TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN:KỸ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn:	ThS.TRẦN PHONG NHÃ
Sinh viên thực hiện:	PHẠM THÀNH ĐẠT
Mã sinh viên:	6551071020
Lớp:	CQ.CNTT.65
Khóa:	65

Tp.Hồ Chí Minh,năm 2025

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN:KỸ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn:	ThS.TRẦN PHONG NHÃ
Sinh viên thực hiện:	PHẠM THÀNH ĐẠT
Mã sinh viên:	6551071020
Lớp:	CQ.CNTT.65
Khóa:	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 thầy đã giúp đỡ tạo điều kiện để em hoàn thành bài tập lớn này. 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 tập lớn 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 GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp.Hồ Chí Minh,ngày....tháng....năm 2025

Giáo viên hướng dẫn

ThS.TRẦN PHONG NHÃ

MỤC LỤC

LỜI CẢM ƠN ..... **i**

NHẬN XÉT CỦA GIẢNG VIÊN ..... **ii**

MỤC LỤC ..... **iii**

A.LÝ THUYẾT **1**

1. HÀM **1**

2. CON TRỎ **3**

3. CON TRỎ MẢNG **5**

4. MẢNG CON TRỎ **6**

5. CON TRỎ HÀM **7**

6. CẤP PHÁT ĐỘNG **9**

7. XỬ LÍ TỆP **11**

8.KIỂU CẤU TRÚC **13**

9. DANH SÁCH LIÊN KẾT **15**

B.ỨNG DỤNG **17**

KẾT LUẬN **32**

TÀI LIỆU THAM KHẢO **33**

A.LÝ THUYẾT

1. HÀM

1.1 Khái niệm: Hàm (function) là một các khối lệnh có nhiệm vụ thực hiện một chức năng nào đó.Hàm có thể chia thành hai loại:

- Hàm được xây dựng sẵn: Là những hàm có sẵn trong ngôn ngữ lập trình, được cung cấp bởi hệ thống hoặc thư viện tiêu chuẩn. Ví dụ, trong code C, printf() là một hàm được xây dựng sẵn để in ra màn hình.
- Hàm do người dùng tự định nghĩa: Là những hàm mà người dùng tự viết để thực hiện một tác vụ cụ thể mà họ cần. Khi tạo hàm, người dùng định nghĩa một khối code có thể được gọi và thực thi bất kỳ lúc nào trong chương trình.

1.2 Một số khái niệm liên quan đến hàm

- Tham số (parameter): là các biến được định nghĩa trong phần khai báo của hàm, phương thức hoặc quy trình trong lập trình. Nó dùng để nhận dữ liệu hoặc giá trị từ bên ngoài và được sử dụng trong quá trình thực hiện của hàm. Tham số giúp mã nguồn linh hoạt hơn, cho phép các hàm xử lý dữ liệu khác nhau mà không cần phải viết lại mã.
- Đối số (argument): là giá trị thực tế được truyền vào một hàm khi hàm đó được gọi. Đây là dữ liệu mà hàm sử dụng để thực hiện các thao tác xử lý và trả về kết quả. Đối số có thể là số, chuỗi, biến hoặc thậm chí là các đối tượng phức tạp trong ngôn ngữ lập trình.
- Gọi hàm(calling a function): Là việc sử dụng tên của hàm cùng với các tham số (nếu có) để thực thi hàm đó. Khi một hàm được gọi, quá trình thực hiện sẽ chuyển tới nội dung của hàm và thực thi nó.

Việc sử dụng hàm giúp rất nhiều trong việc tổ chức code, làm cho code dễ đọc hơn, giảm sự lặp lại và tăng khả năng tái sử dụng.

1.3 Ví dụ

```
#include <stdio.h>


void sayHello() {
    printf("Hello, world!\n");
```

```
}

int main() {
    sayHello();
    return 0;
}
```

2. CON TRỎ

2.1 Khái niệm: Con trỏ là một khái niệm quan trọng trong lập trình, đặc biệt là trong ngôn ngữ C và C++. Con trỏ là một biến chứa địa chỉ của một biến khác trong bộ nhớ. Điều này cho phép bạn thực hiện các thao tác trên bộ nhớ và truy cập trực tiếp vào dữ liệu được lưu trữ trong đó.

2.2 Một số khái niệm liên quan đến con trỏ

- Địa chỉ (Address): Mỗi biến trong bộ nhớ có một địa chỉ duy nhất để xác định vị trí của nó trong bộ nhớ.
- Toán tử con trỏ (& và *):

  + Toán tử &: Trả về địa chỉ của một biến.

  + Toán tử *: Trả về giá trị của biến được trỏ tới bởi một con trỏ.

- Khai báo và sử dụng con trỏ:

  + Để khai báo một con trỏ, bạn sử dụng dấu * trước tên biến.

