

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP.HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO TIỂU LUẬN**

**MÔN: KĨ THUẬT LẬP TRÌNH**

Giảng viên hướng dẫn: ThS. TRẦN PHONG NHÃ

Sinh viên thực hiện: THÁI TRẦN MINH HƯNG

NGUYỄN THANH HUY

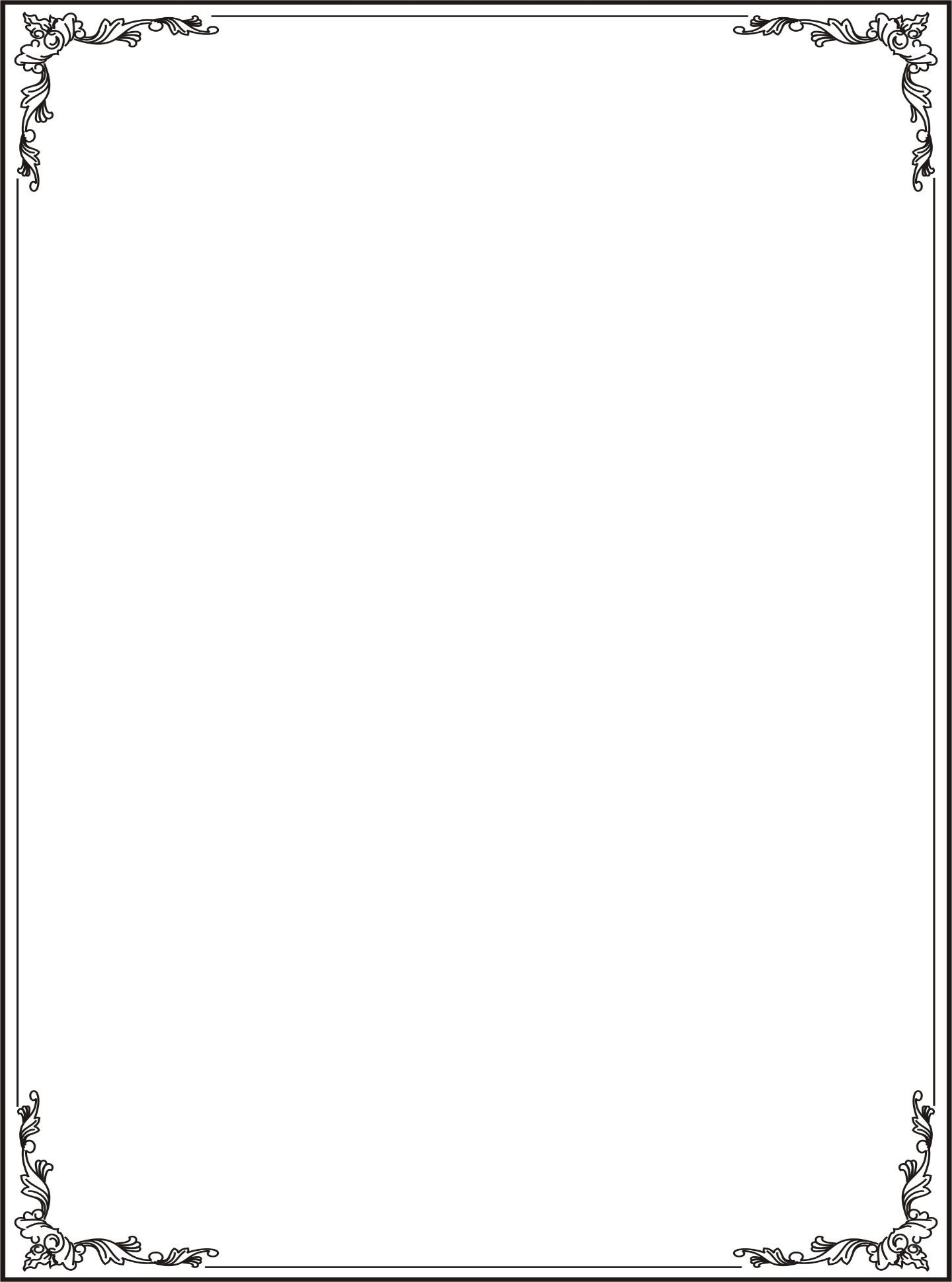
Mã số sinh viên: 6551071038

6551071036

Lớp: CQ.65.CNTT

Khóa: 65

Tp. Hồ Chí Minh, 24 tháng 5 năm 2025



**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP.HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO TIỂU LUẬN**

