

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**  
**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**  
**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN HỌC**  
**MÔN: KỸ THUẬT LẬP TRÌNH**

**ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG**  
**QUẢN LÝ RÁP CHIẾU PHIM**

Giảng viên hướng dẫn: **TRẦN PHONG NHÃ**  
Sinh viên thực hiện: **NGUYỄN NGỌC TIÊN**  
**NGÔ THANH TIẾN**  
Lớp: **CQ.65.CNTT**  
Khoá: **65**

Hồ Chí Minh, 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 MÔN HỌC**  
**MÔN: KỸ THUẬT LẬP TRÌNH**

**ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG**  
**QUẢN LÝ RÁP CHIẾU PHIM**

Giảng viên hướng dẫn: **TRẦN PHONG NHÃ**  
Sinh viên thực hiện: **NGUYỄN NGỌC TIÊN**  
**NGÔ THANH TIẾN**  
Lớp: **CQ.65.CNTT**  
Khoá: **65**

Hồ Chí Minh, tháng 5 năm 2025

## LỜI CẢM ƠN

Lời nói đầu tiên, em xin gửi lời tri ân sâu sắc đến Quý Thầy Cô Bộ môn Công nghệ Thông tin, Phân hiệu Trường Đại học Giao thông Vận tải tại TP. Hồ Chí Minh. Kính chúc Quý Thầy Cô luôn dồi dào sức khỏe và thành công trong sự nghiệp.

Em xin bày tỏ lòng biết ơn chân thành đến quý thầy cô đã tạo điều kiện thuận lợi để em hoàn thành bài tập lớn môn học Kỹ thuật lập trình với đề tài "Xây dựng ứng dụng quản lý rạp chiếu phim". Đặc biệt, em xin gửi lời cảm ơn sâu sắc đến thầy Trần Phong Nhã. Nhờ sự hướng dẫn tận tình và những kiến thức chuyên môn sâu sắc về lập trình, đặc biệt là ngôn ngữ C, Thầy đã giúp em có được nền tảng vững chắc để xây dựng và phát triển ứng dụng này. Những chỉ dẫn quý báu của Thầy không chỉ giúp em giải quyết các vấn đề kỹ thuật trong quá trình hiện thực hóa các chức năng của ứng dụng quản lý rạp chiếu phim bằng ngôn ngữ C, mà còn khơi gợi niềm đam mê và hứng thú của em đối với lĩnh vực lập trình.

Mặc dù đã cố gắng hết sức trong quá trình nghiên cứu và thực hiện, bài tập lớn của em chắc chắn vẫn còn nhiều thiếu sót do hạn chế về kiến thức và kinh nghiệm thực tế trong lập trình C. Em rất mong nhận được những ý kiến đóng góp chân thành từ quý thầy cô để bài tập lớn được hoàn thiện hơn nữa và có thể ứng dụng vào thực tế. Em xin chân thành cảm ơn quý thầy cô đã dành thời gian quý báu để đọc và cho em những lời khuyên quý báu. Em luôn sẵn sàng lắng nghe và tiếp thu mọi ý kiến đóng góp để cải thiện bài tập lớn.

Cuối cùng, em xin gửi lời cảm ơn sâu sắc nhất đến Quý Thầy Cô Bộ môn Công nghệ Thông tin, đặc biệt là thầy Trần Phong Nhã. Kính chúc Quý Thầy Cô luôn mạnh khỏe, hạnh phúc và thành công trên con đường sự nghiệp.

## NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

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

Giảng viên hướng dẫn

GV: Trần Phong Nhã

**BỘ MÔN: CÔNG NGHỆ THÔNG TIN**  
**GIỚI THIỆU SƠ LƯỢC**

-----\*\*\*-----

Nhóm sinh viên thực hiện:

**MSSV**

**Họ và tên**

6551071079

NGUYỄN NGỌC TIÊN

6551071082

NGÔ THANH TIẾN

Lớp:

CQ.65.CNTT

**1. Tên đề tài**

Xây dựng ứng dụng quản lý rạp chiếu phim

**2. Mục đích, yêu cầu**

**a. Mục đích**

Đề tài nhằm xây dựng một ứng dụng quản lý toàn diện cho rạp chiếu phim, hỗ trợ chủ rạp và nhân viên trong việc vận hành các hoạt động hàng ngày một cách hiệu quả và chuyên nghiệp. Ứng dụng sẽ giải quyết các thách thức trong quản lý thủ công, giúp tối ưu hóa quy trình làm việc, tiết kiệm thời gian, giảm thiểu sai sót và nâng cao trải nghiệm cho cả nhân viên lẫn khách hàng.

Hệ thống được xây dựng trên nền tảng ứng dụng máy tính.

**b. Yêu cầu**

- Yêu cầu công nghệ:
  - Sử dụng ngôn ngữ lập trình C.
  - Sử dụng danh sách liên kết đơn.
  - Sử dụng github.
- Chức năng cần đáp ứng:
  - Quản lý thông tin phim (thêm, xóa, sửa).
  - Quản lý bán vé (chọn ghế, tính tiền)

- Quản lý thông tin khách hàng (nếu có chương trình thành viên).
- Thống kê doanh thu
- Giao diện:
  - Thân thiện với người dùng, trực quan và dễ dàng thao tác cho cả nhân viên và người quản lý.
  - Thiết kế rõ ràng, các chức năng được bố trí hợp lý.

### 3. Nội dung và phạm vi đề tài

#### a. Nội dung

- Tổng quan bài toán quản lý rạp chiếu phim.
- Tổng quan về các công nghệ đang sử dụng (ngôn ngữ lập trình C, danh sách liên kết đơn).
- Phân tích và thiết kế chương trình quản lý rạp chiếu phim.
- Kiểm tra và chạy thử chương trình.
- Kết quả thu được sau khi hoàn thành đề tài.

#### b. Phạm vi

- Áp dụng lý thuyết về con trỏ trong C.
- Sử dụng kỹ thuật cấp phát động bộ nhớ.
- Xử lý tệp để lưu trữ và đọc dữ liệu.
- Sử dụng kiểu cấu trúc (struct) để tổ chức dữ liệu.
- Áp dụng cấu trúc dữ liệu danh sách liên kết đơn để quản lý dữ liệu.
- Tập trung giải quyết bài toán quản lý các hoạt động cơ bản của rạp chiếu phim (thay vì quản lý khách sạn).

### 4. Công nghệ, công cụ và ngôn ngữ lập trình

- Sử dụng ngôn ngữ lập trình C.
- Sử dụng cấu trúc dữ liệu danh sách liên kết đơn.
- Sử dụng Git/GitHub để quản lý mã nguồn và cộng tác.

### 5. Các kết quả chính dự kiến sẽ đạt được và ứng dụng

- Xây dựng thành công chương trình quản lý rạp chiếu phim đầy đủ các chức năng đã nêu trên.
- Ứng dụng có thể được triển khai và sử dụng trong các rạp chiếu phim nhỏ hoặc làm nền tảng để phát triển các hệ thống lớn hơn.

# MỤC LỤC

<b>LỜI CẢM ƠN.....</b>	<b>i</b>
<b>NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN.....</b>	<b>ii</b>
<b>GIỚI THIỆU SƠ LƯỢC.....</b>	<b>iii</b>
1. Tên đề tài.....	iii
2. Mục đích, yêu cầu .....	iii
a. Mục đích.....	iii
b. Yêu cầu .....	iii
3. Nội dung và phạm vi đề tài .....	iv
a. Nội dung.....	iv
b. Phạm vi .....	iv
4. Công nghệ, công cụ và ngôn ngữ lập trình.....	iv
5. Các kết quả chính dự kiến sẽ đạt được và ứng dụng.....	iv
<b>MỤC LỤC .....</b>	<b>v</b>
<b>CHƯƠNG 1. MỞ ĐẦU.....</b>	<b>1</b>
1.1. Lý do chọn đề tài .....	1
1.2. Hướng tiếp cận của đề tài .....	1
1.3. Mục tiêu nghiên cứu.....	2
1.4. Đối tượng và phạm vi nghiên cứu .....	2
<b>CHƯƠNG 2: CƠ SỞ LÝ THUYẾT.....</b>	<b>3</b>
2.1. Hàm .....	3
2.1.1. Mục đích và Lợi ích .....	3
2.1.2. Cú pháp .....	3
2.1.2. Truyền tham số.....	4
2.1.3. Giá trị trả về .....	4
2.1.4. Ví dụ Minh họa .....	5

2.2. Con trỏ .....	5
2.2.1 Định nghĩa và Địa chỉ Bộ nhớ .....	5
2.2.2. Khai báo và Khởi tạo .....	5
2.2.3. Các Toán tử Liên quan.....	6
2.2.4. Số học Con trỏ Cơ bản: .....	6
2.2.5. Các Lỗi Thường Gặp với Con trỏ: .....	7
2.2.6. Ví dụ Minh họa .....	7
2.3. Con trỏ và Mảng.....	8
2.3.1. Mối quan hệ giữa con trỏ và mảng trong C .....	8
2.3.2. Ví dụ Minh họa .....	9
2.4. Mảng Con trỏ.....	10
2.4.1. Định nghĩa và Cú pháp Khai báo .....	10
2.4.2. Trường hợp Sử dụng .....	10
2.4.3. So sánh với Mảng 2 Chiều .....	10
2.4.4. Khởi tạo và Sử dụng .....	10
2.4.5. Ví dụ Minh họa .....	11
2.5. Con trỏ Hàm .....	12
2.5.1. Định nghĩa và Mục đích.....	12
2.5.2. Cú pháp Khai báo.....	12
2.5.3. Gán Địa chỉ Hàm.....	13
2.5.4. Gọi Hàm qua Con trỏ .....	13
2.5.5. Trường hợp Sử dụng .....	13
2.5.6. Ví dụ Minh họa .....	14
2.6. Cấp phát Động.....	14
2.6.1. Khái niệm: Heap và Stack.....	14
2.6.2. Các Hàm Cấp phát và Giải phóng.....	14



2.6.3. Xử lý Lỗi:.....	15
2.6.4. Tầm quan trọng của free(): .....	15
2.6.5. Ví dụ Minh họa .....	16
2.7. Xử lý Tập.....	17
2.7.1. Khái niệm và Con trỏ FILE .....	17
2.7.2. Phân loại Tập .....	18
2.7.3. Mở và Đóng Tập .....	18
2.7.4. Thao tác Vào/Ra trên Tập Văn bản.....	19
2.7.5. Thao tác Vào/Ra trên Tập Nhị phân .....	20
2.7.6. Định vị trong Tập.....	20
2.7.6. Xử lý Lỗi Tập.....	20
2.7.7. Ví dụ Minh họa .....	21
2.8. Kiểu Cấu trúc (Struct) .....	22
2.8.1. Định nghĩa và Mục đích.....	22
2.8.2. Cú pháp Khai báo và typedef.....	22
2.8.3. Truy cập Thành viên .....	22
2.8.4. Con trỏ tới Struct.....	23
2.8.5. Mảng Struct.....	23
2.8.6. Ví dụ Minh họa .....	23
2.9. Danh sách Liên kết .....	24
2.9.1. Khái niệm (Danh sách Liên kết Đơn) .....	24
2.9.2. Cấu trúc Nút .....	25
2.9.3. Triển khai bằng Con trỏ và Cấp phát Động .....	25
2.9.4. Các Thao tác Cơ bản (Danh sách Liên kết Đơn) .....	25
2.9.5. Ví dụ Minh họa .....	27
<b>CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ .....</b>	<b>29</b>

3.1. Đặc tả bài toán .....	29
3.2. Yêu cầu hệ thống .....	29
3.3. Phân tích và thiết kế chương trình.....	29
3.4. Cấu trúc dữ liệu chương trình .....	32
3.5. Các chức năng của chương trình .....	33
3.5.1. Chức năng Tổng quan và Luồng hoạt động chính.....	34
3.5.2. Chức năng dành cho Khách hàng (User) .....	35
3.5.3. Chức năng dành cho Quản trị viên (Admin).....	37
3.5.4. Các chức năng Hỗ trợ và Quản lý Dữ liệu nền .....	39
<b>CHƯƠNG 4: THỬ NGHIỆM ỨNG DỤNG.....</b>	<b>41</b>
4.1. Giao diện đăng nhập / đăng ký .....	41
4.2. Giao diện Khách hàng (User) sau khi đăng nhập.....	41
4.3. Giao diện Admin sau khi đăng nhập .....	41
4.4. Chức năng Khách hàng: Xem danh sách phim .....	42
4.5. Chức năng Khách hàng: Đặt vé.....	42
4.6. Chức năng Khách hàng: Xem vé đã đặt .....	43
4.7. Chức năng Khách hàng: Hủy vé.....	43
4.8. Chức năng Admin: Quản lý phim .....	44
4.8.1. Giao diện thêm phim.....	44
4.8.2. Giao diện sửa thông tin phim .....	44
4.8.3. Giao diện xóa phim .....	45
4.9. Chức năng Admin: Quản lý vé và Thống kê.....	46
4.9.1. Giao diện xem tất cả vé.....	46
4.9.2. Giao diện thống kê doanh thu .....	46
4.10. Chức năng Admin: Quản lý tài khoản người dùng .....	47

4.10.1. Giao diện xem danh sách người dùng.....	47
4.10.2. Giao diện xóa tài khoản người dùng.....	47
4.10.3. Giao diện thêm tài khoản Admin .....	48
4.11. Giao diện đổi mật khẩu .....	48
4.12. Giao diện sau khi thoát/đăng xuất .....	49
<b>CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....</b>	<b>50</b>
5.1. Kết luận .....	50
5.1.1. Kết quả đạt được .....	50
5.1.2. Hạn chế .....	50
5.2. Hướng phát triển.....	51
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>52</b>

# CHƯƠNG 1. MỞ ĐẦU

## 1.1. Lý do chọn đề tài

Trong bối cảnh cuộc sống hiện đại và sự phát triển của công nghệ thông tin, nhu cầu giải trí, đặc biệt là xem phim tại rạp, ngày càng tăng. Việc quản lý rạp chiếu phim với các khía cạnh phức tạp như lịch chiếu, vé, phòng chiếu, thông tin phim, nhân viên và doanh thu gặp nhiều khó khăn với phương pháp truyền thống (tốn thời gian, dễ sai sót, khó thống kê).

Do đó, xây dựng một chương trình quản lý rạp chiếu phim hiệu quả bằng ngôn ngữ C là giải pháp tối ưu giúp:

- Tối ưu hóa quy trình quản lý: Tự động hóa đặt vé, xếp lịch, quản lý chỗ ngồi.
- Nâng cao hiệu quả hoạt động: Giảm thiểu sai sót, tiết kiệm nguồn lực.
- Cải thiện trải nghiệm khách hàng: Dễ dàng tra cứu thông tin và đặt vé.
- Hỗ trợ ra quyết định: Cung cấp báo cáo, thống kê chi tiết.
- Tăng cường tính chuyên nghiệp: Tạo hình ảnh hiện đại cho rạp.

## 1.2. Hướng tiếp cận của đề tài

Để xây dựng thành công chương trình quản lý rạp chiếu phim bằng ngôn ngữ C, đề tài sẽ tiếp cận theo các hướng sau:

- Nắm vững kiến thức nền tảng về ngôn ngữ lập trình C: Tập trung vào các cấu trúc dữ liệu (như danh sách liên kết đơn để quản lý thông tin phim, lịch chiếu, vé...), thuật toán và kỹ thuật lập trình cơ bản.
- Phân tích và thiết kế hệ thống: Xác định rõ các chức năng cần có của chương trình (quản lý phim, quản lý lịch chiếu, quản lý phòng chiếu, quản lý vé, quản lý người dùng, báo cáo...).
- Xây dựng cấu trúc dữ liệu phù hợp: Thiết kế các cấu trúc dữ liệu (struct) để lưu trữ thông tin về phim, suất chiếu, vé, phòng chiếu, v.v.
- Triển khai các chức năng: Viết mã C để thực hiện các chức năng đã thiết kế.

- Phát triển giao diện người dùng (nếu có): Tùy thuộc vào yêu cầu cụ thể, có thể phát triển giao diện dạng dòng lệnh (console application) hoặc giao diện đồ họa đơn giản (nếu sử dụng thư viện hỗ trợ).
- Kiểm thử và đánh giá: Tiến hành kiểm thử kỹ lưỡng các chức năng của chương trình, sửa lỗi và đánh giá hiệu quả hoạt động.