  + Để gán địa chỉ của một biến cho một con trỏ, bạn sử dụng toán tử &.

  + Để truy cập giá trị của biến được trỏ tới bởi một con trỏ, bạn sử dụng toán tử *.

- Dùng con trỏ để thực hiện các thao tác trên bộ nhớ: Bằng cách sử dụng con trỏ, bạn có thể thực hiện các thao tác như cấp phát bộ nhớ động, truy cập mảng, và chuyển đổi dữ liệu.

2.3 Ví dụ

```c
#include <stdio.h>


int main() {
    int x = 10;
    int *ptr = &x;
```

```
    printf("Giá trị của x: %d\n", *ptr);

    return 0;

}
```

3. CON TRỎ MẢNG

3.1 Khái niệm

- Con trỏ mảng là khái niệm liên quan đến việc sử dụng con trỏ để tham chiếu đến các phần tử của mảng trong lập trình.
- Trong ngôn ngữ lập trình C và C++, một mảng thực chất là một con trỏ. Khi khai báo một mảng là cũng đang tạo một con trỏ trỏ tới địa chỉ của phần tử đầu tiên trong mảng. Cụ thể, tên của mảng là một con trỏ không thể thay đổi (const pointer) trỏ tới phần tử đầu tiên của mảng.

3.2 Một số khái niệm liên quan đến con trỏ mảng

- Tham chiếu đến phần tử của mảng: Bạn có thể sử dụng con trỏ để truy cập đến các phần tử của mảng bằng cách sử dụng toán tử chỉ mục hoặc toán tử pointer.
- Duyệt mảng: Bạn có thể sử dụng con trỏ để duyệt qua tất cả các phần tử của mảng bằng cách di chuyển con trỏ từ phần tử đầu tiên đến phần tử cuối cùng của mảng.
- Tính toán địa chỉ: Bạn có thể sử dụng con trỏ để tính toán địa chỉ của các phần tử của mảng và truy cập trực tiếp vào bộ nhớ.

3.3 Ví dụ

```c
#include <stdio.h>

int main() {
    int arr[3] = {1, 2, 3};
    int *ptr = arr;

    for (int i = 0; i < 3; i++) {
        printf("%d ", *(ptr + i));
    }

    return 0;
}
```

4. MẢNG CON TRỎ

4.1 Khái niệm: Mảng con trỏ là một mảng mà mỗi phần tử của nó là một con trỏ. Điều này thường được sử dụng để lưu trữ một tập hợp các địa chỉ bộ nhớ, ví dụ như một mảng các chuỗi ký tự (mỗi chuỗi là một mảng ký tự và mỗi phần tử của mảng con trỏ sẽ trỏ đến đầu của một chuỗi).

4.2 Một số khái niệm liên quan đến mảng con trỏ

- Khai báo mảng: Để khai báo một mảng con trỏ, bạn cần chỉ định kiểu dữ liệu mà các con trỏ trong mảng sẽ trỏ tới, theo sau là dấu *, tên mảng và kích thước của mảng trong dấu ngoặc vuông [].
- Khởi tạo mảng: Các phần tử của mảng con trỏ có thể được khởi tạo để trỏ đến các biến đã tồn tại hoặc các vùng nhớ được cấp phát động.
- Truy cập giá trị thông qua mảng con trỏ: Để truy cập giá trị mà một con trỏ trong mảng đang trỏ tới, bạn sử dụng toán tử giải tham chiếu * kết hợp với chỉ số của mảng.

4.3 Ví dụ

```c
#include <stdio.h>

int main() {
    char *colors[] = {"Red", "Green", "Blue"};

    for (int i = 0; i < 3; i++) {
        printf("%s\n", colors[i]);
    }

    return 0;
}
```

5. CON TRỎ HÀM

5.1 Khái niệm

Như chúng ta đã biết, con trỏ trong C là một biến được dùng để lưu trữ địa chỉ của dữ liệu trong bộ nhớ máy tính.

Sau khi khai báo thì hàm cũng được lưu trữ tại một địa chỉ trong bộ nhớ, 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 lưu trữ địa chỉ của một hàm là con trỏ hàm trong C, và sử dụng nó để truy cập vào địa chỉ của hàm, cũng như thực thi các xử lý bên trong hàm đó.

