1 Định nghĩa Hàm 5](#_Toc199017445)

[1.2 Lời gọi Hàm 6](#_Toc199017446)

[1.3 Đối Số Và Tham Số 6](#_Toc199017447)

[1.4 Câu lệnh return trong Hàm 7](#_Toc199017448)

[2. Con trỏ 8](#_Toc199017449)

[2.1 Con trỏ và địa chỉ 8](#_Toc199017450)

[2.2 Tham chiếu và giải tham chiếu 11](#_Toc199017451)

[2.3 Hàm và con trỏ 12](#_Toc199017452)

[2.4 Con trỏ cấp 2 14](#_Toc199017453)

[2.5 Hàm Và Con Trỏ Cấp 2 16](#_Toc199017454)

[3. Con trỏ mảng 18](#_Toc199017455)

[3.1 Khai báo con trỏ mảng trong C 18](#_Toc199017456)

[3.2 Con trỏ và địa chỉ trong mảng 19](#_Toc199017457)

[3.3 Truy xuất giá trị phần tử trong mảng bằng con trỏ 22](#_Toc199017458)

[3.4 Nhập xuất mảng bằng con trỏ trong c 24](#_Toc199017459)

[4. Mảng con trỏ 25](#_Toc199017460)

[4.1 Định nghĩa mảng con trỏ 1 chiều 25](#_Toc199017461)

[4.2 Phân biệt mảng con trỏ với các khái niệm khác 28](#_Toc199017462)

[4.3 Mảng con trỏ 2 chiều 28](#_Toc199017463)

[5. Con trỏ hàm 29](#_Toc199017464)

[5.1 Khai báo con trỏ hàm 30](#_Toc199017465)

[5.2 Khởi tạo 30](#_Toc199017466)

[5.3 Thuộc tính của con trỏ hàm 31](#_Toc199017467)

[5.4 Một số ví dụ ứng dụng con trỏ hàm 31](#_Toc199017468)

[6. Cấp phát động 35](#_Toc199017469)

[6.1 Hàm malloc() 36](#_Toc199017470)

[6.2 Hàm calloc() 38](#_Toc199017471)

[6.3 Hàm free() 40](#_Toc199017472)

[6.4 Hàm realloc() 41](#_Toc199017473)

[7. Xử lí tệp 43](#_Toc199017474)

[7.1 Tạo file 43](#_Toc199017475)

[7.2 Ghi nội dung vào file 45](#_Toc199017476)

[7.3 Đọc nội dung từ file 48](#_Toc199017477)

[7.4 Hàm fgets() 49](#_Toc199017478)

[8. Kiểu cấu trúc 51](#_Toc199017479)

[8.1 Struct 51](#_Toc199017480)

[8.2Struct lồng nhau 55](#_Toc199017481)

[8.3 Struct làm tham số cho hàm 57](#_Toc199017482)

[9. Danh sách liên kết 58](#_Toc199017483)

[***B. Ứng Dụng*** 65](#_Toc199017484)

[**Bài 1:** Quản lí sinh viên 65](#_Toc199017485)

[**Bài 2:** Quản lí đoàn viên 67](#_Toc199017486)

[**Bài 3:** Quản lí ngân hàng 68](#_Toc199017487)

[**Bài 4:** Quản lí đa thức 70](#_Toc199017488)

[**Bài 5:** Quản lí ma trận 72](#_Toc199017489)

[**Bài 6:** Đăng kí và đăng nhập 73](#_Toc199017490)

[**Bài 7:** Quản lí món ăn 74](#_Toc199017491)

**BÀI TẬP LỚN**

# ***A.Lý Thuyết***

## 1. Hàm

### 1.1 Định nghĩa Hàm

* Hàm (function) là một các khối lệnh có nhiệm vụ thực hiện một chức năng nào đó.

- Lợi ích của việc dùng hàm:

* Code trở nên mạch lạc dễ đọc
* Dễ bug khi gặp lỗi
* Dễ bảo trì khi cần thay đổi 1 chức năng nhỏ
* Có khả năng tái sử dụng lại code

Cú pháp:

data\_type function\_name(type1 parameter1, type2 parameter2...){

//code

}

**-** Các thành phần của hàm :

* data\_type : Kiểu trả về của hàm, có thể là các kiểu dữ liệu như int, long long, float, char, double, hoặc void (tương ứng với kiểu trả về là rỗng)
* function\_name : Tên của hàm, cần tuân theo quy tắc như đặt tên biến
* parameter : Tham số của hàm, đây được coi như đầu vào của hàm. Bạn có thể xây dựng bao nhiêu tham số tùy ý và lựa chọn kiểu dữ liệu cho từng tham số.
* code : Các câu lệnh bên trong của hàm

### 1.2 Lời gọi Hàm

- Sau khi xây dựng hàm xong để hàm có thể thực thi bạn cần gọi nó trong hàm main và truyền cho nó tham số nếu cần.

- Khi bạn gọi hàm trong hàm main thì các câu lệnh bên trong hàm sẽ được thực thi, sau khi thực thi hết các câu lệnh thì hàm kết thúc và chương trình tiếp tục thực hiện các câu lệnh bên dưới hàm.

- Mỗi lần bạn gọi hàm thì các câu lệnh trong hàm sẽ được thực hiện.

VD:

#include<stdio.h>

void Output (){

printf(“Hello World\n”);//code

}

int main(){

Output(); // Lời gọi hàm

}

Kết quả:

Hello World

### 1.3 Đối Số Và Tham Số

*- Tham số* (Parameter) hay tham số hình thức là các thành phần khi bạn xây dựng hàm, xem xét ví dụ dưới đây thì a, b, c sẽ được gọi là tham số

*- Đối số* (Argument) hay tham số chính thức là các giá trị bạn truyền vào cho hàm khi gọi hàm, xem xét ví dụ dưới đây thì m, n, p được gọi là đối số

- Khi bạn gọi hàm thì lần lượt giá trị của các đối số sẽ được gán cho tham số, trong ví dụ dưới thì m được gán cho a, n được gán cho b, p được gán cho c. -- Những gì thay đổi trên tham số sẽ không có ảnh hưởng gì tới đối số

**- Chú ý :** Kiểu dữ liệu của đối số và tham số nên trùng nhau hoặc của tham số nên là kiểu dữ liệu lớn hơn kiểu dữ liệu của đối số. Ví dụ bạn xây dựng 1 hàm có tham số là long long thì nó có thể áp dụng với 1 số int nhưng ngược lại thì không.

Ví Dụ:

#include <stdio.h>

void display(int a, int b, int c){

printf("%d %d %d\n", a, b, c);

}

int main(){

int m = 100, n = 200, p = 300;

display(m, n, p);

return 0;

}

Output:

100 200 300

### 1.4 Câu lệnh return trong Hàm

- Khi hàm của bạn thực hiện các chức năng tính toán và mong muốn trả về 1 giá trị cụ thể thì bạn cần câu lệnh return, trong các ví dụ trên thì hàm của mình đều không cần trả về giá trị nào nên mình để kiểu trả về là void.

- Giả sử bạn cần viết hàm tính tổng của 3 số nguyên, khi đó hàm sẽ nhận vào tham số là 3 số nguyên và cần trả về tổng 3 số. Vậy bạn cần xác định tổng của 3 số đó sẽ có kiểu là gì để làm kiểu trả về cho hàm và bổ sung thêm câu lệnh return kèm giá trị bạn muốn trả về.

- Có thể hiểu đơn giản giá trị đi kèm với return chính là kết quả mà hàm trả về cho bạn khi bạn gọi hàm.

- Ví Dụ: Hàm tính tổng 3 số nguyên int

#include <stdio.h>

int tong(int a, int b, int c){

int sum = a + b + c;

return sum;

}

int main(){

printf("%d\n", tong(10, 20, 30));

printf("%d\n", tong(28, 28, 28));

return 0;

}

Output:

60

84

84

84

## 2. Con trỏ

### 2.1 Con trỏ và địa chỉ

- Con trỏ hay biến con trỏ cũng là một biến thông thường nhưng giá trị mà nó lưu lại là địa chỉ của 1 biến khác.

- Ví dụ biến kiểu int N trong chương trình sẽ có địa chỉ nhất định trong bộ nhớ, để lưu trữ giá trị địa chỉ này ta cần biến con trỏ kiểu int

- Khi khai báo biến con trỏ ta thêm dấu \* vào trước tên biến.

- Cú pháp khai báo : Kiểu\_Dữ\_Liệu \*Tên\_Biến\_Con\_Trỏ;

- Ví dụ:

#include <stdio.h>

int main(){

//Dấu \* thể hiện ptr là con trỏ

int \*ptr; // con trỏ kiểu int

//Dấu \* có thể đặt cạnh tên biến hoặc cạnh kiểu dữ liệu

long long\* ptr2; // con trỏ kiểu long long

char \*ptr3;// con trỏ kiểu kí tự

return 0;

}

- Mỗi biến trong chương trình đều được cấp phát vùng nhớ để lưu trữ giá trị của nó, ví dụ biến int sẽ được cấp phát 4 byte liên tiếp để lưu trữ và lấy địa chỉ của byte đầu tiên làm địa chỉ cho biến.

- Để in ra địa chỉ của biến bạn dùng toán tử &, ví dụ &N sẽ cho bạn địa chỉ của biến N.

Ví dụ:

#include <stdio.h>

int main(){

int N = 28;

long long M = 10000012828;

printf("Dia chi cua N trong bo nho : %d\n", &N);

printf("Dia chi cua M trong bo nho : %d\n", &M);

return 0;

}

Output:

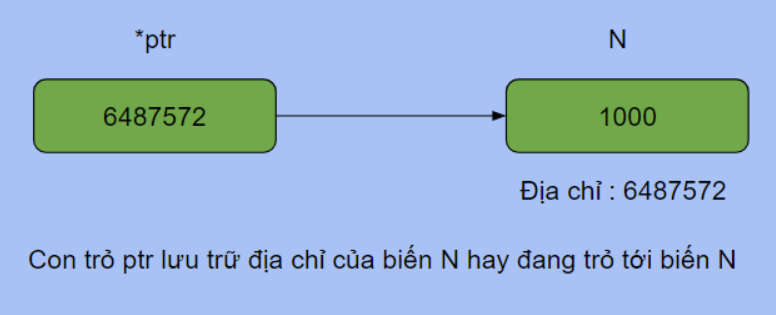
Dia chi cua N trong bo nho : 6487572

Dia chi cua M trong bo nho : 6488592

- Chú ý : Khi bạn chạy code trên thì địa chỉ của N và M sẽ khác, và mỗi lần bạn chạy có thể sẽ có một địa chỉ khác nhau.

- Con trỏ được sinh ra để lưu địa chỉ của biến ở kiểu dữ liệu tương ứng với nó, ví dụ biến con trỏ kiểu int sẽ lưu được địa chỉ của biến int.

- Chương trình sau sẽ gán địa chỉ của N cho biến con trỏ ptr, khi đó ta nói con trỏ ptr trỏ tới biến N .