### 1.3. Mục tiêu nghiên cứu

Mục tiêu chính của đề tài nghiên cứu này là:

- Phát triển thành công chương trình quản lý rạp chiếu phim đáp ứng được các yêu cầu cơ bản về quản lý lịch chiếu, bán vé và quản lý thông tin liên quan.
- Vận dụng và củng cố kiến thức về ngôn ngữ lập trình C cùng các cấu trúc dữ liệu đã học vào việc xây dựng một ứng dụng thực tế.
- Tạo ra một công cụ hữu ích có khả năng hỗ trợ các rạp chiếu phim trong việc quản lý hoạt động kinh doanh, góp phần vào sự phát triển của ngành dịch vụ giải trí.

### 1.4. Đối tượng và phạm vi nghiên cứu

- Đối tượng nghiên cứu: Các quy trình và nghiệp vụ quản lý trong một rạp chiếu phim (bao gồm quản lý phim, lịch chiếu, phòng chiếu, vé, khách hàng).
- Đối tượng sử dụng: Người quản lý rạp chiếu phim, nhân viên bán vé, và có thể mở rộng cho khách hàng (trong trường hợp có chức năng tra cứu thông tin).
- Phạm vi ứng dụng: Chương trình được thiết kế để áp dụng cho việc quản lý tại các rạp chiếu phim có quy mô vừa và nhỏ, tập trung vào các chức năng quản lý cốt lõi.

## CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

### 2.1. Hàm

Hàm là một khối các câu lệnh được nhóm lại với nhau để thực hiện một nhiệm vụ cụ thể. Chúng là các khối xây dựng cơ bản của một chương trình C, đóng vai trò trung tâm trong việc cấu trúc hóa mã nguồn<sup>1</sup>

#### 2.1.1. Mục đích và Lợi ích

- Tái sử dụng: Hàm cho phép định nghĩa một đoạn mã một lần và gọi nó nhiều lần từ các vị trí khác nhau trong chương trình, tránh việc lặp lại mã.
- Tính module: Chia chương trình lớn thành các hàm nhỏ hơn, mỗi hàm thực hiện một chức năng cụ thể, giúp mã nguồn dễ đọc, dễ hiểu, dễ quản lý, gỡ lỗi và bảo trì hơn.
- Trừu tượng hóa: Che giấu chi tiết triển khai bên trong hàm, người dùng chỉ cần biết chức năng và cách gọi hàm.

#### 2.1.2. Cú pháp

- Khai báo hàm:
  - Mục đích: Thông báo cho trình biên dịch về tên hàm, kiểu trả về và kiểu của các tham số trước khi hàm được gọi. Điều này giúp trình biên dịch kiểm tra kiểu dữ liệu của các đối số truyền vào khi gọi hàm, đảm bảo tính đúng đắn và phát hiện lỗi sớm.
  - Cú pháp:

```
kiểu_trả_về tên_hàm(danh_sách_kiểu_tham_số);
```

Ví dụ:

```
int tinhTong(int, int);           // Khai báo hợp lệ
void hienThiThongBao(void);       // Khai báo hàm không có tham số
```

- Vị trí: Thường đặt ở đầu tệp mã nguồn hoặc trong các tệp tiêu đề (.h) để sử dụng trong nhiều tệp. Khai báo là không bắt buộc nếu định nghĩa hàm xuất hiện trước lời gọi hàm đầu tiên. Việc thiếu khai báo có thể dẫn đến cảnh báo hoặc lỗi, đặc biệt khi trình biên dịch mặc định giả định kiểu trả về là int.
- Định nghĩa hàm:

- Cung cấp phần thân (body) của hàm, chứa các câu lệnh thực thi nhiệm vụ của hàm.
- Cú pháp:

```
kiểu_trả_về tên_hàm(danh_sách_tham_số_hình_thức) {
    // Thân hàm: các câu lệnh
    [return giá_trị_trả_về;]
}
```

- Các thành phần bao gồm: kiểu\_trả\_về (có thể là void), tên\_hàm, danh\_sách\_tham\_số\_hình\_thức (formal parameters - các biến nhận giá trị đầu vào), và thân\_hàm.

#### - Gọi hàm:<sup>2</sup>

- Thực thi mã lệnh bên trong thân hàm bằng cách sử dụng tên hàm và cung cấp các đối số thực tế tương ứng với danh sách tham số.
- Cú pháp:

```
[biến_nhận_kết_quả = ] tên_hàm(danh_sách_đối_số);
```

- Khi một hàm được gọi, luồng điều khiển của chương trình sẽ nhảy đến điểm bắt đầu của định nghĩa hàm, thực thi các câu lệnh trong thân hàm. Sau khi hàm kết thúc (gặp lệnh return hoặc dấu } cuối cùng), luồng điều khiển quay trở lại vị trí ngay sau lời gọi hàm trong mã nguồn gọi nó.

#### 2.1.2. Truyền tham số

- Đối số là giá trị truyền vào khi gọi hàm. Tham số là biến được khai báo để nhận các giá trị này.
- Truyền bằng giá trị:
  - Cơ chế mặc định trong C.
  - Bản sao của đối số được tạo và gán cho tham số.
  - Thay đổi tham số không ảnh hưởng đến đối số gốc.
- Truyền bằng tham chiếu (sử dụng con trỏ):
  - C không hỗ trợ trực tiếp.
  - Truyền địa chỉ của đối số (&). Tham số là con trỏ.
  - Hàm có thể thay đổi giá trị của đối số gốc qua con trỏ.

#### 2.1.3. Giá trị trả về

- Hàm trả về một giá trị duy nhất qua return.
- Kiểu trả về được khai báo ở phần khai báo và định nghĩa.

- Hàm không trả về giá trị thì có kiểu void.
- return kết thúc hàm và trả quyền điều khiển về nơi gọi.
- Hàm không trả về trực tiếp mảng, mà trả về con trỏ tới phần tử đầu tiên của mảng.

#### 2.1.4. Ví dụ Minh họa

- Ví dụ 1: Hàm trả về giá trị, truyền tham số theo giá trị

```
int timMax(int so1, int so2) {
    // so1 và so2 là bản sao của a và b
    if (so1 > so2) {
        return so1; // Trả về giá trị lớn hơn
    } else {
        return so2;
    }
}
```

- Ví dụ 2: Hàm mô phỏng truyền tham chiếu bằng con trỏ

```
void hoanDoi(int *px, int *py) {
    // px giữ địa chỉ của x, py giữ địa chỉ của y
    int temp = *px; // Lấy giá trị tại địa chỉ px (giá trị của x)
    *px = *py; // Gán giá trị tại địa chỉ py cho địa chỉ px
    *py = temp; // Gán giá trị temp cho địa chỉ py
}
```

- Ví dụ 3: Hàm void không trả về giá trị

```
void inLoiChao(void) {
    printf("Xin chao the gioi!\n");
    // Không có lệnh return giá trị
}
```

## 2.2. Con trỏ

Con trỏ trong C cho phép thao tác trực tiếp bộ nhớ.

### 2.2.1 Định nghĩa và Địa chỉ Bộ nhớ

Con trỏ là biến lưu địa chỉ bộ nhớ của biến khác, cho phép truy cập gián tiếp dữ liệu.

Mỗi biến có địa chỉ duy nhất.

Con trỏ cần thiết cho cấp phát động, cấu trúc dữ liệu phức tạp và truyền tham số hiệu quả.

### 2.2.2. Khai báo và Khởi tạo

- Cú pháp:

```
kiểu_dữ_liệu *tên_con_trỏ;
```



- `kiểu_dữ_liệu`: Chỉ định kiểu của biến mà con trỏ sẽ trỏ tới (ví dụ: `int`, `char`, `float`, `struct SinhVien`). Kiểu dữ liệu này quan trọng vì nó cho trình biên dịch biết kích thước của dữ liệu tại địa chỉ được trỏ tới, cần thiết cho phép toán số học trên con trỏ.
  - `*`: Dấu hoa thị cho biết đây là khai báo một biến con trỏ.
- Khởi tạo:
- Gán địa chỉ biến:
- ```
tên_con_trỏ = &tên_biến;
```
- Gán NULL:
- ```
tên_con_trỏ = NULL;
```
- Gán địa chỉ từ hàm cấp phát động (`malloc`, `calloc`, `realloc`).
  - Con trỏ NULL: Biểu thị con trỏ không trỏ đến vùng nhớ hợp lệ, cần để tránh con trỏ hoang. Luôn kiểm tra NULL trước khi giải tham chiếu.

### 2.2.3. Các Toán tử Liên quan

- Toán tử lấy địa chỉ `&`: Trả về địa chỉ bộ nhớ của biến.
- Toán tử giải tham chiếu `*`:
- Trong khai báo: chỉ định biến là con trỏ.
- Trong biểu thức: truy cập giá trị tại địa chỉ con trỏ trỏ tới.

### 2.2.4. Số học Con trỏ Cơ bản:

- Các phép toán `++`, `--`, `+`, `-` trên con trỏ dựa trên kích thước kiểu dữ liệu trỏ tới.
- `ptr++/ptr + 1`: Di chuyển đến phần tử kế tiếp (`địa_chỉ_mới = địa_chỉ_cũ + sizeof(*ptr)`).
- `ptr--/ptr - 1`: Di chuyển đến phần tử trước đó (`địa_chỉ_mới = địa_chỉ_cũ - sizeof(*ptr)`).
- `ptr + n`: Di chuyển đến phần tử thứ `n` (`địa_chỉ_mới = địa_chỉ_cũ + n * sizeof(*ptr)`).
- `ptr - n`: Di chuyển lùi lại `n` phần tử (`địa_chỉ_mới = địa_chỉ_cũ - n * sizeof(*ptr)`).
- Quan trọng khi làm việc với mảng.
- So sánh con trỏ (`==`, `!=`, `<`, `>`, `<=`, `>=`) so sánh địa chỉ.
- Con trỏ Void (`void`):\*

- Con trỏ tổng quát, lưu địa chỉ mọi kiểu dữ liệu nhưng không có thông tin về kiểu.
- Dùng trong hàm cấp phát động và hàm generic.
- Không thể giải tham chiếu trực tiếp void\*, cần ép kiểu trước.
- Phép toán số học trên void\* không chuẩn.

#### 2.2.5. Các Lỗi Thường Gặp với Con trỏ:

- Con trỏ Chưa Khởi tạo: Chứa địa chỉ rác, gây lỗi khi giải tham chiếu. Luôn khởi tạo (ít nhất là NULL).
- Giải Tham chiếu Con trỏ NULL: Truy cập địa chỉ NULL, gây lỗi nghiêm trọng. Luôn kiểm tra NULL.
- Con trỏ: Trỏ đến vùng nhớ đã giải phóng hoặc ngoài phạm vi. Gây hành vi không xác định. Gán NULL sau khi free().
- Rò rỉ Bộ nhớ: Bộ nhớ cấp phát động không được giải phóng.
  - Con trỏ mang lại hiệu năng và linh hoạt nhưng đòi hỏi lập trình viên quản lý bộ nhớ cẩn thận để tránh lỗi. void\* hữu ích cho tính tổng quát nhưng cần thận trọng khi ép kiểu để đảm bảo an toàn kiểu dữ liệu.

#### 2.2.6. Ví dụ Minh họa

- Ví dụ: Khai báo, khởi tạo, giải tham chiếu cơ bản

```
#include <stdio.h>

int main() {
    int soNguyen = 100;
    int *ptrSoNguyen;           // Khai báo con trỏ tới int

    // Khởi tạo: gán địa chỉ của soNguyen cho con trỏ
    ptrSoNguyen = &soNguyen;

    printf("Gia tri cua soNguyen: %d\n", soNguyen);
    printf("Dia chi cua soNguyen: %p\n", &soNguyen);
    printf("Gia tri cua ptrSoNguyen (dia chi cua soNguyen): %p\n", ptrSoNguyen);

    // Giải tham chiếu để lấy giá trị
    printf("Gia tri tai dia chi ma ptrSoNguyen tro toi: %d\n", *ptrSoNguyen);

    // Thay đổi giá trị của soNguyen thông qua con trỏ
    *ptrSoNguyen = 200;
    printf("Gia tri moi cua soNguyen sau khi thay doi qua con tro: %d\n", soNguyen);

    return 0;
}
```

## 2.3. Con trỏ và Mảng

### 2.3.1. Mối quan hệ giữa con trỏ và mảng trong C

#### - Quan hệ: Sự Suy biến của Tên Mảng:

- Trong hầu hết các ngữ cảnh biểu thức, tên mảng (ví dụ: `mang`) tự động suy biến thành con trỏ trỏ đến phần tử đầu tiên của mảng (`&mang`). Do đó, `mang` chứa địa chỉ của `mang[0]`.
- Tên mảng hoạt động như một hằng con trỏ, giá trị (địa chỉ bắt đầu) không thể thay đổi sau khi khai báo (không thể gán như `mang = &biến_khac` hoặc thay đổi địa chỉ như `mang++`).

#### - Truy cập Phần tử Mảng:

- Ký pháp Chỉ số (`mang[i]`): Cách truy cập phần tử tại chỉ số `i` thông thường.
- Ký pháp Con trỏ (`*(mang + i)`): Tương đương với `mang[i]`.
  - `mang`: Suy biến thành địa chỉ phần tử đầu tiên.
  - `+ i`: Thực hiện số học con trỏ, di chuyển đến địa chỉ phần tử thứ `i` (`địa_chỉ_mang + i * sizeof(kiểu_phần_tử)`).
  - `*`: Giải tham chiếu, lấy giá trị tại địa chỉ kết quả.
- Ký pháp Biến Con trỏ (`*(ptr + i)` hoặc `ptr[i]`): Nếu `ptr` trỏ đến mảng (ví dụ: `int *ptr = mang;`), có thể truy cập phần tử qua `ptr` bằng cả hai cách.

#### - Số học Con trỏ với Mảng:

- Các phép toán `++`, `--`, `+ i`, `- i` trên con trỏ trỏ vào mảng sẽ di chuyển con trỏ đến các phần tử liền kề hoặc cách `i` phần tử dựa trên kích thước kiểu dữ liệu. Đây là cơ chế duyệt mảng bằng con trỏ.
- So sánh con trỏ (`ptr1 < ptr2`, `ptr1 == &mang[k]`) hợp lệ và hữu ích khi chúng cùng trỏ vào các phần tử của cùng mảng.

Sự tương đương giữa `a[i]` và `*(a + i)` cho phép C xử lý mảng và con trỏ linh hoạt. Khi truyền mảng vào hàm, chỉ địa chỉ phần tử đầu tiên (con trỏ) được truyền, làm mất

thông tin về kích thước gốc của mảng. Do đó, việc truyền kích thước mảng như một tham số riêng là thông lệ bắt buộc. Bên trong hàm, `sizeof(mang)` (dù khai báo là `int mang[]` hay `int *mang`) sẽ trả về kích thước của con trỏ, không phải kích thước mảng. Mặc dù trình biên dịch hiện đại thường tối ưu hóa cả hai cách truy cập để có hiệu năng tương đương, duyệt mảng bằng cách tăng trực tiếp con trỏ (`ptr++`) đôi khi được coi là cách viết "C" tự nhiên hơn và có thể gần với mã máy hơn trong một số trường hợp. Hiểu cả hai phương pháp là cần thiết để đọc và viết mã C hiệu quả.

### 2.3.2. Ví dụ Minh họa

- Ví dụ 1: Truy cập các phần tử mảng bằng con trỏ.

```
#include <stdio.h>

int main() {
    int numbers = {1, 2, 3, 4, 5};
    int *ptr = numbers; // ptr trỏ tới numbers

    printf("Phan tu 1: %d\n", *ptr);    // Output: Phan tu 1: 1
    printf("Phan tu 2: %d\n", *(ptr + 1)); // Output: Phan tu 2: 2
    printf("Phan tu 3: %d\n", ptr);    // Output: Phan tu 3: 3
    printf("Phan tu 4: %d\n", numbers); // Output: Phan tu 4: 4
    printf("Phan tu 5: %d\n", *(numbers + 4)); // Output: Phan tu 5: 5

    return 0;
}
```

Ví dụ này cho thấy nhiều cách khác nhau để truy cập các phần tử của mảng `numbers` bằng cách sử dụng con trỏ `ptr` và tên mảng `numbers`<sup>3</sup>

- Ví dụ 2: Mảng truyền vào hàm

```
#include <stdio.h>

// Hàm nhận mảng (thực chất là con trỏ) và kích thước
void inMang(int *arr, int size) {                // hoặc int arr
    printf("Ben trong ham inMang:\n");
    printf("Kich thuc cua tham so 'arr' (kich thuc con tro): %zu bytes\n", sizeof(arr));
    for (int i = 0; i < size; i++) {
        printf("arr[%d] = %d (tai dia chi %p)\n", i, *(arr + i), arr + i);
    }
}

int main() {
    int duLieu = {11, 22, 33};
    int kichThuoc = sizeof(duLieu) / sizeof(duLieu);

    printf("Ben trong ham main:\n");
    printf("Kich thuc cua mang 'duLieu': %zu bytes\n", sizeof(duLieu));
    inMang(duLieu, kichThuoc);                  // Truyền mảng và kích thước
}
```

```
return 0;  
}
```

## 2.4. Mảng Con trỏ

Mảng con trỏ là cấu trúc dữ liệu mạnh mẽ trong C, quản lý hiệu quả tập hợp dữ liệu, đặc biệt là chuỗi ký tự có độ dài khác nhau.<sup>4</sup>

### 2.4.1. Định nghĩa và Cú pháp Khai báo

Mảng con trỏ là mảng mà mỗi phần tử là một biến con trỏ, trỏ đến một vị trí trong bộ nhớ.