5.2 Lợi ích khi sử dụng con trỏ hàm

- Chuyển đổi linh hoạt giữa các hàm: Con trỏ hàm cho phép bạn lưu trữ địa chỉ của các hàm khác nhau và có thể gọi chúng một cách linh hoạt. Điều này rất hữu ích trong việc triển khai các máy trạng thái (state machine) hoặc các chương trình phục vụ ngắt (ISR).
- Tăng cường tính linh hoạt của mã nguồn: Khi sử dụng con trỏ hàm, bạn có thể dễ dàng thay đổi hàm thực hiện mà không cần sửa đổi cấu trúc mã chính. Điều này giúp mã nguồn dễ bảo trì và mở rộng.
- Đơn giản hóa việc triển khai callback: Con trỏ hàm thường được sử dụng để triển khai các hàm callback trong các giao tiếp như UART, I2C, SPI, cho phép xử lý các sự kiện như dữ liệu đã nhận hoặc hoàn tất truyền.
- Giảm sự lặp lại của mã: Thay vì viết các hàm với logic tương tự, bạn có thể sử dụng con trỏ hàm để tạo ra các biến thể khác nhau của một hàm duy nhất, làm cho mã trở nên ngắn gọn và hiệu quả hơn.
- Quản lý bộ nhớ động hiệu quả: Các hàm trả về con trỏ có thể được sử dụng để quản lý bộ nhớ động, giúp tối ưu hóa việc sử dụng tài nguyên trong các hệ thống có bộ nhớ hạn chế.

Sử dụng con trỏ hàm một cách hiệu quả sẽ giúp bạn tối đa hóa hiệu suất của chương trình, đồng thời tăng cường khả năng bảo trì và mở rộng mã nguồn trong các dự án lập trình phức tạp.

5.3 Ví dụ

```c
#include <stdio.h>


void greet() {
    printf("Hello from function pointer!\n");
}


int main() {
    void (*funcPtr)() = greet;
    funcPtr();
    return 0;
}
```

6. CẤP PHÁT ĐỘNG

6.1 Khái niệm

Cấp độ phát động (phân bổ động) là một khái niệm quan trọng trong trình cài đặt và quản lý bộ nhớ, cho phép chương trình yêu cầu và giải phóng bộ nhớ trong thời gian chạy. Điều này rất hữu ích khi kích thước của dữ liệu không thể xác định trước hoặc khi cần quản lý bộ nhớ bằng cách hoạt động.

6.2 Một số từ khóa liên quan đến cấp phát động

- malloc(): Phát một khối bộ nhớ cụ thể và trả về con trỏ đến khối bộ nhớ đó. Nếu không đủ bộ nhớ, nó sẽ trả về NULL.
- calloc(): Bộ nhớ phát hiện cấp độ cho một mảng và khởi tạo tất cả các phần tử thành 0.
- realloc(): Thay đổi kích thước của bộ nhớ đã được phát.
- free(): Giải phóng bộ nhớ đã được cấp phát.

6.3 So sánh cấp phát động và mảng tĩnh

- Về kích thước

+ Mảng tĩnh: Kích thước bộ nhớ phải được xác định trước và không thể thay đổi trong quá trình thực hiện.

+ Cấp phát động: Kích thước của bộ nhớ có thể thay đổi trong quá trình thực hiện, cho phép hoạt động tốt hơn trong quá trình quản lý bộ nhớ.

- Về quản lý bộ nhớ

+ Mảng tĩnh: Bộ nhớ được tự động giải nén khi chương trình kết thúc hoặc khi biến thể phạm vi .Không cần giải phóng bộ nhớ.

+ Cấp phát động: Trình cài đặt phải tự quản lý bộ nhớ, bao gồm việc giải phóng bộ nhớ khi không sử dụng.Nếu không có thể dẫn đến rò rỉ bộ nhớ.

- Về tốc độ xử lý

+ Mảng tĩnh: Thường nhanh hơn vì bộ nhớ đã được phát trước đó và không cần phải tìm kiếm bộ nhớ trống.

+ Cấp phát động: Có thể làm chậm hơn việc tìm kiếm bộ nhớ trống và quản lý bộ nhớ.

- Về điều kiện sử dụng

+ Mảng tĩnh: Thích hợp cho các biến có cố định kích thước và không thay đổi trong suốt vòng đời của chương trình.

+ Cấp phát động: Thích hợp cho dữ liệu cấu trúc có thay đổi kích thước, chẳng hạn như danh sách liên kết, cây và sơ đồ.

6.4 Ví dụ

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *p = (int*)malloc(sizeof(int));
    if (p == NULL) {
        printf("Loi cap phat.\n");
        return 1;
    }

    *p = 42;
    printf("Gia tri: %d\n", *p);
    free(p);
    return 0;
}
```

7. XỬ LÍ TỆP

7.1 Khái niệm

Xử lý tệp trong trình lập ngôn ngữ C là một phần quan trọng cho phép chương trình đọc và ghi dữ liệu từ và vào tệp. Điều này rất hữu ích khi bạn cần lưu trữ dữ liệu dài hoặc trao đổi dữ liệu giữa các chương trình.

7.2 Các thao tác liên quan đến xử lí tệp

- Mở tệp: Sử dụng hàm ‘fopen’ để mở tệp.

- Đọc tệp: Sử dụng các hàm như ‘fgetc’, ‘fgets’ hoặc ‘fread’ để đọc dữ liệu từ tệp.

- Ghi tệp: Sử dụng các hàm như ‘fputc’, ‘fputs’ hoặc ‘fwrite’ để ghi dữ liệu vào tệp.

-Đóng tệp: Sử dụng hàm ‘fclose’ để đóng tệp.

7.3 Ví dụ