-Ví dụ:

#include <stdio.h>

int main(){

int N = 1000;

printf("Dia chi cua N : %d\n", &N);

int \*ptr;

//Gán địa chỉ của N cho ptr

ptr = &N;

printf("Gia tri cua ptr : %d\n", ptr);

return 0;

}

- Output:

Dia chi cua N : 6487572

Gia tri cua ptr : 6487572

### 2.2 Tham chiếu và giải tham chiếu

- Khi con trỏ ptr trỏ tới hay tham chiếu tới biến N thì thông qua con trỏ ptr ta có thể truy xuất, thay đổi giá trị của biến N mà không cần dùng N.

- Để truy xuất tới giá trị của biến mà con trỏ đang trỏ tới ta dùng toán tử giải tham chiếu \* (dereference)

- Sau khi con trỏ ptr trỏ tới biến N thì N và \*ptr là một, đều truy xuất đến ô nhớ mà N đang chiếm để lấy giá trị tại ô nhớ đó.

* Lưu ý là bạn cần phân biệt dấu \* khi khai báo con trỏ ptr và dấu \* khi giải tham chiếu con trỏ ptr. Dấu \* khi khai báo thể hiện ptr là một con trỏ còn dấu \* trước ptr ở những câu lệnh sau là toán tử giải tham chiếu.

#include <stdio.h>

int main(){

int N = 1000;

printf("Dia chi cua N : %d\n", &N);

int \*ptr = &N; // ptr trỏ tới N

printf("Gia tri cua ptr : %d\n", ptr);

//Toán tử giải tham chiếu

printf("Gia tri cua bien ma con tro ptr tro toi : %d\n", \*ptr);

//Thay đổi N bằng ptr, \*ptr và N là một

\*ptr = 280;

printf("Gia tri cua N sau thay doi : %d %d\n", N, \*ptr);

return 0;

}

- Out put:

Dia chi cua N : 6487572

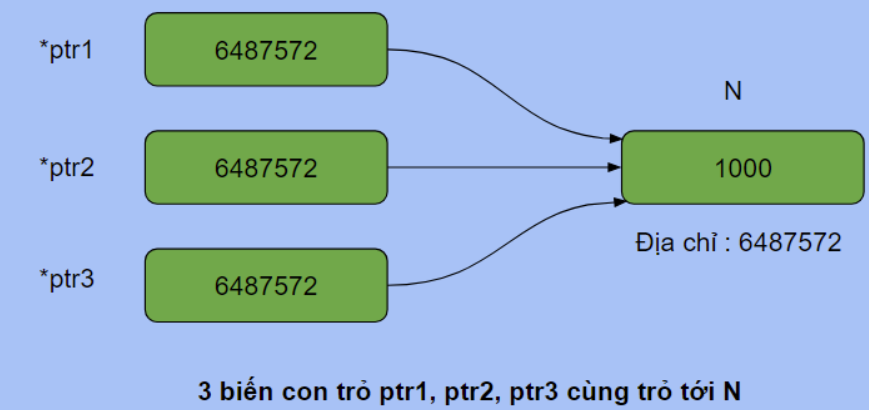
Gia tri cua ptr : 6487572

Gia tri cua bien ma con tro ptr tro toi : 1000

Gia tri cua N sau thay doi : 280 280

- Một biến có thể được trỏ tới bởi nhiều con trỏ, khi đó bạn có thể thông qua bất cứ 1 con trỏ nào để thay đổi giá trị của biến mà nó đang trỏ tới.

- Trong ví dụ dưới đây 3 biến con trỏ ptr1, ptr2, ptr3 đều trỏ tới N nên \*ptr1, \*ptr2, \*ptr3 và N đều có giá trị giống nhau.



### 2.3 Hàm và con trỏ

*Con trỏ làm tham số cho hàm*

- Để thay đổi giá trị của 1 biến sau khi hàm kết thúc thì việc truyền giá trị là không hợp lý, thay vì đó bạn hãy sử dụng con trỏ với mục đích là thay đổi giá trị của biến thông qua con trỏ.

- Khi hàm có tham số là một con trỏ thì khi gọi hàm bạn cần truyền vào một giá trị phù hợp, có thể là địa chỉ của 1 biến hoặc một con trỏ khác.

Ví dụ 1: Thay đổi giá trị của biến sau khi hàm kết thúc

#include <stdio.h>

//\*x ở đây là một con trỏ

void change(int \*x){

printf("Gia tri cua con tro x : %d\n", x);

printf("Gia tri cua bien ma x đang tro toi : %d\n", \*x);

//Đây là tham chiếu tới giá trị của biến

//mà con trỏ x đang trỏ tới để thay đổi nó thành 1000

\*x = 1000;

}

int main(){

int N = 28;

printf("Dia chi cua N : %d\n", &N);

change(&N); // truyền địa chỉ của N vào

printf("Gia tri cua N : %d\n", N);

return 0;

}

- Output:

Dia chi cua N : 6487580

Gia tri cua con tro x : 6487580

Gia tri cua bien ma x đang tro toi : 28

Gia tri cua N : 1000

Giải thích :

- Hàm change có tham số là một con trỏ kiểu int có tên là x, trong main bạn gọi change và truyền địa chỉ của N vào.

- Khi đó x sẽ được gán giá trị là địa chỉ của N, trong hàm change thì câu lệnh \*x = 1000 sẽ truy xuất tới ô nhớ mà x đang trỏ tới và gán giá trị 1000. Mà x lại đang tham chiếu tới N nên giá trị của N đã bị thay đổi.

- Sau khi hàm change kết thúc thì giá trị của N đã bị thay đổi thực sự.

Ví dụ 2 : Hoán đổi giá trị của 2 biến

#include <stdio.h>

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

int tmp = \*x;

\*x = \*y;

\*y = tmp;

}

int main(){

int a = 100, b = 200;

swap(&a, &b);

printf("%d %d\n", a, b);

return 0;

}

- Output:

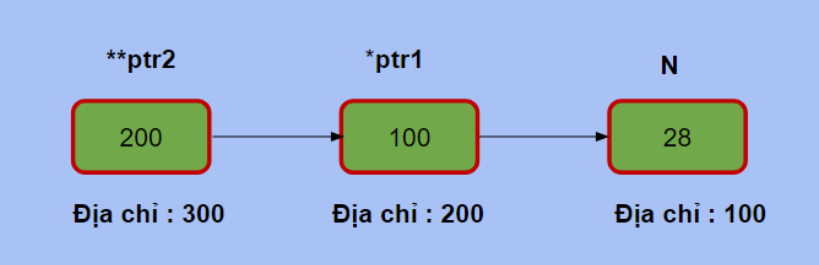
200 100

### 2.4 Con trỏ cấp 2

- Biến con trỏ cũng có địa chỉ riêng của nó, và nếu bạn muốn lưu giữ địa chỉ này thì bạn cần sử dụng con trỏ tới con trỏ.

- Tương tự nếu bạn muốn lưu địa chỉ của con trỏ cấp 2 thì bạn cần con trỏ cấp 3, muốn lưu địa chỉ của con trỏ cấp 3 thì cần con trỏ cấp 4...

- Số lượng dấu \* trước tên biến sẽ thể hiện con trỏ là cấp 1, 2 , hay 3. Ví dụ \*ptr là con trỏ cấp 1, \*\*ptr là con con trỏ cấp 2 và \*\*\*ptr là con trỏ cấp 3.



Ví dụ 1 : Con trỏ cấp 2

#include <stdio.h>

int main(){

int N = 28;

int \*ptr1 = &N;

// con trỏ cấp 2 ptr2 lưu địa chỉ con trỏ ptr1

int \*\*ptr2 = &ptr1;

printf("Dia chi cua N : %d\n", &N);

printf("Gia tri cua ptr1 : %d\n", ptr1);

printf("Dia chi cua ptr1 : %d\n", &ptr1);

printf("Gia tri cua ptr2 : %d\n", ptr2);

printf("Giai tham chieu ptr2 : %d\n", \*ptr2);

printf("Giai tham chieu 2 lan ptr2 : %d\n", \*\*ptr2);

return 0;

}

- Output:

Dia chi cua N : 6487572

Gia tri cua ptr1 : 6487572

Dia chi cua ptr1 : 6487576

Gia tri cua ptr2 : 6487576

Giai tham chieu ptr2 : 6487572

Giai tham chieu 2 lan ptr2 : 28

Giải thích :

- Con trỏ ptr1 lưu địa chỉ của N là 6487572 hay giá trị của ptr1 là 6487572.

- Con trỏ ptr2 trỏ tới ptr1 nên giá trị của ptr2 chính là địa chỉ của ptr1 là 6487576.

- Giải tham chiếu ptr2 sẽ giúp bạn truy xuất giá trị của biến mà ptr2 đang trỏ tới, mà ptr2 đang trỏ tới ptr1 nên giá trị khi giải tham chiếu ptr2 sẽ là ptr1 hay địa chỉ của N.

- Giải tham chiếu 2 lần sẽ giúp bạn truy xuất giá trị của N, giải tham chiếu lần 1 ta đang truy xuất tới ptr1, giải tham chiếu 2 lần sẽ truy xuất tới N.

### 2.5 Hàm Và Con Trỏ Cấp 2

- Để thay đổi giá trị của biến thông thường bằng hàm ta dùng con trỏ, vậy để thay đổi biến con trỏ ta cần cùng tới con trỏ cấp 2. Tương tự bạn muốn thay đổi giá trị của con trỏ cấp 2 thông qua hàm ta cần xây dựng tham số là con trỏ cấp 3.

- Nếu chúng ta nhầm tưởng để thay đổi giá trị của biến thông qua hàm chỉ cần sử dụng con trỏ, điều này chỉ đúng khi bạn muốn thay đổi giá trị của biến thông thường, còn thay đổi con trỏ thì không thể chỉ dùng con trỏ.

Ví dụ 1 : Thay đổi giá trị của con trỏ thông qua hàm

change(ptr1, ptr2);

printf("Sau khi goi ham : \n");

printf("Gia tri cua ptr1 : %d\n", ptr1);

printf("Gia tri cua ptr2 : %d\n", ptr2);

return 0;

}

#include <stdio.h>

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

x = y;}

int main(){

int N = 28, M = 56;

int \*ptr1 = &N;

int \*ptr2 = &M;

printf("Truoc khi goi ham : \n");

printf("Gia tri cua ptr1 : %d\n", ptr1);

printf("Gia tri cua ptr2 : %d\n", ptr2);

-Output:

Truoc khi goi ham :

Gia tri cua ptr1 : 6487560

Gia tri cua ptr2 : 6487564

Sau khi goi ham :

Gia tri cua ptr1 : 6487560

Gia tri cua ptr2 : 6487564

* Giải thích :

- Hàm change nhận 2 tham số là ptr1 và ptr2, khi đó giá trị của con trỏ x = ptr1 và y = ptr2, trong hàm change gán con trỏ y cho con trỏ x. Nhưng vấn đề ở đây là ptr1 với x khác nhau và ptr2 với y khác nhau, chỉ đơn thuần ta gán giá trị của ptr1 và ptr2 cho x và y, do đó x y có thay đổi gì thì không ảnh hưởng tới ptr1 và ptr2. Do đó sau khi hàm change kết thúc thì không có gì thay đổi đối với ptr1 và ptr2.