- Cú pháp:

```
kiểu_dữ_liệu *tên_mảng[số_lượng];
```

**kiểu\_dữ\_liệu:** Kiểu dữ liệu của đối tượng mà con trỏ trong mảng sẽ trỏ tới (ví dụ: int, char, struct SinhVien).

**tên\_mảng:** Tên của mảng con trỏ.

**[số\_lượng]:** Số lượng con trỏ mà mảng chứa.

### 2.4.2. Trường hợp Sử dụng

**Mảng Chuỗi ký tự (char \*strings[]):** Ứng dụng phổ biến nhất. Mỗi phần tử là char\*, trỏ đến ký tự đầu của một chuỗi (mảng ký tự kết thúc bằng \0). Hiệu quả để lưu trữ danh sách chuỗi có độ dài khác nhau.

**Mảng con trỏ tới Struct/Integer/etc.:** Quản lý tập hợp cấu trúc hoặc biến số nguyên, đặc biệt khi cấp phát động hoặc cần sắp xếp/thao tác dựa trên địa chỉ.

### 2.4.3. So sánh với Mảng 2 Chiều

**Sử dụng Bộ nhớ:** char \*mang\_chuoi[] thường tiết kiệm bộ nhớ hơn char mang\_2d[][] khi lưu trữ chuỗi có độ dài thay đổi. Mảng 2D cấp phát SO\_HANG \* SO\_COT byte, có thể lãng phí nếu nhiều chuỗi ngắn hơn SO\_COT. Mảng con trỏ chỉ cấp phát bộ nhớ cho con trỏ cộng với bộ nhớ thực tế cho mỗi chuỗi.

**Tính Linh hoạt:** Mảng con trỏ lưu trữ chuỗi có độ dài bất kỳ, mảng 2D yêu cầu độ dài tối đa cố định cho tất cả chuỗi.

**Thao tác:** Hoán đổi vị trí chuỗi trong mảng con trỏ chỉ cần hoán đổi giá trị con trỏ (địa chỉ), nhanh hơn sao chép toàn bộ nội dung trong mảng 2D.

### 2.4.4. Khởi tạo và Sử dụng

- Khởi tạo mảng con trỏ chuỗi:

```
char *danhSachTen[] = {"An", "Binh", "Cuong"};
```

- Khởi tạo mảng con trỏ số nguyên:

```
int a=1, b=2, c=3;
int *ptrs[] = {&a, &b, &c};
```

- Truy cập dữ liệu được trỏ tới:

```
printf("Ten: %s\n", danhSachTen[i]);
printf("So: %d\n", *ptrs[i]);
```

Việc dùng mảng con trỏ tạo ra mức độ gián tiếp: mảng chứa địa chỉ nơi dữ liệu được lưu trữ, không chứa dữ liệu trực tiếp. Sự tách biệt này cho phép mỗi phần tử dữ liệu (ví dụ: chuỗi) có kích thước riêng, tối ưu hóa bộ nhớ so với cấu trúc cố định như mảng 2D. Các thao tác như sắp xếp, hoán đổi hiệu quả hơn vì chỉ cần thay đổi con trỏ thay vì sao chép dữ liệu lớn. Cần phân biệt rõ cấu trúc bộ nhớ của char mang[C] (khối liên kết R\*C byte) và char \*mang[R] (mảng R con trỏ, mỗi con trỏ trỏ đến chuỗi ở vị trí bất kỳ) để tránh truy cập bộ nhớ sai lệch.

#### 2.4.5. Ví dụ Minh họa

- Ví dụ 1: Mảng con trỏ số nguyên

```
#include <stdio.h>

int main() {
    int num1 = 10, num2 = 20, num3 = 30;
    int *ptr_arr; // Mảng chứa 3 con trỏ tới int

    ptr_arr = &num1;
    ptr_arr = &num2;
    ptr_arr = &num3;

    printf("Gia tri cac bien thong qua mang con tro:\n");
    for (int i = 0; i < 3; i++) {
        printf("ptr_arr[%d] tro toi gia tri: %d tai dia chi %p\n", i, *ptr_arr[i], ptr_arr[i]);
    }
    return 0;
}
```

- Ví dụ 2: Mảng con trỏ tới cấu trúc

```
#include <stdio.h>

// Giả sử chúng ta có một cấu trúc (struct) tên là MyStruct
struct MyStruct {
    int id;
    char name[50];
};
```

```

};

int main() {
    // Khai báo các biến cấu trúc
    struct MyStruct s1 = {1, "Struct 1"};
    struct MyStruct s2 = {2, "Struct 2"};
    struct MyStruct s3 = {3, "Struct 3"};

    // Khai báo một mảng con trỏ tới MyStruct
    // và khởi tạo nó với địa chỉ của các biến cấu trúc
    struct MyStruct *ptrArray[3] = {&s1, &s2, &s3};

    // Truy cập và in dữ liệu thông qua mảng con trỏ
    for (int i = 0; i < 3; i++) {
        printf("ID: %d, Name: %s\n", ptrArray[i]->id, ptrArray[i]->name);
    }

    return 0;
}

```

## 2.5. Con trỏ Hàm

Con trỏ hàm là một biến thể của con trỏ, dùng để lưu trữ địa chỉ của một hàm, cho phép gọi hàm đó một cách gián tiếp.

### 2.5.1. Định nghĩa và Mục đích

- Con trỏ hàm là biến chứa địa chỉ bộ nhớ nơi mã thực thi của một hàm bắt đầu.
- Mục đích chính là cho phép xử lý hàm như dữ liệu, ví dụ: truyền hàm làm đối số (callback), trả về hàm từ hàm khác, tạo mảng hàm (jump table) để gọi hàm động. Điều này giúp tăng tính linh hoạt, tái sử dụng và mở rộng mã nguồn.

### 2.5.2. Cú pháp Khai báo

- Cú pháp: kiểu\_trả\_về (\*tên\_con\_trỏ)(danh\_sách\_kiểu\_tham\_số);.
- kiểu\_trả\_về, tên\_con\_trỏ, và danh\_sách\_kiểu\_tham\_số phải khớp hoàn toàn với chữ ký của hàm mà con trỏ sẽ trỏ tới.
- Dấu ngoặc đơn (\*tên\_con\_trỏ) là bắt buộc để phân biệt với khai báo hàm trả về con trỏ.

Ví dụ :

```

// Con trỏ tới hàm nhận 2 int, trả về int
int (*pCong)(int, int);

// Con trỏ tới hàm nhận char*, trả về void
void (*pInThongBao)(char*);

```

### 2.5.3. Gán Địa chỉ Hàm

- Tên hàm (không có dấu ()) tự động chuyển thành địa chỉ của hàm đó.
- Cú pháp gán:

```
tên_con_trò = tên_hàm; hoặc tên_con_trò = &tên_hàm;
```

Ví dụ :

```
int cong(int a, int b) {  
    return a + b;  
}  
  
pCong = cong;           // Gán địa chỉ hàm cong cho con trỏ pCong
```

### 2.5.4. Gọi Hàm qua Con trỏ

- Sử dụng tên con trỏ hàm theo sau là dấu ngoặc đơn chứa các đối số.

Cú pháp:

```
(*tên_con_trò)(danh_sách_đối_số);  
// hoặc  
tên_con_trò(danh_sách_đối_số);      (cách sau phổ biến hơn)
```

Ví dụ:

```
int ketQua = pCong(5, 3);
```

### 2.5.5. Trường hợp Sử dụng

- Callbacks: Truyền con trỏ hàm vào hàm khác để hàm đó thực thi hàm được truyền vào (ví dụ: hàm qsort trong thư viện C).
- Jump Tables (Mảng Con trỏ Hàm): Lưu trữ địa chỉ nhiều hàm trong mảng, cho phép chọn và thực thi hàm dựa trên chỉ số, thay thế cấu trúc switch-case phức tạp.
- Các ứng dụng khác: Máy trạng thái (State Machines), xử lý sự kiện, kiến trúc plugin, xử lý ngắt trong lập trình nhúng.

Con trỏ hàm cho phép quyết định hàm nào được thực thi tại thời điểm chạy, thay vì xác định cứng lúc biên dịch, tạo nên tính linh hoạt. Cơ chế callback tách logic chung khỏi hành vi cụ thể. Jump tables cung cấp phương pháp hiệu quả để điều phối luồng thực thi. Dù cú pháp phức tạp, việc yêu cầu khớp kiểu trả về và tham số giúp trình biên dịch kiểm tra kiểu, tăng độ an toàn so với dùng void\*.



### 2.5.6. Ví dụ Minh họa

```
#include <stdio.h>

int phepCong(int a, int b) {
    return a + b;
}
int phepTru(int a, int b) {
    return a - b;
}
int main() {
    int (*pTinhToan)(int, int);           // Khai báo con trỏ hàm
    pTinhToan = phepCong;                 // Gán địa chỉ hàm phepCong
    int tong = pTinhToan(10, 5);           // Gọi hàm phepCong qua con trỏ
    printf("Tong: %d\n", tong);            // Output: Tong: 15

    pTinhToan = phepTru;                   // Gán địa chỉ hàm phepTru
    int hieu = pTinhToan(10, 5);           // Gọi hàm phepTru qua con trỏ
    printf("Hieu: %d\n", hieu);            // Output: Hieu: 5
    return 0;
}
```

## 2.6. Cấp phát Động

Cấp phát bộ nhớ động là kỹ thuật cho phép chương trình quản lý bộ nhớ linh hoạt trong quá trình thực thi.<sup>5</sup>

### 2.6.1. Khái niệm: Heap và Stack

- Chương trình C sử dụng hai vùng nhớ chính:
  - Stack: Dùng cho cấp phát tĩnh (biến cục bộ, tham số hàm), quản lý tự động theo cơ chế LIFO, truy cập nhanh, kích thước giới hạn.
  - Heap: Dùng cho cấp phát động, quản lý thủ công bởi lập trình viên, truy cập chậm hơn stack, kích thước linh hoạt và lớn hơn.
- Sự cần thiết: Cần thiết khi kích thước bộ nhớ không biết trước lúc biên dịch, khi cần cấp phát bộ nhớ lớn hơn stack, hoặc khi dữ liệu cần tồn tại lâu hơn phạm vi hàm. Nền tảng cho cấu trúc dữ liệu động.

### 2.6.2. Các Hàm Cấp phát và Giải phóng

```
malloc(size_t size);
```

- Cấp phát một khối bộ nhớ liên tục size byte trên heap.
- Trả về con trỏ void\* đến byte đầu tiên, cần ép kiểu.
- Trả về NULL nếu thất bại.
- Vùng nhớ được cấp phát không được khởi tạo (chứa giá trị rác).

```
calloc(size_t num, size_t size);
```

- Cấp phát bộ nhớ cho mảng num phần tử, mỗi phần tử size byte.
- Trả về void\* hoặc NULL nếu thất bại, cần ép kiểu.
- Khởi tạo tất cả các byte về 0.

```
realloc(void *ptr, size_t new_size);
```

- Thay đổi kích thước khối nhớ ptr thành new\_size byte.
- Nếu new\_size lớn hơn, cố gắng mở rộng; nếu không thể, tìm khối mới, sao chép dữ liệu, giải phóng khối cũ. Phần mở rộng không được khởi tạo.
- Nếu new\_size nhỏ hơn, khối nhớ có thể bị thu hẹp.
- Nếu ptr là NULL, hoạt động như malloc(new\_size).
- Nếu new\_size là 0 và ptr khác NULL, tương đương free(ptr).
- Trả về void\* đến khối nhớ mới (có thể khác địa chỉ cũ) hoặc NULL nếu thất bại. Nếu thất bại, ptr ban đầu vẫn nguyên vẹn.

```
free(void *ptr);
```

- Giải phóng khối bộ nhớ ptr đã cấp phát động.
- Chỉ gọi cho con trỏ hợp lệ từ malloc, calloc, realloc và chưa bị giải phóng.
- free(NULL) không làm gì cả.
- double free hoặc free con trỏ không hợp lệ gây hành vi không xác định.

#### 2.6.3. Xử lý Lỗi:

- Luôn kiểm tra giá trị trả về của malloc, calloc, realloc có phải là NULL không để xử lý lỗi cấp phát.
- Với realloc: Gán kết quả cho biến tạm. Nếu biến tạm khác NULL, mới gán lại cho con trỏ gốc để tránh mất con trỏ cũ khi realloc thất bại.

#### 2.6.4. Tầm quan trọng của free():

- Rò rỉ bộ nhớ (Memory Leak): Xảy ra nếu bộ nhớ động không được giải phóng bằng free() khi không cần thiết, làm chiếm dụng tài nguyên.
- Hậu quả: Có thể làm cạn kiệt bộ nhớ, chương trình chạy chậm, treo hoặc crash.
- Trách nhiệm của Lập trình viên: Phải gọi free() đúng lúc, đúng chỗ, và chỉ một lần.
- Thực hành tốt: Sau free(ptr), gán ptr = NULL; để tránh con trỏ treo.

Cấp phát động là công cụ mạnh mẽ nhưng đòi hỏi quản lý bộ nhớ chặt chẽ. Lựa chọn giữa malloc (nhanh, không khởi tạo) và calloc (chậm hơn, khởi tạo về 0) tùy yêu cầu. realloc cần xử lý cẩn thận. free() là then chốt cho chương trình ổn định.

### 2.6.5. Ví dụ Minh họa

#### Ví dụ 1: malloc và free

```
#include <stdio.h>
#include <stdlib.h> // Cần thư viện này cho malloc, free

int main() {
    int *duLieu;
    int n = 5;

    // Cấp phát bộ nhớ cho mảng 5 số nguyên
    duLieu = (int*)malloc(n * sizeof(int));

    // Kiểm tra cấp phát thành công
    if (duLieu == NULL) {
        fprintf(stderr, "Loi: Khong the cap phat bo nho!\n");
        return 1; // Thoát với mã lỗi
    }

    printf("Da cap phat thanh cong %zu bytes.\n", n * sizeof(int));

    // Sử dụng bộ nhớ (ví dụ: gán giá trị)
    for (int i = 0; i < n; i++) {
        duLieu[i] = i * 10;
        printf("duLieu[%d] = %d\n", i, duLieu[i]);
    }

    // Giải phóng bộ nhớ khi không cần dùng nữa
    free(duLieu);
    printf("Da giai phong bo nho.\n");
    duLieu = NULL; // Gán NULL cho con trỏ sau khi free

    return 0;
}
```

#### Ví dụ 2: malloc, realloc và free

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

int main() {
    int *mang;
    int kichThuocBanDau = 3;
    int kichThuocMoi = 5;

    // Cấp phát bộ nhớ ban đầu
    mang = (int*)malloc(kichThuocBanDau * sizeof(int));
    if (mang == NULL) {
        fprintf(stderr, "Loi cap phat ban dau!\n");
        return 1;
    }
}
```

```

    }
    printf("Cap phat ban dau %d phan tu thanh cong.\n", kichThuocBanDau);
    for(int i=0; i<kichThuocBanDau; ++i) mang[i] = i+1;

    // Thay đổi kích thước mảng
    printf("Thay doi kich thuoc thanh %d phan tu...\n", kichThuocMoi);
    int *mangTam = (int*)realloc(mang, kichThuocMoi * sizeof(int));

    // Kiểm tra realloc thành công
    if (mangTam == NULL) {
        fprintf(stderr, "Loi khi thay doi kich thuoc (realloc)!\n");
        // Quan trọng: giải phóng bộ nhớ cũ nếu realloc thất bại
        free(mang);
        return 1;
    }

    // Chỉ gán lại con trỏ nếu realloc thành công
    mang = mangTam;
    printf("Thay doi kich thuoc thanh cong.\n");

    // Gán giá trị cho phần tử mới (nếu mở rộng)
    if (kichThuocMoi > kichThuocBanDau) {
        for(int i=kichThuocBanDau; i<kichThuocMoi; ++i) mang[i] = (i+1)*10;
    }