**MÔN: KĨ THUẬT LẬP TRÌNH**

Giảng viên hướng dẫn: ThS. TRẦN PHONG NHÃ

Sinh viên thực hiện: THÁI TRẦN MINH HƯNG

NGUYỄN THANH HUY

Mã số sinh viên: 6551071038

6551071036

Lớp: CQ.65.CNTT

Khóa: 65

Tp. Hồ Chí Minh, 24 tháng 5 năm 2025

**MỤC LỤC**

[**LỜI CẢM ƠN** 3](#_Toc198719734)

[**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN** 4](#_Toc198719735)

[**A.LÝ THUYẾT** 5](#_Toc198719736)

[**CHƯƠNG 1: HÀM** 5](#_Toc198719737)

[1.3 Ví dụ minh họa 5](#_Toc198719738)

[**CHƯƠNG 2: CON TRỎ** 7](#_Toc198719739)

[2.1 Định nghĩa 7](#_Toc198719740)

[2.2 Cú pháp 7](#_Toc198719741)

[2.3 Ví dụ minh họa 7](#_Toc198719742)

[**CHƯƠNG 3: CON TRỎ MẢNG** 8](#_Toc198719743)

[3.1 Định nghĩa 8](#_Toc198719744)

[3.2 Cú pháp 8](#_Toc198719745)

[3.3 Ví dụ minh họa 8](#_Toc198719746)

[3.3.1 Ví dụ 8](#_Toc198719747)

[3.3.2 Ví dụ 9](#_Toc198719748)

[3.3.3 Ví dụ 9](#_Toc198719749)

[**CHƯƠNG 4: MẢNG CON TRỎ** 10](#_Toc198719750)

[4.1 Định nghĩa 10](#_Toc198719751)

[4.2 Cú pháp 10](#_Toc198719752)

[4.3 Ví dụ minh họa 10](#_Toc198719753)

[**CHƯƠNG 5: CON TRỎ HÀM** 11](#_Toc198719754)

[5.1 Định nghĩa 11](#_Toc198719755)

[5.2 Cú pháp 11](#_Toc198719756)

[5.3 Ví dụ minh họa 11](#_Toc198719757)

[**CHƯƠNG 6: CẤP PHÁT ĐỘNG** 13](#_Toc198719758)

[6.1 Định nghĩa 13](#_Toc198719759)

[6.2 Hàm malloc() 13](#_Toc198719760)

[6.3 Hàm calloc() 15](#_Toc198719761)

[6.4 Hàm free() 16](#_Toc198719762)

[6.5 Hàm realloc() 17](#_Toc198719763)

[**CHƯƠNG 7: XỬ LÝ TỆP** 19](#_Toc198719764)

[7.1 Tạo file 19](#_Toc198719765)

[7.2 Ghi nội dung vào file 21](#_Toc198719766)

[7.3 Đọc nội dung từ file 22](#_Toc198719767)

[**CHƯƠNG 8: KIỂU CẤU TRÚC** 25](#_Toc198719768)

[8.1 Định nghĩa 25](#_Toc198719769)

[8.2 Cú pháp 25](#_Toc198719770)

[8.3 Ví dụ minh họa 25](#_Toc198719771)

[**CHƯƠNG 9: DANH SÁCH LIÊN KẾT** 30](#_Toc198719772)

[9.1 Định nghĩa 30](#_Toc198719773)

[9.2 Danh sách liên kết đơn 30](#_Toc198719774)