```c
#include <stdio.h>

int main() {
    FILE *f = fopen("data.txt", "w");
    if (f == NULL) {
        printf("Loi mo tep.\n");
        return 1;
    }
    fprintf(f, "Xin chao!\nDay la file van ban.");
    fclose(f);

    f = fopen("data.txt", "r");
    if (f == NULL) {
        printf("Loi mo tep.\n");
        return 1;
    }
```

```
    char c;
    while ((c = fgetc(f)) != EOF)
        putchar(c);
    fclose(f);

    return 0;
}
```

8. KIỂU CẤU TRÚC

8.1 Khái niệm

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.

8.2 Lợi ích của việc sử dụng kiểu cấu trúc

- Gom nhóm các dữ liệu có liên quan: Struct cho phép bạn nhóm nhiều biến (có thể khác kiểu) thành một đơn vị logic duy nhất, giúp dữ liệu rõ ràng và dễ quản lý hơn.
- Dễ mở rộng và bảo trì mã nguồn: Thay vì sửa đổi hàng loạt hàm và biến rải rác, bạn chỉ cần sửa trong cấu trúc.
- Dễ dàng truyền dữ liệu qua hàm: Bạn có thể truyền một struct cho hàm như một đối tượng duy nhất thay vì nhiều đối số rời rạc.
- Quản lý bộ nhớ tốt hơn: Khi dùng struct, bạn có thể khai báo mảng hoặc cấp phát bộ nhớ động cho toàn bộ cấu trúc.
- Tăng tính tái sử dụng: Một khi đã định nghĩa struct, bạn có thể sử dụng lại ở nhiều nơi trong chương trình.

8.3 Ví dụ

```c
#include <stdio.h>
#include <string.h>

struct SinhVien {
    char ten[30];
    int tuoi;
};
```

```c
int main() {
    struct SinhVien sv = {"An", 20};
    printf("Ten: %s, Tuoi: %d\n", sv.ten, sv.tuoi);
    return 0;
}
```

9. DANH SÁCH LIÊN KẾT

9.1 Khái niệm

Danh sách liên kết là một cấu trúc dữ liệu trong lập trình máy tính, được sử dụng để lưu trữ và quản lý một tập hợp các phần tử dữ liệu. Trong danh sách liên kết, mỗi phần tử được gọi là "nút" và bao gồm dữ liệu của nút đó cùng một con trỏ chỉ đến nút tiếp theo trong danh sách.

Mỗi nút trong danh sách liên kết chứa hai phần chính:

1. Dữ liệu: Thông tin được lưu trữ trong nút, có thể là bất kỳ kiểu dữ liệu nào, chẳng hạn như số nguyên, số thực, ký tự, hoặc thậm chí là một cấu trúc phức tạp hơn.

2. Con trỏ tiếp theo: Một con trỏ chỉ đến nút tiếp theo trong danh sách liên kết. Điều này tạo ra một chuỗi các nút mà mỗi nút chỉ biết đến nút tiếp theo của nó, tạo thành một danh sách.

Có hai loại danh sách liên kết chính:

1. Danh sách liên kết đơn: Mỗi nút chỉ trỏ đến nút tiếp theo trong danh sách.

2. Danh sách liên kết đôi: Mỗi nút chứa một con trỏ không chỉ đến nút tiếp theo, mà còn chỉ đến nút trước đó trong danh sách, tạo thành một danh sách có thể được duyệt cả từ phía trước và từ phía sau.

9.2 Lợi ích của việc sử dụng danh sách liên kết

- Cấp phát bộ nhớ linh hoạt (động)

+ Không cần biết trước số lượng phần tử.

+ Dễ dàng thêm hoặc xóa phần tử mà không cần dịch chuyển dữ liệu như mảng.

- Thêm/Xóa phần tử dễ dàng

+ Thêm/Xóa ở đầu/cuối/giữa rất nhanh chỉ bằng cách thay đổi con trỏ.

+ Không cần dịch chuyển toàn bộ như trong mảng.

- Tính linh hoạt khi thao tác với dữ liệu

+ Dễ thêm tính năng tìm kiếm, sắp xếp, lọc.

+ Có thể dùng để đọc/ghi dữ liệu từ/ra tệp theo dòng, cấu trúc...

- Không lãng phí bộ nhớ

+ Chỉ cấp phát đúng số lượng phần tử đang dùng.

+Không cần cấp phát thừa như mảng tĩnh.

9.3 Ví dụ