    // In mảng sau khi thay đổi kích thước
    printf("Mang sau khi realloc:\n");
    for (int i = 0; i < kichThuocMoi; i++) {
        printf("%d ", mang[i]);
    }
    printf("\n");

    // Giải phóng bộ nhớ
    free(mang);
    printf("Da giai phong bo nho.\n");
    mang = NULL;

    return 0;
}

```

## 2.7. Xử lý Tập

Xử lý tập trong C là quá trình tương tác với các tập trên thiết bị lưu trữ, bao gồm tạo, mở, đọc, ghi và đóng tập.<sup>6</sup>

### 2.7.1. Khái niệm và Con trỏ FILE

- C sử dụng cấu trúc FILE (trong <stdio.h>) để làm việc với tập. Cấu trúc này chứa thông tin quản lý luồng dữ liệu vào/ra của tập (vị trí con trỏ tập, trạng thái lỗi, bộ đệm).
- Mọi thao tác tập được thực hiện qua con trỏ tới cấu trúc FILE (con trỏ tập), ví dụ: FILE \*fp;.

### 2.7.2. Phân loại Tập

- Tập Văn bản (Text Files): Lưu dữ liệu dạng chuỗi ký tự đọc được (ASCII, UTF-8), dòng kết thúc bằng \n. Có thể có chuyển đổi ký tự tự động (ví dụ: \n thành \r\n trên Windows). Thích hợp cho tệp cấu hình, mã nguồn.
- Tập Nhị phân (Binary Files): Lưu dữ liệu dạng byte thô như trong bộ nhớ, không có chuyển đổi ký tự. Thích hợp cho dữ liệu có cấu trúc (struct), hình ảnh, âm thanh, đảm bảo tính toàn vẹn byte.

### 2.7.3. Mở và Đóng Tập

- fopen(const char \*filename, const char \*mode):
  - Dùng để mở tệp. filename là tên tệp (có thể kèm đường dẫn), mode là chuỗi chỉ định chế độ truy cập.
  - Trả về FILE\* nếu thành công, NULL nếu lỗi. Cần kiểm tra NULL để đảm bảo mở tệp thành công.
  - Bảng Chế độ Mở Tệp (fopen() Modes):

Mode	Mô tả	Nếu Tệp Tồn tại	Nếu Tệp Không Tồn tại	Kiểu
"r"	Mở để đọc	Đọc từ đầu	Lỗi (NULL)	Text
"w"	Mở để ghi (tạo mới/ghi đè)	Xóa nội dung, ghi từ đầu	Tạo tệp mới	Text
"a"	Mở để ghi nối vào cuối (append)	Ghi vào cuối	Tạo tệp mới	Text
"rb"	Mở để đọc	Đọc từ đầu	Lỗi (NULL)	Binary
"wb"	Mở để ghi (tạo	Xóa nội dung,	Tạo tệp mới	Binary

	mới/ghi đè)	ghi từ đầu		
"ab"	Mở để ghi nối vào cuối (append)	Ghi vào cuối	Tạo tệp mới	Binary
"r+"	Mở để đọc và ghi	Đọc/ghi từ đầu	Lỗi (NULL)	Text
"w+"	Mở để đọc và ghi (tạo mới/ghi đè)	Xóa nội dung, đọc/ghi	Tạo tệp mới	Text
"a+"	Mở để đọc và ghi nối vào cuối	Đọc từ đầu, ghi vào cuối	Tạo tệp mới	Text
"rb+"	Mở để đọc và ghi	Đọc/ghi từ đầu	Lỗi (NULL)	Binary
"wb+"	Mở để đọc và ghi (tạo mới/ghi đè)	Xóa nội dung, đọc/ghi	Tạo tệp mới	Binary
"ab+"	Mở để đọc và ghi nối vào cuối	Đọc từ đầu, ghi vào cuối	Tạo tệp mới	Binary

Bảng 2.7: Bảng Chế độ Mở Tệp

- `fclose(FILE *fp)`:

- Đóng tệp liên kết với `fp`, đảm bảo dữ liệu trong bộ đệm được ghi vào đĩa và giải phóng tài nguyên.
- Trả về 0 nếu thành công, EOF nếu lỗi. Quên đóng tệp có thể gây mất dữ liệu.

#### 2.7.4. Thao tác Vào/Ra trên Tệp Văn bản

- `fprintf(FILE *fp, const char *format, ...)`: Ghi dữ liệu định dạng vào tệp, tương tự `printf`.

- `fscanf(FILE *fp, const char *format, ...)`: Đọc dữ liệu định dạng từ tệp, tương tự `scanf`. Trả về số mục đọc được hoặc EOF.
- `fputs(const char *str, FILE *fp)`: Ghi chuỗi str vào tệp (không tự thêm `\n`).
- `fgets(char *str, int n, FILE *fp)`: Đọc một dòng (tối đa n-1 ký tự) vào str. Bao gồm `\n` (nếu có) và kết thúc bằng `\0`. Trả về str hoặc NULL.
- `fgetc(FILE *fp)` / `fputc(int c, FILE *fp)`: Đọc/ghi một ký tự.

#### 2.7.5. Thao tác Vào/Ra trên Tệp Nhị phân

- `fwrite(const void *ptr, size_t size, size_t count, FILE *fp)`: Ghi count phần tử, mỗi phần tử size byte, từ ptr vào tệp. Trả về số phần tử ghi thành công.
- `fread(void *ptr, size_t size, size_t count, FILE *fp)`: Đọc count phần tử, mỗi phần tử size byte, từ tệp vào ptr. Trả về số phần tử đọc thành công.

#### 2.7.6. Định vị trong Tệp

- `fseek(FILE *fp, long offset, int whence)`: Di chuyển con trỏ tệp. offset là số byte dịch chuyển. whence là vị trí gốc (`SEEK_SET`: đầu tệp, `SEEK_CUR`: vị trí hiện tại, `SEEK_END`: cuối tệp). Trả về 0 nếu thành công.
- `ftell(FILE *fp)`: Trả về vị trí hiện tại của con trỏ tệp (long int) hoặc -1L nếu lỗi. Kết hợp `fseek(fp, 0, SEEK_END)` và `ftell(fp)` để xác định kích thước tệp.
- `rewind(FILE *fp)`: Đặt lại con trỏ tệp về đầu tệp (tương đương `fseek(fp, 0, SEEK_SET)`), xóa cờ lỗi.

#### 2.7.6. Xử lý Lỗi Tệp

- Kiểm tra `fopen()` trả về NULL.
- `feof(FILE *fp)`: Kiểm tra cuối tệp. Trả về true sau khi cố đọc vượt cuối tệp. Dùng để xác nhận nguyên nhân lỗi của hàm đọc là do hết tệp.
- `ferror(FILE *fp)`: Kiểm tra lỗi đọc/ghi. Trả về true nếu có lỗi I/O. Dùng để phân biệt lỗi I/O và cuối tệp. `clearerr(fp)` để xóa cờ lỗi.
- `perror(const char *s)`: In chuỗi s và thông báo lỗi hệ thống tương ứng với `errno`. Hữu ích để biết lý do `fopen` hoặc hàm I/O khác thất bại.

Sự khác biệt giữa chế độ văn bản (chuyển đổi ký tự) và nhị phân (byte-by-byte) là quan trọng. Chế độ nhị phân đảm bảo toàn vẹn dữ liệu phi văn bản. Xử lý lỗi cẩn thận ở mọi bước (mở, đọc/ghi, đóng) là cần thiết để chương trình làm việc với tệp đáng tin cậy.

### 2.7.7. Ví dụ Minh họa

- Ví dụ : Đọc/Ghi Tập Văn bản (dùng fprintf/fscanf)

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

int main() {
    FILE *fp;
    char ten;
    int tuoi;

    // Ghi vào tập văn bản
    fp = fopen("vanban.txt", "w");
    if (fp == NULL) {
        perror("Loi mo file de ghi");
        return 1;
    }
    fprintf(fp, "Nguyen Van A 25\n");
    fprintf(fp, "Tran Thi B 30\n");
    if (fclose(fp) == EOF) {
        perror("Loi dong file sau khi ghi");
        // Có thể vẫn tiếp tục đọc nếu file đã được tạo
    }
    printf("Da ghi du lieu vao vanban.txt\n");

    // Đọc từ tập văn bản
    fp = fopen("vanban.txt", "r");
    if (fp == NULL) {
        perror("Loi mo file de doc");
        return 1;
    }
    printf("\nDoc du lieu tu vanban.txt:\n");
    // Đọc từng dòng cho đến khi hết tập hoặc có lỗi
    while (fscanf(fp, "%s %d", ten, &tuoi) == 2) {
        printf("Ten: %s, Tuoi: %d\n", ten, tuoi);
    }

    // Kiểm tra nguyên nhân dừng vòng lặp
    if (ferror(fp)) {
        perror("Loi trong qua trinh doc file");
    } else if (feof(fp)) {
        printf("Da doc den cuoi file.\n");
    }

    if (fclose(fp) == EOF) {
        perror("Loi dong file sau khi doc");
    }

    return 0;
}
```



## 2.8. Kiểu Cấu trúc (Struct)

Kiểu cấu trúc (struct) trong C là một kiểu dữ liệu do người dùng định nghĩa, cho phép nhóm các biến có kiểu dữ liệu khác nhau thành một đơn vị logic.<sup>7</sup>

### 2.8.1. Định nghĩa và Mục đích

- struct cho phép tạo kiểu dữ liệu phức tạp để biểu diễn đối tượng hoặc bản ghi thực tế có nhiều thuộc tính (ví dụ: sinh viên có mã số, họ tên, điểm; điểm 2D có tọa độ x, y).
- Nhóm các dữ liệu liên quan giúp tổ chức mã nguồn tốt hơn, dễ đọc và quản lý hơn.

### 2.8.2. Cú pháp Khai báo và typedef

- Cú pháp định nghĩa mẫu cấu trúc:

```
struct ten_cau_truc {  
    kieu_du_lieu1 ten_thanh_vien1;  
    kieu_du_lieu2 ten_thanh_vien2;  
    //... các thành viên khác  
}; // Dấu chấm phẩy ở cuối là bắt buộc
```

- Khai báo biến cấu trúc:

```
struct ten_cau_truc ten_bien;
```

- Sử dụng typedef để tạo bí danh, giúp khai báo biến ngắn gọn hơn:

```
// Cách 1: typedef sau khi định nghĩa  
struct ten_cau_truc { /* ... */ };  
typedef struct ten_cau_truc TenKieuMoi;  
  
// Cách 2: typedef kết hợp định nghĩa  
typedef struct { /* ... */ } TenKieuMoi;  
  
// Khai báo biến:  
TenKieuMoi ten_bien;
```

### 2.8.3. Truy cập Thành viên

- Toán tử Chấm (. - Dot Operator): Dùng khi làm việc trực tiếp với biến cấu trúc để truy cập thành viên. Cú pháp:

```
ten_bien_cau_truc.ten_thanh_vien
```

- Toán tử Mũi tên (-> - Arrow Operator): Dùng khi làm việc với con trỏ trỏ đến biến cấu trúc để truy cập thành viên. Cú pháp:

```
ten_con_trỏ_cau_truc->ten_thanh_vien
```

Đây là cách viết tắt cho (\*ten\_con\_trỏ\_cau\_truc).ten\_thanh\_vien.

#### 2.8.4. Con trỏ tới Struct

- Khai báo con trỏ: struct ten\_cau\_truc \*ptr; hoặc TenKieuMoi \*ptr;.
- Gán địa chỉ: ptr = &ten\_bien\_cau\_truc; hoặc cấp phát động: ptr = (TenKieuMoi\*)malloc(sizeof(TenKieuMoi));.
- Truy cập thành viên qua con trỏ bằng toán tử ->.

#### 2.8.5. Mảng Struct

- Khai báo mảng các cấu trúc: struct ten\_cau\_truc ten\_mang[kich\_thuoc]; hoặc TenKieuMoi ten\_mang[kich\_thuoc];.
- Truy cập một phần tử (struct) trong mảng: ten\_mang[chi\_so]
- Truy cập một thành viên của một phần tử struct trong mảng: ten\_mang[chi\_so].ten\_thanh\_vien.

Struct là nền tảng cho việc mô hình hóa dữ liệu phức tạp trong C. Chúng cho phép tạo kiểu dữ liệu mới bằng cách kết hợp các kiểu cơ bản hoặc struct khác (struct lồng nhau). Phân biệt rõ ràng toán tử . (với biến struct) và -> (với con trỏ tới struct) là rất quan trọng. Struct thường được dùng cùng con trỏ và cấp phát động để tạo cấu trúc dữ liệu linh hoạt như danh sách liên kết.

#### 2.8.6. Ví dụ Minh họa

- Ví dụ 1: Định nghĩa, khai báo, truy cập bằng toán tử . (có typedef)

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

// Định nghĩa cấu trúc và dùng typedef
typedef struct {
    char ten;
    int tuoi;
    float chieuCao;
} Nguoi;

int main() {
    Nguoi nguoi1; // Khai báo biến cấu trúc dùng typedef

    // Gán giá trị cho các thành viên dùng toán tử '.'
    strcpy(nguoi1.ten, "Nguyen Van B");
    nguoi1.tuoi = 30;
```

```

    nguoi1.chieuCao = 1.75f;

    // Truy cập và in các thành viên
    printf("Thông tin:\n");
    printf("Ten: %s\n", nguoi1.ten);
    printf("Tuoi: %d\n", nguoi1.tuoi);
    printf("Chieu cao: %.2f m\n", nguoi1.chieuCao);

    return 0;
}

```

## - Ví dụ 2: Mảng các struct

```

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

typedef struct {
    char tenSach;
    int namXB;
} Sach;

int main() {
    Sach thuVien; // Khai báo mảng chứa 3 cấu trúc Sach

    // Khởi tạo các phần tử trong mảng
    strcpy(thuVien.tenSach, "De Men Phieu Luu Ky");
    thuVien.namXB = 1941;

    strcpy(thuVien.tenSach, "So Do");
    thuVien.namXB = 1936;

    strcpy(thuVien.tenSach, "Tat Den");
    thuVien.namXB = 1939;

    printf("Thông tin sach trong thu vien:\n");
    for (int i = 0; i < 3; i++) {
        // Truy cập thành viên của struct trong mảng
        printf("- %s (%d)\n", thuVien[i].tenSach, thuVien[i].namXB);
    }

    return 0;
}

```

## 2.9. Danh sách Liên kết

Danh sách liên kết là một cấu trúc dữ liệu động, tuyến tính, gồm một chuỗi các nút (node), mỗi nút chứa dữ liệu và một con trỏ đến nút tiếp theo.<sup>8</sup>

### 2.9.1. Khái niệm (Danh sách Liên kết Đơn)

- Các nút không lưu trữ ở vị trí bộ nhớ liên tiếp nhau, khác với mảng.
- Mỗi nút (node) thường có hai thành phần:
  - Dữ liệu (Data): Giá trị của phần tử.

- Con trỏ next: Địa chỉ của nút kế tiếp.
- Danh sách được truy cập qua con trỏ head, trỏ đến nút đầu tiên. Nếu rỗng, head là NULL.
- Con trỏ next của nút cuối cùng là NULL, đánh dấu kết thúc danh sách.
- Ưu điểm: Kích thước động (dễ thêm/bớt nút), chèn/xóa hiệu quả (đặc biệt ở đầu/khi biết nút trước) vì chỉ cần thay đổi con trỏ.
- Nhược điểm: Truy cập tuần tự (phải duyệt từ head), không truy cập ngẫu nhiên theo chỉ số ( $O(N)$  để truy cập). Tốn bộ nhớ cho con trỏ next. Có thể gây cache miss nhiều hơn.

### 2.9.2. Cấu trúc Nút

Định nghĩa bằng struct, chứa trường dữ liệu và con trỏ tới chính kiểu cấu trúc đó (struct Node \*next). typedef giúp gọn hơn.

