TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

KỸ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn:	**ThS.TRẦN PHONG NHÃ**
Sinh viên thực hiện:	**ĐẶNG QUỐC QUY**
Mã sinh viên:	**6551071074**
Lớp:	**CQ.CNTT.65**
Khóa:	**65**

Tp.Hồ Chí Minh,năm 2025

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

KỸ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn:	**ThS.TRẦN PHONG NHÃ**
Sinh viên thực hiện:	**ĐẶNG QUỐC QUY**
Mã sinh viên:	**6551071074**
Lớp:	**CQ.CNTT.65**
Khóa:	**65**

Tp.Hồ Chí Minh,năm 2025

LỜI CẢM ƠN

Chào thầy và các bạn sinh viên thân mến!

Lời nói đầu tiên, em xin gửi tới Quý Thầy Cô Bộ môn Công nghệ Thông tin Trường Đại học Giao thông vận tải phân hiệu tại thành phố Hồ Chí Minh lời chúc sức khỏe và lòng biết ơn sâu sắc.

Em xin gửi lời cảm ơn chân thành nhất đến tất cả những người đã đóng góp và hỗ trợ cho bài báo cáo tiểu luận này. Sự giúp đỡ và động viên của cô và các bạn đã đóng vai trò vô cùng quan trọng trong quá trình chuẩn bị và hoàn thiện bài báo cáo này.

Em muốn bày tỏ lòng biết ơn sâu sắc đến các giảng viên và người hướng dẫn đã cung cấp cho em những kiến thức và hướng dẫn quý báu. Nhờ sự chỉ dạy tận tâm của thầy, em đã có cơ hội nắm vững kiến thức và kỹ năng cần thiết để thực hiện bài báo cáo này.

Cuối cùng, em xin gửi lời cảm ơn đến tất cả những người đã đọc báo cáo này và quan tâm đến công trình của em. Sự quan tâm và phản hồi của thầy cô và các bạn là nguồn động lực lớn để em tiếp tục phát triển và nỗ lực hơn nữa trong tương lai.

Một lần nữa, em xin chân thành cảm ơn tất cả mọi người vì sự đóng góp và hỗ trợ của thầy cô và các bạn. Sự giúp đỡ của quý vị đã góp phần quan trọng vào thành công của bài báo cáo này.

Xin cảm ơn và chúc mọi người một ngày thật hạnh phúc và tràn đầy niềm vui!

Trân trọng,

Đặng Quốc Quy

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

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

Giáo viên hướng dẫn

ThS.TRẦN PHONG NHÃ

MỤC LỤC

A. LÝ THUYẾT

1. HÀM:

1.1. Khái niệm:

Trong ngôn ngữ lập trình C, **hàm (function)** là một khối mã có tên, thực hiện một nhiệm vụ cụ thể và có thể được gọi nhiều lần trong chương trình. Hàm giúp chia nhỏ chương trình thành các phần dễ quản lý, tăng tính tái sử dụng và cải thiện cấu trúc mã nguồn. Có nhiều loại hàm trong các ngôn ngữ lập trình khác nhau, nhưng chúng thường chia thành hai loại chính: hàm được xây dựng sẵn và hàm do người dùng tự định nghĩa.

- Hàm được xây dựng sẵn: Là những hàm có sẵn trong ngôn ngữ lập trình, được cung cấp bởi hệ thống hoặc thư viện tiêu chuẩn. Ví dụ, trong code C, **printf()** là một hàm được xây dựng sẵn để in ra màn hình.

- Hàm do người dùng tự định nghĩa: Là những hàm mà người dùng tự viết để thực hiện một tác vụ cụ thể mà họ cần. Khi tạo hàm, người dùng định nghĩa một khối Code có thể được gọi và thực thi bất kỳ lúc nào trong chương trình.

Các khái niệm cơ bản về hàm trong lập trình gồm:

- Tham số (Parameters): Là các giá trị được truyền vào hàm khi gọi nó. Những giá trị này có thể được sử dụng bên trong hàm để thực hiện một số công việc cụ thể.

- Giá trị trả về (Return Value): Là giá trị mà hàm trả về sau khi thực thi xong. Điều này cho phép hàm truyền thông tin hoặc kết quả của nó trở lại cho phần của chương trình gọi hàm.

- Gọi hàm (Calling a Function): Là việc sử dụng tên của hàm cùng với các tham số (nếu có) để thực thi hàm đó. Khi một hàm được gọi, quá trình thực hiện sẽ chuyển tới nội dung của hàm và thực thi nó.

- Phạm vi biến (Variable Scope): Định nghĩa xem một biến có thể truy cập được từ đâu trong chương trình. Điều này có thể liên quan đến biến cục bộ (local variables), chỉ có thể truy cập được từ bên trong hàm mà nó được khai báo, và biến toàn cục (global variables), có thể truy cập được từ bất kỳ nơi nào trong chương trình.

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

1.2 Ví dụ:

-Đoạn code không sử dụng hàm:

```c
#include <stdio.h>

int main() {
    int n, i, is_prime = 1;

    printf("Nhap mot so nguyen: ");
    scanf("%d", &n);

    if (n <= 1) {
        is_prime = 0;
    } else {
        for (i = 2; i * i <= n; i++) {
            if (n % i == 0) {
                is_prime = 0;
                break;
            }
        }
    }

    if (is_prime)
        printf("%d la so nguyen to.\n", n);
    else
        printf("%d khong phai la so nguyen to.\n", n);

    return 0;
}
```

- Đoạn code có sử dụng hàm:

```c
#include <stdio.h>

// Ham kiem tra so nguyen to
int is_prime(int n) {
    if (n <= 1)
        return 0;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0)
            return 0;
    }
    return 1;
}

int main() {
    int n;

    printf("Nhap mot so nguyen: ");
    scanf("%d", &n);

    if (is_prime(n))
        printf("%d la so nguyen to.\n", n);
    else
        printf("%d khong phai la so nguyen to.\n", n);

    return 0;
}
```

Trong đoạn mã đầu tiên, toàn bộ logic kiểm tra số nguyên tố được viết trực tiếp trong hàm **main()**. Trong khi đó, trong đoạn code thứ hai, chương trình được chia thành hai phần: hàm **is_prime()** để kiểm tra số nguyên tố và hàm **main()** để xử lý nhập/xuất dữ liệu. Hàm **is_prime()** nhận vào một số nguyên n và trả về 1 nếu n là số nguyên tố, ngược lại trả về 0. Hàm **main()** chỉ chịu trách nhiệm gọi hàm **is_prime()** và hiển thị kết quả.giúp mã nguồn rõ ràng và dễ quản lý hơn.