[9.2.1 Cấu trúc node trong DSLK đơn: 31](#_Toc198719775)

[9.2.2 Tạo một node mới 31](#_Toc198719776)

[9.2.3 Thêm phần tử vào đầu danh sách 31](#_Toc198719777)

[9.2.4 In danh sách 32](#_Toc198719778)

[9.2.5 Giải phóng bộ nhớ 32](#_Toc198719779)

[9.2.6 Xóa phần tử theo giá trị 32](#_Toc198719780)

[9.2.7 Ví dụ minh họa 33](#_Toc198719781)

[9.3 Danh sách liên kết đôi 35](#_Toc198719782)

[9.3.1 Cấu trúc node trong DSLK đôi 35](#_Toc198719783)

[9.3.2 Thêm phần tử vào đầu danh sách 35](#_Toc198719784)

[9.3.3 In danh sách 36](#_Toc198719785)

[9.3.4 Giải phóng bộ nhớ 36](#_Toc198719786)

[9.3.5 Xóa phần tử đầu 36](#_Toc198719787)

[9.3.6 Ví dụ 37](#_Toc198719788)

[9.4 Danh sách liên kết vòng 38](#_Toc198719789)

[9.4.1 Tạo danh sách liên kết vòng từ danh sách liên kết đơn 38](#_Toc198719790)

[9.4.2 Tạo danh sách liên kết vòng từ danh sách liên kết đôi 41](#_Toc198719791)

[**B.ỨNG DỤNG** 44](#_Toc198719792)

[**TÀI LIỆU THAM KHẢO** 45](#_Toc198719793)

# **LỜI CẢM ƠN**

Lời đầu tiên, em xin gửi tới quý thầy cô bộ môn công nghệ thông tin nói riêng và toàn thể giảng viên của trường Đại Học Giao Thông Vận Tải phân hiệu tại thành phố Hồ Chí Minh lời chúc sức khoẻ trân trọng nhất.

Em chân thành cảm ơn thầy Trần Phong Nhã đã tạo cơ hội cũng như truyền tải cho em những kiến thức cần thiết để em nghiên cứu và hoàn thành bài báo cáo môn học kĩ thuật lập trình. Qua đó giúp em có thêm cơ hội mở rộng kiến thức về lĩnh vực công nghệ thông tin mà mình đang theo học.

Tuy đã cố gắng nghiên cứu nhưng vẫn còn nhiều thiếu sót. Vì vậy em rất mong sẽ nhận được sự đóng góp ý kiến của quý thầy cô để bài báo của em được hoàn chỉnh.

Lời cuối cùng, em xin cảm ơn quý thầy cô bộ môn công nghệ thông tin. Đặc biệt là thầy Trần Phong Nhã đã giúp đỡ và hỗ trợ em trong quá trình nghiên cứu và hoàn thành bài báo cáo.

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

***Tp. Hồ Chí Minh, ngày … tháng … năm …..***

|  |  |
| --- | --- |
|  | **Giáo viên hướng dẫn** |
|  | **ThS. Trần Phong Nhã** |

# **A.LÝ THUYẾT**

# **CHƯƠNG 1: HÀM**

* 1. Định nghĩa

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

* 1. 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

1.3 Ví dụ minh họa

Ví dụ : Hàm tính và in tổng của 3 số (có 3 tham số là a, b, c )

#include<stdio.h>

void inTong(int a, int b, int c){ //tạo hàm

int sum = a + b + c;

printf(“%d”, sum);

}

int main(){

int a = 1;

int b = 2;

int c = 3;

inTong(a, b, c); //gọi hàm

return 0;

}

# **CHƯƠNG 2: CON TRỎ**

## 2.1 Định nghĩa

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.