```
typedef struct Node
{
    int data;                // Ví dụ: dữ liệu là số nguyên
    struct Node *next;      // Con trỏ tới nút kế tiếp
} Node;
```

### 2.9.3. Triển khai bằng Con trỏ và Cấp phát Động

- Các nút được tạo và cấp phát động trên heap bằng malloc.
- Con trỏ head (kiểu Node\*) là điểm bắt đầu để quản lý danh sách, ban đầu khởi tạo là NULL.
- Mọi thao tác (chèn, xóa, duyệt) liên quan đến việc đọc và thay đổi con trỏ next và có thể cả head.

### 2.9.4. Các Thao tác Cơ bản (Danh sách Liên kết Đơn)

- Duyệt/Hiển thị (Traversal/Display): Bắt đầu từ head, dùng con trỏ tạm temp. Khi temp khác NULL, xử lý temp->data, rồi temp = temp->next.
- Chèn (Insertion):
  - Chèn vào đầu (Insert at Head):  $O(1)$ 
    - Tạo newNode (malloc), gán dữ liệu.
    - newNode->next = head.
    - head = newNode.

- Chèn vào cuối (Insert at End):  $O(N)$ 
  - Tạo newNode,  $\text{newNode} \rightarrow \text{next} = \text{NULL}$ .
  - Nếu  $\text{head} == \text{NULL}$ ,  $\text{head} = \text{newNode}$ .
  - Nếu không, duyệt đến nút cuối (có  $\text{next} == \text{NULL}$ ).
  - Đặt next của nút cuối trở về newNode.
- Chèn vào vị trí k (Insert at Position):  $O(N)$ 
  - Tạo newNode.
  - Nếu  $k=0$ , chèn vào đầu.
  - Nếu  $k>0$ , duyệt đến nút tại  $k-1$  (prevNode).
  - Nếu prevNode tồn tại,  $\text{newNode} \rightarrow \text{next} = \text{prevNode} \rightarrow \text{next}$ .
  - $\text{prevNode} \rightarrow \text{next} = \text{newNode}$ . Xử lý k không hợp lệ.
- Xóa (Deletion):
  - Xóa khỏi đầu (Delete from Head):  $O(1)$ 
    - Nếu rỗng, không làm gì.
    - $\text{temp} = \text{head}$ .
    - $\text{head} = \text{head} \rightarrow \text{next}$ .
    - $\text{free}(\text{temp})$ .
  - Xóa khỏi cuối (Delete from End):  $O(N)$ 
    - Nếu rỗng, không làm gì.
    - Nếu chỉ có một nút,  $\text{free}(\text{head})$ ,  $\text{head} = \text{NULL}$ .
    - Nếu nhiều nút, duyệt đến nút áp cuối (prevNode, có  $\text{next} \rightarrow \text{next} == \text{NULL}$ ).
    - $\text{temp} = \text{prevNode} \rightarrow \text{next}$  (nút cuối).
    - $\text{prevNode} \rightarrow \text{next} = \text{NULL}$ .
    - $\text{free}(\text{temp})$ .
  - Xóa tại vị trí k (Delete at Position):  $O(N)$ 
    - Nếu  $k=0$ , xóa ở đầu.
    - Nếu  $k>0$ , duyệt đến nút tại  $k-1$  (prevNode).
    - Nếu prevNode hoặc  $\text{prevNode} \rightarrow \text{next}$  không tồn tại, báo lỗi.
    - $\text{temp} = \text{prevNode} \rightarrow \text{next}$  (nút cần xóa).
    - $\text{prevNode} \rightarrow \text{next} = \text{temp} \rightarrow \text{next}$ .
    - $\text{free}(\text{temp})$ .

- Tìm kiếm (Search):  $O(N)$ . Duyệt từ head, so sánh dữ liệu từng nút. Trả về con trỏ/vị trí nút tìm thấy, hoặc NULL/thông báo nếu không thấy.

Danh sách liên kết là ví dụ của cấu trúc dữ liệu động, bộ nhớ không liên kề. Ưu điểm về linh hoạt kích thước và chèn/xóa (khi biết vị trí) đến từ việc dùng con trỏ. Nhược điểm là truy cập tuần tự, làm tìm kiếm/truy cập phần tử thứ  $N$  chậm hơn mảng. Triển khai đòi hỏi cẩn thận quản lý con trỏ (head, next, kiểm tra NULL) và cấp phát/giải phóng bộ nhớ (malloc/free). Sai sót có thể làm mất liên kết, mất dữ liệu, tạo vòng lặp, rò rỉ bộ nhớ hoặc lỗi con trỏ treo.

#### 2.9.5. Ví dụ Minh họa

- Ví dụ: Tạo Node, Chèn vào đầu, Hiển thị

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

// Định nghĩa cấu trúc Node (như trên)
typedef struct Node {
    int data;
    struct Node *next;
} Node;

// Hàm tạo nút mới
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Loi cap phat bo nho cho node moi!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Hàm chèn vào đầu danh sách
void insertAtFirst(Node** headRef, int data) {
    Node* newNode = createNode(data);
    newNode->next = *headRef;           // Nút mới trỏ đến head cũ
    *headRef = newNode;                // Cập nhật head mới
}

// Hàm hiển thị danh sách
void displayList(Node* head) {
    Node* current = head;
    if (current == NULL) {
        printf("Danh sach rong.\n");
        return;
    }
    printf("Danh sach: ");
    while (current != NULL) {
        printf("%d -> ", current->data);
    }
}
```

```
    current = current->next;
}
printf("NULL\n");
}

int main() {
    Node* head = NULL;           // Khởi tạo danh sách rỗng

    insertAtFirst(&head, 30);
    insertAtFirst(&head, 20);
    insertAtFirst(&head, 10);

    displayList(head);           // Output: Danh sach: 10 -> 20 -> 30 -> NULL

    // Cần thêm hàm giải phóng toàn bộ danh sách ở cuối chương trình
    // freeList(&head);

    return 0;
}
```

## CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ

### 3.1. Đặc tả bài toán

Trong thời đại công nghệ số, việc ứng dụng công nghệ thông tin vào quản lý rạp chiếu phim mang lại nhiều lợi ích. Nhu cầu về một chương trình quản lý rạp chiếu phim hiệu quả là rất lớn. Chương trình này cần hỗ trợ các hoạt động quản lý một cách dễ dàng và tiện lợi, thông qua máy tính. Dữ liệu cần được quản lý tập trung để đảm bảo tính nhất quán. Chương trình cần quản lý thông tin về phim, suất chiếu và vé. Mỗi phim có các thông tin: mã phim, tên phim, thể loại, ngày chiếu, giờ chiếu, phòng chiếu và giá vé. Mỗi vé bao gồm: mã vé, mã phim, tên người dùng, ghế ngồi và trạng thái vé.

### 3.2. Yêu cầu hệ thống

Chương trình quản lý rạp chiếu phim cần thực hiện các chức năng sau: thêm phim, xem danh sách phim, sửa thông tin phim, đặt vé, hủy vé, xem sơ đồ ghế, thống kê doanh thu và in danh sách vé. Chương trình được phát triển bằng ngôn ngữ C và sử dụng các kiến thức lập trình C cốt lõi để xây dựng.

### 3.3. Phân tích và thiết kế chương trình

- Để xây dựng chương trình quản lý rạp chiếu phim, cần thực hiện các bước phân tích và thiết kế sau, đồng thời làm rõ việc áp dụng các kiến thức lập trình C:
- Phân tích yêu cầu:
  - Xác định rõ các chức năng mà chương trình cần cung cấp (quản lý phim, quản lý vé, thống kê,...).
  - Xác định các đối tượng cần quản lý (Phim, Vé, Người dùng).
  - Xác định các thuộc tính của mỗi đối tượng (ví dụ: Phim có mã phim, tên phim,...).
  - Xác định các tương tác giữa các đối tượng.
- Thiết kế chương trình:
  - Thiết kế cấu trúc dữ liệu:
  - Kiểu cấu trúc (struct): Đây là nền tảng để tổ chức dữ liệu. Ví dụ, PhimNode không chỉ là một tập hợp các biến; nó đại diện cho một "bộ phim" với các thuộc tính như MaPhim, TenPhim, v.v. Tương tự cho VeNode và NguoDungNode. Việc sử dụng struct giúp mã nguồn dễ đọc và dễ quản lý hơn bằng cách đóng gói dữ liệu liên quan. Truy cập thành



viên của một cấu trúc được thực hiện bằng toán tử . (ví dụ: *phim.GiaVe*) nếu bạn có biến cấu trúc, hoặc toán tử -> (ví dụ: *phim\_ptr->GiaVe*) nếu bạn có một con trỏ tới cấu trúc.

- Danh sách liên kết đơn: Được chọn vì tính linh hoạt của nó trong việc quản lý một số lượng động các đối tượng. Không giống như mảng có kích thước cố định, danh sách liên kết có thể phát triển hoặc thu nhỏ khi cần thiết trong thời gian chạy. Mỗi phần tử (node) là một kiểu cấu trúc và chứa một con trỏ (next) trỏ đến node tiếp theo, tạo thành một chuỗi. Node cuối cùng trong danh sách sẽ có con trỏ next là NULL.
- Thiết kế các hàm và module:
- Hàm (Function): Chương trình được chia thành các module chức năng, mỗi module bao gồm nhiều hàm. Ví dụ, hàm *themPhim()* đóng gói toàn bộ logic để thêm một bộ phim mới. Điều này thúc đẩy khả năng tái sử dụng mã (ví dụ: *hàm inDanhSachPhim()* có thể được gọi từ cả menu admin và menu người dùng) và giúp việc gỡ lỗi dễ dàng hơn (lỗi có thể được cô lập trong một hàm cụ thể).
  - Tham số hàm: Các hàm nhận tham số để hoạt động trên dữ liệu cụ thể. Ví dụ, *datVe(char \*user)* nhận một con trỏ tới chuỗi *user* để biết ai đang đặt vé.
  - Giá trị trả về: Hàm có thể trả về giá trị để thông báo kết quả hoạt động của chúng, ví dụ *dangNhap()* trả về *UserType (int)* để cho biết loại người dùng hoặc đăng nhập thất bại.
- Con trỏ (Pointer): Là công cụ không thể thiếu trong C, đặc biệt quan trọng trong chương trình này:
- Truyền tham chiếu: Để một hàm có thể sửa đổi biến bên ngoài phạm vi của nó, chúng ta truyền địa chỉ của biến đó (sử dụng toán tử &). Bên trong hàm, biến này được truy cập thông qua một con trỏ (sử dụng toán tử \* để giải tham chiếu). Ví dụ rõ ràng nhất là các hàm như *docPhimTuFile(PhimNode \*\*head)*. Ở đây, *head* là con trỏ đầu của danh sách liên kết. Để hàm có thể thay đổi *head* (ví dụ, khi danh sách ban đầu rỗng hoặc khi thêm node vào đầu), chúng ta cần truyền địa chỉ của con trỏ *head* (tức là một con trỏ tới con trỏ).

- Làm việc với chuỗi: Trong C, chuỗi ký tự là mảng các ký tự kết thúc bằng ký tự null \0. Con trỏ kiểu char\* thường được sử dụng để làm việc với chuỗi, ví dụ như truyền tên người dùng, tên phim cho các hàm.
- Duyệt danh sách liên kết: Con trỏ được sử dụng để di chuyển từ node này sang node khác trong danh sách liên kết (ví dụ: temp = temp->next;).
- Thiết kế giao diện người dùng: Giao diện dòng lệnh được xây dựng bằng cách sử dụng các hàm printf() để hiển thị menu và thông báo, và scanf() để nhận đầu vào từ người dùng.
- Thiết kế luồng xử lý: Hàm main() thường điều phối luồng chính, gọi các hàm menu và hàm chức năng dựa trên lựa chọn của người dùng.
- Mô tả chi tiết các chức năng chính: (Như đã mô tả ở phiên bản trước, tập trung vào việc các hàm cụ thể sử dụng các khái niệm C như thế nào)
- Cấp phát động:
  - Khi một thực thể mới cần được tạo (ví dụ: một bộ phim mới được thêm, một vé mới được đặt, một người dùng mới đăng ký), bộ nhớ cho node tương ứng trong danh sách liên kết được cấp phát từ vùng nhớ heap bằng cách sử dụng malloc(). Ví dụ: *PhimNode \*newPhim = (PhimNode \*)malloc(sizeof(PhimNode));*. Lệnh này yêu cầu một khối bộ nhớ đủ lớn để chứa một PhimNode và trả về một con trỏ tới khối bộ nhớ đó.
  - Điều quan trọng là phải kiểm tra xem malloc() có trả về NULL hay không (nghĩa là cấp phát thất bại).
  - Khi một node không còn cần thiết nữa (ví dụ: khi xóa phim), bộ nhớ đã cấp phát cho nó phải được giải phóng bằng free() (ví dụ: free(phimCanXoa);) để tránh rò rỉ bộ nhớ (memory leaks).
- Xử lý tệp:
  - Dữ liệu của chương trình (người dùng, phim, vé) được lưu trữ bền vững trong các tệp văn bản (nguoidung.txt, phim.txt, ve.txt).
  - Con trỏ tệp (FILE \*): Một con trỏ kiểu FILE (ví dụ: FILE \*f\_phim;) được sử dụng để tham chiếu đến một tệp.
  - fopen(): Hàm này được sử dụng để mở một tệp. Nó nhận tên tệp và chế độ mở (ví dụ: "r" để đọc, "w" để ghi – ghi đè nếu tệp tồn tại, "a" để ghi nối

vào cuối tệp). Nó trả về một con trỏ FILE\* hoặc NULL nếu mở tệp thất bại.

- `fscanf()` và `fprintf()`: Được sử dụng để đọc dữ liệu có định dạng từ tệp và ghi dữ liệu có định dạng vào tệp. Ví dụ, khi đọc thông tin phim: `fscanf(f_phim, "%[^/]|%[^/]|...", temp->MaPhim, temp->TenPhim, ...);`.
- `fclose()`: Đóng một tệp đã mở, giải phóng tài nguyên liên quan đến tệp. Việc đóng tệp sau khi hoàn thành thao tác là rất quan trọng.
- Các hàm như `docPhimTuFile()` sẽ mở tệp phim ở chế độ đọc, đọc từng dòng, phân tích cú pháp dữ liệu, cấp phát động một `PhimNode` mới, điền dữ liệu vào node và thêm nó vào danh sách liên kết. Ngược lại, các hàm như `ghiDanhSachPhimVaoFile()` (nếu có, hoặc logic tương tự trong `suaPhim`, `xoaPhim`) sẽ mở tệp ở chế độ ghi và duyệt qua danh sách liên kết, ghi thông tin của mỗi node vào tệp.

### 3.4. Cấu trúc dữ liệu chương trình

- Chương trình sử dụng các kiểu cấu trúc (struct) sau để định nghĩa và lưu trữ thông tin, kết hợp với danh sách liên kết đơn và cấp phát động:
  - Cấu trúc *PhimNode*: Lưu trữ thông tin về một bộ phim.

```
typedef struct PhimNode {  
    char MaPhim[15];           // Mảng ký tự. Khi truyền vào hàm, tên mảng  
                               // hoạt động như một con trỏ tới phần tử đầu tiên.  
    char TenPhim[50];          // Tương tự, là một chuỗi ký tự.  
    char TheLoai[20];  
    char NgayChieu[11];  
    char GioChieu[6];  
    char PhongChieu[10];  
    int GiaVe;  
    struct PhimNode *next;     // Con trỏ tự trỏ, liên kết đến PhimNode tiếp theo.  
} PhimNode;
```

- Mảng và Con trỏ mảng: Các trường như *MaPhim*, *TenPhim* là các mảng ký tự. Trong C, tên của một mảng, khi được sử dụng trong hầu hết các ngữ cảnh (ví dụ, khi truyền cho một hàm), sẽ phân rã thành một con trỏ trỏ đến phần tử đầu tiên của mảng. Vì vậy, *MaPhim* có thể được coi là *char\** trong nhiều trường hợp.

- Trường *next* là một con trỏ kiểu *struct PhimNode \**, là cốt lõi của việc xây dựng danh sách liên kết.
- Cấu trúc *VeNode*: Lưu trữ thông tin về một vé.

```
typedef struct VeNode {
    char MaVe[15];
    char MaPhim[15];
    char TenNguoiDung[15];
    int GheNgoi[40];           // Mảng số nguyên. Tên mảng 'GheNgoi' khi
                                // truyền cho hàm sẽ là một con trỏ tới int (int*).
    int TrangThai;             // 0: chưa đặt, 1: đã đặt, 2: đã hủy
    struct VeNode *next;       // Con trỏ tới node VeNode tiếp theo.
} VeNode;
```

- Trường *GheNgoi* là một mảng 40 số nguyên. Khi bạn truyền *GheNgoi* cho một hàm, bạn thực sự đang truyền địa chỉ của phần tử đầu tiên của mảng, do đó hàm nhận nó như một con trỏ mảng (cụ thể là *int\**). Các thao tác trên mảng bên trong hàm sẽ ảnh hưởng đến mảng gốc.
- Trường *next* là một con trỏ kiểu *struct VeNode \**.
- Cấu trúc *NguoiDungNode*: Lưu trữ thông tin về một người dùng.

```
typedef struct NguoiDungNode {
    char username[15];         // Mảng ký tự cho tên người dùng.
    char password[15];         // Mảng ký tự cho mật khẩu.
    int type;                   // 1: admin, 2: user
    struct NguoiDungNode *next; // Con trỏ tới node
                                // NguoiDungNode tiếp theo.
} NguoiDungNode;
```

- Trường *next* là một con trỏ kiểu *struct NguoiDungNode \**.