2. CON TRỎ:

2.1 Khái niệm:

Con trỏ là một khái niệm quan trọng trong lập trình. Con trỏ là một biến đặc biệt dùng để lưu trữ địa chỉ bộ nhớ của một biến khác, thay vì lưu giá trị trực tiếp. Điều này cho phép bạn thao tác trực tiếp với bộ nhớ, truy cập và thay đổi giá trị của biến thông qua địa chỉ của nó.

Một số khái niệm cơ bản về con trỏ:

1. Địa chỉ (Address): Mỗi biến trong bộ nhớ có một địa chỉ duy nhất để xác định vị trí của nó trong bộ nhớ.

2. Con trỏ (Pointer): Là một biến chứa địa chỉ của một biến khác.

3. Toán tử con trỏ (**&** và ***):

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

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

4. Khai báo và sử dụng con trỏ:

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

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

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

5. Dùng con trỏ để thực hiện các thao tác trên bộ nhớ:

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

2.2 Ví dụ:

-Đoạn code không sử dụng con trỏ:

```c
#include <stdio.h>

void tangGiaTri(int n) {
    n = n + 1;
    printf("Gia tri trong ham tangGiaTri: %d\n", n);
}

int main() {
    int so = 5;
    tangGiaTri(so);
    printf("Gia tri trong ham main: %d\n", so);
    return 0;
}
```

- Đoạn code có sử dụng con trỏ:

```c
#include <stdio.h>

void tangGiaTri(int *n) {
    *n = *n + 1;
    printf("Gia tri trong ham tangGiaTri: %d\n", *n);
}

int main() {
    int so = 5;
    tangGiaTri(&so);
    printf("Gia tri trong ham main: %d\n", so);
    return 0;
}
```

Trong đoạn mã đầu tiên, hàm **tangGiaTri** nhận một bản sao của biến **so,** việc thay đổi giá trị của n trong hàm không ảnh hưởng đến **so** trong hàm **main**.Còn trong đoạn mã thứ 2, hàm **tangGiaTri** nhận địa chỉ của biến **so** thông qua con trỏ **n**. Việc thay đổi giá trị tại địa chỉ đó ảnh hưởng trực tiếp đến biến so trong hàm **main**.

3. CON TRỎ MẢNG:

3.1. Khái niệm:

Con trỏ mảng là khái niệm liên quan đến việc sử dụng con trỏ để tham chiếu đến các phần tử của mảng trong lập trình.

Con trỏ mảng là một con trỏ trỏ đến phần tử đầu tiên của một mảng. Nó cho phép bạn truy cập và thao tác các phần tử trong mảng thông qua địa chỉ bộ nhớ thay vì chỉ số. Tên của mảng là một con trỏ không thể thay đổi (const pointer) trỏ tới phần tử đầu tiên của mảng.

Một số khái niệm quan trọng liên quan đến con trỏ mảng:

1. Tham chiếu đến phần tử của mảng: Con trỏ có thể được sử dụng để truy cập đến các phần tử của mảng bằng cách sử dụng toán tử chỉ mục **[]** hoặc toán tử pointer *****. Con trỏ di chuyển trong mảng dựa trên kích thước kiểu dữ liệu.

2. Duyệt mảng: Có thể sử dụng con trỏ để duyệt qua tất cả các phần tử của mảng bằng cách di chuyển con trỏ từ phần tử đầu tiên đến phần tử cuối cùng của mảng.

3. Tính toán địa chỉ: Bạn có thể sử dụng con trỏ để tính toán địa chỉ của các phần tử của mảng và truy cập trực tiếp vào bộ nhớ.

4. Truyền mảng vào hàm bằng con trỏ: Có thể truyền mảng vào hàm bằng cách sử dụng con trỏ, cho phép hàm thực hiện các thao tác trên mảng mà không cần phải sao chép toàn bộ mảng.

3.2 Ví dụ:

```c
#include <stdio.h>

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int *p = arr;  // con tro p tro den phan tu dau tien cua mang

    // Duyet mang bang con tro
    for (int i = 0; i < 5; i++) {
        printf("arr[%d] = %d\n", i, *(p + i));
    }

    return 0;
}
```

Trong ví dụ này:

- Chúng ta khai báo một mảng **arr** có 5 phần tử và một con trỏ **p** kiểu **int**.

- Chúng ta gán địa chỉ của phần tử đầu tiên của mảng cho con trỏ **p** bằng cách sử dụng tên mảng (mảng là một con trỏ tới phần tử đầu tiên của nó).

- Sau đó, chúng ta sử dụng con trỏ để truy cập và in ra các phần tử của mảng bằng cách sử dụng toán tử * để lấy giá trị của phần tử mà con trỏ đang trỏ đến.

4. MẢNG CON TRỎ

4.1 Khái niệm:

Mảng con trỏ là một mảng mà mỗi phần tử trong mảng là một con trỏ. Thường dùng để lưu địa chỉ của các biến, chuỗi hoặc mảng khác.

kiểu_dữ_liệu *tên_mảng_con_trỏ[kích_thước_mảng];

Trong đó:

- **kiểu_dữ_liệu**: là kiểu dữ liệu của các phần tử mà mỗi phần tử của mảng con trỏ sẽ trỏ đến.
- **tên_mảng_con_trỏ**: là tên của mảng con trỏ.
- **kích_thước_mảng**: là số lượng phần tử trong mảng con trỏ.

Cần phân biệt được con trỏ mảng và mảng con trỏ:

+ Con trỏ mảng: Con trỏ trỏ tới 1 mảng. VD: int *p[5]

+ Mảng con trỏ: Nhiều con trỏ trong 1 mảng. VD: int *Arr[5]

Trong ví dụ này:

- **int** là kiểu dữ liệu của các phần tử mà mỗi phần tử của mảng con trỏ **Arr** sẽ trỏ đến.
- **Arr** là tên của mảng con trỏ.
- **5** là số lượng phần tử trong mảng con trỏ **Arr**.