```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};
int main() {
    struct Node* head = malloc(sizeof(struct Node));
    head->data = 1;
    head->next = malloc(sizeof(struct Node));
    head->next->data = 2;
    head->next->next = NULL;

    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    return 0;
}
```

B.ỨNG DỤNG

XÂY DỰNG ỨNG DỤNG CHO VIỆC QUẢN LÝ THƯ VIỆN

Các tính năng cần thiết:

1. **Thêm sách**
2. **Xóa sách**
3. **Tìm kiếm sách theo mã số sách**
4. **Thêm người mượn**
5. **Xóa người mượn**
6. **Tìm kiếm người mượn theo mã thẻ thư viện**
7. **Hiển thị danh sách sách**
8. **Hiển thị danh sách người mượn**
9. **In danh sách sách ra file**
10. **In danh sách người mượn**

Mục đích:

- Nhằm xây dựng một hệ thống quản lý thư viện đơn giản, phục vụ cho việc theo dõi sách và hoạt động mượn trả sách, em đặt ra bài toán: làm sao để lưu trữ thông tin sách trong thư viện và quản lý việc mượn trả một cách linh hoạt, dễ mở rộng.
- Để giải quyết bài toán này, em quyết định kết hợp giữa việc sử dụng mảng và danh sách liên kết nhằm thiết kế một mô hình quản lý dữ liệu cơ bản, làm nền tảng cho các tính năng nâng cao sau này.
    - Về danh sách liên kết: em sử dụng danh sách liên kết đơn để lưu trữ các thông tin về sách hoặc người mượn. Cụ thể, mỗi node sách sẽ chứa các thông tin như: “Mã sách”, “Tên sách”, “Tác giả”, “Ngày xuất bản”, “Số lượng” và “Giá sách”. Tương tự, danh sách người mượn cũng lưu các trường như: “Mã thành viên”, “Họ tên”, “Số điện thoại”, “Tên sách mượn”, “Ngày mượn”, “Ngày trả”, v.v.
    - Về mảng: mảng có thể được sử dụng để phân loại sách theo thể loại hoặc nhóm người mượn theo tháng/quý nếu mở rộng hệ thống. Mỗi phần tử trong mảng sẽ là một con trỏ trỏ đến danh sách liên kết, từ đó tạo ra một mảng các danh sách liên kết, thuận tiện cho việc truy xuất và xử lý theo nhóm.

- Bằng việc sử dụng con trỏ và cấp phát động, hệ thống quản lý thư viện trở nên linh hoạt hơn trong việc thêm, xóa, tìm kiếm, sắp xếp hoặc cập nhật dữ liệu, đồng thời tiết kiệm tài nguyên bộ nhớ. Đây là một hướng tiếp cận phù hợp để mô phỏng hệ thống quản lý thư viện thực tế một cách đơn giản, rõ ràng và có khả năng phát triển thêm nhiều chức năng nâng cao trong tương lai.

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct Sach { //Dinh nghia cau truc Sach
    char maSach[10];
    char tenSach[100];
    char tacGia[50];
    char ngayXB[15];
    int soLuong;
    int giaSach;
    struct Sach* next;
} Sach;

typedef struct NguoiMuon { //Dinh nghia cau truc NguoiMuon
      char mttv[10];
      char hoTen[50];
      char sdt[10];
      char maSach[10];
      char tenSach[100];
      int soLuong;
      char ngayMuon[15];
      char ngayTra[15];
      struct NguoiMuon* next;
} NguoiMuon;
```

```
// Ham tao nut Sach moi
Sach* taoNodeSach(char* maSach, char* tenSach, char* tacGia, char*
ngayXB, int soLuong, int giaSach) {
    Sach* newNode = (Sach*)malloc(sizeof(Sach));
    strcpy(newNode->maSach, maSach);
    strcpy(newNode->tenSach, tenSach);
    strcpy(newNode->tacGia, tacGia);
    strcpy(newNode->ngayXB, ngayXB);
    newNode->soLuong = soLuong;
    newNode->giaSach = giaSach;
    newNode->next = NULL;
    return newNode;
}

// Ham tao nut NguoiMuon moi
NguoiMuon* taoNodeNguoiMuon(char* mttv, char* hoTen, char* sdt, char*
maSach, char* tenSach, int soLuong, char* ngayMuon, char* ngayTra) {
    NguoiMuon* newNode = (NguoiMuon*)malloc(sizeof(NguoiMuon));
    strcpy(newNode->mttv, mttv);
    strcpy(newNode->hoTen, hoTen);
    strcpy(newNode->sdt, sdt);
    strcpy(newNode->maSach, maSach);
    strcpy(newNode->tenSach, tenSach);
    strcpy(newNode->ngayMuon, ngayMuon);
    strcpy(newNode->ngayTra, ngayTra);
    newNode->soLuong = soLuong;
    newNode->next = NULL;
    return newNode;
}

// Ham them sach
void themSach(Sach** head) {
```

```
    char tiepTucThemSach;
    do {
        char maSach[10], tenSach[100], tacGia[50], ngayXB[15];
        int soLuong, giaSach;
        printf("Nhap ma sach: ");
        scanf("%s", maSach);
