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: KĨ THUẬT LẬP TRÌNH

ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG QUẢN LÝ RẠP 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 4 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: NHẬP MÔN CÔNG NGHỆ THÔNG TIN

ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG QUẢN LÝ RẠP 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 4 năm 2025

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. Hồ CHÍ MINH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

GIỚI THIỆU SƠ LƯỢC

BỘ MÔN: CÔNG NGHỆ THÔNG TIN

-------***-------

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.

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

MỤC LỤC

DANH MỤC CHỮ VIẾT TẮT

STT	Mô tả	Ý nghĩa	Ghi chú
1			
2			
3			
4			
5			
6			
7			

BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ

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

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.[5]
- 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.[1]
- 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 (Function Definition):
  - 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 (Function Call):
  - Thực thi mã lệnh bên trong thân hàm bằng cách sử dụng tên hàm và cung cấp các đối số thực tế (actual arguments) 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

2.3.2. Ví dụ Minh họa

- **Ví dụ 1: Duyệt mảng bằng con trỏ**

```
#include <stdio.h>

int main() {
    int mang = {10, 20, 30, 40, 50};
    int n = sizeof(mang) / sizeof(mang);
    int *ptr = mang;                    // ptr trỏ đến phần tử đầu tiên
(mang)
    int i;

    printf("Duyet mang bang con tro:\n");
    for (i = 0; i < n; i++) {
        printf("Gia tri: %d, Dia chi: %p\n", *ptr, ptr);
        ptr++; // Di chuyển con trỏ đến phần tử tiếp theo
    }

    // ptr bây giờ đang trỏ ra ngoài mảng (vị trí sau phần tử cuối)

    return 0;
}
```

- **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 thuoc cua tham so 'arr' (kich thuoc 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 thuoc 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.

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 liền kề 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_trỏ = tên_hàm; hoặc tên_con_trỏ = &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_trỏ)(danh_sách_đối_số);
// hoặc
tên_con_trỏ(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.

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 (<stdlib.h>)

```
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;
}
```

OK, đây là tóm tắt cho phần "2.7. Xử lý Tệp (File Handling)":

2.7. Xử lý Tệp

Xử lý tệp trong C là quá trình tương tác với các tệp trên thiết bị lưu trữ, bao gồm tạo, mở, đọc, ghi và đóng tệp.

2.7.1. Khái niệm và Con trỏ FILE

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

2.7.2. Phân loại Tệp

- **Tệp 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.
- **Tệp 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 Tệp

- **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 mới/ghi đè)	Xóa nội dung, ghi từ đầu	Tạo tệp mới	Binary
"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

- **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 Tệp Văn bản (dùng fprintf/fscanf)

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

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

    // Ghi vào tệp 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 tệp 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 tệp 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.

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_tro_cau_truc->ten_thanh_vien
```

Đây là cách viết tắt cho (*ten_con_tro_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("Thong 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 3: 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("Thong 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.

2.9.1. Khái niệm (Danh sách Liên kết Đơn)

- Các nút không lưu trữ ở vị trí bộ nhớ liền kề 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, newNode->next = NULL.
        - Nếu head == NULL, head = newNode.
        - Nếu không, duyệt đến nút cuối (có next == NULL).
        - Đặt next của nút cuối trỏ đến 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, newNode->next = prevNode->next.
        - prevNode->next = newNode. Xử lý k không hợp lệ.
    - **Xóa (Deletion):**
        - **Xóa khỏi đầu (Delete from Head):** O(1)
            - Nếu rỗng, không làm gì.
            - temp = head.
            - head = head->next.
            - free(temp).
        - **Xóa khỏi cuối (Delete from End):** O(N)
            - Nếu rỗng, không làm gì.
            - Nếu chỉ có một nút, free(head), head = NULL.
            - Nếu nhiều nút, duyệt đến nút áp cuối (prevNode, có next->next == NULL).
            - temp = prevNode->next (nút cuối).
            - prevNode->next = NULL.

        - free(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 prevNode->next không tồn tại, báo lỗi.
        - temp = prevNode->next (nút cần xóa).
        - prevNode->next = temp->next.
        - free(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 liền 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 cấu trúc dữ liệu danh sách liên kết đơn.

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:

- **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:** Chọn cấu trúc dữ liệu phù hợp để lưu trữ thông tin (trong trường hợp này là danh sách liên kết đơn).
    - **Thiết kế các hàm và module:** Chia chương trình thành các hàm nhỏ, mỗi hàm thực hiện một chức năng cụ thể. Nhóm các hàm có liên quan vào các module.

- **Thiết kế giao diện người dùng:** (Nếu có) Xác định cách người dùng sẽ tương tác với chương trình.
- **Thiết kế luồng xử lý:** Xác định trình tự các bước thực hiện của mỗi chức năng.

- **Mô tả chi tiết các chức năng chính:**
  - **Quản lý người dùng:**
    - Chức năng đăng nhập cho phép người dùng truy cập vào hệ thống.
    - Chức năng đăng ký cho phép người dùng tạo tài khoản mới.
    - Hệ thống quản lý thông tin người dùng, bao gồm tên đăng nhập và mật khẩu.
  - **Quản lý phim:**
    - Cho phép thêm phim mới vào danh sách.
    - Cho phép sửa đổi thông tin của phim.
    - Cho phép xóa phim khỏi danh sách.
    - Hiển thị danh sách phim hiện có.
  - **Quản lý vé:**
    - Cho phép người dùng đặt vé cho một suất chiếu.
    - Cho phép người dùng hủy vé đã đặt.
    - Hiển thị sơ đồ ghế ngồi để người dùng chọn ghế.
    - Hiển thị thông tin vé đã đặt của người dùng.
  - **Thống kê:**
    - Thống kê doanh thu bán vé.
- **Thiết kế chi tiết:**
  - **Cấu trúc dữ liệu:**
    - Sử dụng danh sách liên kết đơn để lưu trữ thông tin về phim, vé và người dùng.
    - Mỗi node trong danh sách liên kết tương ứng với một phim, một vé hoặc một người dùng.
  - **Các hàm chính:**
    - Hàm main(): Hàm chính của chương trình, điều khiển luồng thực thi.
    - Các hàm quản lý người dùng:

dangNhap(), dangKy(), docTaiKhoan().

- Các hàm quản lý phim:

  themPhim(), suaPhim(), xoaPhim(), inDanhSachPhim().

- Các hàm quản lý vé:

  datVe(), huyVe(), inSoDoGhe(), inVeDaDat().

- Các hàm hỗ trợ:

  xoaMH(), stop(), taoMaVe().

3.4. Cấu trúc dữ liệu chương trình

Chương trình sử dụng các cấu trúc dữ liệu sau để lưu trữ thông tin:

- **Cấu trúc** PhimNode**:** Lưu trữ thông tin về một bộ phim.

```
typedef struct PhimNode {
    char MaPhim[15];
    char TenPhim[50];
    char TheLoai[20];
    char NgayChieu[11];
    char GioChieu[6];
    char PhongChieu[10];
    int GiaVe;
    struct PhimNode *next;
} PhimNode;
```

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];
    int TrangThai; // 0: chưa đặt, 1: đã đặt, 2: đã hủy
    struct VeNode *next;
} VeNode;
```

Cấu trúc NguoiDungNode**:** Lưu trữ thông tin về một người dùng.

```
typedef struct NguoiDungNode {
    char username[15];
    char password[15];
    int type; // 1: admin, 2: user
    struct NguoiDungNode *next;
} NguoiDungNode;
```

TÀI LIỆU THAM KHẢO