Mảng con trỏ này sẽ chứa 5 con trỏ, mỗi con trỏ có thể trỏ đến một vùng nhớ khác nhau trong bộ nhớ, nơi lưu trữ các giá trị kiểu `int`.

4.2 Ví dụ:

```c
#include <stdio.h>

int main() {
    int a = 10, b = 20, c = 30;
    int *arr[3];  // mang gom 3 con tro kieu int

    arr[0] = &a;  // con tro dau tien tro den bien a
    arr[1] = &b;  // con tro thu hai tro den bien b
    arr[2] = &c;  // con tro thu ba tro den bien c

    for (int i = 0; i < 3; i++) {
        printf("Gia tri arr[%d] = %d\n", i, *arr[i]);
    }

    return 0;
}
```

Giải thích:

- Trong ví dụ này, ta có một mảng con trỏ ***Arr** có 3 phần tử a,b,c kiểu **int**.

- Sau đó, ta khai báo và khởi tạo các biến a,b và lần lượt là 10,20 và 30.

-Tiếp theo, ta gán địa chỉ của các biến này cho các phần tử của mảng con trỏ ***Arr** bằng cách sử dụng toán tử **&**.

- Cuối cùng, ta in ra giá trị của các phần tử của mảng con trỏ.

Trong ví dụ này, mỗi phần tử của mảng con trỏ ***Arr** đều trỏ đến một biến khác nhau trong bộ nhớ, cho phép chúng ta lưu trữ địa chỉ của các biến khác nhau trong một mảng.

5. CON TRỎ MẢNG:

5.1 Khái niệm:

Con trỏ hàm là một khái niệm trong lập trình được sử dụng để tham chiếu đến một hàm cụ thể trong mã chương trình.

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

Trong một số ngôn ngữ lập trình như C và C++, con trỏ hàm được khai báo bằng cách chỉ ra kiểu dữ liệu của hàm mục tiêu cùng với tên của con trỏ và dấu sao (*). Ví dụ:

int (*funcPtr)(int, int);

Đây là một con trỏ hàm có kiểu trả về là int và nhận hai tham số kiểu int.

5.2 Lợi ích khi sử dụng hàm con trỏ:

Việc sử dụng hàm con trỏ mang lại một số lợi ích và tác dụng quan trọng trong lập trình, bao gồm:

1. Linh hoạt: Cho phép chọn và gọi hàm một cách linh hoạt tại thời điểm chạy chương trình.

2. Tách biệt logic: Việc sử dụng con trỏ hàm cho phép bạn tách biệt logic thực thi của hàm ra khỏi việc quyết định loại hàm nào sẽ được sử dụng.

3. Tái sử dụng mã nguồn: Giảm sự lặp lại mã nguồn bằng cách sử dụng con trỏ hàm trong các cấu trúc dữ liệu hoặc thuật toán chung.

4. Hỗ trợ callback: Hữu ích trong việc triển khai các hàm callback, nơi một hàm được truyền vào một hàm khác để được gọi lại sau.

5.3. Ví dụ:

```c
#include <stdio.h>

// Dinh nghia cac ham thuc hien phep toan
int cong(int a, int b) {
    return a + b;
}

int tru(int a, int b) {
    return a - b;
}

int main() {
    int x = 10, y = 5;
    int (*pheptoan)(int, int);  // Khai bao con tro ham

    // Chon phep toan cong
    pheptoan = cong;
    printf("Ket qua: %d\n", pheptoan(x, y));

    // Chon phep toan tru
    pheptoan = tru;
    printf("Ket qua: %d\n", pheptoan(x, y));

    return 0;
}
```

Giải thích:

- Ta định nghĩa hai hàm **cong** và **tru** để thực hiện phép cộng và phép trừ.

- Khai báo một con trỏ hàm **pheptoan** có kiểu trả về là **int** và nhận hai tham số kiểu **int**.

- Trong hàm **main**, chúng ta gán địa chỉ của hàm **cong** cho con trỏ **pheptoan** và gọi hàm thông qua con trỏ này. Sau đó, chúng ta gán địa chỉ của hàm **tru** cho con trỏ và gọi hàm tương tự.

6. CẤP PHÁT ĐỘNG:

6.1. Khái niệm:

Cấp phát động là quá trình trong lập trình khi bạn cần cấp phát bộ nhớ trong quá trình thực thi của chương trình, thay vì cấp phát bộ nhớ tĩnh trong quá trình biên dịch. Quá trình này cho phép bạn tạo ra các cấu trúc dữ liệu có kích thước có thể thay đổi hoặc không biết trước tại thời điểm biên dịch.

Các từ khóa quan trọng trong cấp phát động:

- malloc: Hàm này cấp phát một lượng bộ nhớ cụ thể và trả về một con trỏ tới vùng nhớ được cấp phát.

- calloc: Tương tự như malloc, nhưng calloc cấp phát một số lượng các đối tượng và khởi tạo tất cả các byte của bộ nhớ được cấp phát thành giá trị 0.

- realloc: Hàm này thay đổi kích thước của vùng nhớ đã được cấp phát trước đó. Nó có thể làm tăng hoặc giảm kích thước của vùng nhớ và trả về một con trỏ mới.

- free: Hàm này được sử dụng để giải phóng bộ nhớ đã được cấp phát trước đó, giúp trả lại tài nguyên cho hệ thống.

Việc sử dụng cấp phát động cho phép tạo ra các cấu trúc dữ liệu linh hoạt như mảng động, danh sách liên kết, cây nhị phân, và đồ thị. Tuy nhiên, cần lưu ý rằng việc quản lý bộ nhớ động có thể phức tạp và có thể dẫn đến các vấn đề như rò rỉ bộ nhớ và xung đột bộ nhớ, do đó, việc sử dụng cấp phát động cần được thực hiện một cách cẩn thận.

6.2. Lợi ích khi sử dụng cấp phát động:

Việc sử dụng cấp phát động mang lại một số lợi ích quan trọng trong lập trình, bao gồm:

1. Tính linh hoạt: Cấp phát động cho phép bạn tạo ra các cấu trúc dữ liệu có kích thước có thể thay đổi trong quá trình thực thi của chương trình. Điều này làm cho mã của bạn trở nên linh hoạt hơn và có thể xử lý các tình huống đa dạng.