### 3.5. Các chức năng của chương trình

Chương trình quản lý rạp chiếu phim được xây dựng với mục tiêu cung cấp một giải pháp toàn diện cho việc vận hành và quản lý các hoạt động của rạp. Các chức năng được thiết kế để phục vụ hai đối tượng người dùng chính: Quản trị viên (Admin) và

Khách hàng (User), với các quyền hạn và tác vụ riêng biệt. Việc triển khai các chức năng này đều dựa trên các kiến thức C đã đề cập.

### 3.5.1. Chức năng Tổng quan và Luồng hoạt động chính

Chương trình bắt đầu với một menu đăng nhập hoặc đăng ký.

- Hiện thị menu đăng nhập (*menudangNhap()*): Hàm này chịu trách nhiệm hiển thị giao diện cho người dùng.

```
void menudangNhap(void) {
    xoaMH(); // Gọi hàm xóa màn hình
    printf(BOLD BLUE"+-----+\n"RESET_COLOR);
    printf(BOLD BLUE|"RESET_COLOR);
    printf(BOLD BLUE"====" RED" HE THONG DAT VE XEM PHIM"
    BOLD BLUE" =====RESET_COLOR);
    printf(BOLD BLUE"\n");
    printf("+-----+\n"RESET_COLOR);
    printf(BOLD BLUE|"RESET_COLOR,RED"      Chao mung! "
    RESET_COLOR"\n");
    printf(BOLD YELLOW" 1. Dang nhap          "RESET_COLOR);
    printf(BOLD BLUE"\n");
    printf(BOLD BLUE"|");
    printf(BOLD YELLOW" 2. Dang ky tai khoan moi    "RESET_COLOR);
    printf(BOLD BLUE"\n");
    printf("|");
    printf(BOLD YELLOW" 3. Thoat              "RESET_COLOR);
    printf(BOLD BLUE"\n");
    printf(BOLD BLUE"+-----+\n"RESET_COLOR);
    printf(BOLD GREEN);
    printf("Nhap lua chon cua ban: ");
    printf(RESET_COLOR);
}
```

- Xác thực người dùng và phân loại vai trò (Admin hoặc User) dựa trên thông tin đăng nhập hoặc đăng ký thành công (*dangNhap()*, *dangKy()*). Các hàm này sử dụng con trỏ (cho chuỗi user) và xử lý tệp để đọc thông tin tài khoản.
- Sau khi đăng nhập thành công, chương trình chuyển hướng người dùng đến menu chức năng tương ứng với vai trò của họ (*chonMenu(UserType, user)*). Hàm

*chonMenu* nhận *UserType* (kiểu int) và *user* (một con trỏ kiểu char \*) làm tham số.

### 3.5.2. Chức năng dành cho Khách hàng (User)

Sau khi đăng nhập thành công với vai trò là khách hàng, người dùng có thể truy cập các chức năng sau thông qua hàm *menuKhachHang(user)* (nhận con trỏ *user*):

- Xem danh sách phim (*inDanhSachPhim()*):
  - Hàm này hiển thị danh sách các bộ phim.
  - Thông tin chi tiết của mỗi phim được đọc từ danh sách liên kết các *PhimNode*.
  - Chức năng này sử dụng hàm *docPhimTuFile()* để đọc thông tin từ tệp *phim.txt* (ví dụ về xử lý tệp) và nạp vào danh sách liên kết được quản lý bằng con trỏ và cập phát động.
- Đặt vé xem phim (*datVe(user)*):
  - Hàm này cho phép người dùng đặt vé. Tham số *user* là một con trỏ kiểu *char \**.
  - Hiển thị danh sách phim để người dùng lựa chọn.
  - Sau khi chọn phim, hiển thị sơ đồ ghế ngồi (*inSoDoGhe()*). Hàm *inSoDoGhe* nhận vào con trỏ đầu của danh sách liên kết vé (*VeNode \*headVe*) và con trỏ tới mã phim (*const char \*maPhim*). Mảng *GheNgoi* trong *VeNode* được sử dụng để xác định trạng thái ghế.
  - Người dùng chọn ghế. Hệ thống kiểm tra tính hợp lệ.
  - Nếu ghế hợp lệ, hệ thống tạo mã vé (*taoMaVe()*), cập phát động một *VeNode* mới, lưu thông tin vé vào node này và thêm vào đầu danh sách liên kết vé. Sau đó, ghi lại toàn bộ danh sách vé vào tệp *ve.txt* (*ghiDanhSachVeVaoFile()*) – một ví dụ về xử lý tệp.
  - Thông báo đặt vé thành công.
- Xem vé đã đặt (*inVeDaDat(user)*):
  - Hàm này hiển thị danh sách vé người dùng đã đặt.
  - Đọc dữ liệu từ tệp *ve.txt* vào danh sách liên kết *VeNode* bằng hàm *docVeTuFile()*.
  - Duyệt danh sách liên kết bằng con trỏ để tìm và hiển thị các vé của người dùng.

- Hủy vé đã đặt (*huyVeDaDat(user)*):
  - Hàm này cho phép hủy vé.
  - Hiện thị danh sách vé đã đặt của người dùng.
  - Người dùng chọn vé muốn hủy.
  - Hệ thống cập nhật trạng thái của *VeNode* tương ứng trong danh sách liên kết và ghi lại vào tệp *ve.txt*. Trạng thái ghế trong mảng *GheNgoi* cũng được cập nhật.
- Đăng xuất:
  - Cho phép người dùng thoát và quay lại menu đăng nhập.

```
void menuKhachHang(char *user) { // user là một con trỏ kiểu char*
    xoaMH();
    printf(BOLD BLUE"+-----+\n");
    printf("|");
    printf(BOLD RED);
    printf("    MENU KHACH HANG    ");
    printf(RESET_COLOR);
    printf(BOLD BLUE"\n");
    printf(BOLD BLUE"+-----+\n");
    printf("|" BOLD GREEN " Chao mung: %-21s " RESET_COLOR, user);
    printf(BOLD BLUE"\n"); // Sử dụng con trỏ user
    printf("+-----+\n");
    printf("|"); printf(BOLD YELLOW " 1. Xem danh sach phim    ");
    printf(BOLD BLUE "\n");
    printf("|"); printf(BOLD YELLOW " 2. Dat ve xem phim    ");
    printf(BOLD BLUE "\n");
    printf("|"); printf(BOLD YELLOW " 3. Xem ve da dat    ");
    printf(BOLD BLUE "\n");
    printf("|"); printf(BOLD YELLOW " 4. Huy ve da dat    ");
    printf(BOLD BLUE "\n");
    printf("|"); printf(BOLD YELLOW " 5. Dang xuat    ");
    printf(BOLD BLUE "\n");
    printf("+-----+\n"RESET_COLOR);
    printf(YELLOW);
    printf("Nhap lua chon cua ban: ");
    printf(RESET_COLOR);
}
```

### 3.5.3. Chức năng dành cho Quản trị viên (Admin)

Sau khi đăng nhập với vai trò quản trị viên, người dùng có quyền truy cập vào các chức năng quản lý cấp cao thông qua hàm *menuQuanTriVien(user)*:

- Quản lý Phim (*menuQuanLyPhim(user)*):
  - Thêm phim mới (*themPhim()*):
    - Admin nhập thông tin phim.
    - Cấp phát động một *PhimNode* mới.
    - Kiểm tra tính hợp lệ và trùng lặp (duyệt danh sách liên kết hiện có bằng con trỏ).
    - Lưu thông tin phim mới vào danh sách liên kết phim và cập nhật vào tệp *phim.txt* (sử dụng hàm *fprintf()* cho xử lý tệp).
  - Sửa thông tin phim (*suaPhim()*):
    - Admin chọn phim cần sửa.
    - Tìm *PhimNode* tương ứng trong danh sách liên kết bằng con trỏ.
    - Cập nhật thông tin và ghi lại vào tệp *phim.txt*.
  - Xóa phim (*xoaPhim()*):
    - Admin chọn phim cần xóa.
    - Tìm và xóa *PhimNode* khỏi danh sách liên kết (cần xử lý các con trỏ next một cách cẩn thận và sử dụng *free()* để giải phóng bộ nhớ động của node bị xóa).
    - Cập nhật tệp *phim.txt*.
  - Hiện thị danh sách phim (*inDanhSachPhim()*): Tương tự như của User.
- Quản lý Dữ liệu Vé và Thống kê (*menuQuanLyVeVaThongKe(user)*):
  - Xem danh sách tất cả vé đã đặt (*inDanhSachVe()*):
    - Đọc dữ liệu từ *ve.txt* vào danh sách liên kết *VeNode*.
    - Hiện thị thông tin từ danh sách liên kết.
  - Thống kê doanh thu (*thongKeDoanhThu()*):
    - Đọc dữ liệu phim và vé vào các danh sách liên kết.
    - Duyệt qua danh sách liên kết vé, tham chiếu đến thông tin phim (qua *MaPhim*) để tính tổng doanh thu.
- Quản lý Tài khoản Người dùng (*menuQuanLyTaiKhoanNguoiDung(user)*):
  - Xem danh sách người dùng (*inDanhSachNguoiDung()*):



- Đọc `nguoidung.txt` vào danh sách liên kết `NguoiDungNode`.
- Hiển thị thông tin (không hiển thị mật khẩu).
- Xóa tài khoản người dùng (`xoaTaiKhoanNguoiDung(adminUser)`):
  - Admin chọn tài khoản cần xóa.
  - Tìm và xóa `NguoiDungNode` khỏi danh sách liên kết (sử dụng `free()` để giải phóng bộ nhớ động).
  - Cập nhật tệp `nguoidung.txt`.
- Đăng xuất:
  - Cho phép Admin thoát.

```
void menuQuanTriVien(char *user) { // user là một con trỏ kiểu char*
    xoaMH();
    printf(BLUE"+-----+\n");
    printf("|");
    printf(BOLD YELLOW);
    printf(RED"    MENU QUAN TRI VIEN    "RESET_COLOR);
    printf(BLUE"\n");
    printf("+-----+\n");
    printf("|" BOLD GREEN " Admin: %-28s " RESET_COLOR , user);
    printf(BLUE "\n");
    printf("+-----+\n");
    printf("|"); printf(BOLD YELLOW " 1. Quan ly Phim          ");
    printf(BLUE"\n");
    printf("|"); printf(BOLD YELLOW " 2. Quan ly Du lieu Ve va Thong ke  ");
    printf( BLUE"\n");
    printf("|"); printf(BOLD YELLOW " 3. Quan ly Tai khoan Nguoi dung  ");
    printf( BLUE"\n");
    printf("|"); printf( BOLD YELLOW" 4. Dang xuat          ");
    printf( BLUE"\n");
    printf("+-----+\n");
    printf(GREEN);
    printf("Nhap lua chon cua ban: ");
    printf(RESET_COLOR);
}
```

### 3.5.4. Các chức năng Hỗ trợ và Quản lý Dữ liệu nền

Các hàm này hoạt động ngầm để hỗ trợ các tác vụ chính và quản lý dữ liệu, thể hiện rõ việc áp dụng các kiến thức C:

- Quản lý Tài khoản:
  - *dangNhap(char \*user)*: Hàm xử lý đăng nhập, sử dụng con trỏ user và pass (được cấp phát động bên trong hàm), gọi hàm *docTaiKhoan()* để xử lý tệp.
  - *dangKy(char \*user)*: Hàm đăng ký, sử dụng con trỏ user, cấp phát động cho pass và *confirmPass*, làm việc với danh sách liên kết *NguoiDungNode* (đọc từ tệp, thêm node mới được cấp phát động) và xử lý tệp để ghi tài khoản mới.
  - *docTaiKhoan(char \*user, char \*pass)*: Hàm đọc tệp *nguoidung.txt* để tìm và xác thực tài khoản. Nhận vào các con trỏ user và pass.
  - *docTaiKhoanTuFile(NguoiDungNode \*\*head)*: Hàm đọc toàn bộ tài khoản từ *nguoidung.txt* và lưu vào danh sách liên kết *NguoiDungNode*. Tham số head là một con trỏ tới con trỏ (*NguoiDungNode \*\**), cho phép hàm thay đổi giá trị của con trỏ head ở nơi gọi. Mỗi node được cấp phát động.
  - *ghiDanhSachNguoiDungVaoFile(NguoiDungNode \*head)*: Hàm ghi thông tin từ danh sách liên kết người dùng (truyền vào qua con trỏ head) vào tệp *nguoidung.txt*.
- Quản lý Dữ liệu Phim:
  - *docPhimTuFile(PhimNode \*\*head)*: Tương tự *docTaiKhoanTuFile*, đọc từ *phim.txt* vào danh sách liên kết *PhimNode*, sử dụng con trỏ tới con trỏ và cấp phát động.
  - Các hàm thêm, sửa, xóa phim thao tác trên danh sách liên kết này (sử dụng con trỏ để duyệt, cấp phát động cho node mới, *free()* cho node bị xóa) và sau đó cập nhật lại vào tệp *phim.txt*.
- Quản lý Dữ liệu Vé:
  - *docVeTuFile(VeNode \*\*head)*: Tương tự, đọc từ *ve.txt* vào danh sách liên kết *VeNode*. Mảng *GheNgoi* trong mỗi *VeNode* được đọc từ chuỗi trong tệp.

- *ghiDanhSachVeVaoFile(VeNode \*headVe)*: Ghi danh sách liên kết vé vào tệp *ve.txt*.
  - *taoMaVe(char \*maVe)*: Hàm tạo mã vé, nhận vào một con trỏ *maVe* để ghi mã vé được tạo.
  - *inSoDoGhe(VeNode \*headVe, const char \*maPhim)*: Hàm hiển thị sơ đồ ghế, làm việc với danh sách liên kết vé và mảng *GheNgoi*.
- Tiện ích Giao diện:
- *xoaMH(void)*: Hàm xóa màn hình console.
  - *stop(int t, const char \*s)*: Hàm dừng chương trình, nhận *s* là một con trỏ hằng trỏ tới chuỗi.
- Chương trình sử dụng các tệp văn bản (*nguoidung.txt*, *phim.txt*, *ve.txt*) để lưu trữ dữ liệu một cách bền vững thông qua các hàm xử lý tệp. Dữ liệu trong các tệp này được đọc vào các danh sách liên kết (được tạo bởi các kiểu cấu trúc và quản lý bằng con trỏ, với các node được cấp phát động) khi chương trình cần truy xuất hoặc xử lý, và được ghi lại sau khi có sự thay đổi, đảm bảo tính nhất quán của dữ liệu. Mảng (như *GheNgoi* hoặc các chuỗi ký tự) cũng là một phần quan trọng của các cấu trúc dữ liệu, và việc xử lý chúng thường liên quan đến con trỏ mảng.

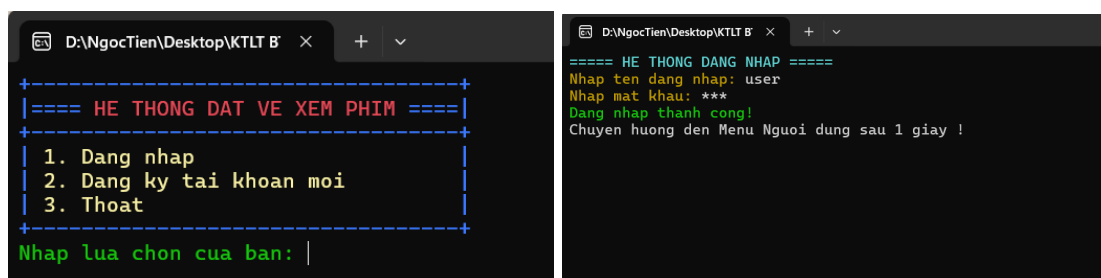
## CHƯƠNG 4: THỬ NGHIỆM ỨNG DỤNG

Chương này mô tả quá trình thử nghiệm các chức năng chính của ứng dụng quản lý rạp chiếu phim, thể hiện qua các giao diện dòng lệnh (CLI) mà người dùng tương tác.

### 4.1. Giao diện đăng nhập / đăng ký

Khi khởi chạy chương trình, người dùng được chào đón bằng menu chính cho phép lựa chọn:

Người dùng nhập lựa chọn và chương trình sẽ điều hướng đến màn hình tương ứng. Ví dụ, nếu chọn đăng nhập, chương trình sẽ yêu cầu nhập tên đăng nhập và mật khẩu. Mật khẩu khi nhập sẽ được ẩn bằng ký tự \*.