- Nếu bạn muốn thay đổi ptr1 thì bạn cần tìm nhờ hàm change tìm tới địa chỉ của ptr1 để thay thế, và khi đó thì ta cần con trỏ cấp 2 làm tham số cho hàm change.

Ví dụ 2 : Thay đổi con trỏ cấp 1 thông qua hàm bằng con trỏ cấp 2

printf("Gia tri cua ptr1 : %d\n", ptr1);

printf("Gia tri cua ptr2 : %d\n", ptr2);

change(&ptr1, ptr2);

printf("Sau khi goi ham : \n");

printf("Gia tri cua ptr1 : %d\n", ptr1);

printf("Gia tri cua ptr2 : %d\n", ptr2);

return 0;

}

#include <stdio.h>

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

\*x = y;

}

int main(){

int N = 28, M = 56;

int \*ptr1 = &N;

int \*ptr2 = &M;

printf("Truoc khi goi ham : \n");

-Output:

Truoc khi goi ham :

Gia tri cua ptr1 : 6487576

Gia tri cua ptr2 : 6487580

Sau khi goi ham :

Gia tri cua ptr1 : 6487580

Gia tri cua ptr2 : 6487580

## 

## 3. Con trỏ mảng

* Mảng cũng là một loại dữ liệu trong C, và do đó, chúng ta cũng có thể sử dụng con trỏ để lưu trữ địa chỉ và qua đó thao tác với chúng. Chúng ta gọi con trỏ sử dụng trong mảng là con trỏ mảng trong c.

3.1 Khai báo con trỏ mảng trong C

- Cách khai báo con trỏ chuỗi trong c cũng tương tự như cách khai báo các loại con trỏ khác. Chúng ta viết kiểu của con trỏ, tiếp theo là dấu hoa thị \* và tên của con trỏ với cú pháp sau đây:

type \*p;

- Trong đó type là kiểu dữ liệu của con trỏ chuỗi, và p là tên con trỏ. Lưu ý kiểu type của con trỏ phải giống với kiểu của mảng mà nó gán địa chỉ.

- Sau khi khai báo con trỏ mảng, chúng ta có thể tiến hành gán địa chỉ của mảng cho nó.

- Tuy nhiên khác với cách gán địa chỉ cho con trỏ với các loại dữ liệu khác thì với con trỏ mảng trong C, chúng ta không dùng toán tử & khi gán địa chỉ của mảng cho con trỏ.

- Khi khai báo một mảng, tên biến dùng để khai báo mảng sẽ biểu thị địa chỉ của điểm bắt đầu của vùng lưu mảng trong bộ nhớ.

- Do tên biến dùng để khai báo mảng đã là một địa chỉ, nên khi gán địa chỉ của mảng cho con trỏ, chúng ta không cần thêm toán tử & như các loại dữ liệu khác.  
- Ngược lại, nếu bạn viết thêm toán tử & thì khi biên dịch sẽ xảy ra lỗi.Ví dụ khi gán địa chỉ của biến hay cấu trúc cho con trỏ, chúng ta cần dùng tới toán tử &.

- Tuy nhiên chúng ta lại không cần tới toán tử & khi gán địa chỉ mảng cho con trỏ.

- VD: Cách khai báo con trỏ không cần dùng tới &

int nums[] = {1,2,3,4}, \*p;

p = nums;

### 3.2 Con trỏ và địa chỉ trong mảng

* *Con trỏ mảng biểu thị địa chỉ của điểm bắt đầu mảng trong bộ nhớ*

- Con trỏ mảng biểu thị địa chỉ của điểm bắt đầu vùng lưu trữ mảng trong bộ nhớ. Địa chỉ này cũng chính là địa chỉ của phần tử đầu tiên (có index bằng 0) của mảng trong bộ nhớ.

- Sau khi gán địa chỉ của mảng cho con trỏ, chúng ta có thể xuất ra địa chỉ của điểm bắt đầu mảng (cũng chính là địa chỉ của phần tử đầu tiên) bằng cách in giá trị của con trỏ mảng như sau:

#include <stdio.h>

int main(){

int nums[] = {10,20,30,44,55}, \*p;

p = nums;

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

return 0;

}

// Output : 6487552

* *Chỉ định địa chỉ trong mảng thông qua dịch chuyển con trỏ*

- Chúng ta dịch chuyển con trỏ bằng cách cộng trừ nó cho một số đơn vị. Và bằng cách dịch chuyển con trỏ, chúng ta có thể chỉ định tới một địa chỉ trong mảng mà chúng ta cần làm việc với phần tử tại địa chỉ đó.

- Ví dụ, do con trỏ p biểu thị địa chỉ trỏ tới phần tử đầu tiên (index bằng 0), nên chúng ta có thể chỉ định địa chỉ của phần tử thứ 2 (có index bằng 1) trong mảng bằng cách cộng thêm 1 đơn vị vào con trỏ, hoặc phần tử thứ 4 (có index bằng 3) trong mảng bằng cách cộng thêm 3 đơn vị vào con trỏ như sau:

#include <stdio.h>

int main(){

int nums[] = {10,20,30,44,55}, \*p;

p = nums;

printf("array[0] address: %d\n", p); //Địa chỉ phần tử đầu tiên

printf("array[1] address: %d\n", p + 1); //Địa chỉ phần tử thứ hai

printf("array[3] address: %d\n", p + 3); //Địa chỉ phần tử thứ tư

return 0;

}

- Output:

array[0] address: 6487552

array[1] address: 6487556

array[3] address: 6487564

- Hãy chú ý vào địa chỉ của array[0] và array[1]. Mặc dù dịch chuyển con trỏ 1 đơn vị, nhưng bạn có thể thấy địa chỉ trên bộ nhớ được dịch chuyển tới 4 byte như trên. Lý do là bởi 1 phần tử trong mảng số kiểu int có kích thước là 4 byte, nên khi dịch chuyển sang 1 phần tử trong mảng, thì địa chỉ trên bộ nhớ sẽ được dịch chuyển tương ứng là 4 byte.

* *Chỉ định địa chỉ bằng con trỏ để nhập phần tử trong mảng*

- Thông thường để nhập giá trị từ bàn phím vào phần tử trong mảng, chúng ta chỉ định vị trí của phần tử cần nhập thông qua index, và dùng hàm scanf() để nhập giá trị vào. Ví dụ như scanf("%d",&a[0]) hoặc scanf("%d",&a[1]) chẳng hạn.

- Tuy nhiên bằng cách sử dụng con trỏ để chỉ định địa chỉ các phần tử, chúng ta cũng có thể tiến hành nhập giá trị từ bàn phím vào các phần tử. Ví dụ:

#include <stdio.h>

int main(){

int nums[100], \*p;

p = nums;

//Nhập giá trị tại địa chỉ phần tử đầu tiên

printf("Nhap nums[0] = ");

scanf("%d", p);

//Nhập giá trị tại địa chỉ phần tử thứ hai

printf("Nhap nums[1] = ");

scanf("%d", (p + 1));

//Nhập giá trị tại địa chỉ phần tử thứ ba

printf("Nhap nums[2] = ");

scanf("%d", (p + 2));

printf("nums[0]= %d\n", nums[0]);

printf("nums[1]= %d\n", nums[1]);

printf("nums[2]= %d\n", nums[2]);

return 0;

}

- Output:

Nhap nums[0] = 2

Nhap nums[1] = 4

Nhap nums[2] = 5

nums[0]= 2

nums[1]= 4

nums[2]= 5

### 3.3 Truy xuất giá trị phần tử trong mảng bằng con trỏ

- Trong bài con trỏ C chúng ta đã biết để truy xuất giá trị tại vị trí con trỏ chỉ đến, chúng ta viết dấu hoa thị \* vào đằng trước tên con trỏ.

- Cách truy xuất các ký tự trong mảng bằng con trỏ cũng tương tự như vậy.

- Giả sử chúng ta có mảng array và cho con trỏ mảng p lưu địa chỉ của nó như sau:

int a[n], \*p;

p = a;

- Thông thường để truy cập vào phần tử trong mảng, chúng ta chỉ định index của phần tử đó, ví dụ như a[0] hoặc a[1] chẳng hạn.

- Tuy nhiên bằng cách kết hợp tên con trỏ với dấu hoa thị \*, chúng ta cũng có thể truy cập và lấy giá trị của các phần tử trong mảng array với cú pháp sau đây:

\*(p + index );

- Trong đó index là index của phần tử cần lấy giá trị trong mảng và p là con trỏ mảng.

- Ở đây, (p + index) có ý nghĩa tăng con trỏ một số index đơn vị, nhằm chỉ định địa chỉ của phần tử cần truy cập trên bộ nhớ. Và việc thêm hoa thị \*(p + index) nhằm lấy giá trị phần tử tại vị trí này.

- Và trong trường hợp index bằng 0, chúng ta có thể hiểu rằng cách viết \*p biểu thị giá trị của phần tử đầu tiên trong mảng.

- Cách truy cập vào phần tử trong mảng bằng index và bằng con trỏ có thể so sánh như bảng dưới đây:



- Ví dụ cụ thể:

#include <stdio.h>

int main(){

int nums[] = {10,20,30,44,55}, \*p;

p = nums;

printf("%d\n", nums[2]);

printf("%d\n", \*(p +2) );

printf("%d\n", nums[4]);

printf("%d\n", \*(p +4) );

return 0;

}

- Output:

30

30

55

55

3.4 Nhập xuất mảng bằng con trỏ trong c

- Bằng cách sử dụng con trỏ mảng, chúng ta có thể chỉ định vị trí các phần tử trong mảng, cũng như là truy cập và lấy giá trị của các phần tử đó.

Ứng dụng điều này, chúng ta cũng có thể nhập xuất mảng bằng con trỏ trong c như sau.

- Ví dụ: Chương trình mẫu nhập xuất mảng bằng con trỏ trong c

#include <stdio.h>

void input(int \*p, int n){

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

printf("Nhap array[%d]= ", i);

scanf("%d", (p+i));

}

}

void output(int \*p, int n){

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

printf("%d", \*(p+i));

printf("\n");}

}

int main(){

int n;

printf("Nhap n so phan tu: ");

scanf("%d", &n);

int array[n], \*p = array;

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

input(p, n);

output(p, n);

return 0;

}

- Output:

Nhap n so phan tu: 3

Nhap cac phan tu:

Nhap array[0]= 2

Nhap array[1]= 4

Nhap array[2]= 6

2

4

6

## 4. Mảng con trỏ

### 4.1 Định nghĩa mảng con trỏ 1 chiều

* Mảng con trỏ là một mảng, trong đó mỗi phần tử là một con trỏ.
* Nói cách khác, thay vì lưu giá trị như mảng bình thường, mỗi phần tử của mảng sẽ lưu một địa chỉ (pointer).

- Cú pháp:

<kiểu dữ liệu> \*ten\_mang[so\_luong];

- Ví dụ:

int \*arr[5]; // Mảng gồm 5 con trỏ kiểu int \*

char \*names[3]; // Mảng gồm 3 con trỏ kiểu char \* (thường dùng cho chuỗi)

* Tại sao phải dùng mảng con trỏ?
* Khi bạn muốn **lưu nhiều địa chỉ**, ví dụ:

- Nhiều chuỗi ký tự (vì chuỗi là mảng ký tự và thường dùng con trỏ).