2.2 Cú pháp

Kiểu\_Dữ\_Liệu \*Tên\_Biến\_Con\_Trỏ;

2.3 Ví dụ minh họa

#include <stdio.h>

int main() {

int a = 10; // Biến thường

int \*p; // Khai báo con trỏ p

p = &a; // Gán địa chỉ của biến a cho con trỏ p

printf("Giá trị của a: %d\n", a);

printf("Địa chỉ của a: %p\n", &a);

printf("Giá trị của con trỏ p: %p\n", p);

printf("Giá trị mà con trỏ p trỏ tới: %d\n", \*p); // Truy xuất giá trị thông qua con trỏ

return 0;

}

# **CHƯƠNG 3: CON TRỎ MẢNG**

## 3.1 Định nghĩa

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.

Bằng cách sử dụng **con trỏ mảng trong C**, chúng ta có thể truy cập vào địa chỉ của mảng mà nó chỉ đến trong bộ nhớ, cũng như là thao tác và xử lý trực tiếp với mảng này. Lại nữa, chúng ta cũng có thể sử dụng con trỏ mảng để xử lý gián tiếp mảng trong hàm, qua đó nâng cao việc sử dụng hiệu quả bộ nhớ và giảm thời gian thực thi chương trình.

## 3.2 Cú pháp

Kiểu\_Dữ\_Liệu \*Tên\_Biến\_Con\_Trỏ = Tên\_Mảng;

Lưu ý:

* Kiểu\_Dữ\_Liệu của con trỏ phải giống với kiểu của mảng mà nó gán địa chỉ.
* 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:

int nums[] = {1,2,3,4}, \*p ;

p = nums ;

## 3.3 Ví dụ minh họa

Mảng 1 chiều A[] thì con trỏ trỏ tới phần tử A[i] là A + i, vậy (A + i) tương đương với &A[i] hay địa chỉ của A[i] và \*(A + i) tương đương với A[i].

### 3.3.1 Ví dụ

#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;}

### 3.3.2 Ví dụ

/\*Tạo hàm nhập mảng bằng con trỏ trong C\*/

void input\_array(int \*array, int length){

//array: tên mảng

//length: độ dài mảng

for (short i = 0; i < length; i++) scanf("%d", (array + i));

}

### 3.3.3 Ví dụ

/\*Tạo hàm xuất mảng bằng con tror chiều trong C\*/

void show\_array(int \*array, int length){

//array: tên mảng

//length: độ dài mảng

for(short i = 0; i < length; i++) printf("%d ", \*(array + i));

printf("\n");

}

# **CHƯƠNG 4: MẢNG CON TRỎ**

## 4.1 Định nghĩa

Mảng con trỏ là 1 mảng mà phần tử trong mảng là con trỏ

## 4.2 Cú pháp

Kiểu\_Dữ\_Liệu \*Tên\_Mảng[Số\_Lượng];

## 4.3 Ví dụ minh họa

#include <stdio.h>

int main() {

int x = 10, y = 20, z = 30;

int \*arr[3]; // Mảng 3 con trỏ

arr[0] = &x;

arr[1] = &y;

arr[2] = &z;

// In ra giá trị thông qua con trỏ

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

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

}

return 0;

}

# **CHƯƠNG 5: CON TRỎ HÀM**

## 5.1 Định nghĩa

Con trỏ hàm là biến dùng để lưu trữ địa chỉ hàm, khi muốn sử dụng thì chỉ cần gọi con trỏ.

Con trỏ tới một hàm cho phép bạn tự động quyết định hàm nào sẽ được gọi trong thời gian chạy, thay vì thời gian biên dịch. Điều này mang lại sự linh hoạt và có thể được sử dụng để triển khai các lệnh gọi lại (callbacks), trình xử lý sự kiện (event handlers), và để triển khai các cấu trúc mã trừu tượng hơn như máy trạng thái (State Machine).