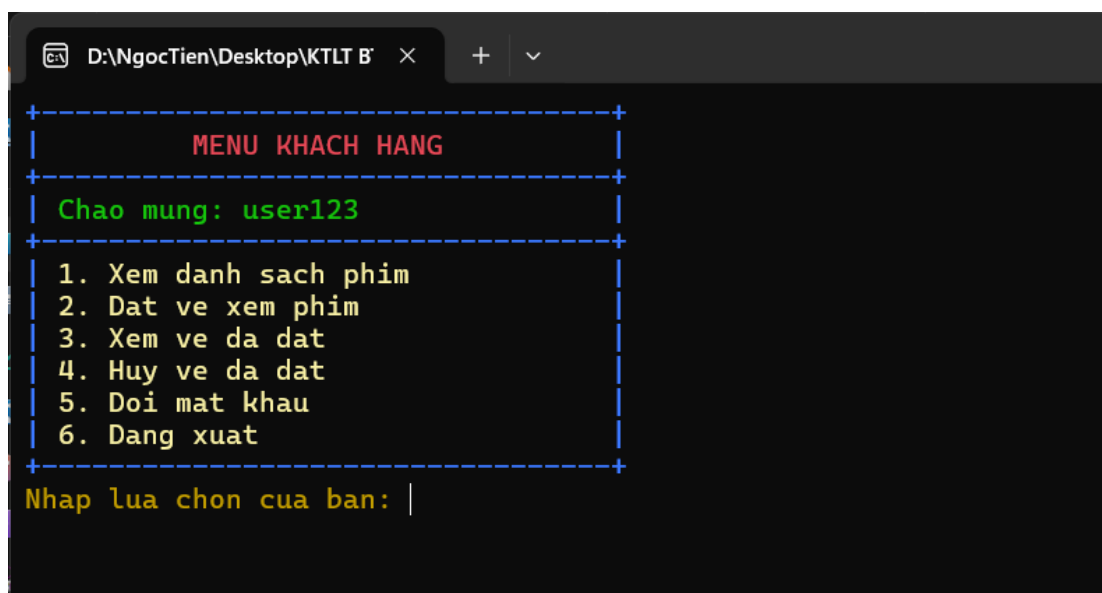


```
D:\NgocTien\Desktop\KTLT B' x + v
+-----+
|==== HE THONG DAT VE XEM PHIM ====|
+-----+
| 1. Đăng nhập                        |
| 2. Đăng ký tài khoản mới          |
| 3. Thoát                          |
+-----+
Nhập lựa chọn của bạn: |

D:\NgocTien\Desktop\KTLT B' x + v
===== HE THONG DANG NHAP =====
Nhập tên đăng nhập: user
Nhập mật khẩu: ***
Đăng nhập thành công!
Chuyển hướng đến Menu Người dùng sau 1 giây !
```

### 4.2. Giao diện Khách hàng (User) sau khi đăng nhập

Sau khi Khách hàng (User) đăng nhập thành công, màn hình sẽ hiển thị menu dành riêng cho User:



```
D:\NgocTien\Desktop\KTLT B' x + v
+-----+
|              MENU KHACH HANG              |
+-----+
| Chào mừng: user123                      |
+-----+
| 1. Xem danh sách phim                    |
| 2. Đặt vé xem phim                      |
| 3. Xem vé đã đặt                       |
| 4. Hủy vé đã đặt                       |
| 5. Đổi mật khẩu                        |
| 6. Đăng xuất                          |
+-----+
Nhập lựa chọn của bạn: |
```

Người dùng nhập số tương ứng với chức năng muốn thực hiện.

### 4.3. Giao diện Admin sau khi đăng nhập

Sau khi Quản trị viên (Admin) đăng nhập thành công, màn hình sẽ hiển thị menu

```

D:\NgocTien\Desktop\KTLT B' x + v
+-----+
|          MENU QUAN TRI VIEN          |
+-----+
| Admin: admin123                       |
+-----+
| 1. Quan ly Phim                       |
| 2. Quan ly Du lieu Ve va Thong ke    |
| 3. Quan ly Tai khoan Nguoi dung      |
| 4. Doi mat khau                      |
| 5. Dang xuat                         |
+-----+
| Nhap lua chon cua ban: |

```

#### 4.4. Chức năng Khách hàng: Xem danh sách phim

Khi User chọn "Xem danh sách phim", chương trình sẽ hiển thị một bảng liệt kê các phim hiện có trong hệ thống. Thông tin bao gồm: Mã phim, Tên phim, Thể loại, Ngày chiếu, Giờ chiếu, Phòng chiếu và Giá vé. Danh sách này được đọc từ tệp phim.txt.

```

D:\NgocTien\Desktop\KTLT B' x + v
DANH SACH PHIM
+-----+
| Ma Phim | Ten Phim | The Loai | Ngay | Gio | Phong | Gia ve |
+-----+
| BOLB    | Mada    | Tinh Cam | 15/08/2006 | 5:15 | 2E6    | 90000  |
| CNTT    | Tham tu | Trinh Tham | 06/09/2006 | 5:15 | 3E4    | 90000  |
| ASM     | Hello World | JBH    | 19/03/2006 | 19:23 | UTC2   | 50000  |
| PH005    | Phim E  | Phieu luu | 2024/07/21 | 19:00 | PB     | 95000  |
| PH004    | Phim D  | Vien tuong | 2024/07/22 | 14:00 | P1     | 85000  |
| PH002    | Phim B  | Kinh di   | 2024/07/21 | 20:00 | P2     | 90000  |
| PH001    | Phim A  | Hanh dong | 2024/07/20 | 15:30 | P1     | 80000  |
| PH001    | Phim A  | Hanh dong | 2024/07/20 | 10:00 | P1     | 80000  |
| AAA     | ASDF    | DFGH     | 19/02/2006 | 11:00 | 2E6    | 50000  |
+-----+
| Nhap 0 de quay lai: |

```

#### 4.5. Chức năng Khách hàng: Đặt vé

Khi User chọn "Đặt vé xem phim":

1. Chương trình hiển thị danh sách phim để User chọn bằng cách nhập Mã phim.
2. Sau khi chọn phim, sơ đồ ghế của phòng chiếu tương ứng được hiển thị, với ký hiệu O cho ghế trống và X cho ghế đã đặt.

```

PH001    Phim A    Hanh dong    2024/07/20 10:00    P1    80000
AAA      ASDF      DFGH        19/02/2006 11:00    2E6    50000
+-----+
| Nhap ma phim muon dat ve (0 de huy): PH005 |
+-----+
| So do ghe (X: da dat, 0: trong):          |
+-----+
| A | 0 X 0 0 0 0 0 0 |
| B | 0 0 0 0 0 0 0 0 |
| C | 0 0 0 0 0 0 0 0 |
| D | 0 0 0 0 0 0 0 0 |
| E | 0 0 0 0 0 0 0 0 |
+-----+
|      1 2 3 4 5 6 7 8 |
| Nguoi dat ve: user   |
| Nhap ghe ngoi (VD: A1, 0 de huy): |

```

3. User nhập vị trí ghế muốn đặt (ví dụ: A1, B5).

- Hệ thống kiểm tra tính hợp lệ. Nếu hợp lệ, vé được tạo, lưu vào ve.txt và thông báo đặt vé thành công cùng mã vé.

```

So do ghe (X: da dat, 0: trong):
-----
A | 0 X 0 0 0 0 0 0
B | 0 0 0 0 0 0 0 0
C | 0 0 0 0 0 0 0 0
D | 0 0 0 0 0 0 0 0
E | 0 0 0 0 0 0 0 0
-----
      1 2 3 4 5 6 7 8
Nguoi dat ve: user
Nhap ghe ngoi (VD: A1, 0 de huy): A5
Dat ve thanh cong!
Ma ve: VE9938
Ghe da dat: A5
Nhap 0 de quay lai: |

```

#### 4.6. Chức năng Khách hàng: Xem vé đã đặt

User có thể xem lại các vé mình đã đặt. Chương trình sẽ hiển thị danh sách các vé thuộc về User đó, bao gồm Mã vé, Mã phim, Tên phim, Ghế ngồi và Trạng thái vé ("Đã đặt" hoặc "Đã hủy").

D:\NgocTien\Desktop\KTLT B' x + v

DANH SACH VE DA DAT

Ma Ve	Ma Phim	Ten Nguoi Dat	Ghe Ngoi	Trang Thai
VE3237	ASM	user	A1	Da dat
VE3448	PH005	user	A2	Da dat
VE3412	PH005	user	XX	Da huy
VE3501	PH005	user	XX	Da huy
VE2845	BOLB	user	A3	Da dat
VE9938	PH005	user	A5	Da dat

Nhap 0 de quay lai: |

#### 4.7. Chức năng Khách hàng: Hủy vé

Khi User chọn "Hủy vé đã đặt":

- Chương trình hiển thị danh sách các vé "Đã đặt" của User.
- User nhập Mã vé muốn hủy.
- Hệ thống yêu cầu xác nhận.

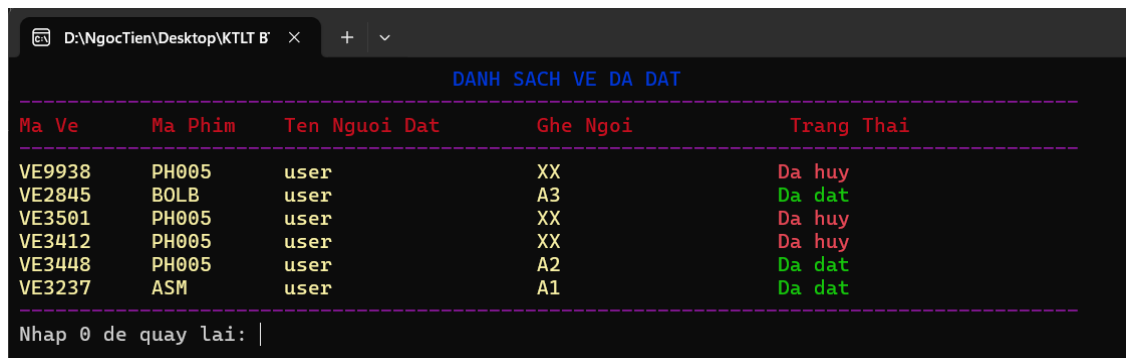
D:\NgocTien\Desktop\KTLT B' x + v

DANH SACH VE DA DAT

Ma Ve	Ma Phim	Ten Nguoi Dat	Ghe Ngoi	Trang Thai
VE3237	ASM	user	A1	Da dat
VE3448	PH005	user	A2	Da dat
VE3412	PH005	user	XX	Da huy
VE3501	PH005	user	XX	Da huy
VE2845	BOLB	user	A3	Da dat
VE9938	PH005	user	A5	Da dat

Nhap ma ve muon huy (0 de thoat): VE9938  
 Ban chac chan muon huy ve: A5 ?  
 Nhap 1 de xac nhan, 0 de huy thao tac: 1  
 Huy ve thanh cong!  
 Nhap 0 de quay lai: |

4. Nếu xác nhận, trạng thái vé được cập nhật thành "Đã hủy" trong ve.txt, và ghế ngồi tương ứng được đánh dấu lại là trống.



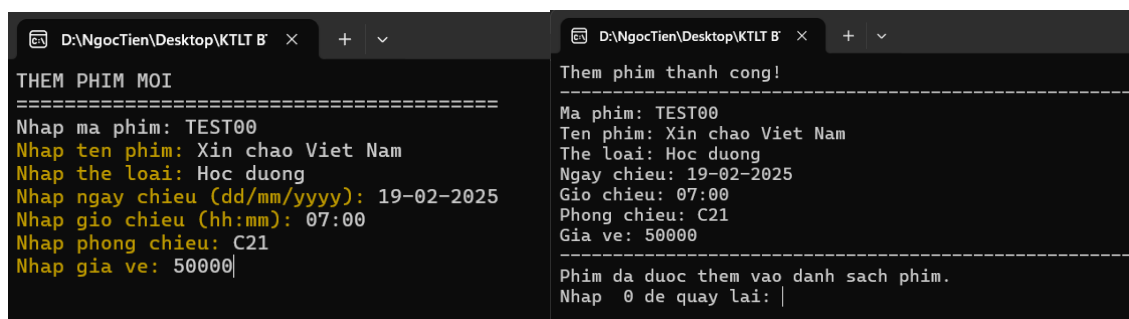
Ma Ve	Ma Phim	Ten Ngươi Dat	Ghế Ngồi	Trạng Thái
VE9938	PH005	user	XX	Đã hủy
VE2845	B0LB	user	A3	Đã đặt
VE3501	PH005	user	XX	Đã hủy
VE3412	PH005	user	XX	Đã hủy
VE3448	PH005	user	A2	Đã đặt
VE3237	ASM	user	A1	Đã đặt

Nhập 0 để quay lại: |

#### 4.8. Chức năng Admin: Quản lý phim

##### 4.8.1. Giao diện thêm phim

Admin chọn chức năng "Thêm phim mới" từ menu quản lý phim. Chương trình yêu cầu Admin nhập các thông tin: Mã phim, Tên phim, Thể loại, Ngày chiếu, Giờ chiếu, Phòng chiếu, Giá vé. Hệ thống kiểm tra Mã phim không trùng và lịch chiếu không trùng trong cùng một phòng. Nếu hợp lệ, phim mới được lưu vào phim.txt.



```
THEM PHIM MOI
=====
Nhap ma phim: TEST00
Nhap ten phim: Xin chao Viet Nam
Nhap the loai: Hoc duong
Nhap ngay chiếu (dd/mm/yyyy): 19-02-2025
Nhap giờ chiếu (hh:mm): 07:00
Nhap phong chiếu: C21
Nhap giá vé: 50000

Them phim thanh cong!
Ma phim: TEST00
Ten phim: Xin chao Viet Nam
The loai: Hoc duong
Ngay chiếu: 19-02-2025
Gio chiếu: 07:00
Phong chiếu: C21
Gia vé: 50000

Phim đã được thêm vào danh sách phim.
Nhap 0 để quay lại: |
```

##### 4.8.2. Giao diện sửa thông tin phim

Admin chọn "Sửa thông tin phim". Chương trình hiển thị danh sách phim, Admin chọn phim cần sửa bằng Mã phim.



Ma Phim	Ten Phim	The Loai	Ngay	Gio	Phong	Gia vé
TEST00	Xin Chao VN	Hoc duong	19-02-2025	07:00	C21	50000
TEST01	Xin Chao ITK65	Hoc Duong	19-02-2025	19:00	A34	50000
B0LB	Mada	Tinh Cam	15/08/2006	5:15	2E6	90000
CNTT	Tham tu kien	Trinh Tham	06/09/2006	5:15	3E4	90000
ASM	Hello World	JBH	19/03/2006	19:23	UTC2	50000
PH005	Phim E	Phieu lưu	2024/07/21	19:00	PB	95000
PH004	Phim D	Vien tuong	2024/07/22	14:00	P1	85000
PH002	Phim B	Kinh di	2024/07/21	20:00	P2	90000
PH001	Phim A	Hanh dong	2024/07/20	15:30	P1	80000
PH001	Phim A	Hanh dong	2024/07/20	10:00	P1	80000
AAA	ASDF	DFGH	19/02/2006	11:00	2E6	50000

Nhập mã phim muốn sửa: AAA|

Sau đó, Admin có thể chọn mục thông tin muốn sửa và nhập giá trị mới. Thông tin cập nhật được lưu lại vào phim.txt.

```

----- MENU -----
1. Sua ma phim
2. Sua ten phim
3. Sua the loai
4. Sua ngay chieu
5. Sua gio chieu
6. Sua phong chieu
7. Sua gia ve
0. Thoat sua quay lai quan li phim
-----
Chon: |

```

#### 4.8.3. Giao diện xóa phim

DANH SÁCH PHIM

Ma Phim	Ten Phim	The Loai	Ngay	Gio	Phong	Gia ve
AAA	ASDF	DFGH	19/02/2006	11:00	2E6	50000
PH001	Phim A	Hanh dong	2024/07/20	10:00	P1	80000
PH001	Phim A	Hanh dong	2024/07/20	15:30	P1	80000
PH002	Phim B	Kinh di	2024/07/21	20:00	P2	90000
PH004	Phim D	Vien tuong	2024/07/22	14:00	P1	85000
PH005	Phim E	Phieu luu	2024/07/21	19:00	PB	95000
ASM	Hello World	JBH	19/03/2006	19:23	UTC2	50000
CNTT	Tham tu kien	Trinh Tham	06/09/2006	5:15	3E4	90000
BOLB	Mada	Tinh Cam	15/08/2006	5:15	2E6	90000
TEST01	Xin Chao ITK65	Hoc Duong	19-02-2025	19:00	A34	50000
TEST00	Xin Chao VN	Hoc duong	19-02-2025	07:00	C21	50000

Nhap ma phim muon xoa (0 de thoat): TEST01

Admin chọn "Xóa phim". Sau khi Admin chọn Mã phim cần xóa và xác nhận, phim đó sẽ được loại bỏ khỏi phim.txt.