        getchar();
        printf("Nhap ten sach: ");
        fgets(tenSach, 100, stdin);
        tenSach[strcspn(tenSach, "\n")] = 0;
        printf("Nhap tac gia: ");
        fgets(tacGia, 50, stdin);
        tacGia[strcspn(tacGia, "\n")] = 0;
        printf("Nhap ngay xuat ban: ");
        scanf("%s", ngayXB);
        printf("Nhap so luong: ");
        scanf("%d", &soLuong);
        printf("Nhap gia sach: ");
        scanf("%d", &giaSach);
        Sach* newNode = taoNodeSach(maSach, tenSach, tacGia, ngayXB,
soLuong, giaSach);
        Sach* current = *head;
        Sach* prev = NULL;

        while (current != NULL && strcmp(current->maSach, maSach) < 0) {
            prev = current;
            current = current->next;
        }

        if (prev == NULL) {
            newNode->next = *head;
            *head = newNode;
```

```
        } else {
            newNode->next = current;
            prev->next = newNode;
        }

        printf("Da them sach ma so %s\n", maSach);
        printf("Ban co muon them sach nua khong?: ");
        getchar();
        scanf("%c", &tiepTucThemSach);
        getchar();

    } while (tiepTucThemSach == 'y' || tiepTucThemSach == 'Y');
}

// Ham xoa sach theo MS sach
void xoaSach(Sach** head) {
     char tiepTucXoaSach;
     do {
        char maSachCanXoa[10];
        printf("Nhap ma sach can xoa: ");
        scanf("%s", maSachCanXoa);
     Sach* current = *head;
     Sach* prev = NULL;

    while (current != NULL && strcmp(current->maSach, maSachCanXoa) !=
0) {
        prev = current;
        current = current->next;
    }

    if (current == NULL) {
        printf("Khong tim thay sach voi MS %s\n", maSachCanXoa);
```

```
        return;
    }

    if (prev == NULL) {
        *head = current->next;
    } else {
        prev->next = current->next;
    }

    free(current);
    printf("Da xoa sach voi MS %s\n", maSachCanXoa); // Thong bao xoa
sach thanh cong
    printf("Ban co muon xoa sach nua khong?: ");
        getchar();
        scanf("%c", &tiepTucXoaSach);
        getchar();
    } while (tiepTucXoaSach == 'y' || tiepTucXoaSach == 'Y');
}

// Ham them nguoi muon
void themNguoiMuon(NguoiMuon** head){
    char tiepTucThemNguoiMuon;
    do {
    char mttv[10], hoTen[50], sdt[10], maSach[10], tenSach[100],
ngayMuon[15], ngayTra[15];
    int soLuong;
    printf("Nhap ma the thu vien: ");
    scanf("%s", mttv);
    getchar(); // Xoa bo dem
    printf("Nhap ho ten: ");
    fgets(hoTen, 50, stdin);
    hoTen[strcspn(hoTen, "\n")] = 0;
    printf("Nhap so dien thoai: ");
```

```
    scanf("%s", sdt);
    getchar();
    printf("Nhap ma sach: ");
    scanf("%s", maSach);
    getchar();
    printf("Nhap ten sach: ");
    fgets(tenSach, 100, stdin);
    tenSach[strcspn(tenSach, "\n")] = 0; // Xoa ky tu xuong dong
    printf("Nhap so luong: ");
    scanf("%d", &soLuong);
    getchar();
    printf("Nhap ngay muon: ");
    scanf("%s", ngayMuon);
    getchar();
    printf("Nhap ngay tra: ");
    scanf("%s", ngayTra);
    getchar();
    NguoiMuon* newNode = taoNodeNguoiMuon(mttv, hoTen, sdt, maSach,
tenSach, soLuong, ngayMuon, ngayTra);
    NguoiMuon* current = *head;
    NguoiMuon* prev = NULL;

    // Tim vi tri chen
    while (current != NULL && strcmp(current->mttv, mttv) < 0) {
        prev = current;
        current = current->next;
    }

    // Chen nut
    if (prev == NULL) {
        newNode->next = *head;
        *head = newNode;
```

```
    } else {
        newNode->next = current;
        prev->next = newNode;
    }

    printf("Da them nguoi muon voi ma the %s\n", mttv); // Thong bao
them nguoi muon thanh cong
    printf("Ban co muon them nguoi muon nua khong?: ");
    fflush(stdin);
    scanf(" %c", &tiepTucThemNguoiMuon);
    getchar();
     } while (tiepTucThemNguoiMuon == 'y' || tiepTucThemNguoiMuon ==
'Y');
}



// Ham xoa nguoi muon theo ma the thu vien
void xoaNguoiMuon(NguoiMuon** head) {
    char tiepTucXoaNguoiMuon;
    do {
        char mttvCanXoa[10];