- Nhiều biến động (cấp phát động).

- Truy xuất nhiều mảng nhỏ khác nhau (mảng 2 chiều kiểu "rời rạc").

Ví dụ 1: Mảng con trỏ chứa các chuỗi

#include <stdio.h>

int main() {

char \*names[] = {"An", "Binh", "Cuong"};

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

printf("Ten %d: %s\n", i + 1, names[i]);

}

return 0;

}

-Output:

Ten 1: An

Ten 2: Binh

Ten 3: Cuong

Ví dụ 2: Mảng con trỏ trỏ đến các biến int

#include <stdio.h>

int main() {

int a = 10, b = 20, c = 30;

int \*arr[3] = {&a, &b, &c};

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

printf("Gia tri arr[%d] = %d\n", i, \*arr[i]);

}

return 0;

}

- Output:

Gia tri arr[0] = 10

Gia tri arr[1] = 20

Gia tri arr[2] = 30

- Giải thích:

+ arr là mảng chứa địa chỉ của các biến a, b, c.

+ Truy xuất giá trị bằng \*arr[i].

Ví dụ 3: Mảng con trỏ dùng với cấp phát động

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*arr[3];

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

arr[i] = (int \*)malloc(sizeof(int));

\*arr[i] = (i + 1) \* 10; // Lưu 10, 20, 30

}

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

printf("arr[%d] = %d\n", i, \*arr[i]);

free(arr[i]); // Giải phóng vùng nhớ

}

return 0;

}

- Output:

arr[0] = 10

arr[1] = 20

arr[2] = 30

- Giải thích:

+ Mỗi phần tử arr[i] là một con trỏ int \* trỏ đến vùng nhớ được cấp phát bằng malloc.

+ Lưu giá trị 10, 20, 30.

### 4.2 Phân biệt mảng con trỏ với các khái niệm khác



### 4.3 Mảng con trỏ 2 chiều

- Ví dụ: Mảng con trỏ 2 chiều

#include <stdio.h>

int main() {

int row1[] = {1, 2}, row2[] = {3, 4}, row3[] = {5, 6};

int \*matrix[] = {row1, row2, row3};

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

for (int j = 0; j < 2; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

return 0;

}

- Output:

1 2

3 4

5 6

- Lưu ý: Mảng 2 chiều không đều (jagged array) có thể tạo được nhờ mảng con trỏ.

## 5. Con trỏ hàm

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

Chúng ta hãy xem một ví dụ:

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

int main() {

int (\*fptr)(int, int);

// Assign to add()

fptr = &add;

// Call the function via ptr

printf("%d", fptr(10, 5));

return 0;

}

- Output:

15

- Giải thích: Trong chương trình này, chúng ta định nghĩa hàm add() , gán địa chỉ của hàm này cho con trỏ hàm fptr và gọi hàm thông qua con trỏ để in tổng của hai số nguyên.

### 5.1 Khai báo con trỏ hàm

- Con trỏ hàm được khai báo theo chữ ký của hàm mà chúng sẽ trỏ tới. Dưới đây là cú pháp chung của khai báo con trỏ hàm :

return\_type (\*pointer\_name)(parameter\_types);

Trong đó: - return\_type: Kiểu giá trị mà hàm trả về.

- parameter\_types: Kiểu tham số mà hàm sử dụng.

- pointer\_name**:** Tên của con trỏ hàm.

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

- Kiểu của hàm được quyết định bởi kiểu trả về, số lượng và kiểu tham số. Vì vậy, con trỏ hàm phải được khai báo theo cách khớp với chữ ký của hàm mà sau đó nó trỏ tới. Ví dụ, trong đoạn mã trên, con trỏ hàm được khai báo là:

int (\*fpr)(int, int);

khớp với chữ ký của hàm add() mà nó trỏ tới sau đó.

### 5.2 Khởi tạo

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

pointer\_name = &function\_name

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

pointer\_name = function\_name;

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

### 5.3 Thuộc tính của con trỏ hàm

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

+ Trỏ tới địa chỉ bộ nhớ của một hàm trong đoạn mã.

+ Yêu cầu chữ ký hàm chính xác (kiểu trả về và danh sách tham số).

+ Có thể trỏ tới các chức năng khác nhau có chữ ký trùng khớp.

+ Không thể thực hiện các phép tính số học như tăng hoặc giảm.

+ Hỗ trợ chức năng giống mảng cho các bảng con trỏ hàm.

### 5.4 Một số ví dụ ứng dụng con trỏ hàm

- Các chương trình sau đây liệt kê một số ứng dụng phổ biến của con trỏ hàm cùng với các ví dụ mã.

- Một trong những ứng dụng hữu ích nhất của con trỏ hàm là truyền hàm làm đối số cho các hàm khác. Điều này cho phép bạn chỉ định hàm nào sẽ gọi khi chạy.

- Ví dụ 1: Con trỏ hàm làm đối số (Gọi lại)

- Giải thích: Hàm calc chấp nhận một phép toán con trỏ hàm được sử dụng để thực hiện một phép toán cụ thể (như phép cộng hoặc phép trừ) trên hai số nguyên a và b. Bằng cách truyền hàm add hoặc minus cho calc, hàm đúng sẽ được thực thi động.

#include <stdio.h>

// A simple addition function

int add(int a, int b) {

return a + b;

}

// A simple subtraction function

int subtract(int a, int b) {

return a - b;

}

void calc(int a, int b, int (\*op)(int, int)) {

printf("%d\n", op(a, b));

}

int main() {

// Passing different

// functions to 'calc'

calc(10, 5, add);

calc(10, 5, subtract);

return 0;

}

- Output:

15

5

- Ví dụ 2: Mô phỏng các hàm thành viên trong cấu trúc

Chúng ta có thể tạo một thành viên dữ liệu bên trong struct , nhưng chúng ta

không thể định nghĩa một hàm bên trong nó.

Nhưng chúng ta có thể định nghĩa các con trỏ hàm, sau đó có thể được sử dụng để gọi các hàm được gán.

#include <stdio.h>

typedef struct Rect {

int w, h;

void (\*set)(struct Rect\*, int, int);

int (\*area)(struct Rect\*);

void (\*show)(struct Rect\*);

} Rect;

int area(Rect\* r) {

return r->w \* r->h;

}

void show(Rect\* r) {

printf("Chieu rong cua hinh chu nhat: %d, "

"Chieu cao: %d\n", r->w, r->h);

}

void set(Rect\* r, int w, int h) {

r->w = w;

r->h = h;

}

void constructRect(Rect\* r) {

r->w = 0;

r->h = 0;

r->set = set;

r->area = area;

r->show = show;

}

int main() {

// Create a Rectangle object

Rect r;

constructRect(&r);

// Use r as a Rectangle

r.set(&r, 10, 5);

r.show(&r);

printf("Dien tich hinh chu nhat: %d", r.area(&r));

return 0;

}

- Output:

Chieu rong cua hinh chu nhat: 10, Chieu cao: 5

Dien tich hinh chu nhat: 50

- Bạn cũng có thể sử dụng con trỏ hàm trong mảng để triển khai một tập hợp hàm một cách động.

- Ví dụ 3: Mảng các con trỏ hàm

#include <stdio.h>

// Khai báo hàm

int add(int a, int b) {

return a + b;

}

int sub(int a, int b) {

return a - b;

}

int mul(int a, int b) {

return a \* b;

}

int divd(int a, int b) {

return a / b;

}

int main() {

// Khai báo 1 man cac con tro ham

int (\*farr[])(int, int) = {add, sub, mul, divd};

int x = 10, y = 5;

// Goi ham dong bang cach su dung ham

printf("Sum: %d\n", farr[0](x, y));

printf("Difference: %d\n", farr[1](x, y));

printf("Product: %d", farr[2](x, y));

return 0;

}

- Output:

Sum: 15

Difference: 5

Product: 50

## 6. Cấp phát động

- Cấp phát động (Dynamic memory allocation) là một kỹ thuật giúp bạn có thể xin cấp phát một vùng nhớ phù hợp với nhu cầu của bài toán trong lúc thực thi thay vì phải khai báo cố định.

- Cấp phát động thường được sử dụng để cấp phát mảng động hoặc sử dụng trong các cấu trúc dữ liệu.

- Có nhiều cấu trúc dữ liệu quan trọng dựa trên kỹ thuật này như : Danh sách liên kết, cây nhị phân hay các cấu trúc dữ liệu dạng mảng động.

- Khi bạn sử dụng cấp phát động thì vùng nhớ cấp phát sẽ là vùng nhớ heap.

- Trong ngôn ngữ lập trình C cung cấp cho bạn 4 hàm trong thư viện để bạn có thể thao tác với việc cấp phát động vùng nhớ và giải phóng vùng nhớ sau khi sử dụng, bao gồm :

* malloc()
* calloc()
* free()
* realloc()

### 6.1 Hàm malloc()

- Hàm malloc() viết tắt của từ memory allocation tức là cấp phát động vùng nhớ, hàm này được sử dụng để xin cấp phát khối bộ nhớ theo kích thước byte mong muốn.

- Giá trị trả về của hàm là một con trỏ kiểu void, bạn nên ép kiểu sang kiểu dữ liệu mà bạn cần dùng.

- Các giá trị trong các ô nhớ được cấp phát là giá trị rác

- Cú pháp : ptr = (cast\_type\*)malloc(byte\_size)

Trong đó :

* ptr là con trỏ lưu trữ ô nhớ đầu tiên của vùng nhớ được cấp phát
* cast\_type\* là kiểu con trỏ mà bạn muốn ép kiểu sang
* byte\_size là kích thước theo byte bạn muốn cấp phát

Ví dụ 1:

#include <stdio.h>

#include <stdlib.h>

int main(){

//Cấp phát vùng nhớ tương đương mảng 100 phần tử int

//sizeof(int) = 4

int \*a = (int\*)malloc(100 \* sizeof(int));

//Cấp phát vùng nhớ tương đương mảng 1000 phần tử char

//sizeof(char) = 1

char \*c = (char\*)malloc(1000 \* sizeof(char));

return 0;

}

- Trong ví dụ trên sau khi cấp phát động thì con trỏ a và c sẽ lưu giữ địa chỉ của ô nhớ đầu tiên trong các ô nhớ được cấp phát. Bạn có thể sử dụng con trỏ này như mảng cũng được.

- Toán tử sizeof trả về kích thước theo byte của kiểu dữ liệu, ví dụ bạn muốn cấp phát vùng nhớ cho 100 phần tử kiểu int thì bạn cần 400 byte. Đơn giản hơn là bạn nhân số phần tử cần cấp phát với kích thước của 1 phần tử.

- Trong trường hợp không cấp phát đủ vùng nhớ thì hàm malloc sẽ trả về con trỏ NULL.

- Ví dụ 2:

#include <stdio.h>

#include <stdlib.h>

int main(){

int n = 10;

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

if(a == NULL){

printf("Cap phat khong thanh cong !\n");

}

else{

printf("Cap phat thanh cong !\n");

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

a[i] = 28 + i; // \*(a + i) = 28 cũng được

}

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

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

}

}

return 0;

}

- Output:

Cap phat thanh cong !