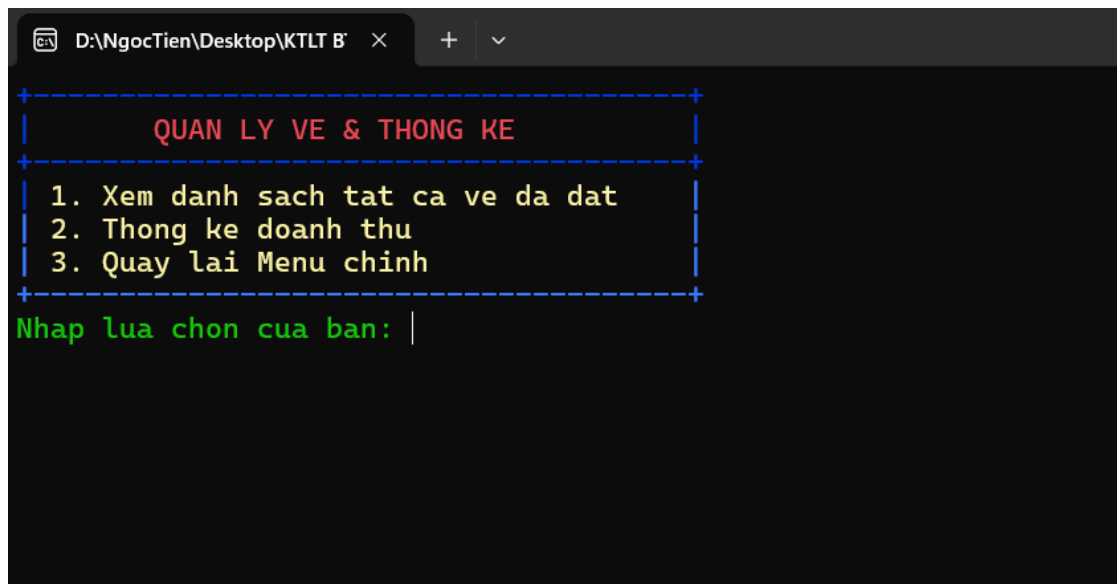
```

Nhap ma phim muon xoa (0 de thoat): TEST01
Ban co chac chan muon xoa phim sau khong?
-----
Ma phim: TEST01
Ten phim: Xin Chao ITK65
The loai: Hoc Duong
Ngay chieu: 19-02-2025
Gio chieu: 19:00
Phong chieu: A34
Gia ve: 50000
-----
Nhap 1 de xoa, 0 de huy: 1
Xoa phim thanh cong!
Nhap 0 de quay lai: |

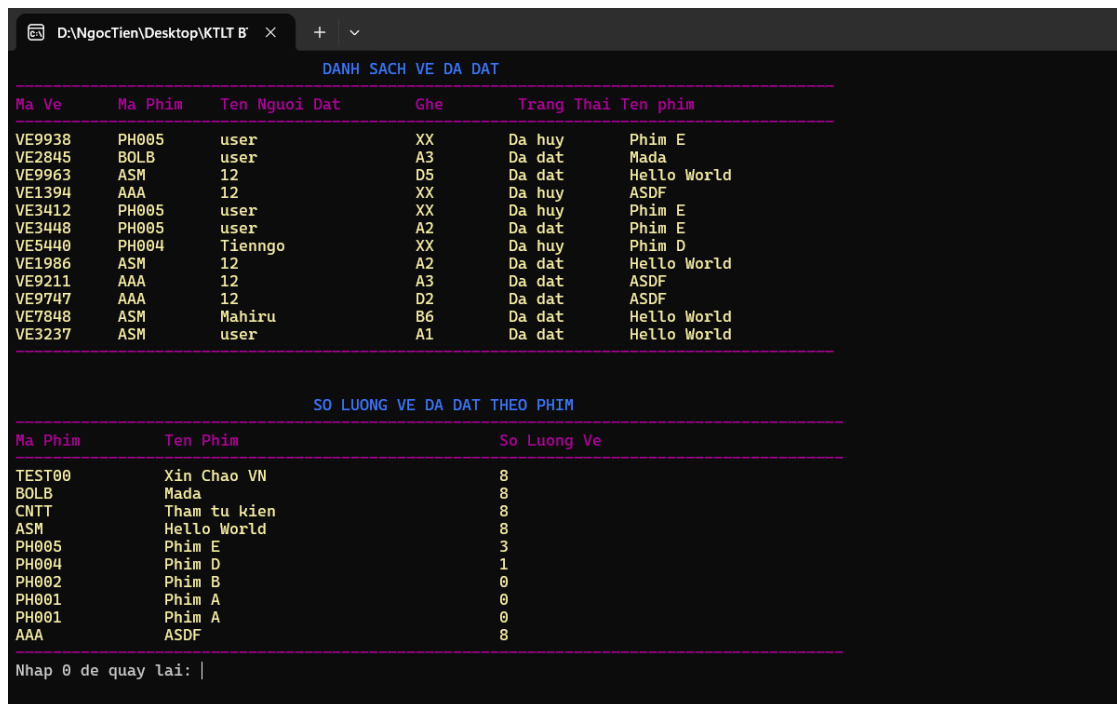
```



#### 4.9. Chức năng Admin: Quản lý vé và Thống kê



##### 4.9.1. Giao diện xem tất cả vé



##### 4.9.2. Giao diện thống kê doanh thu

Chức năng này cho phép Admin xem tổng doanh thu từ việc bán vé cho từng bộ phim, chỉ tính các vé có trạng thái "Đã đặt".

D:\NgocTien\Desktop\KTLT B' x + v

THONG KE DOANH THU

Ma Phim	Ten Phim	Doanh Thu
TEST00	Xin Chao VN	0
BOLB	Mada	90000
CNTT	Tham tu kien	0
ASM	Hello World	200000
PH005	Phim E	95000
PH004	Phim D	0
PH002	Phim B	0
PH001	Phim A	0
PH001	Phim A	0
AAA	ASDF	100000

Nhap 0 de quay lai: |

#### 4.10. Chức năng Admin: Quản lý tài khoản người dùng

D:\NgocTien\Desktop\KTLT B' x + v

QUAN LY TAI KHOAN NGUOI DUNG

1. Xem danh sach nguoi dung
2. Xoa tai khoan nguoi dung
3. Them tai khoan Admin
4. Quay lai Menu chinh

Nhap lua chon cua ban: |

##### 4.10.1. Giao diện xem danh sách người dùng

Admin có thể xem danh sách tất cả tài khoản người dùng trong hệ thống, bao gồm Tên đăng nhập và Loại tài khoản (Admin/User). Mật khẩu không được hiển thị.

D:\NgocTien\Desktop\KTLT B' x + v

DANH SACH NGUOI DUNG

Ten Dang Nhap	Loai Tai Khoan
Tienngo12	User
Tienngo	User
user	User
ngoctien	User
tester	Admin
admin123	Admin
Tien	User
tienngo	User

Nhap 0 de quay lai: |

##### 4.10.2. Giao diện xóa tài khoản người dùng

Admin chọn tài khoản cần xóa bằng Tên đăng nhập. Sau khi xác nhận, tài khoản sẽ bị xóa khỏi nguoidung.txt. Admin không thể xóa tài khoản admin đang sử dụng.

```
D:\NgocTien\Desktop\KTLT B' x + v
DANH SACH NGUOI DUNG
-----
Ten Dang Nhap      Loi Tai Khoan
-----
Tienngo12          User
Tienngo            User
user               User
ngoctien           User
tester             Admin
admin123           Admin
Tien               User
tienngo            User
-----

Nhap ten dang nhap cua nguoi dung muon xoa: Tien
Ban co chac chan muon xoa tai khoan cua nguoi dung 'Tien' khong?
Nhap 1 de xac nhan, 0 de huy: 1
Xoa tai khoan nguoi dung 'Tien' thanh cong!
Nhap 0 de quay lai: |
```

### 4.10.3. Giao diện thêm tài khoản Admin

Chỉ tài khoản Admin mới có thể thêm được tài khoản Admin mới.

Người dùng không thể thêm hoặc đăng kí loại tài khoản này.

```
D:\NgocTien\Desktop\KTLT B' x + v
-----
|          THEM TAI KHOAN ADMIN          |
-----

Nhap ten Admin: NewAdmin
Nhap mat khai: 123123
Nhap lai mat khai de xac nhan: 123123|
```

### 4.11. Giao diện đổi mật khẩu

Khi người dùng chọn đổi mật khẩu sẽ cho xác nhận lại mật khẩu cũ và nhập mật khẩu mới hai lần để xác nhận.

```
D:\NgocTien\Desktop\KTLT B' x + v
-----
|          DOI MAT KHAU:          |
|          Ten Dang Nhap: user          |
-----

Nhap mat khai cu: user123
Nhap mat khai moi: 12345
Nhap lai mat khai moi de xac nhan: 12345|
```

#### 4.12. Giao diện sau khi thoát/đăng xuất

Khi người dùng chọn "Đăng xuất" từ menu của mình, hoặc "Thoát" từ menu đăng nhập ban đầu, chương trình sẽ hiển thị thông báo cảm ơn và kết thúc phiên làm việc.



```
D:\NgocTien\Desktop\KTLT B' x + v
CAM ON BAN DA SU DUNG CHUONG TRINH!
-----
|                                |
|          NHOM THUC HIEN      |
|                                |
|          Nguyen Ngoc Tien - 6551071079      |
|          Ngo Thanh Tien - 6551071082      |
|                                |
|                                |
Chuyen huong den Exit sau 2 giay !|
```

## CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 5.1. Kết luận

#### 5.1.1. Kết quả đạt được

Chương trình quản lý rạp chiếu phim bằng ngôn ngữ C đã được phát triển thành công, đáp ứng các yêu cầu cơ bản đã đề ra. Các chức năng chính bao gồm quản lý người dùng (đăng nhập, đăng ký), quản lý phim (thêm, sửa, xóa, xem danh sách), quản lý vé (đặt vé, hủy vé, xem vé đã đặt, xem sơ đồ ghế), và các chức năng thống kê cho quản trị viên (xem danh sách vé, thống kê doanh thu, quản lý tài khoản người dùng) đều hoạt động ổn định.

Chương trình sử dụng hiệu quả các kiến thức lập trình C như cấu trúc, danh sách liên kết đơn, con trỏ, cấp phát bộ nhớ động và xử lý tệp để lưu trữ và thao tác dữ liệu. Giao diện người dùng dạng dòng lệnh (CLI) tuy đơn giản nhưng rõ ràng, dễ thao tác và đáp ứng được các nhu cầu quản lý cơ bản. Dữ liệu được lưu trữ bền vững trong các tệp văn bản, đảm bảo tính nhất quán.

#### 5.1.2. Hạn chế

- Bên cạnh những kết quả đạt được, chương trình vẫn còn một số hạn chế nhất định:
- Giao diện người dùng: Giao diện dòng lệnh (CLI) có thể không trực quan và thân thiện bằng giao diện đồ họa (GUI), đặc biệt với người dùng không quen với việc nhập lệnh.
- Xử lý lỗi và ràng buộc dữ liệu: Mặc dù có một số kiểm tra cơ bản (ví dụ: trùng mã phim, trùng lịch chiếu), việc xử lý lỗi đầu vào và các ràng buộc dữ liệu phức tạp hơn (ví dụ: định dạng ngày tháng, giờ chiếu, giá vé hợp lệ) có thể cần được cải thiện để tăng tính mạnh mẽ của chương trình.
- Tìm kiếm và lọc dữ liệu: Các chức năng tìm kiếm và lọc dữ liệu hiện tại còn hạn chế (ví dụ: tìm phim theo tên, lọc vé theo ngày).
- Bảo mật: Mật khẩu người dùng được lưu trữ dưới dạng văn bản thuần trong tệp, điều này không đảm bảo an toàn. Cần có cơ chế mã hóa mật khẩu.
- Khả năng mở rộng: Việc thêm các tính năng phức tạp mới có thể gặp khó khăn do cấu trúc hiện tại.

## 5.2. Hướng phát triển

- Để hoàn thiện và nâng cao chất lượng của chương trình quản lý rạp chiếu phim, có thể xem xét các hướng phát triển sau:
- Phát triển giao diện đồ họa (GUI): Sử dụng các thư viện đồ họa như GTK+, Qt (với C++) hoặc chuyển sang một ngôn ngữ khác có hỗ trợ GUI mạnh mẽ hơn để tạo trải nghiệm người dùng tốt hơn.
- Cải thiện xử lý lỗi và xác thực dữ liệu: Bổ sung các cơ chế kiểm tra đầu vào nghiêm ngặt hơn, xử lý các trường hợp ngoại lệ một cách linh hoạt.
- Nâng cao chức năng tìm kiếm và lọc: Cho phép người dùng tìm kiếm phim, vé theo nhiều tiêu chí khác nhau (tên, thể loại, ngày chiếu, trạng thái vé,...).
- Tăng cường bảo mật: Áp dụng các kỹ thuật mã hóa mật khẩu (ví dụ: hashing với salt) để bảo vệ thông tin người dùng.
- Quản lý lịch chiếu nâng cao: Cho phép quản lý nhiều suất chiếu cho cùng một phim, quản lý các phòng chiếu khác nhau một cách linh hoạt hơn.
- Thêm tính năng quản lý khuyến mãi, thành viên: Xây dựng hệ thống quản lý khách hàng thân thiết, các chương trình giảm giá, tích điểm.
- In ấn và báo cáo: Phát triển chức năng in vé, in báo cáo doanh thu theo các định dạng chuẩn.
- Tích hợp cơ sở dữ liệu: Thay vì sử dụng tệp văn bản, có thể chuyển sang sử dụng một hệ quản trị cơ sở dữ liệu (ví dụ: SQLite, MySQL) để quản lý dữ liệu hiệu quả và an toàn hơn, đặc biệt với khối lượng dữ liệu lớn.
- Phân quyền chi tiết hơn: Nếu cần, có thể phát triển hệ thống phân quyền chi tiết hơn cho các loại người dùng khác nhau trong hệ thống (ví dụ: nhân viên bán vé, quản lý rạp).

## TÀI LIỆU THAM KHẢO

[1] “C++ Functions - Tổng quan về Hàm trong C++ - Lập trình - Điện tử” ,

<https://www.laptrinhdientu.com/2023/09/cpp-function-overview.html>

[Truy cập vào 8/5/2025]

[2] “Hàm trong lập trình C - QuanTriMang.com”,

<https://quantrimang.com/hoc/hamtrong-lap-trinh-c-156118>

[Truy cập vào 8/5/2025]

[3] “Truyền mảng vào hàm trong C | Khóa học c - kungfu tech”,

<https://kungfutech.edu.vn/bai-viet/c/truyen-mang-vao-ham-trong-c>

[Truy cập vào 8/5/2025]

[4] “Con trỏ làm tham số cho hàm | Khóa học c - kungfu tech”,

<https://kungfutech.edu.vn/bai-viet/c/con-tro-lam-tham-so-cho-ham>

[Truy cập vào 8/5/2025]

[5] “Quản lý Bộ nhớ động Viblo”, [https://viblo.asia/p/quan-li-bo-nho-dong-](https://viblo.asia/p/quan-li-bo-nho-dong-EvbLbaobJnk)

[EvbLbaobJnk](https://viblo.asia/p/quan-li-bo-nho-dong-EvbLbaobJnk)

[Truy cập vào 8/5/2025]

[6] [Lập trình C] Xử lý file - file handling - Lam Dev Blog,

<https://dev.phamvanlam.com/lap-trinh-c-xu-ly-file/>

[Truy cập vào 8/5/2025]

[7] “Ví dụ về file nhị phân lưu struct Kỹ thuật lập trình v1.0”,

<https://ktlt.fita.edu.vn/cacbai-code-mau-tham-khao/vi-du-ve-file-nhi-phan-luu-struct>

[Truy cập vào 8/5/2025]

[8] “C Program to Implement Singly Linked List | GeeksforGeeks”,

<https://www.geeksforgeeks.org/c-program-to-implement-singly-linked-list/>

[Truy cập vào 8/5/2025]