## 5.2 Cú pháp

Kiểu\_Dữ\_Liệu (\*Tên\_Con\_Trỏ)(Tham\_Số);

## 5.3 Ví dụ minh họa

#include <stdio.h>

// khai báo một hàm đơn giản để được trỏ tới

void sayHello(){

printf("Hello World");

}

// một hàm khác cũng sẽ được trỏ tới

void sayGoodbye(){

printf("Goodbye, World!\n");

}

int main(){

// khai báo một con trỏ hàm void

void (\*funcPtr)();

funcPtr = sayHello;

// gọi hàm thông qua con trỏ

funcPtr(); // -> in ra: Hello World

funcPtr = sayGoodbye;

funcPtr(); // -> in ra: Goodbye, World!

return 0;

}

# **CHƯƠNG 6: CẤP PHÁT ĐỘNG**

## 6.1 Định nghĩa

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()

## 6.2 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

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

Ví dụ minh họa

#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;

}

## 6.3 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 minh họa

#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;

}

## 6.4 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 minh họa

#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;

}

## 6.5 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 minh họa

#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;}

# **CHƯƠNG 7: XỬ LÝ 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.

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 :

|  |  |  |
| --- | --- | --- |
| Mode | Ý nghĩa | Chú ý |
| r | Mở file để đọc | Nếu file không tồn tại thì hàm fopen() trả về con trỏ NULL |
| rb | Mở file để đọc theo kiểu file nhị phân | Nếu file không tồn tại thì hàm fopen() trả về con trỏ NULL |
| w | Mở file để ghi | Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới |
| wb | Mở file để ghi theo kiểu file nhị phân | Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới |
| a | Mở file text lên để ghi tiếp vào cuối file mà không xóa nội dung cũ trong file | Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới |
| ab | Mở file nhị phân lên để ghi tiếp vào cuối file mà không xóa nội dung cũ trong file | Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới |
| r+ | Mở file để vừa đọc vừa ghi | Nếu file không tồn tại thì hàm fopen() trả về con trỏ NULL |
| rb+ | Mở file để vừa đọc vừa ghi theo kiểu nhị phân | Nếu file không tồn tại thì hàm fopen() trả về con trỏ NULL |
| w+ | Mở file để vừa đọc vừa ghi | Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới |
| wb+ | Mở file để vừa đọc vừa ghi theo kiểu nhị phân | Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới |
| a+ | Mở file lên vừa để đọc và ghi vào cuối file | Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới |
| ab+ | Mở file để vừa đọc vừa ghi vào cuối file theo kiểu nhị phân | Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới |

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.

Cú pháp:

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

Ví dụ: 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;

}

## 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()

Hàm 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, ... );

Ví dụ: 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;

}

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:

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:

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;

}

# **CHƯƠNG 8: KIỂU CẤU TRÚC**

## 8.1 Định nghĩa

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.

## 8.2 Cú pháp

struct ten\_struct{

data\_type1 data\_field1;

data\_type2 data\_field2;

....

data\_typen data\_fieldn;

};

## 8.3 Ví dụ minh họa

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).

#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;

}

Ví dụ: struct lồng nhau

#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;

}

Ví dụ: struct làm tham số cho hàm

#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;

}

# **CHƯƠNG 9: DANH SÁCH LIÊN KẾT**

## 9.1 Định nghĩa

Danh sách liên kết (linked list) là một cấu trúc dữ liệu linh hoạt và phổ biến trong lập trình. Nó cho phép lưu trữ dữ liệu dưới dạng các nút (node) liên kết với nhau thông qua các con trỏ. Danh sách liên kết có 3 dạng: danh sách liên kết đơn, danh sách liên kết đôi, danh sách liên kết vòng.

## 9.2 Danh sách liên kết đơn

Danh sách liên kết đơn là một cấu trúc dữ liệu, nó tương tự như một mảng động với những tính chất quan trọng như sau:

* Mở rộng và thu hẹp một cách linh hoạt
* Các phần tử trong DSLK gọi là node và được cấp phát động khi cần
* Số lượng phần tử trong DSLK phụ thuộc vào bộ nhớ heap
* Dễ dàng chèn và xóa phần tử
* Các phần tử trong DSLK không có thứ tự
* Truy cập phần tử trong DSLK cần truy cập tuần tự không thể truy cập qua chỉ số
* Mỗi node trong DSLK cần có thêm 1 con trỏ để lưu liên kết

Mỗi phần tử trong DSLK được gọi là một node hay nút, node sẽ lưu thông tin dữ liệu (ví dụ như một số nguyên, 1 chuỗi ký tự, 1 sinh viên...) và ngoài ra cần có phần liên kết, phần liên kết này giúp các node có thể liên lạc với nhau. Mỗi node sẽ lưu thêm địa chỉ của node phía sau nó trong DSLK thông qua 1 thuộc tính con trỏ.