        printf("Nhap ma the thu vien can xoa: ");
        scanf("%s", mttvCanXoa);
    NguoiMuon* current = *head;
    NguoiMuon* prev = NULL;

    while (current != NULL && strcmp(current->mttv, mttvCanXoa) != 0) {
        prev = current;
        current = current->next;
    }

    if (current == NULL) {
        printf("Khong tim thay nguoi muon voi ma the %s\n", mttvCanXoa);
```

```
        return;
    }

    if (prev == NULL) {
        *head = current->next;
    } else {
        prev->next = current->next;
    }

    free(current);
    printf("Da xoa nguoi muon voi ma the %s\n", mttvCanXoa); // Thong
bao xoa nguoi muon thanh cong
    printf("Ban co muon xoa nguoi muon nua khong?: ");
        fflush(stdin);
        scanf(" %c", &tiepTucXoaNguoiMuon);
    } while (tiepTucXoaNguoiMuon == 'y' || tiepTucXoaNguoiMuon == 'Y');
}

// Ham tim kiem sach theo ma so sach
Sach* timSach(Sach* head, char* maSach) {
    while (head != NULL) {
        if (strcmp(head->maSach, maSach) == 0)
            return head;
        head = head->next;
    }
    return NULL;
}

// Ham tim kiem nguoi muon theo ma the thu vien
NguoiMuon* timNguoiMuon(NguoiMuon* head, char* mttv) {
    while (head != NULL) {
        if (strcmp(head->mttv, mttv) == 0)
```

```
            return head;
        head = head->next;
    }
    return NULL;
}

// Ham hien thi danh sach sach
void hienthiSach(Sach* head) {
    printf("\nDanh sach sach\n");
    printf("%-10s %-20s %-20s %-10s %10s %10s\n", "MS sach", "Ten sach",
"Tac gia", "Ngay xuat ban", "So luong", "Gia sach");
    while (head != NULL) {
        printf("%-10s %-20s %-20s %-10s %10d %10d\n", head->maSach,
head->tenSach, head->tacGia, head->ngayXB, head->soLuong, head-
>giaSach);
        head = head->next;
    }
}

// Ham hien thi danh sach nguoi muon
void hienthiNguoiMuon(NguoiMuon* head) {
    printf("\nDanh sach nguoi muon\n");
    printf("%-10s %-20s %-10s %-10s %-20s %-10s %-10s %-10s\n", "Ma
the", "Ten nguoi muon", "SDT", "MS sach", "Ten sach", "So Luong", "Ngay
muon", "Ngay tra");
    while (head != NULL) {
        printf("%-10s %-20s %-10s %-10s %-20s %-10d %-10s %-10s\n",
head->mttv, head->hoTen, head->sdt, head->maSach, head->tenSach, head-
>soLuong, head->ngayMuon, head->ngayTra);
        head = head->next;
    }
}

// Ham ghi danh sach ra file text
```

```
void ghifileSach(Sach* head, const char* filename) {
    FILE* f = fopen(filename, "w");
    if (f == NULL) {
        printf("Khong the mo file %s\n", filename);
        return;
    }

    while (head != NULL) {
        fprintf(f, "%-10s %-20s %-20s %-10s %10d %10d\n", head->maSach,
head->tenSach, head->tacGia, head->ngayXB, head->soLuong, head-
>giaSach);
        head = head->next;
    }

    fclose(f);
    printf("Da ghi danh sach vao file %s\n", filename);
}

// Ham ghi danh sach ra file text
void ghifileNguoiMuon(NguoiMuon* head, const char* filename) {
    FILE* f = fopen(filename, "w");
    if (f == NULL) {
        printf("Khong the mo file %s\n", filename);
        return;
    }

    while (head != NULL) {
        fprintf(f, "%-10s %-20s %-10s %-10s %-20s %-10d %-10s %-10s\n",
head->mttv, head->hoTen, head->sdt, head->maSach, head->tenSach, head-
>soLuong, head->ngayMuon, head->ngayTra);
        head = head->next;
    }
```

```
    fclose(f);
    printf("Da ghi danh sach vao file %s\n", filename);
}

// Ham main
int main() {
    Sach* sachhead = NULL;
    NguoiMuon* muonhead = NULL;
    int choice;

    do {
        printf("\n[============== QUAN LY THU VIEN ==============]\n");
        printf("| 1. Them sach                                  |\n");
        printf("| 2. Xoa sach                                   |\n");
        printf("| 3. Tim kiem sach theo ma so sach              |\n");
        printf("| 4. Them nguoi muon                            |\n");
        printf("| 5. Xoa nguoi muon                             |\n");
        printf("| 6. Tim kiem nguoi muon theo ma the thu vien   |\n");