28 29 30 31 32 33 34 35 36 37

### 6.2 Hàm calloc()

Hàm calloc() viết tắt của contiguous allocation tương tự như malloc() sử dụng để cấp phát vùng nhớ động nhưng các giá trị của các vùng nhớ được cấp phát sẽ có giá trị mặc định là 0 thay vì giá trị rác như hàm malloc()

Cú pháp : ptr = (cast\_type\*) calloc(n, element\_size)

Trong đó :

* ptr là con trỏ lưu trữ ô nhớ đầu tiên của vùng nhớ được cấp phát
* cast\_type\* là kiểu con trỏ mà bạn muốn ép kiểu sang
* n là số lượng phần tử bạn muốn cấp phát
* element\_size là kích thước theo byte của 1 phần tử

Ví dụ:

#include <stdio.h>

#include <stdlib.h>

int main(){

int n = 10;

int \*a = (int\*)calloc(n, sizeof(int));

if(a == NULL){

printf("Cap phat khong thanh cong !\n");

}else{

printf("Cap phat thanh cong !\n");

printf("Mang ban dau : ");

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

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

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

a[i] = 28 + i; // \*(a + i) = 28 cũng được

}

printf("\nMang sau khi thay doi : ");

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

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

}

}

return 0;

}

- Output:

Cap phat thanh cong !

Mang ban dau : 0 0 0 0 0 0 0 0 0 0

Mang sau khi thay doi : 28 29 30 31 32 33 34 35 36 37

### 6.3 Hàm free()

- Hàm malloc() và calloc() xin cấp phát vùng nhớ nhưng lại không tự giải phóng vùng nhớ mà nó xin cấp phát, hàm free() có chắc năng giải phóng vùng nhớ mà malloc() hoặc calloc() đã xin cấp phát.

- Việc sử dụng free() sau khi sử dụng malloc() và calloc() là cần thiết để tránh lãng phí bộ nhớ

- Cú pháp : free(ptr)

Ví dụ :

#include <stdio.h>

#include <stdlib.h>

int main(){

int n = 10;

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

if(a == NULL){

printf("Cap phat khong thanh cong !\n");

}

else{

printf("Cap phat thanh cong !\n");

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

a[i] = 28 + i; // \*(a + i) = 28 cũng được

}

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

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

}

free(a);

printf("\nGiai phong thanh cong !\n");

}

return 0;

}

- Output:

Cap phat thanh cong !

28 29 30 31 32 33 34 35 36 37

Giai phong thanh cong !

### 6.4 Hàm realloc()

- Hàm realloc() viết tắt của re-allocation tức là cấp phát lại, trong trường hợp sử dụng malloc() và calloc() nhưng cần bổ sung thêm bạn sử dụng realloc().

- Realloc() giúp bạn giữ lại các giá trị trên vùng nhớ cũ và bổ sung thêm vùng nhớ mới với các giá trị rác.

- Ví dụ khi bạn đang xin cấp phát 5 phần tử nhưng lại muốn cấp phát lại thành 10 phần tử và giữ nguyên giá trị của 5 phần tử trước đó. Nếu bạn sử dụng malloc() hay calloc() thì 5 phần tử cũ sẽ không còn vì bạn được cấp phát 1 vùng nhớ mới, calloc() thì chỉ bổ sung thêm vùng nhớ cho 5 phần tử mới còn 5 phần tử cũ thì vấn giữ nguyên.

- Cú pháp : ptr = (cast\_type\*)realloc(ptr, new\_size).

Ví dụ:

#include <stdio.h>

#include <stdlib.h>

int main(){

int n = 5;

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

if(a == NULL){

printf("Cap phat khong thanh cong !\n");

}

else{

printf("Cap phat thanh cong !\n");

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

a[i] = 28 + i; // \*(a + i) = 28 cũng được

}

printf("Mang truoc khi cap phat lai : ");

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

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

}

a = (int\*)realloc(a, 10);

printf("\nMang sau khi cap phat lai : ");

for(int i = 0; i < 10; i++){

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

}

}

return 0;

}

- Output:

Cap phat thanh cong !

Mang truoc khi cap phat lai : 28 29 30 31 32

Mang sau khi cap phat lai : 28 29 30 31 32 1313428052 -2116255851 46732 12350032 0

## 7. Xử lí tệp

### 7.1 Tạo file

**-** Để làm việc với file thì đầu tiên bạn cần mở file đó lên, trong C để chương trình có thể làm việc với file bạn cần thông qua 1 con trỏ kiểu FILE

- Có 2 kiểu file là file text (bạn có thể đọc được nội dung) và file nhị phân, trong bài học này mình sẽ hướng dẫn cách dùng file text trước.

- Cú pháp khai báo con trỏ FILE :

FILE \*f;

- Sau khi khai báo con trỏ file thì bạn có thể dùng nó để mở file với hàm fopen(), hàm này nằm trong thư viện

Cú pháp :

FILE \*f;

f = fopen ("ten\_file", "mode");

- Tham số thứ nhất trong fopen() là tên file mà bạn muốn mở, nếu bạn sử dụng file nằm trong cùng thư mục với file mã nguồn thì bạn chỉ cần tên file

Ví dụ :

#include <stdio.h>

int main(){

FILE \*f;

f = fopen("input.txt", "r");

}

- Nếu bạn muốn sử dụng file nằm đâu đó ở các thư mục khác với file mã nguồn thì bạn cần truyền vào tên file là đường dẫn duyệt đối.

Ví dụ :

#include <stdio.h>

int main(){

FILE \*f;

f = fopen("C:/CNTT/28Tech/input.txt", "r");

}

- Tham số thứ 2 trong hàm fopen() chính là mode mở file, nó sẽ chỉ định việc bạn mở file lên để đọc, ghi file và đang đọc ghi với file text hay file nhị phân

- Các mode mở file chính :



- Sau khi đọc ghi xong file bạn nên đóng file lại bằng hàm fclose()

Cú pháp :

FILE \*f;

fclose( f );

### 7.2 Ghi nội dung vào file

- Để ghi nội dung vào file bạn làm tương tự như bạn in nội dung ra màn hình, các hàm ghi nội dung vào file thường có thêm chữ f ở trước.

- Để ghi nội dung vào file thì bạn chỉ cần dùng hàm fprintf() là đủ.

- Hàm fprintf() được dùng để ghi nội dung vào file tương tự như hàm printf

Cú pháp : int fprintf ( FILE \* f, const char \* format, ... );

Ví dụ 1 : Ghi 1 số nguyên, 1 số thực, 1 kí tự và 1 chuỗi ký tự vào trong file output.txt

#include <stdio.h>

int main(){

FILE \*f;

f = fopen("output.txt", "w");

char s[100] = "28tech.com.vn";

printf("%d %.2lf %.c %s\n", 100, 3.14, '@', s);

fclose(f);

return 0;

}

- Output: File output.txt

100 3.14 @ 28tech.com.vn

- Ví dụ 2 : Ghi vào file prime.txt các số nguyên tố từ 1 tới 200, mỗi dòng tối đa 10 số

#include <stdio.h>

#include <math.h>

int prime(int n){

if(n < 2) return 0;

for(int i = 2; i <= sqrt(n); i++){

if(n % i == 0){

return 0;}}

return 1;

}

int main(){

FILE \*f;

f = fopen("prime.txt", "w");

int dem = 0;

for(int i = 1; i <= 200; i++){

if(prime(i)){

fprintf(f, "%d ", i);

++dem;

if(dem % 10 == 0){

//ghi dấu xuống dòng

fprintf(f, "\n"); }

}

}

fclose(f);

return 0;

}

- Output: File prime.txt

2 3 5 7 11 13 17 19 23 29

31 37 41 43 47 53 59 61 67 71

73 79 83 89 97 101 103 107 109 113

127 131 137 139 149 151 157 163 167 173

179 181 191 193 197 199

Ví dụ 3 : Mở file output.txt và ghi tiếp vào cuối file

- Để nội dung cũ trong file không bị xóa hết trước khi đọc thì bạn cần dùng mode mở là "a".

- Bạn có thể chạy mã nguồn này nhiều lần và sẽ thấy nội dung cũ được giữ nguyên và việc ghi nội dung được thực hiện tiếp tục từ cuối file

#include <stdio.h>

#include <math.h>

int main(){

FILE \*f;

f = fopen("output.txt", "a");

fprintf(f, "28tech\n");

fclose(f);

return 0;

}

- File output.txt sau 4 lần chạy mã nguồn

28tech

28tech

28tech

28tech

### 7.3 Đọc nội dung từ file

- Để đọc nội dung từ file bạn sử dụng 2 hàm chính là fscanf() và fgets(), ngoài ra còn có fgetchar().

- fscanf() - hàm này tương tự hàm scanf() mà bạn hay dùng khi đọc từ bàn phím, bây giờ bạn đọc từ file.

Cú pháp : int fscanf ( FILE \* f, const char \* format, ... );

- Để đọc được file thì bạn cần biết file được lưu trữ như thế nào

Ví dụ 1 : Đọc số nguyên int, số thực double, chuỗi ký tự không có dấu cách trong file input.txt, sau đó in ra màn hình

#include <stdio.h>

#include <math.h>

int main(){

FILE \*f;

f = fopen("input.txt", "r");

if(f == NULL){

printf("Cannot open file !\n");

}

else{

int n;

double d;

char s[1000];

fscanf(f, "%d %lf %s", &n, &d, s);

printf("Data : %d %.2lf %s\n", n, d, s);

fclose(f);

}

return 0;

}

### 7.4 Hàm fgets()

- Hàm fgets() được sử dụng tương tự hàm gets(), dùng để đọc chuỗi ký tự có dấu cách.

- Hàm này cũng bị trôi lệnh như hàm gets nên bạn cần lưu ý xử lý, ngoài ra nó còn đọc cả ký tự enter ở cuối dòng nên bạn cần loại bỏ ký tự enter này khỏi chuỗi

- Cú pháp : char \* fgets ( char \* str, int num, FILE \* f);

Tham số :

* char\* str : Xâu bạn muốn lưu nội dung đọc được
* num : Số lượng kí tự tối đa bạn muốn đọc
* f : Con trỏ file

Ví dụ 1:

#include <stdio.h>

#include <math.h>

#include <string.h>

int main(){

FILE \*f;

f = fopen("input.txt", "r");

if(f == NULL){

printf("Cannot open file !\n");

}

else{

char s[1000];

fgets(s, 1000, f);

//Xoa enter

s[strlen(s) - 1] = '\0';

printf("Data : %s", s);

fclose(f);

}

return 0;

}

Ví dụ 2 : Xử lý trôi lệnh khi dùng fgets()

- Nếu trước fgets() bạn dùng fscanf() và để dư kí tự enter sẽ bị trôi lệnh, cách xử lý là loại bỏ enter bằng hàm fgetc(), hàm này sẽ đọc 1 kí tự trong file

#include <stdio.h>

#include <math.h>

#include <string.h>

int main(){

FILE \*f;

f = fopen("input.txt", "r");

if(f == NULL){

printf("Cannot open file !\n");

}

else{

int n;

fscanf(f, "%d", &n);

//xử lý kí tự enter

fgetc(f);

char s[1000];

fgets(s, 1000, f);

//Xoa enter

s[strlen(s) - 1] = '\0';

printf("Data : %d\n%s", n, s);

fclose(f);

}

return 0;

}

Chúng ta cũng có thể thay fgetc() bằng  fscanf(f, "\n");

## 8. Kiểu cấu trúc

### 8.1 Struct

- Struct là giải pháp khi bạn cần giải quyết các bài toán thực tế khi mà đối tượng bạn cần lưu lại trong chương trình cần rất nhiều thông tin.