Node cuối cùng trong danh sách liên kết thì phần liên kết của nó sẽ lưu con trỏ NULL.

Để xây dựng một node cho DSLK bạn dùng struct, mỗi khi bạn cần tạo ra một node mới trong DSLK thì bạn cần cấp phát động(dùng malloc).

Một node trong DSLK đơn gồm 2 phần :

* Dữ liệu của node lưu thông tin của một node, có thể là kiểu dữ liệu như số, chuỗi, sinh viên, ...
* Phần liên kết - Đây là một con trỏ để lưu địa chỉ của node kế tiếp nó trong DSLK, thường đặt tên là next.

### 9.2.1 Cấu trúc node trong DSLK đơn:

#include <stdio.h>

#include <stdlib.h>

// Định nghĩa cấu trúc node

struct Node {

int data;

struct Node\* next;

};

### 9.2.2 Tạo một node mới

struct Node\* createNode(int value) {

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

if (!newNode) {

printf("Lỗi cấp phát bộ nhớ\n");

exit(1);

}

newNode->data = value;

newNode->next = NULL;

return newNode;

}

### 9.2.3 Thêm phần tử vào đầu danh sách

void insertAtHead(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

newNode->next = \*head;

\*head = newNode;

}

### 9.2.4 In danh sách

void printList(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

printf("%d -> ", current->data);

current = current->next;

}

printf("NULL\n");

}

### 9.2.5 Giải phóng bộ nhớ

void freeList(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

struct Node\* next = current->next;

free(current);

current = next;

}

}

### 9.2.6 Xóa phần tử theo giá trị

void deleteByValue(struct Node\*\* head, int value) {

if (\*head == NULL) {

printf("Danh sách rỗng, không thể xóa.\n");

return;

}

struct Node\* current = \*head;

struct Node\* prev = NULL;

// Nếu nút đầu chứa giá trị cần xóa

if (current->data == value) {

\*head = current->next;

free(current);

return;

}

// Duyệt danh sách để tìm nút chứa giá trị

while (current != NULL && current->data != value) {

prev = current;

current = current->next;

}

// Nếu không tìm thấy giá trị

if (current == NULL) {

printf("Không tìm thấy giá trị %d trong danh sách.\n", value);

return;

}

// Bỏ qua nút cần xóa

prev->next = current->next;

free(current);

}

### 9.2.7 Ví dụ minh họa

int main() {

struct Node\* head = NULL;

insertAtHead(&head, 10);

insertAtHead(&head, 20);

insertAtHead(&head, 30);

insertAtHead(&head, 40);

printf("Danh sách ban đầu:\n");

printList(head); // 40 -> 30 -> 20 -> 10 -> NULL

deleteByValue(&head, 30);

printf("Sau khi xóa giá trị 30:\n");

printList(head); // 40 -> 20 -> 10 -> NULL

deleteByValue(&head, 100); // Không tồn tại

deleteByValue(&head, 40); // Xoá đầu danh sách

printf("Sau khi xóa thêm:\n");

printList(head); // 20 -> 10 -> NULL

freeList(head);

return 0;

}

Chú ý:

* Mỗi node trong DSLK được cấp phát động
* Mỗi node trong DSLK thực chất là một con trỏ, nó là địa chỉ một ô nhớ mà ô nhớ đó được sử dụng để lưu trữ thông tin về một node.
* Sử dụng toán tử -> khi bạn cần truy cập vào data và next của một node trong DSLK thông qua con trỏ quản lý node đó
* DSLK thực chất là 1 danh sách quản lý các con trỏ kiểu node

## 9.3 Danh sách liên kết đôi

Node mở đầu gọi là Head để nhận biết vị trí bắt đầu của dãy, mỗi node sẽ có hai con trỏ prev, next trong đó next là con trỏ được trỏ đến phần tử kế tiếp, prev là con trỏ được trỏ đến phần tử liền trước. Với phần tử đầu tiên (head), prev trỏ đến NULL, với phần tử cuối cùng, next trỏ đến NULL.