        printf("| 7. Hien thi danh sach sach                    |\n");
        printf("| 8. Hien thi danh sach nguoi muon              |\n");
        printf("| 9. Ghi danh sach sach ra file                 |\n");
        printf("| 10. Ghi danh sach nguoi muon ra file          |\n");
        printf("| 0. Thoat                                      |\n");
        printf("[===============================================]\n");
        printf("Nhap lua chon: ");
        scanf("%d", &choice);
        getchar();

        switch (choice) {
            case 1:
                themSach(&sachhead);
                break;
```

```
            case 2:
                xoaSach(&sachhead);
                break;
            case 3: {
                char maSach[10];
                printf("Nhap MS sach can tim: ");
                scanf("%s", maSach);
                Sach* s = timSach(sachhead, maSach);
                if (s) {
                    printf("%-10s %-20s %-20s %-10s %10d %10d\n",
"MaSach", "TenSach", "TacGia", "NgayXB", "SoLuong", "GiaSach");
                    printf("%-10s %-20s %-20s %-10s %10d %10d\n", s-
>maSach, s->tenSach, s->tacGia, s->ngayXB, s->soLuong, s->giaSach);
                } else {
                    printf("Khong tim thay sach\n"); }
                break;
            }
            case 4:
                themNguoiMuon(&muonhead);
                break;
            case 5: {
                xoaNguoiMuon(&muonhead);
                break;
                  }
            case 6: {
                char mttv[10];
                  printf("Nhap ma the thu vien cua nguoi muon can tim:
");
                  scanf("%s", mttv);
                  NguoiMuon* nm = timNguoiMuon(muonhead, mttv);
                if (nm) {
                      printf("%-10s %-20s %-10s %-10s %-20s %-10d %-10s
%-10s\n", "MaThe", "HoTen", "SDT", "MaSach", "TenSach", "SoLuong",
"NgayMuon", "NgayTra");
```

```
                    printf("%-10s %-20s %-10s %-10s %-20s %-10d %-10s
%-10s\n", nm->mttv, nm->hoTen, nm->sdt, nm->maSach, nm->tenSach, nm-
>soLuong, nm->ngayMuon, nm->ngayTra);
                } else {
                    printf("Khong tim thay nguoi muon\n"); }
                      break;
                  }
            case 7:
                hienthiSach(sachhead);
                break;
            case 8:
                hienthiNguoiMuon(muonhead);
                break;
            case 9:
                ghifileSach(sachhead, "sach.txt");
                break;
            case 10:
                ghifileNguoiMuon(muonhead, "nguoi_muon.txt");
                break;
            case 0:
                printf("Thoat chuong trinh.\n");
                break;
            default:
                printf("Lua chon khong hop le!\n");
                break;

    }
} while (choice != 0);

    // Giai phong bo nho
    while (sachhead != NULL) {
        Sach* temp = sachhead;
        sachhead = sachhead->next;
```

```
        free(temp);
    }
    while (muonhead != NULL) {
      NguoiMuon* temp = muonhead;
      muonhead = muonhead->next;
      free(temp);
      }

    return 0;
}
```

KẾT LUẬN

Thông qua cách xây dựng bài toán như đã trình bày, kết hợp giữa mảng và danh sách liên kết, em bước đầu đã đáp ứng được các yêu cầu cơ bản trong việc quản lý thư viện. Tuy nhiên, đây mới chỉ là ý tưởng ban đầu và vẫn đang trong quá trình hoàn thiện, nên không thể tránh khỏi một số thiếu sót trong thiết kế chương trình.

Em rất mong nhận được những góp ý, nhận xét từ thầy để có thể cải thiện hơn nữa kỹ năng lập trình của bản thân. Đồng thời, từ những ý kiến đóng góp đó, em hy vọng có thể nâng cấp bài toán này theo hướng tối ưu hơn trong việc quản lý thư viện một cách hiệu quả và khoa học hơn

TÀI LIỆU THAM KHẢO

[1]. “Tài Liệu Lập Trình C Và Cách Học Lập Trình C Hiệu Quả” https://blog.28tech.com.vn/c-tai-lieu-lap-trinh-c-va-cach-hoc-lap-trinh-c-hieu-qua [Accessed 25/04/2025].

[2]. “Hướng dẫn C - W3Schools https://www.w3schools.com/c/index.php [Accessed 25/04/2025]