2. Tiết kiệm bộ nhớ: Trong một số trường hợp, cấp phát động có thể giúp tiết kiệm bộ nhớ bằng cách chỉ cấp phát bộ nhớ khi cần và giải phóng nó khi không cần thiết nữa. Điều này giúp tránh lãng phí tài nguyên.

3. Hỗ trợ cấu trúc dữ liệu linh hoạt: Cấp phát động làm cho việc triển khai các cấu trúc dữ liệu động như danh sách liên kết, cây nhị phân, và đồ thị trở nên dễ dàng hơn. Bạn có thể thay đổi kích thước của cấu trúc dữ liệu và thực hiện các thao tác chèn, xóa và sắp xếp một cách linh hoạt.

4. Quản lý bộ nhớ hiệu quả: Cấp phát động cho phép bạn quản lý bộ nhớ của chương trình một cách hiệu quả hơn. Bạn có thể chỉ cấp phát bộ nhớ khi cần và giải phóng nó khi không cần thiết nữa, giúp tránh các vấn đề như rò rỉ bộ nhớ và xung đột bộ nhớ.

5. Hỗ trợ các cấu trúc dữ liệu phức tạp: Cấp phát động làm cho việc triển khai các cấu trúc dữ liệu phức tạp như bảng băm, hàng đợi ưu tiên và vùng nhớ đệm trở nên dễ dàng hơn. Điều này giúp tạo ra các ứng dụng phần mềm phức tạp và hiệu quả hơn.

Tóm lại, việc sử dụng cấp phát động mang lại tính linh hoạt, tiết kiệm bộ nhớ, hỗ trợ cấu trúc dữ liệu linh hoạt, quản lý bộ nhớ hiệu quả và hỗ trợ các cấu trúc dữ liệu phức tạp trong lập trình.

6.3. So sánh với mảng tĩnh:

1. Kích thước:

- Mảng tĩnh: Kích thước của mảng tĩnh được xác định tại thời điểm biên dịch và không thể thay đổi trong quá trình thực thi của chương trình.
- Cấp phát động: Kích thước của vùng nhớ được cấp phát động có thể thay đổi trong quá trình thực thi của chương trình.

2. Quản lý bộ nhớ:

- Mảng tĩnh: Bộ nhớ cho mảng tĩnh được quản lý tự động, do đó không cần phải giải phóng bộ nhớ thủ công.
- Cấp phát động: Bạn phải tự quản lý việc cấp phát và giải phóng bộ nhớ khi sử dụng cấp phát động.

3. Tính linh hoạt:

- Mảng tĩnh: Mảng tĩnh không linh hoạt về kích thước và không thể thay đổi trong quá trình thực thi của chương trình.
- Cấp phát động: Cấp phát động cho phép bạn tạo ra các cấu trúc dữ liệu có kích thước linh hoạt và có thể thay đổi tùy ý trong quá trình thực thi của chương trình.

4. Hiệu suất:

- Mảng tĩnh: Truy cập vào các phần tử của mảng tĩnh thường nhanh hơn do vị trí của các phần tử được biết trước và không thay đổi.
- Cấp phát động: Truy cập vào các phần tử của cấu trúc dữ liệu cấp phát động thường có hiệu suất kém hơn một chút do cần thêm thời gian để truy cập thông qua con trỏ.

5. Điều kiện sử dụng:

- Mảng tĩnh: Thích hợp cho các trường hợp mà kích thước của dữ liệu là cố định và không thay đổi.

- Cấp phát động: Thích hợp cho các trường hợp mà kích thước của dữ liệu có thể thay đổi hoặc không biết trước.

Tóm lại, cả mảng tĩnh và cấp phát động đều có những ưu điểm và nhược điểm riêng của chúng và thích hợp cho các tình huống sử dụng khác nhau trong lập trình. Lựa chọn giữa chúng phụ thuộc vào yêu cầu cụ thể của dự án và tính linh hoạt cần thiết.

Ví dụ:

6.4. Ví dụ:

```c
#include <stdio.h>
#include <stdlib.h> // Thu vien de su dung cac ham cap phat dong

int main() {
    int n; // So luong phan tu cua mang, nhap tu nguoi dung
    printf("Nhap so luong phan tu cua mang: ");
    scanf("%d", &n);
    // Cap phat dong mot mang co n phan tu kieu int
    int *arr = (int *)malloc(n * sizeof(int));
    // Kiem tra xem cap phat da thanh cong chua
    if (arr == NULL) {
        printf("Khong du bo nho!");
        return 1; // Ket thuc chuong trinh voi ma loi
    }
    // Nhap cac phan tu cua mang tu nguoi dung
    printf("Nhap cac phan tu cua mang:\n");
    for (int i = 0; i < n; i++) {
        printf("arr[%d] = ", i);
        scanf("%d", &arr[i]);
    }
    // In cac phan tu cua mang
    printf("Cac phan tu cua mang la:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    // Giai phong bo nho da cap phat dong
    free(arr);
    return 0;
}
```

Trong ví dụ này:

1. Người dùng nhập số lượng phần tử của mảng từ bàn phím.

2. Sử dụng hàm **malloc** để cấp phát động một mảng có số lượng phần tử là **n**.

3. Kiểm tra xem việc cấp phát đã thành công hay không. Nếu **malloc** trả về **NULL**, điều này có thể chỉ ra rằng không đủ bộ nhớ trống còn.

4. Nhập các phần tử của mảng từ người dùng.

5. In các phần tử của mảng ra màn hình.

6. Sử dụng hàm **free** để giải phóng bộ nhớ đã cấp phát động khi không cần thiết nữa.

Ví dụ này minh họa việc sử dụng cấp phát động để tạo ra một mảng động và giải phóng bộ nhớ sau khi đã sử dụng xong.

7. XỬ LÍ TỆP

7.1 Khái niệm:

Xử lý tệp trong C là quá trình giúp chương trình có thể mở, đọc, ghi và đóng các tệp trên ổ đĩa bằng cách sử dụng con trỏ FILE và các hàm như fopen, fclose, fprintf, fscanf, nhằm lưu trữ hoặc truy xuất dữ liệu một cách lâu dài thay vì chỉ xử lý tạm thời trong bộ nhớ RAM.

Các hoạt động cơ bản trong xử lý tệp bao gồm:

1. Mở tệp (fopen) : Đây là bước đầu tiên trong xử lý tệp, trong đó bạn mở một tệp để đọc hoặc ghi dữ liệu vào đó.

2. Đọc/ghi từ tệp (fscanf, fprintf, fgets, fputs, fread, fwrite, v.v.) : Bạn có thể đọc/ghi dữ liệu từ tệp đã mở và sử dụng nó trong ứng dụng của mình.

3. Đóng tệp (fclose): Sau khi bạn đã hoàn thành công việc của mình với một tệp, bạn cần đóng nó. Điều này giải phóng tài nguyên hệ thống và đảm bảo rằng không có ai khác có thể truy cập vào tệp đó trong khi bạn không sử dụng.

4. Xử lý thư mục: Ngoài việc xử lý các tệp, bạn cũng có thể cần làm việc với các thư mục trong hệ thống tệp. Điều này có thể bao gồm việc tạo mới, di chuyển, xóa thư mục và thậm chí là liệt kê các tệp trong một thư mục.

7.1. Tại sao cần xử lí tệp:

1. Lưu trữ và truy cập dữ liệu: Tệp là nơi lưu trữ dữ liệu trong lập trình. Bằng cách sử dụng tệp, chương trình có thể lưu trữ dữ liệu vào bộ nhớ không thể thay đổi và truy cập nó lại sau này.

2. Dữ liệu lớn: Trong các ứng dụng thực tế, dữ liệu thường lớn và không thể lưu trữ hoặc xử lý trong bộ nhớ RAM. Sử dụng tệp cho phép lưu trữ và xử lý các tập tin dữ liệu lớn mà không cần phải tải toàn bộ dữ liệu vào bộ nhớ.

3. Giao tiếp với người dùng: Các chương trình thường cần đọc dữ liệu từ người dùng hoặc ghi dữ liệu vào tệp để lưu trữ thông tin nhập liệu hoặc kết quả của chương trình.

4. Đồng bộ hóa và chia sẻ dữ liệu: Trong các ứng dụng đa tiến trình hoặc đa luồng, việc sử dụng tệp để lưu trữ và truy cập dữ liệu giữa các tiến trình hoặc luồng là cần thiết để đồng bộ hóa và chia sẻ dữ liệu.

5. Bảo trì và quản lý dữ liệu: Sử dụng tệp cho phép dễ dàng sao lưu, di chuyển, xóa và quản lý các tập tin và thư mục, giúp cho việc bảo trì và quản lý dữ liệu trở nên dễ dàng hơn.

6. Tích hợp với hệ thống và ứng dụng khác: Sử dụng tệp cho phép các chương trình tương tác với các tệp và thư mục được tạo bởi các ứng dụng và hệ thống khác, cho phép tích hợp và tương tác giữa các ứng dụng khác nhau.

7.3 Ví dụ:

```c
#include <stdio.h>

int main() {
    FILE *file;

    // Ghi du lieu vao tep
    file = fopen("data.txt", "w"); // mo tep o che do ghi
    if (file == NULL) {
        printf("Khong mo duoc tep de ghi!\n");
        return 1;
    }
    fprintf(file, "Xin chao!\n");
    fprintf(file, "Day la dong thu hai.\n");
    fclose(file); // dong tep sau khi ghi xong

    // Doc du lieu tu tep
    file = fopen("data.txt", "r"); // mo tep o che do doc
    if (file == NULL) {
        printf("Khong mo duoc tep de doc!\n");
        return 1;
    }
    char line[100];
    printf("Noi dung trong tep:\n");
    while (fgets(line, sizeof(line), file)) {
        printf("%s", line);
    }
    fclose(file); // dong tep sau khi doc xong

    return 0;
}
```

Kết quả khi chạy chương trình này sẽ là hiển thị nội dung của tệp "data.txt" lên màn hình:

Xin chao!

Day la dong thu hai.

- Chúng ta sử dụng hàm **fopen** để mở tệp **"input.txt"** để đọc (**"r"** là chế độ đọc).

- Sử dụng một vòng lặp để đọc từng dòng của tệp bằng cách sử dụng hàm **fgets**.

- Hiển thị từng dòng được đọc lên màn hình bằng cách sử dụng hàm **printf**.

- Cuối cùng, chúng ta sử dụng hàm **fclose** để đóng tệp sau khi đã đọc xong.

8. KIỂU CẤU TRÚC:

8.1 Khái niệm:

Kiểu cấu trúc (struct) trong C là một kiểu dữ liệu do người lập trình định nghĩa, dùng để nhóm nhiều biến (thuộc nhiều kiểu dữ liệu khác nhau) lại thành một đơn vị logic duy nhất.Struct là giải pháp khi bạn cần giải quyết các bài toán thực tế khi mà đối tượng bạn cần lưu lại trong chương trình cần rất nhiều thông tin.

Struct giống như một "hộp chứa nhiều biến liên quan", ví dụ như thông tin của một sinh viên gồm: họ tên (kiểu chuỗi), tuổi (int), điểm (float) — ta gộp tất cả vào một struct sinhvien.

8.2 Những tính năng mở rộng trong kiểu cấu trúc struct:

1. typedef struct: typedef giúp bạn đặt tên kiểu dữ liệu mới cho struct, để khi khai báo biến khỏi phải viết dài dòng.

VD:

```
typedef struct {
char ten[50];
int tuoi;
float diem;
} SinhVien;
```

2. Mảng struct: Bạn có thể khai báo mảng chứa nhiều struct, rất hữu ích khi lưu danh sách nhiều đối tượng.

VD:

```
SinhVien ds[100];//Mang 100 sinh vien
```

3. Struct lồng nhau : Struct có thể chứa một hoặc nhiều struct khác bên trong, giúp tổ chức dữ liệu phức tạp hơn.

VD:

```
typedef struct {
int ngay;
int thang;
int nam;
} Ngay;
```

```
typedef struct {
    char ten[50];
    Ngay sinhnhat; // struct ben trong struct
    float diem;
} SinhVien;
```

9. DANH SÁCH LIÊN KẾT:

9.1 Khái niệm:

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

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

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

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

Có hai loại danh sách liên kết chính:

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

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

Các thao tác phổ biến trên danh sách liên kết bao gồm: thêm một nút mới vào đầu hoặc cuối danh sách, xóa một nút, tìm kiếm một giá trị trong danh sách, và duyệt qua các nút của danh sách để thực hiện các thao tác khác nhau.

Danh sách liên kết thường được sử dụng trong các ứng dụng cần thêm/xóa dữ liệu một cách linh hoạt hoặc trong các trường hợp mà kích thước của dữ liệu không biết trước.

9.2 Lợi ích khi sử dụng danh sách liên kết:

Sử dụng danh sách liên kết mang lại nhiều lợi ích quan trọng trong lập trình, bao gồm:

1. Khả năng thêm/xóa linh hoạt: Danh sách liên kết cho phép thêm và xóa các phần tử một cách linh hoạt mà không cần phải di chuyển hoặc sao chép các phần tử khác như trong mảng. Điều này làm cho việc thêm/xóa các phần tử trở nên đơn giản và hiệu quả hơn.

2. Khả năng mở rộng động: Danh sách liên kết có thể mở rộng động, tức là bạn có thể thêm bất kỳ số lượng phần tử nào vào danh sách mà không gặp phải giới hạn của kích thước tĩnh như trong mảng.

3. Duyệt dữ liệu dễ dàng: Duyệt qua các phần tử trong danh sách liên kết là dễ dàng, vì mỗi phần tử chỉ cần trỏ đến phần tử tiếp theo. Điều này giúp thực hiện các thao tác duyệt và truy cập dữ liệu một cách dễ dàng và hiệu quả.

4. Khả năng chia sẻ tài nguyên: Danh sách liên kết cho phép chia sẻ tài nguyên giữa các phần tử một cách dễ dàng. Ví dụ, nếu nhiều danh sách chia sẻ các nút có cùng dữ liệu, không cần phải lưu trữ nhiều bản sao của dữ liệu đó.

5. Xử lý dữ liệu không đồng nhất: Danh sách liên kết cho phép bạn lưu trữ dữ liệu không đồng nhất, tức là các phần tử trong danh sách có thể có các kiểu dữ liệu khác nhau. Điều này làm cho việc lưu trữ và quản lý các dữ liệu có cấu trúc phức tạp trở nên dễ dàng hơn.

6. Điều chỉnh kích thước một cách linh hoạt: Bạn có thể dễ dàng điều chỉnh kích thước của danh sách liên kết, không như mảng có kích thước cố định. Điều này giúp tiết kiệm bộ nhớ và làm cho việc quản lý dữ liệu trở nên linh hoạt hơn.

9.3 Ví dụ:

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

// Khai bao cau truc Node
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Ham tao Node moi
Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Cap phat bo nho that bai!\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// Them Node vao dau danh sach
void insertAtHead(Node** head, int value) {
    Node* newNode = createNode(value);
    newNode->next = *head;
    *head = newNode;
}
```

```c
// In danh sach lien ket
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Giai phong bo nho danh sach
void freeList(Node* head) {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    Node* head = NULL; // danh sach rong

    // Chen gia tri vao danh sach
    insertAtHead(&head, 10);
    insertAtHead(&head, 20);
    insertAtHead(&head, 30);

    // In danh sach
    printList(head);

    // Giai phong bo nho
    freeList(head);

    return 0;
}
```

Đoạn code thực hiện:

1. Định nghĩa struct Node với trường data và con trỏ next trỏ đến nút kế tiếp.
2. Sử dụng con trỏ head để quản lý đầu danh sách liên kết.
3. Hàm createNode(int x) tạo nút mới với dữ liệu x và cấp phát bộ nhớ động.
4. Hàm insertAtHead(Node** head, Node* new_node) thêm nút mới vào đầu danh sách, cập nhật head.
5. Trong main(), nhập số phần tử, tạo nút và thêm vào đầu danh sách trong vòng lặp.
6. Cuối cùng, duyệt danh sách và in ra dữ liệu từng nút.

B.ỨNG DỤNG

XÂY DỰNG ỨNG DỤNG CHO VIỆC QUẢN LÝ THỜI GIAN VÀ LỊCH HỌC CÁ NHÂN

1. Các tính năng cần thiết:

- Thêm sự kiện học (môn học, thời gian, địa điểm) vào một ngày cụ thể.

- Xem lịch học theo ngày.

- Hiển thị toàn bộ lịch học đã lưu.

- Tìm kiếm sự kiện theo tên môn học.

- Xóa sự kiện theo ngày và vị trí.

2. Mục đích của bài toán:

- Nhằm xây dựng một hệ thống quản lý lịch học theo ngày, chương trình cần giải quyết bài toán làm sao để phân chia các sự kiện học theo từng ngày cụ thể, trong đó mỗi ngày có thể có nhiều sự kiện khác nhau như môn học, thời gian và địa điểm học. Để thực hiện điều đó, em quyết định kết hợp việc sử dụng mảng động để lưu các sự kiện trong một ngày và danh sách liên kết để quản lý các ngày học khác nhau.
- Việc sử dụng danh sách liên kết giúp tạo ra một chuỗi các ngày học liên tiếp, trong đó mỗi ngày được lưu trữ dưới dạng một node có thể chứa nhiều sự kiện. Các sự kiện trong một ngày được lưu trong một mảng động, giúp dễ dàng mở rộng số lượng sự kiện theo nhu cầu thực tế.
- Sử dụng con trỏ và cấp phát động hỗ trợ việc mở rộng dung lượng của mảng sự kiện khi số lượng sự kiện tăng lên, đồng thời danh sách liên kết giúp quản lý hiệu quả các ngày học riêng biệt. Từ đó, chương trình có thể thuận tiện trong việc nhập liệu, xem, tìm kiếm, và xóa các sự kiện học một cách rõ ràng và linh hoạt nhất.

-Phần code:

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

typedef struct {
    char monHoc[50];
    char thoiGian[10];
    char diaDiem[50];
} SuKien;
```

```c
typedef struct NgayHoc {
    char ngay[11];  // "dd/mm/yyyy"
    SuKien* dsSuKien;
    int soLuong;
    int sucChua;
    struct NgayHoc* tiep;
} NgayHoc;

NgayHoc* danhSachNgay = NULL;

// ham xoa bo dem stdin
void donSachBoDem() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}

// kiem tra nam nhuan
int laNamNhuan(int nam) {
    return (nam % 400 == 0) || (nam % 4 == 0 && nam % 100 != 0);
}

// kiem tra ngay hop le
int kiemTraNgayHopLe(const char* ngay) {
    if (strlen(ngay) != 10) return 0;
    if (ngay[2] != '/' || ngay[5] != '/') return 0;

    for (int i = 0; i < 10; i++) {
        if (i == 2 || i == 5) continue;
        if (!isdigit(ngay[i])) return 0;
    }

    int dd = (ngay[0] - '0') * 10 + (ngay[1] - '0');
    int mm = (ngay[3] - '0') * 10 + (ngay[4] - '0');
    int yyyy = (ngay[6] - '0') * 1000 + (ngay[7] - '0') * 100 + (ngay[8] - '0') * 10 + (ngay[9] - '0');

    if (yyyy < 1900 || yyyy > 2100) return 0;
    if (mm < 1 || mm > 12) return 0;

    int soNgayTrongThang[] = {31,28,31,30,31,30,31,31,30,31,30,31};
    if (laNamNhuan(yyyy)) soNgayTrongThang[1] = 29;

    if (dd < 1 || dd > soNgayTrongThang[mm - 1]) return 0;

    return 1;
}
```

```
// kiem tra thoi gian hop le
int kiemTraThoiGianHopLe(const char* tg) {
    int len = strlen(tg);
    if (len < 4 || len > 5) return 0;  // vi du: 7:00 (4 ky tu) hoac 07:00 (5 ky tu)

    int colonPos = -1;
    for (int i = 0; i < len; i++) {
        if (tg[i] == ':') {
            colonPos = i;
            break;
        }
    }
    if (colonPos == -1) return 0; // khong co dau ':'

    // kiem tra phan gio
    for (int i = 0; i < colonPos; i++) {
        if (!isdigit(tg[i])) return 0;
    }
    // kiem tra phan phut
    for (int i = colonPos + 1; i < len; i++) {
        if (!isdigit(tg[i])) return 0;
    }

    // chuyen phan gio va phut thanh so nguyen
    int gio = 0, phut = 0;
    for (int i = 0; i < colonPos; i++) {
        gio = gio * 10 + (tg[i] - '0');
    }
    for (int i = colonPos + 1; i < len; i++) {
        phut = phut * 10 + (tg[i] - '0');
    }

    if (gio < 0 || gio > 23) return 0;
    if (phut < 0 || phut > 59) return 0;

    return 1;
}
```

```
// tim ngay trong danh sach lien ket
NgayHoc* timNgay(char* ngay) {
    NgayHoc* p = danhSachNgay;
    while (p != NULL) {
        if (strcmp(p->ngay, ngay) == 0) return p;
        p = p->tiep;
    }
    return NULL;
}
```

```c
// them ngay moi (neu chua co)
NgayHoc* themNgay(char* ngay) {
    NgayHoc* moi = (NgayHoc*)malloc(sizeof(NgayHoc));
    if (moi == NULL) {
        printf("Loi cap phat bo nho!\n");
        return NULL;
    }
    strcpy(moi->ngay, ngay);
    moi->dsSuKien = (SuKien*)malloc(2 * sizeof(SuKien));  // ban dau chua 2 su kien
    if (moi->dsSuKien == NULL) {
        printf("Loi cap phat bo nho!\n");
        free(moi);
        return NULL;
    }
    moi->soLuong = 0;
    moi->sucChua = 2;
    moi->tiep = danhSachNgay;
    danhSachNgay = moi;
    return moi;
}

// kiem tra su kien da ton tai chua (cung ngay, cung mon, cung thoi gian)
int suKienTrungLap(NgayHoc* ngayHoc, const char* monHoc, const char* thoiGian) {
    for (int i = 0; i < ngayHoc->soLuong; i++) {
        if (strcmp(ngayHoc->dsSuKien[i].monHoc, monHoc) == 0 &&
            strcmp(ngayHoc->dsSuKien[i].thoiGian, thoiGian) == 0) {
            return 1; // trung lap
        }
    }
    return 0; // khong trung lap
}
```

```c
void themSuKien() {
    char ngay[11];
    do {
        printf("Nhap ngay (dd/mm/yyyy): ");
        scanf("%10s", ngay);
        donSachBoDem();
        if (!kiemTraNgayHopLe(ngay)) {
            printf("Ngay khong hop le. Vui long nhap lai.\n");
        } else break;
    } while (1);

    NgayHoc* n = timNgay(ngay);
    if (n == NULL) {
        n = themNgay(ngay);
        if (n == NULL) return;
    }
```

```
    // Khai bao bien tam luu thong tin su kien
    char monHoc[50];
    char thoiGian[10];
    char diaDiem[50];

    printf("Nhap ten mon hoc: ");
    if (fgets(monHoc, sizeof(monHoc), stdin) == NULL) {
        printf("Loi nhap!\n");
        return;
    }
    monHoc[strcspn(monHoc, "\n")] = '\0';

    do {
        printf("Nhap thoi gian (h:mm hoac hh:mm): ");
        scanf("%9s", thoiGian);
        donSachBoDem();
        if (!kiemTraThoiGianHopLe(thoiGian)) {
            printf("Thoi gian khong hop le. Vui long nhap lai.\n");
        } else break;
    } while (1);

    // Kiem tra trung lap
    if (suKienTrungLap(n, monHoc, thoiGian)) {
        printf("Su kien da ton tai (trung ngay, mon hoc va thoi gian).\n");
        return;
    }

    printf("Nhap dia diem: ");
    if (fgets(diaDiem, sizeof(diaDiem), stdin) == NULL) {
        printf("Loi nhap!\n");
        return;
    }
    diaDiem[strcspn(diaDiem, "\n")] = '\0';
```

```
    // Them su kien vao danh sach
    if (n->soLuong >= n->sucChua) {
        n->sucChua *= 2;
        SuKien* tmp = (SuKien*)realloc(n->dsSuKien, n->sucChua * sizeof(SuKien));
        if (tmp == NULL) {
            printf("Khong du bo nho!\n");
            return;
        }
        n->dsSuKien = tmp;
    }

    SuKien* s = &n->dsSuKien[n->soLuong];
    strcpy(s->monHoc, monHoc);
    strcpy(s->thoiGian, thoiGian);
    strcpy(s->diaDiem, diaDiem);

    n->soLuong++;
    printf("Da them su kien thanh cong!\n");
}
```

```c
// hien thi su kien cua ngay
void hienThiNgay() {
    char ngay[11];
    printf("Nhap ngay muon xem (dd/mm/yyyy): ");
    scanf("%10s", ngay);
    donSachBoDem();

    if (!kiemTraNgayHopLe(ngay)) {
        printf("Ngay khong hop le!\n");
        return;
    }

    NgayHoc* n = timNgay(ngay);
    if (n == NULL || n->soLuong == 0) {
        printf("Khong co lich hoc trong ngay nay.\n");
        return;
    }

    printf("== Lich hoc ngay %s ==\n", ngay);
    for (int i = 0; i < n->soLuong; i++) {
        printf("%d. %s - %s - %s\n", i + 1,
               n->dsSuKien[i].thoiGian,
               n->dsSuKien[i].monHoc,
               n->dsSuKien[i].diaDiem);
    }
}
```

```c
// hien thi tat ca lich hoc
void hienThiTatCa() {
    NgayHoc* p = danhSachNgay;
    if (p == NULL) {
        printf("Chua co lich hoc nao.\n");
        return;
    }

    while (p != NULL) {
        printf("== Ngay %s ==\n", p->ngay);
        for (int i = 0; i < p->soLuong; i++) {
            printf(" %d. %s - %s - %s\n", i + 1,
                   p->dsSuKien[i].thoiGian,
                   p->dsSuKien[i].monHoc,
                   p->dsSuKien[i].diaDiem);
        }
        p = p->tiep;
    }
}
```

```c
// tim kiem theo mon hoc
void timKiemTheoMon() {
    char mon[50];
    donSachBoDem();
    printf("Nhap ten mon hoc can tim: ");
    if (fgets(mon, sizeof(mon), stdin) == NULL) {
        printf("Loi nhap!\n");
        return;
    }
    mon[strcspn(mon, "\n")] = '\0';

    NgayHoc* p = danhSachNgay;
    int timThay = 0;
    while (p != NULL) {
        for (int i = 0; i < p->soLuong; i++) {
            if (strstr(p->dsSuKien[i].monHoc, mon) != NULL) {
                printf("Ngay %s - %s - %s - %s\n",
                       p->ngay,
                       p->dsSuKien[i].thoiGian,
                       p->dsSuKien[i].monHoc,
                       p->dsSuKien[i].diaDiem);
                timThay = 1;
            }
        }
        p = p->tiep;
    }

    if (!timThay) {
        printf("Khong tim thay lich hoc voi mon '%s'.\n", mon);
    }
}

// xoa su kien theo ngay va vi tri
void xoaSuKien() {
    char ngay[11];
    int viTri;
    printf("Nhap ngay (dd/mm/yyyy): ");
    scanf("%10s", ngay);
    donSachBoDem();

    if (!kiemTraNgayHopLe(ngay)) {
        printf("Ngay khong hop le!\n");
        return;
    }
```

```c
    NgayHoc* n = timNgay(ngay);
    if (n == NULL || n->soLuong == 0) {
        printf("Khong co su kien trong ngay.\n");
        return;
    }

    printf("Nhap vi tri su kien can xoa (1-%d): ", n->soLuong);
    scanf("%d", &viTri);
    donSachBoDem();

    if (viTri < 1 || viTri > n->soLuong) {
        printf("Vi tri khong hop le!\n");
        return;
    }

    for (int i = viTri - 1; i < n->soLuong - 1; i++) {
        n->dsSuKien[i] = n->dsSuKien[i + 1];
    }
    n->soLuong--;

    printf("Da xoa su kien.\n");
}
```

```c
// menu chinh
void menu() {
    int chon;
    do {
        printf("\n==== QUAN LY LICH HOC ====\n");
        printf("1. Them su kien\n");
        printf("2. Xem lich theo ngay\n");
        printf("3. Hien thi tat ca lich hoc\n");
        printf("4. Tim kiem theo mon hoc\n");
        printf("5. Xoa su kien theo ngay\n");
        printf("0. Thoat\n");
        printf("Chon: ");
        scanf("%d", &chon);
        donSachBoDem();

        switch (chon) {
            case 1: themSuKien(); break;
            case 2: hienThiNgay(); break;
            case 3: hienThiTatCa(); break;
            case 4: timKiemTheoMon(); break;
            case 5: xoaSuKien(); break;
            case 0: printf("Tam biet!\n"); break;
            default: printf("Lua chon khong hop le!\n");
        }

    } while (chon != 0);
}

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

KẾT LUẬN

Thông qua cách xây dựng bài toán trên, cùng với việc kết hợp giữa Mảng và Danh sách Liên kết, cơ bản đã đáp ứng được các điều cần có của việc quản lí lịch học. Tuy nhiên, đây chỉ mới là ý tưởng và đang trong quá trình cải thiện, nâng cấp. Không tránh khỏi một số sai sót trong khâu thiết kế chương trình.

Bên cạnh đó, em mong được nhận những nhận xét từ thầy để phần nào cải thiện khả năng lập trình của mình. Cũng như có thể nâng cấp được bài toán đã trình bày trên từ đó mà tối ưu nhất cách thực thi của bài toán trong việc quản lí lịch học cá nhân.

TÀI LIỆU THAM KHẢO

[1]. “C Tutorial – Learning C Programming”,[Online]. Available: C Tutorial (w3schools.com). [Accessed 8/05/2025].

[2]. “Lập trình C cơ bản – Giới thiệu ngôn ngữ C”,[Online]. Available: Lập trình C cơ bản - Giới thiệu ngôn ngữ C (200lab.io). [Accessed 10/05/2025].

[3]. “Stack Overflow”, [Online]. Available: Stack Overflow - Where Developers Learn, Share, & Build Careers. [Accessed 12/5/2025].

[4]. “GitHub”, [Online]. Available: GitHub (https://github.com). [Accessed 19/05/2025].