### 9.3.1 Cấu trúc node trong DSLK đôi

#include <stdio.h>

#include <stdlib.h>

// Cấu trúc node của danh sách liên kết đôi

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

### 9.3.2 Thêm phần tử vào đầu danh sách

void insertAtHead(struct Node\*\* head, int value) {

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

newNode->data = value;

newNode->prev = NULL;

newNode->next = \*head;

if (\*head != NULL)

(\*head)->prev = newNode;

\*head = newNode;

}

### 9.3.3 In danh sách

void printList(struct Node\* head) {

struct Node\* temp = head;

printf("Danh sách: ");

while (temp != NULL) {

printf("%d <-> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

### 9.3.4 Giải phóng bộ nhớ

void freeList(struct Node\* head) {

struct Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

}

### 9.3.5 Xóa phần tử đầu

void deleteHead(struct Node\*\* head) {

if (\*head == NULL)

return;

struct Node\* temp = \*head;

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

if (\*head != NULL)

(\*head)->prev = NULL;

free(temp);

}

### 9.3.6 Ví dụ

int main() {

struct Node\* head = NULL;

insertAtHead(&head, 10);

insertAtHead(&head, 20);

insertAtHead(&head, 30);

printList(head); // 30 <-> 20 <-> 10 <-> NULL

deleteHead(&head);

printList(head); // 20 <-> 10 <-> NULL

freeList(head);

return 0;

}

## 9.4 Danh sách liên kết vòng

Danh sách liên kết vòng (Circular Linked List) là một biến thể của Danh sách liên kết (Linked List), trong đó phần tử đầu tiên trỏ tới phần tử cuối cùng và phần tử cuối cùng trỏ tới phần tử đầu tiên.

Cả hai loại Danh sách liên kết đơn (Singly Linked List) và Danh sách liên kết đôi (Doubly Linked List) đều có thể được tạo thành dạng Danh sách liên kết vòng.

### 9.4.1 Tạo danh sách liên kết vòng từ danh sách liên kết đơn

#include <stdio.h>

#include <stdlib.h>

// Cấu trúc node của danh sách liên kết đơn

struct Node {

int data;

struct Node\* next;

};

// Hàm tạo node mới

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->next = NULL;

return newNode;

}

// Thêm phần tử vào cuối danh sách đơn

void insertEnd(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

// Chuyển danh sách đơn thành danh sách liên kết vòng

void convertToCircular(struct Node\* head) {

if (head == NULL) return;

struct Node\* temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = head; // Nối node cuối về node đầu

}

// In danh sách liên kết vòng (dùng do-while)

void printCircularList(struct Node\* head) {

if (head == NULL) {

printf("Danh sách rỗng.\n");

return;

}

struct Node\* temp = head;

do {

printf("%d -> ", temp->data);

temp = temp->next;

} while (temp != head);

printf("(vòng lại)\n");

}

// Chương trình chính

int main() {

struct Node\* head = NULL;

// Tạo danh sách liên kết đơn

insertEnd(&head, 10);

insertEnd(&head, 20);

insertEnd(&head, 30);

printf("Trước khi chuyển thành danh sách liên kết vòng:\n");

struct Node\* temp = head;

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

// Chuyển sang danh sách liên kết vòng

convertToCircular(head);

printf("Sau khi chuyển thành danh sách liên kết vòng:\n");

printCircularList(head);

return 0;