- Ví dụ khi bạn muốn lưu thông tin của một sinh viên thì không có kiểu dữ liệu nào trong C phù hợp cả, mà bạn cần tập hợp nhiều kiểu dữ liệu vào làm một.

- Struct hay cấu trúc là một kiểu dữ liệu mà tự bạn định nghĩa ra bằng cách gộp nhiều kiểu dữ liệu có sẵn lại nhằm mục đích có thể mô tả nhiều trường thông tin của đối tượng bạn cần lưu

- Cú pháp khai báo :

struct ten\_struct{

data\_type1 data\_field1;

data\_type2 data\_field2;

....

data\_typen data\_fieldn;

};

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

- Ví dụ bạn muốn xây dựng một cấu trúc lưu sinh viên gồm :

* Họ tên - Chuỗi ký tự không quá 50
* Email - Chuỗi ký tự không quá 50
* Tên lớp - Chuỗi ký tự không quá 30
* Điểm gpa - Số thực

- Ta sẽ xây dựng cấu trúc sinh viên như sau :

struct SinhVien{

char hoten[50];

char email[50];

char lop[30];

double gpa;

};

- Sau khi xây dựng xong cấu trúc thì bạn có thể dùng nó như kiểu dữ liệu thông thường và khai báo các biến thuộc kiểu cấu trúc này.

- Lưu ý là struct trong C khi khai báo bạn cần thêm từ khóa struct ở trước, nếu bạn không muốn khai báo từ khóa struct mỗi khi dùng cấu trúc thì có thể typedef để định nghĩa lại kiểu cho cấu trúc.

- Để truy cập vào các trường dữ liệu của cấu trúc bạn dùng toán tử '.' (dot operator)

Ví dụ 1 :

#include <stdio.h>

#include <string.h>

struct SinhVien{

char hoten[50];

char email[50];

char lop[30];

double gpa;

};

//Thay struct SinhVien = SinhVien

typedef struct SinhVien SinhVien;

int main(){

SinhVien s; // struct SinhVien cung duoc

strcpy(s.hoten, "Nguyen Van 28Tech");

strcpy(s.email, "28tech@gmail.com");

strcpy(s.lop, "CNTT");

s.gpa = 2.1;

printf("Thong tin sinh vien : \n");

printf("Ho ten : %s\n", s.hoten);

printf("Email : %s\n", s.email);

printf("Lop : %s\n", s.lop);

printf("Diem gpa : %.2lf\n", s.gpa);

return 0;

}

- Output:

Thong tin sinh vien :

Ho ten : Nguyen Van 28Tech

Email : 28tech@gmail.com

Lop : CNTT

Diem gpa : 2.10

- Bạn có thể gán 2 cấu trúc cho nhau, khi đó các trường thông tin của 2 biến struct là giống nhau.

- Ví dụ 2 : Nhập thông tin struct từ bàn phím

#include <stdio.h>

#include <string.h>

struct SinhVien{

char hoten[50];

char email[50];

char lop[30];

double gpa;

};

//Thay struct SinhVien = SinhVien

typedef struct SinhVien SinhVien;

int main(){

SinhVien s; // struct SinhVien cung duoc

printf("Nhap ten : ");

gets(s.hoten);

printf("Nhap email : ");

gets(s.email);

printf("Nhap lop : ");

gets(s.lop);

printf("Nhap gpa : ");

scanf("%lf", &s.gpa);

printf("Thong tin sinh vien : \n");

printf("Ho ten : %s\n", s.hoten);

printf("Email : %s\n", s.email);

printf("Lop : %s\n", s.lop);

printf("Diem gpa : %.2lf\n", s.gpa);

return 0;

}

### 8.2Struct lồng nhau

- Struct có thể làm kiểu dữ liệu cho một trường thông tin trong một struct khác, khi đó ta có struct trong struct

- Ví dụ bạn cần quản lý thông tin của 1 cuốn sách, cuốn sách này có tác giả cần lưu thông tin về tên và quốc tịch của tác giả đó

- Khi đó bạn hãy xây dựng trước cấu trúc lưu tác giả, sau đó xây dựng struct sách. Trong struct sách sẽ có 1 biến thuộc kiểu tác giả để lưu thông tin

Code :

#include <stdio.h>

#include <string.h>

struct TacGia{

char hoten[100];

char quoctich[100];

};

typedef struct TacGia TacGia;

struct Sach{

TacGia tg;

char tensach[100];

int giaban;

};

typedef struct Sach Sach;

int main(){

Sach s;

strcpy(s.tensach, "Hanh trinh vo dich WC 2022");

s.giaban = 500000;

strcpy(s.tg.hoten, "Lionel Messi");

strcpy(s.tg.quoctich, "Argentina");

printf("Thong tin sach : \n");

printf("Tieu de : %s\n", s.tensach);

printf("Ten tac gia : %s\n", s.tg.hoten);

printf("Quoc tich tac gia : %s\n", s.tg.quoctich);

printf("Gia ban : %dVND\n", s.giaban);

return 0;

}

- Output:

Thong tin sach :

Tieu de : Hanh trinh vo dich WC 2022

Ten tac gia : Lionel Messi

Quoc tich tac gia : Argentina

Gia ban : 500000VND

### 8.3 Struct làm tham số cho hàm

- Sau khi xây dựng xong struct thì nó cũng là một kiểu dữ liệu vì thế mà bạn có thể viết hàm có tham số là struct hoặc trả về struct.

- Trong ví dụ dưới đây mình sẽ viết hàm trả về 1 struct được nhập từ bàn phím và 1 hàm in ra thông tin của 1 struct ra màn hình.

Code :

#include <stdio.h>

#include <string.h>

struct CauThu{

char hoten[50];

char clb[50];

char quoctich[50];

int banthang, kientao;

};

typedef struct CauThu CauThu;

CauThu nhap(){

CauThu x;

printf("Nhap ho ten : "); gets(x.hoten);

printf("Nhap clb : "); gets(x.clb);

printf("Quoc tich : "); gets(x.quoctich);

printf("So ban thang, kien tao : ");

scanf("%d%d", &x.banthang, &x.kientao);

return x;

}

void xuat(CauThu x){

printf("Thong tin cau thu : \n");

printf("Ho ten : %s\n", x.hoten);

printf("CLB : %s\n", x.clb);

printf("Quoc tich : %s\n", x.quoctich);

printf("Ban thang : %d, kien tao : %d\n", x.banthang, x.kientao);

}

int main(){

CauThu s = nhap();

xuat(s);

return 0;

}

void xuat(CauThu x){

printf("Thong tin cau thu : \n");

printf("Ho ten : %s\n", x.hoten);

printf("CLB : %s\n", x.clb);

printf("Quoc tich : %s\n", x.quoctich);

printf("Ban thang : %d, kien tao : %d\n", x.banthang, x.kientao);

}

int main(){

CauThu s = nhap();

xuat(s);

return 0;

}

## 9. Danh sách liên kết

- Danh sách liên kết là một tập hợp các nút được phân bổ động, được sắp xếp theo cách mà mỗi nút chứa một giá trị và một con trỏ. Con trỏ luôn trỏ đến thành viên tiếp theo của danh sách. Nếu con trỏ là NULL, thì đó là nút cuối cùng trong danh sách.

- Danh sách liên kết được lưu giữ bằng cách sử dụng biến con trỏ cục bộ trỏ đến mục đầu tiên của danh sách. Nếu con trỏ đó cũng là NULL, thì danh sách được coi là rỗng.

- Hãy định nghĩa một nút danh sách liên kết:

typedef struct node {

int val;

struct node \* next;

} node\_t;

- Lưu ý rằng chúng ta đang định nghĩa struct theo cách đệ quy, điều này có thể thực hiện được trong C. Hãy đặt tên cho loại nút của chúng ta node\_tlà .

- Bây giờ chúng ta có thể sử dụng các nút. Hãy tạo một biến cục bộ trỏ đến mục đầu tiên của danh sách (gọi là head).

node\_t \* head = NULL;

head = (node\_t \*) malloc(sizeof(node\_t));

if (head == NULL) {

return 1;

}

head->val = 1;

head->next = NULL;

- Chúng ta vừa tạo biến đầu tiên trong danh sách. Chúng ta phải đặt giá trị và mục tiếp theo là rỗng nếu chúng ta muốn hoàn tất việc điền danh sách. Lưu ý rằng chúng ta phải luôn kiểm tra xem malloc có trả về giá trị NULL hay không.

- Để thêm một biến vào cuối danh sách, chúng ta có thể tiếp tục chuyển đến con trỏ tiếp theo:

node\_t \* head = NULL;

head = (node\_t \*) malloc(sizeof(node\_t));

head->val = 1;

head->next = (node\_t \*) malloc(sizeof(node\_t));

head->next->val = 2;

head->next->next = NULL;

- Việc này có thể tiếp tục mãi, nhưng điều chúng ta thực sự nên làm là chuyển đến mục cuối cùng của danh sách, cho đến khi nextbiến là NULL.

* *Lặp lại một danh sách*

- Hãy xây dựng một hàm in ra tất cả các mục của một danh sách. Để làm điều này, chúng ta cần sử dụng một currentcon trỏ sẽ theo dõi nút mà chúng ta hiện đang in. Sau khi in giá trị của nút, chúng ta đặt current con trỏ đến nút tiếp theo và in lại, cho đến khi chúng ta đến cuối danh sách (nút tiếp theo là NULL).

void print\_list(node\_t \* head) {

node\_t \* current = head;

while (current != NULL) {

printf("%d\n", current->val);

current = current->next;

}

}

* *Thêm một mục vào cuối danh sách*

- Để lặp lại tất cả các thành viên của danh sách liên kết, chúng ta sử dụng một con trỏ có tên là current. Chúng ta thiết lập nó bắt đầu từ phần đầu và sau đó ở mỗi bước, chúng ta chuyển con trỏ đến mục tiếp theo trong danh sách, cho đến khi chúng ta đến mục cuối cùng.

void push(node\_t \* head, int val) {

node\_t \* current = head;

while (current->next != NULL) {

current = current->next;}

current->next = (node\_t \*) malloc(sizeof(node\_t));

current->next->val = val;

current->next->next = NULL;

}

- Các trường hợp sử dụng tốt nhất cho danh sách được liên kết là ngăn xếp và hàng đợi, chúng ta sẽ triển khai chúng ngay bây giờ:

* *Thêm một mục vào đầu danh sách (đẩy vào danh sách)*

- Để thêm vào đầu danh sách, chúng ta cần thực hiện như sau:

* Tạo một mục mới và thiết lập giá trị của nó
* Liên kết mục mới để trỏ đến phần đầu của danh sách
* Đặt phần đầu của danh sách là mục mới của chúng ta

- Thao tác này sẽ tạo ra một mục mới cho danh sách với giá trị mới và giữ phần còn lại của danh sách được liên kết với mục đó.