}

### 9.4.2 Tạo danh sách liên kết vòng từ danh sách liên kết đôi

#include <stdio.h>

#include <stdlib.h>

// Cấu trúc node của danh sách liên kết đôi

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

// Tạo một node mới

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

// Thêm node vào cuối danh sách liên kết đôi

void insertEnd(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

newNode->prev = temp;

}

// Chuyển danh sách liên kết đôi thành danh sách liên kết đôi vòng

void convertToCircular(struct Node\* head) {

if (head == NULL) return;

struct Node\* last = head;

while (last->next != NULL)

last = last->next;

// Nối node cuối với node đầu

last->next = head;

head->prev = last;

}

// In danh sách liên kết đôi vòng (dùng do-while để tránh lặp vô tận)

void printCircularDoublyList(struct Node\* head) {

if (head == NULL) {

printf("Danh sách rỗng.\n");

return;

}

struct Node\* temp = head;

printf("Danh sách liên kết đôi vòng:\n");

do {

printf("%d <-> ", temp->data);

temp = temp->next;

} while (temp != head);

printf("(vòng lại)\n");

}

int main() {

struct Node\* head = NULL;

insertEnd(&head, 10);

insertEnd(&head, 20);

insertEnd(&head, 30);

printf("Trước khi chuyển thành danh sách liên kết đôi vòng:\n");

struct Node\* temp = head;

while (temp != NULL) {

printf("%d <-> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

convertToCircular(head); // Chuyển sang dạng vòng

printCircularDoublyList(head); // In ra danh sách vòng

return 0;

}

**B.ỨNG DỤNG**

XÂY DỰNG ỨNG DỤNG CHO VIỆC QUẢN LÍ THÔNG TIN SINH VIÊN

Các tính năng cần thiết:

1. Thêm thông tin sinh viên.
2. Xóa thông tin sinh viên.
3. In danh sách sinh viên.
4. In danh sách sinh viên theo ngành.
5. In danh sách sinh viên theo GPA.
6. Chỉnh sửa thông tin sinh viên.
7. Tìm kiếm sinh viên theo mã sinh viên.
8. Tìm kiếm sinh viên theo tên.

# **TÀI LIỆU THAM KHẢO**

[1].https://blog.28tech.com.vn/c-ham “[C]. Hàm – 28Tech”.

[2].https://blog.28tech.com.vn/c-ly-thuyet-con-tro “[C]. Lý Thuyết Con Trỏ - 28Tech”.

[3].https://laptrinhcanban.com/c/lap-trinh-c-co-ban/mang-trong-c/con-tro-mang-trong-c/ “Con trỏ mảng trong c – Laptrinhcanban.com”.

[4].https://vietjack.com/lap\_trinh\_c/mang\_con\_tro\_trong\_c.jsp “Mảng Con trỏ trong C (chi tiết nhất)”.

[5].https://viblo.asia/p/con-tro-ham-cc-obA46dlwLKv “Con Trỏ Hàm – C/C++”.

[6].https://blog.28tech.com.vn/c-malloc-calloc-free-va-realloc “[C]. malloc(), calloc(), free() và realloc()”.

[7].https://blog.28tech.com.vn/c-doc-ghi-file-text “[C]. Đọc Ghi File Text - 28Tech”.

[8].https://blog.28tech.com.vn/c-struct-kieu-cau-truc “[C]. Struct - Kiểu Cấu Trúc - 28Tech”.

[9].https://blog.28tech.com.vn/dslk-don-them-node-vao-dau-dslk “[DSLK Đơn]. Danh Sách Liên Kết Đơn - Singly Linked List - 28Tech”.

<https://howkteam.vn/course/cau-truc-du-lieu-va-giai-thuat/linked-list-4273> “Linked List | How Kteam”.

<https://vietjack.com/cau-truc-du-lieu-va-giai-thuat/cau-truc-du-lieu-danh-sach-lien-ket-vong.jsp> “Cấu trúc dữ liệu Danh sách liên kết vòng (Circular Linked List)”.