- Vì chúng ta sử dụng một hàm để thực hiện thao tác này, chúng ta muốn có thể sửa đổi biến head. Để làm điều này, chúng ta phải truyền một con trỏ đến biến con trỏ (một con trỏ double) để chúng ta có thể sửa đổi chính con trỏ đó.

void push(node\_t \*\* head, int val) {

node\_t \* new\_node;

new\_node = (node\_t \*) malloc(sizeof(node\_t));

new\_node->val = val;

new\_node->next = \*head;

\*head = new\_node;

}

* *Xóa mục đầu tiên (nổi lên khỏi danh sách)*

- Để bật một biến, chúng ta cần đảo ngược hành động này:

1. Lấy mục tiếp theo mà đầu trỏ tới và lưu nó lại
2. Giải phóng mục đầu
3. Đặt đầu là mục tiếp theo mà chúng ta đã lưu trữ ở bên cạnh

Đây là mã:

int pop(node\_t \*\* head) {

int retval = -1;

node\_t \* next\_node = NULL;

if (\*head == NULL) {

return -1;

}

next\_node = (\*head)->next;

retval = (\*head)->val;

free(\*head);

\*head = next\_node;

return retval;

}

* *Xóa mục cuối cùng của danh sách*

- Việc xóa phần tử cuối cùng khỏi danh sách cũng rất giống với việc thêm phần tử đó vào cuối danh sách, nhưng có một ngoại lệ lớn - vì chúng ta phải thay đổi một phần tử trước phần tử cuối cùng, nên chúng ta phải xem xét hai phần tử phía trước và xem liệu phần tử tiếp theo có phải là phần tử cuối cùng trong danh sách hay không:

int remove\_last(node\_t \* head) {

int retval = 0;

/\* if there is only one item in the list, remove it \*/

if (head->next == NULL) {

retval = head->val;

free(head);

return retval;

}

/\* get to the second to last node in the list \*/

node\_t \* current = head;

while (current->next->next != NULL) {

current = current->next;

}

/\* now current points to the second to last item of the list, so let's remove current->next \*/

retval = current->next->val;

free(current->next);

current->next = NULL;

return retval;

}

* *Xóa một mục cụ thể*

- Để xóa một mục cụ thể khỏi danh sách, theo chỉ mục của mục đó từ đầu danh sách hoặc theo giá trị của mục đó, chúng ta sẽ cần duyệt qua tất cả các mục, liên tục nhìn về phía trước để tìm hiểu xem chúng ta đã đến nút trước mục mà chúng ta muốn xóa hay chưa. Điều này là do chúng ta cũng cần thay đổi vị trí đến nơi nút trước đó trỏ tới.

- Sau đây là thuật toán:

1. Lặp lại đến nút trước nút mà chúng ta muốn xóa
2. Lưu nút mà chúng ta muốn xóa trong một con trỏ tạm thời
3. Đặt con trỏ tiếp theo của nút trước đó để trỏ đến nút sau nút mà chúng ta muốn xóa
4. Xóa nút bằng con trỏ tạm thời

- Có một số trường hợp ngoại lệ mà chúng ta cần xử lý, vì vậy hãy đảm bảo rằng bạn hiểu mã.

int remove\_by\_index(node\_t \*\* head, int n) {

int i = 0;

int retval = -1;

node\_t \* current = \*head;

node\_t \* temp\_node = NULL;

if (n == 0) {

return pop(head);}

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

if (current->next == NULL) {

return -1;}

current = current->next;

}

if (current->next == NULL) {

return -1;}

temp\_node = current->next;

retval = temp\_node->val;

current->next = temp\_node->next;

free(temp\_node);

return retval;

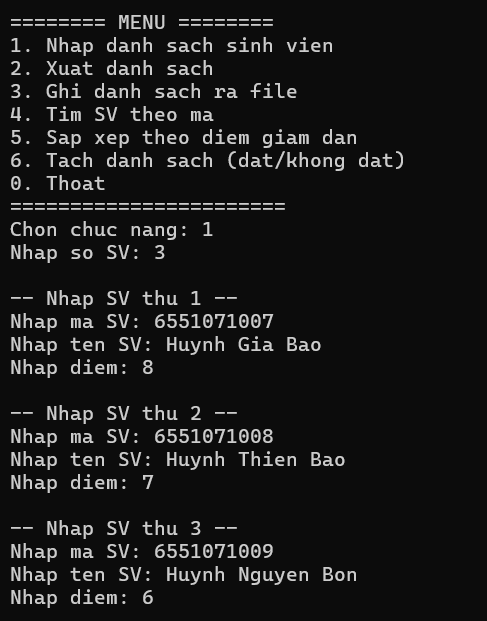
}

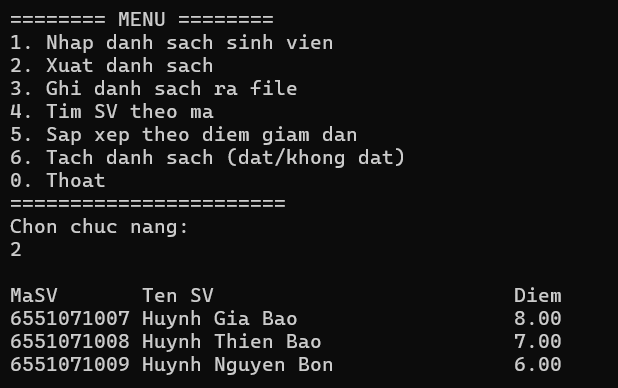
# ***B. Ứng Dụng***

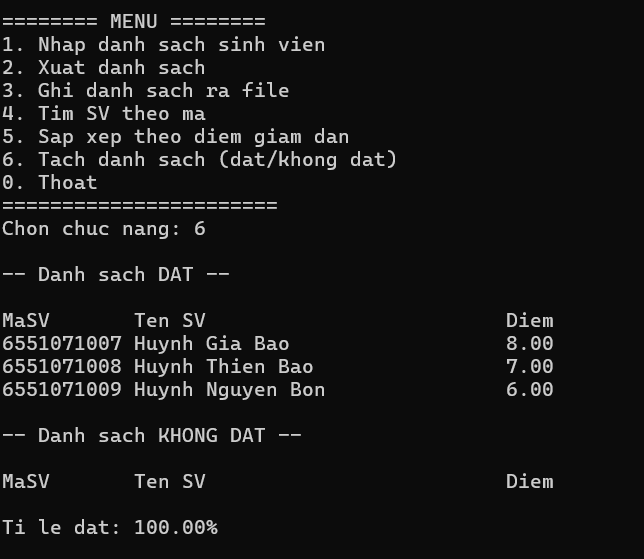
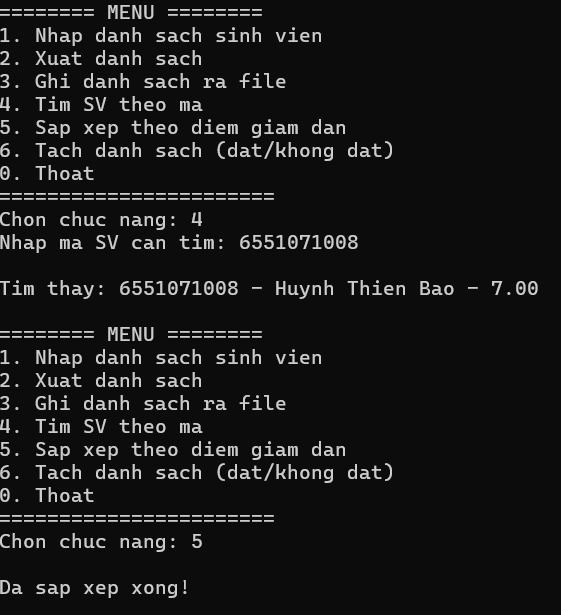
## **Bài 1:** Quản lí sinh viên

- Code: Trong Github.

- Output:



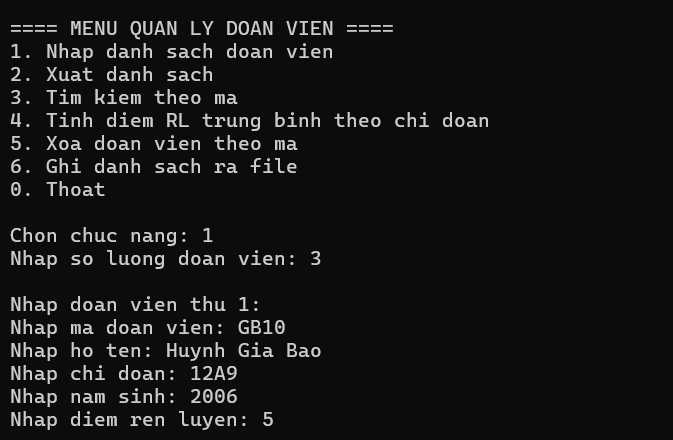


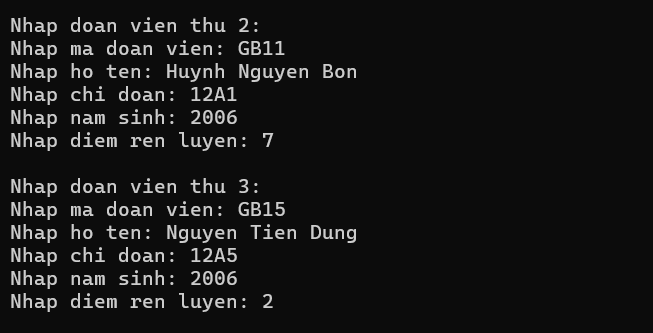


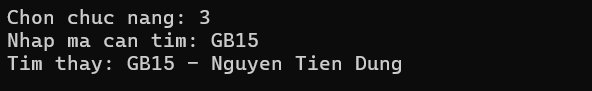
## **Bài 2:** Quản lí đoàn viên

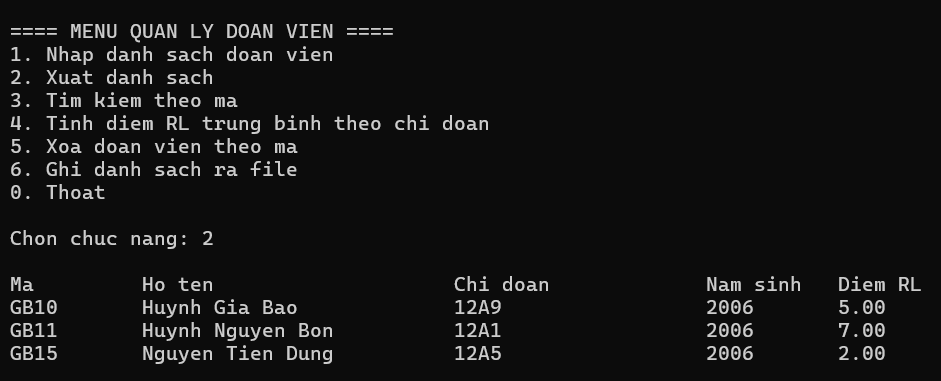
- Code: Trong Github

- Output:





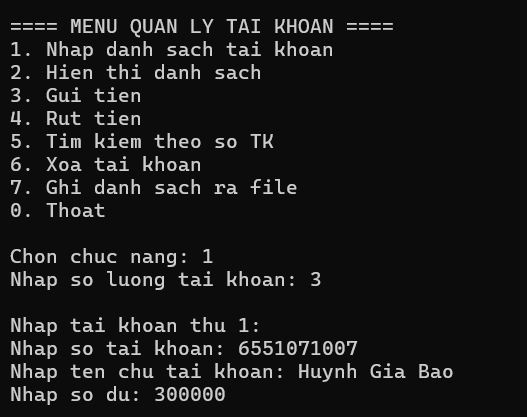


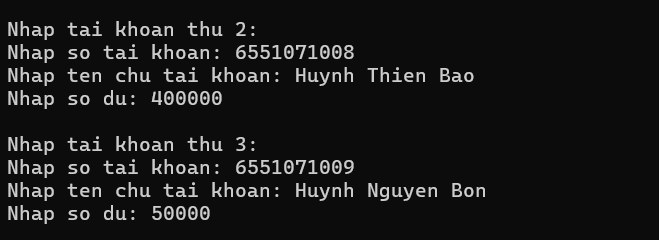


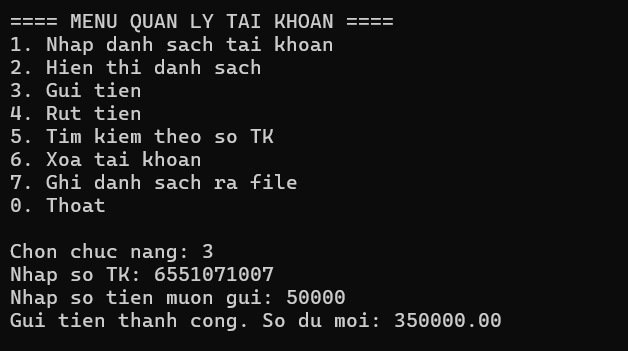
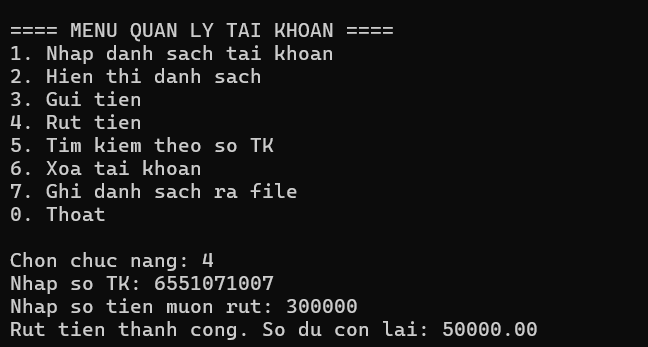
## **Bài 3:** Quản lí ngân hàng

- Code: Trong Github

- Output:



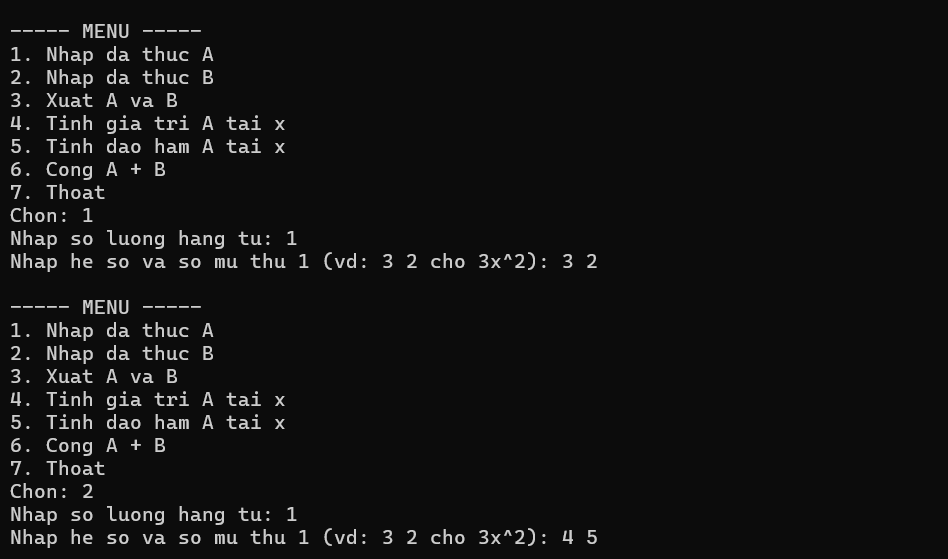
****



## **Bài 4:** Quản lí đa thức

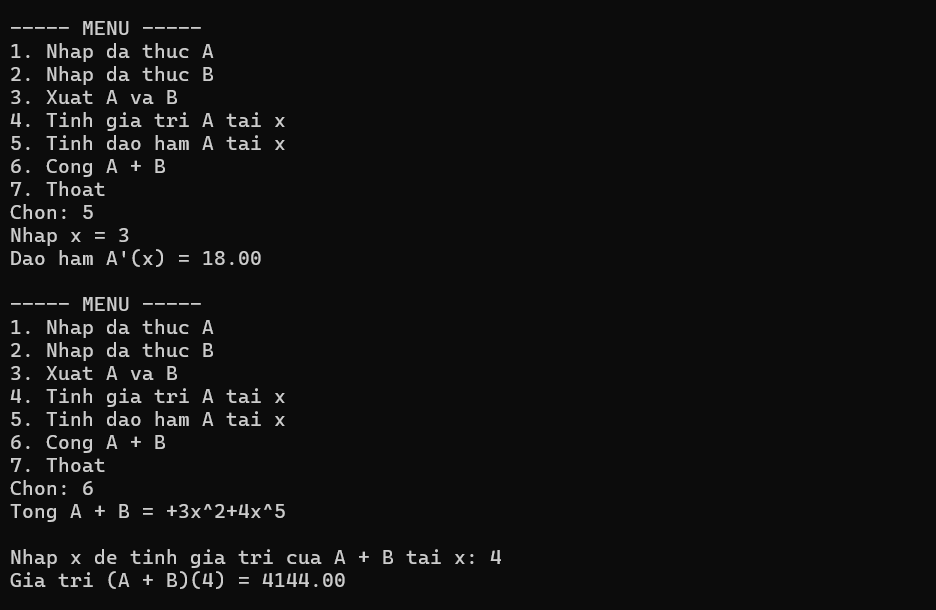
- Code: Trong Github

- Output:





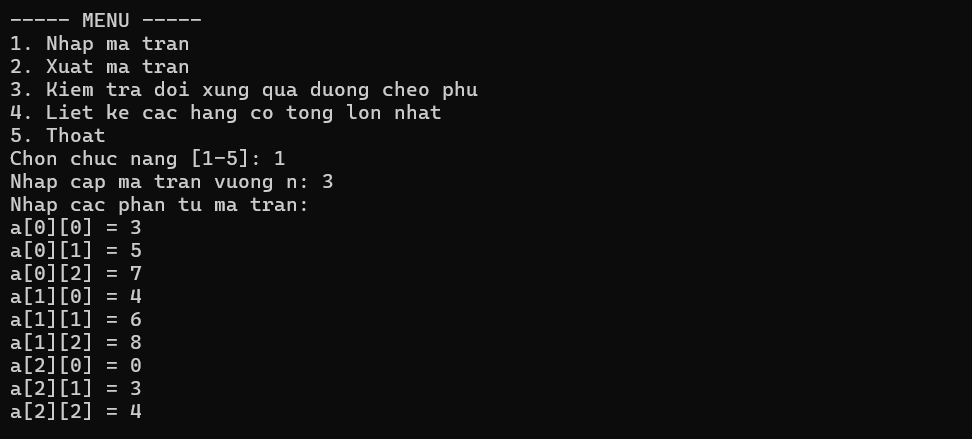


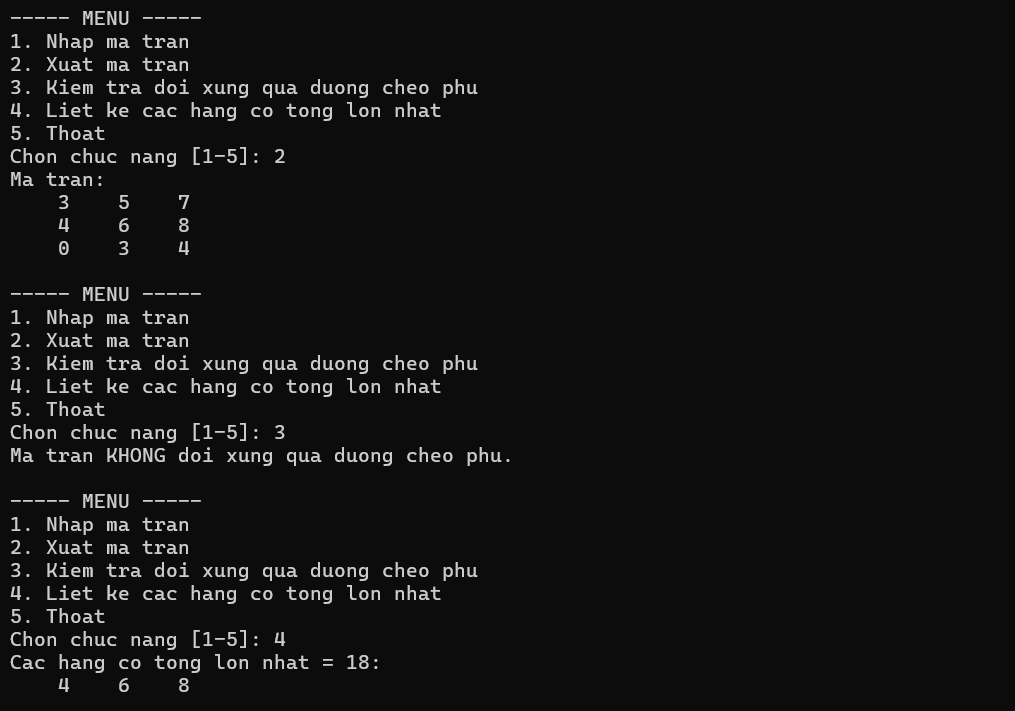


## **Bài 5:** Quản lí ma trận

- Code: Trong Github

- Output:

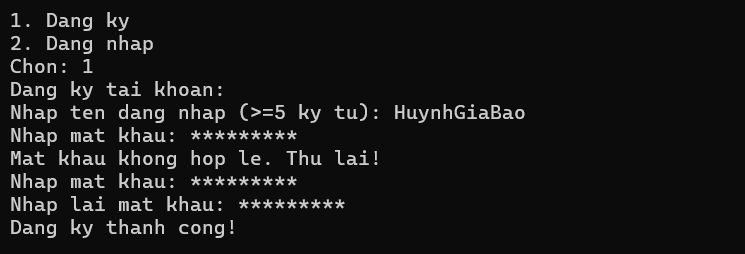




## **Bài 6:** Đăng kí và đăng nhập

- Code: Trong Gibhub

- Output:





## **Bài 7:** Quản lí món ăn

- Code: Trong Github

- Output:



