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

BÁO CÁO

MÔN: KỸ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn: ThS. TRẦN PHONG NHÃ

Sinh viên thực hiện: BÙI HOÀNG QUÂN

Lớp: CQ.65.CNTT

Khoá: 65

Tp. Hồ Chí Minh, 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: KỸ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn: ThS. TRẦN PHONG NHÃ

Sinh viên thực hiện: BÙI HOÀNG QUÂN

Lớp: CQ.65.CNTT

Khoá: 65

Tp. Hồ Chí Minh, năm 2025

LỜI CẢM ƠN

Lời nói đầu tiên, nhóm 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.

Đặc biệt xin cảm ơn thầy Trần Phong Nhã – giảng viên môn Kỹ thuật lập trình đã giúp đỡ tạo điều kiện để em hoàn thành báo cáo "Bài Tập Lớn"., hướng dẫn cho em kiến thức, định hướng và kỹ năng để có thể hoàn thành bài báo cáo này này.

Tuy đã cố gắng trong quá trình nghiên cứu tìm hiểu tuy nhiên do kiến thức còn hạn chế nên vẫn còn tồn tại nhiều thiếu sót. Vì vậy em rất mong nhận được sự đóng góp ý kiến của Quý thầy cô bộ môn để đề tài của em có thể hoàn thiện hơn.

Lời sau cùng, nhóm em xin gửi lời chúc tới Quý Thầy Cô Bộ môn Công nghệ thông tin và hơn hết thầy Trần Phong Nhã có thật nhiều sức khỏe, có nhiều thành công trong công việc.

Xin chân thành cảm ơn!

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

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

Giáo viên hướng dẫn

Trần Phong Nhã

MỤC LỤC

PHẦN A - LÝ THUYẾT

I. Hàm

1. Khái niệm

_ Một hàm trong C được hiểu theo nghĩa là một “Routine” hoặc “subprogram"

_ Hàm là một đơn vị độc lập trong C

• Không được xây dựng hàm bên trong 1 hàm khác

• Mỗi hàm có thể có các biến, hằng, mảng riêng

_ Một chương trình viết bằng C gồm 1 hoặc nhiều hàm, trong đó có 1 hàm chính là hàm “main()” _ Hàm có thể có giá trị trả về (kết quả của hàm) hoặc không có giá trị trả về (chỉ đơn thuần thực hiện 1 công việc nào đó)

- Hàm có thể có hoặc không có tham số

2. Quy tắc hoạt động của hàm:

_ Lời gọi hàm có dạng tổng quát như sau:

Tên_hàm ([danh sách tham số thực])

_ Số lượng tham số thực trong lời gọi hàm phải bằng số lượng tham số hình thức (trong khai báo hàm)

_ Kiểu của các tham số thực phải tương ứng với kiểu của tham số hình thức

_ Khi gặp 1 lời gọi hàm tại 1 vị trí nào đó trong chương trình, máy sẽ dời vị trí đó chuyển đến thực hiện các lệnh của hàm được gọi

_Thứ tự thực hiện khi có 1 lời gọi hàm

• Cấp phát bộ nhớ cho các biến cục bộ

• Gán giá trị của tham số thực sự cho tham số hình thức

• Thực hiện các lệnh trong thân của hàm

• Gặp lệnh return hoặc dấu } kết thúc hàm thì xóa vùng nhớ đã cấp cho các biến cục bộ và rời khỏi hàm -> trở về vị trí đã dừng sau lời gọi hàm.

•Nếu thoát khỏi hàm từ câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

3. Ví dụ minh họa

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

```
int tong(int a, int b) {
    return a + b;
}

int main() {
    printf("Tổng: %d\n", tong(5, 3));
    return 0;
}
```

II. Con trỏ

1. Khái niệm

_ Trong ngôn ngữ lập trình C, con trỏ (pointer) là một biến đặc biệt dùng để lưu trữ địa chỉ của một biến khác trong bộ nhớ.

_ Mỗi biến trong C đều được lưu trữ tại một địa chỉ cụ thể trong bộ nhớ.

_Con trỏ là biến chứa địa chỉ của biến đó, chứ không phải giá trị thực của nó.

2.Khai báo con trỏ

_ Cú pháp:

```
<kiểu dữ liệu> *<tên_con_trỏ>;
```

3. Ví dụ minh họa

```
#include <stdio.h>
int main() {
    int x = 10;
    int *ptr = &x;

    printf("Giá trị: %d\n", *ptr);  // Truy cập giá trị qua con trỏ

    printf("Địa chỉ: %p\n", ptr);   // In địa chỉ
    return 0;

}
```

III. Con trỏ mảng

1.Mối quan hệ giữa con trỏ và mảng

_ Là tập hợp các phần tử cùng kiểu dữ liệu, được lưu trữ liên tiếp trong bộ nhớ.

_ Con trỏ à biến dùng để lưu địa chỉ của một biến khác.

Mối quan hệ:

_ Tên mảng thực chất là một con trỏ hằng (không thay đổi được), trỏ đến phần tử đầu tiên của mảng.

_Do đó, bạn có thể dùng con trỏ để truy cập các phần tử của mảng.

2. Ví dụ minh họa về con trỏ mảng

```
#include <stdio.h>
int main() {
int a[5] = {10, 20, 30, 40, 50};
int (*p)[5] = &a;
printf("Phan tu thu 3: %d\n", (*p)[2]);
for (int i = 0; i < 5; i++) {
        printf("a[%d] = %d\n", i, (*p)[i]); }
 return 0;
}
```

IV. Mảng con trỏ

1. Khái niệm

Trong C, mảng con trỏ (pointer array) là một khái niệm liên quan đến việc sử dụng con trỏ để tạo và quản lý mảng. Cụ thể, mảng con trỏ là một mảng mà mỗi phần tử trong mảng là một con trỏ trỏ đến một giá trị hoặc một đối tượng khác, thay vì lưu trữ giá trị trực tiếp.

Mảng con trỏ là một mảng mà mỗi phần tử trong mảng chứa địa chỉ (pointer) trỏ tới một biến hoặc một đối tượng. Điều này cho phép bạn làm việc với các đối tượng động (như chuỗi ký tự, mảng động, hoặc các đối tượng có kích thước thay đổi trong thời gian thực) một cách linh hoạt hơn.

2.Cách khai báo mảng con trỏ trong C

_ Để khai báo một mảng con trỏ, sử dụng cú pháp sau:

type *array_name[size];

3. Ví dụ minh họa

```
#include <stdio.h>


int main() {
    const char *ds[] = {"apple", "banana", "cherry"};
    for (int i = 0; i < 3; i++) {
        printf("%s\n", ds[i]);
    }
    return 0;
}
```

V. Con trỏ hàm

1. Khái niệm

Khái niệm con trỏ hàm trong C là một con trỏ (pointer) dùng để lưu trữ địa chỉ của một hàm thay vì lưu trữ địa chỉ của một biến hoặc một mảng. Điều này cho phép chúng ta gọi hàm thông qua con trỏ, thay vì gọi trực tiếp tên hàm.

2. Khai báo con trỏ hàm

_ Cách khai báo:

```
int (*func_ptr)(int, int);
```

3. Gán địa chỉ hàm cho con trỏ hàm

_Để gán hàm add cho con trỏ hàm func_ptr, ta làm như sau:

```
func_ptr = add;
```

4. Sử dụng con trỏ hàm

_Để gọi hàm thông qua con trỏ, bạn sử dụng cú pháp sau:

```
int result = func_ptr(3, 4);  // Tương đương với add(3, 4)


printf("Kết quả: %d\n", result);
```

5.Ví dụ minh họa

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

```
int cong(int a, int b) { return a + b; }

int main() {
    int (*funcPtr)(int, int) = cong;
    printf("Kết quả: %d\n", funcPtr(2, 3));
    return 0;
}
```

VI. Cấp phát động

1. Khái niệm

Cấp phát động (dynamic memory allocation) là kỹ thuật trong lập trình C cho phép bạn cấp phát bộ nhớ trong thời gian chạy chương trình, thay vì xác định kích thước bộ nhớ tại thời điểm biên dịch.

2. Các hàm cấp phát động trong C

Trong ngôn ngữ lập trình C, cấp phát động (dynamic memory allocation) cho phép chúng ta cấp phát bộ nhớ trong lúc chương trình đang chạy, thay vì phải khai báo kích thước cố định trước.

Các hàm cấp phát động trong C đều được định nghĩa trong thư viện <stdlib.h>.

3. Ví dụ minh họa

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

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

    arr = (int *)malloc(n * sizeof(int));
    if (arr == NULL) return 1;

    for (int i = 0; i < n; i++) {
```

```
        arr[i] = i * 2;
        printf("%d ", arr[i]);
    }

    free(arr);
    return 0;
}
```

VII. Tệp

1. Khái niệm tệp trong C

Trong ngôn ngữ lập trình C, tệp (file) là một cấu trúc dữ liệu được sử dụng để lưu trữ thông tin lâu dài trên bộ nhớ ngoài (như ổ cứng), cho phép chương trình đọc và ghi dữ liệu một cách có tổ chức thông qua các hàm xử lý tệp trong thư viện chuẩn.

2. Các bước làm việc

Khi xử lý tệp trong C, bạn thường thực hiện theo 4 bước:

-Mở tệp (sử dụng fopen( ))

-Xử lý tệp (đọc/ghi dữ liệu)

-Đóng tệp (sử dụng fclose( ))

-(Tùy chọn) Kiểm tra lỗi

3. Các chế độ mở tệp (modes)

Khi làm việc với tệp trong ngôn ngữ lập trình C, ta sử dụng hàm fopen() để mở tệp. Khi mở, cần chỉ định chế độ (mode) để xác định tệp sẽ được đọc, ghi, hay cả hai, và tệp có bị ghi đè hay không.

Các chế độ mở tệp thường dùng:

Chế độ	Mô tả
"r"	Mở tệp để đọc. Tệp phải tồn tại, nếu không sẽ lỗi.
"w"	Mở tệp để ghi. Nếu tệp tồn tại, sẽ xóa trắng nội dung cũ. Nếu chưa có, sẽ tạo mới.
"a"	Mở tệp để ghi nối vào cuối. Nếu chưa có tệp, sẽ tạo mới. Không xóa nội dung cũ.
"r+"	Mở tệp để đọc và ghi. Tệp phải tồn tại.
"w+"	Mở tệp để đọc và ghi. Nếu có tệp, sẽ xóa trắng nội dung; nếu không, sẽ tạo mới.
"a+"	Mở tệp để đọc và ghi nối vào cuối. Nếu không có, sẽ tạo mới.

4. Các thao tác với tệp

_ Mở/đóng tệp:

```
FILE *fopen(const char *filename, const char *mode);
 int fclose(FILE *stream);
```

_ Đọc dữ liệu:

fgetc(FILE *stream) – đọc 1 ký tự

fgets(char *str, int n, FILE *stream) – đọc 1 dòng

fread() – đọc khối dữ liệu nhị phân

_ Ghi dữ liệu:

fputc(int char, FILE *stream) – ghi 1 ký tự

fputs(const char *str, FILE *stream) – ghi 1 chuỗi

fwrite() – ghi khối dữ liệu nhị phân

5. Kiểm tra lỗi và kiểm soát con trỏ tệp

feof(FILE *stream) – kiểm tra kết thúc tệp

ferror(FILE *stream) – kiểm tra lỗi

rewind(FILE *stream) – đưa con trỏ về đầu tệp

fseek() và ftell() – di chuyển và xác định vị trí con trỏ tệp

6. Tệp văn bản vs. tệp nhị phân

Trong C, tệp (file) có thể được chia làm 2 loại chính:

Tiêu chí	Tệp văn bản (Text File)	Tệp nhị phân (Binary File)
Định dạng lưu trữ	Dữ liệu lưu dưới dạng ký tự ASCII (có thể đọc được bằng Notepad).	Dữ liệu lưu dưới dạng nhị phân (mã máy), không thể đọc trực tiếp.
Kích thước tệp	Thường lớn hơn, do dữ liệu được chuyển sang dạng ký tự.	Nhỏ gọn hơn, vì lưu trực tiếp dưới dạng nhị phân.
Tốc độ truy xuất	Chậm hơn, do phải chuyển đổi giữa ký tự và dữ liệu.	Nhanh hơn, vì làm việc trực tiếp với bộ nhớ.
Dễ đọc thủ công	Có – mở bằng Notepad, VS Code,… để xem nội dung.	Không – mở bằng phần mềm sẽ thấy ký tự lạ.

Tiêu chí	Tệp văn bản (Text File)	Tệp nhị phân (Binary File)
Cách mở tệp	Dùng mode: "r", "w", "a", "r+", "w+", "a+"	Dùng mode: "rb", "wb", "ab", "rb+", "wb+", "ab+"
Hàm ghi/đọc	fprintf(), fscanf(), fgets(), fputs()	fread(), fwrite()

7. Ứng dụng

_Lưu trữ kết quả chương trình

_Đọc dữ liệu cấu hình từ file

_Quản lý danh sách, cơ sở dữ liệu đơn giản

_Ghi log, thống kê

VIII. Kiểu cấu trúc

1. Khái niệm

Trong ngôn ngữ lập trình C, kiểu cấu trúc (struct) là một kiểu dữ liệu do người dùng định nghĩa cho phép nhóm nhiều biến có các kiểu dữ liệu khác nhau lại thành một đơn vị duy nhất. Kiểu cấu trúc rất hữu ích khi bạn muốn biểu diễn một thực thể trong thế giới thực có nhiều thuộc tính khác nhau, ví dụ như một sinh viên có mã số, tên, điểm trung bình,…

2. Khái báo kiểu cấu trúc

_ Cú pháp cơ bản:

```
struct TênKiểuCấuTrúc {

   kiểu_dữ_liệu thành_phan_1;

   kiểu_dữ_liệu thành_phan_2;

   ...
```

```
};
```

_ Ví dụ:

```
struct SinhVien {
    int maSV;
    char ten[50];
    float diemTB;
};
```

3. Khai báo biến kiểu cấu trúc

Có hai cách khai báo biến kiểu cấu trúc:

Cách 1: Khai báo sau khi định nghĩa cấu trúc

```
struct SinhVien sv1, sv2;
```

Cách 2: Khai báo biến ngay khi định nghĩa cấu trúc

```
struct SinhVien {
    int maSV;
    char ten[50];
```

```
    float diemTB;

} sv1, sv2;
```

4. Truy cập thành phần của cấu trúc

Dùng dấu chấm (.) để truy cập các thành phần:

```
    sv1.maSV = 123;

    strcpy(sv1.ten, "Nguyen Van A");

    sv1.diemTB = 8.5;
```

Nếu dùng con trỏ đến cấu trúc, dùng dấu ->:

```
    struct SinhVien *ptr = &sv1;

     ptr->maSV = 124;
```

5. Lồng cấu trúc

Cấu trúc có thể chứa một cấu trúc khác:

6. Sử dụng từ khóa typedef để đơn giản hóa

7. Sao chép và so sánh cấu trúc

_ Gán một biến cấu trúc cho biến khác:

```
SinhVien sv2 = sv1; // Sao chép nội dung sv1 sang sv2
```

_Tuy nhiên, không thể so sánh trực tiếp hai cấu trúc bằng toán tử ==. Phải so sánh từng trường một hoặc dùng hàm memcmp.

IX. Danh sách liên kết

1. Khái niệm danh sách liên

Danh sách liên kết (Linked List) một cấu trúc dữ liệu bao gồm các phần tử được gọi là nút (node), mỗi nút chứa dữ liệu và một con trỏ trỏ đến nút tiếp theo trong danh sách. Các nút này được cấp phát bộ nhớ động và liên kết với nhau bằng con trỏ, cho phép chèn, xóa phần tử linh hoạt mà không cần dịch chuyển các phần tử khác.

2.Khai báo cấu trúc danh sách liên kết đơn trong C:

```
struct Node {

int data; // Dữ liệu của nút

struct Node* next; // Con trỏ trỏ đến nút tiếp theo
};
```

3. Tạo ra một nút mới.

Trong danh sách liên kết đơn, mỗi phần tử được gọi là nút (node). Để làm việc với danh sách, bước đầu tiên là tạo ra một nút mới.

Khi tạo một nút mới, ta cần:

_ Cấp phát bộ nhớ cho nút bằng cách sử dụng hàm malloc trong thư viện stdlib.h.

_Gán giá trị dữ liệu cho trường data của nút.

_Khởi tạo con trỏ next trỏ tới NULL (vì nút mới chưa liên kết với nút nào khác).

Mục đích:

_Tạo nút để có thể chèn vào đầu, cuối hoặc giữa danh sách liên kết.

_Là bước cơ bản để xây dựng và xử lý danh sách liên kết trong C.

Việc tạo nút mới là phần quan trọng trong việc thao tác và quản lý cấu trúc dữ liệu danh sách liên kết.

4. Thêm phần tử vào danh sách

Trong danh sách liên kết đơn, để thêm (chèn) một phần tử mới, ta có thể thực hiện ở các vị trí:

_Thêm vào đầu danh sách (Insert at Head)

Ví dụ:

_ Thêm vào cuối danh sách (Insert at Tail)

Ví dụ:

5. In danh sách

Để in danh sách liên kết, ta duyệt từng nút từ đầu danh sách (head) cho đến khi gặp NULL, và in ra dữ liệu của từng nút.

_ Hàm in danh sách:

```
void printList(struct Node* head) {
        struct Node* temp = head;
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
}
```

Ví dụ:

```
int main() {
        struct Node* head = NULL;

                                                                insertAt
Head(&head, 30);

                                                                insertAt
Head(&head, 20);

                                                                insertAt
Head(&head, 10);

                                                                printList
(head); // Kết quả: 10 -> 20 -> 30 -> NULL
        return 0;
}
```

6. Xóa phần tử đầu danh sách

Để xóa phần tử đầu của danh sách liên kết đơn, ta làm các bước sau:

_Cập nhật head: Đưa con trỏ head trỏ đến nút kế tiếp của nút đầu.

_Giải phóng bộ nhớ: Dùng free để giải phóng bộ nhớ của nút đầu.

Ví dụ:

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

// Khai báo cấu trúc nút
struct Node {
int data;
```

```
struct Node* next;
};

// Hàm tạo một nút mới
structNode*createNode(int                                                    value){
        struct Node* newNode = (struct
        Node*)malloc(sizeof(struct Node));
        newNode->data = value;
        newNode->next = NULL;
        return newNode;
}

// Hàm thêm vào đầu danh sách
void insertAtHead(struct Node** head, int value) {
        struct Node* newNode = createNode(value);
        newNode->next = *head;
        *head = newNode;
}

// Hàm xóa phần tử đầu danh sách
void deleteAtHead(struct Node** head) {
        if (*head == NULL) {
        printf("Danh sach rong!\n");
        return;
        }
        struct Node* temp = *head;
                                                                        *head =
(*head)->next;
        free(temp);
}


// Hàm in danh sách
void printList(struct Node* head) {
   struct Node* temp = head;
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}
```

```
int main() {
    struct Node* head = NULL;
    insertAtHead(&head, 10);
    insertAtHead(&head, 20);
    insertAtHead(&head, 30);
    printf("Danh sach ban dau: ");
    printList(head);  // Kết quả: 30 -> 20 -> 10 -> NULL
    deleteAtHead(&head);  // Xóa phần tử đầu tiên (30)
    printf("Danh sach sau khi xoa phan tu dau: ");
    printList(head);  // Kết quả: 20 -> 10 -> NULL
    return 0;
}
```

7. Xóa phần tử theo giá trị

Để xóa phần tử theo giá trị trong danh sách liên kết đơn, ta cần thực hiện các bước sau:

_Duyệt qua danh sách: Tìm phần tử có giá trị cần xóa.

_Xử lý các trường hợp:

+Nếu phần tử cần xóa là phần tử đầu (tức là head).

+ Nếu phần tử cần xóa không phải là phần tử đầu, cần cập nhật con trỏ next của nút trước nó.

_Giải phóng bộ nhớ: Sau khi xóa, cần giải phóng bộ nhớ của phần tử đó.

Ví dụ:

```
#include <stdio.h>

#include <stdlib.h>
```

```
// Khai báo cấu trúc nút
struct Node {
    int data;
    struct Node* next;
};

// Hàm tạo một nút mới
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// Hàm thêm vào đầu danh sách
```

```
void insertAtHead(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *head;
    *head = newNode;
}


// Hàm in danh sách
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

```
// Hàm xóa phần tử theo giá trị

void deleteNode(struct Node** head, int key) {

    struct Node* temp = *head;

    struct Node* prev = NULL;



    // Nếu phần tử cần xóa là phần tử đầu
```

PHẦN B – ỨNG DỤNG

- Xây dựng ứng dụng cho bài toán cụ thể với đầy đủ tính năng cần thiết
- Ứng dụng cho bài quản lý sinh viên.

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

// === Cấu trúc Sinh viên ===
typedef struct {
    char msv[20];
    char hoTen[50];
    int namSinh;
    char lop[20];
    float dtb;
    char diaChi[100];
} SinhVien;

// === Cấu trúc danh sách liên kết để ghi chú thao tác ===
typedef struct Node {
    char ghiChu[100];
    struct Node *next;
} Node;

// ==== Hàm menu ====
void menu() {
    printf("\n====== MENU ======\n");
    printf("1. Nhập danh sách sinh viên\n");
    printf("2. Hiển thị danh sách sinh viên\n");
    printf("3. Ghi danh sách vào file\n");
```

```
    printf("4. Sắp xếp theo họ tên (A-Z)\n");
    printf("5. Sắp xếp theo điểm TB (giảm dần)\n");
    printf("6. Hiển thị ghi chú thao tác\n");
    printf("0. Thoát\n");
    printf("==================\n");
}

// ==== Hàm thêm ghi chú vào danh sách liên kết ====
void themGhiChu(Node **head, const char *noiDung) {
    Node *newNode = (Node*)malloc(sizeof(Node));
    strcpy(newNode->ghiChu, noiDung);
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
    } else {
        Node *p = *head;
        while (p->next != NULL) p = p->next;
        p->next = newNode;
    }
}

// ==== Hiển thị ghi chú thao tác ====
void hienThiGhiChu(Node *head) {
    printf("\n--- Ghi chú thao tác ---\n");
    int i = 1;
    while (head != NULL) {
        printf("%d. %s\n", i++, head->ghiChu);
        head = head->next;
    }
```

```
}

// ==== Nhập danh sách sinh viên ====
void nhapDanhSach(SinhVien **ds, int *n) {
    printf("Nhập số lượng sinh viên: ");
    scanf("%d", n);
    getchar();

    *ds = (SinhVien*)malloc((*n) * sizeof(SinhVien));
    if (*ds == NULL) {
        printf("Lỗi cấp phát bộ nhớ!\n");
        exit(1);
    }

    for (int i = 0; i < *n; i++) {
        printf("\n--- Sinh viên %d ---\n", i + 1);

        printf("Mã SV: "); fgets((*ds)[i].msv, 20, stdin); (*ds)[i].msv[strcspn((*ds)[i].msv,
"\n")] = '\0';
        printf("Họ tên: "); fgets((*ds)[i].hoTen, 50, stdin);
(*ds)[i].hoTen[strcspn((*ds)[i].hoTen, "\n")] = '\0';
        printf("Năm sinh: "); scanf("%d", &(*ds)[i].namSinh); getchar();
        printf("Lớp: "); fgets((*ds)[i].lop, 20, stdin); (*ds)[i].lop[strcspn((*ds)[i].lop, "\n")] =
'\0';
        printf("Điểm TB: "); scanf("%f", &(*ds)[i].dtb); getchar();
        printf("Địa chỉ: "); fgets((*ds)[i].diaChi, 100, stdin);
(*ds)[i].diaChi[strcspn((*ds)[i].diaChi, "\n")] = '\0';
    }
}
```

```
// ==== Hiển thị danh sách sinh viên ====
void hienThiDanhSach(SinhVien *ds, int n) {
    printf("\n--- Danh sách sinh viên ---\n");
    for (int i = 0; i < n; i++) {
        printf("STT %d\n", i + 1);
        printf("MSV    : %s\n", ds[i].msv);
        printf("Họ tên : %s\n", ds[i].hoTen);
        printf("Năm sinh: %d\n", ds[i].namSinh);
        printf("Lớp    : %s\n", ds[i].lop);
        printf("Điểm TB: %.2f\n", ds[i].dtb);
        printf("Địa chỉ: %s\n", ds[i].diaChi);
        printf("--------------------------\n");
    }
}

// ==== Ghi danh sách ra file ====
void ghiFile(SinhVien *ds, int n, const char *tenTep) {
    FILE *f = fopen(tenTep, "w");
    if (f == NULL) {
        printf("Không thể mở file để ghi!\n");
        return;
    }

    for (int i = 0; i < n; i++) {
        fprintf(f, "STT %d\n", i + 1);
        fprintf(f, "%s\n", ds[i].msv);
        fprintf(f, "%s\n", ds[i].hoTen);
        fprintf(f, "%d\n", ds[i].namSinh);
        fprintf(f, "%s\n", ds[i].lop);
        fprintf(f, "%.2f\n", ds[i].dtb);
```

```
        fprintf(f, "%s\n", ds[i].diaChi);
    }

    fclose(f);
    printf("Đã ghi danh sách vào file '%s'\n", tenTep);
}

// ==== Hàm so sánh theo tên dùng cho con trỏ hàm ====
int soSanhTen(SinhVien a, SinhVien b) {
    return strcmp(a.hoTen, b.hoTen);
}

// ==== Hàm so sánh theo điểm TB ====
int soSanhDTB(SinhVien a, SinhVien b) {
    return b.dtb > a.dtb ? 1 : -1;
}

// ==== Hàm sắp xếp dùng con trỏ hàm ====
void sapXep(SinhVien *ds, int n, int (*cmp)(SinhVien, SinhVien)) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (cmp(ds[i], ds[j]) > 0) {
                SinhVien tmp = ds[i];
                ds[i] = ds[j];
                ds[j] = tmp;
            }
        }
    }
}
```

```
int main() {
    SinhVien *ds = NULL;
    int n = 0;
    Node *ghiChu = NULL;
    int luaChon;

    do {
        menu();
        printf("Chọn: ");
        scanf("%d", &luaChon);
        getchar();

        switch (luaChon) {
            case 1:
                nhapDanhSach(&ds, &n);
                themGhiChu(&ghiChu, "Đã nhập danh sách sinh viên.");
                break;
            case 2:
                hienThiDanhSach(ds, n);
                break;
            case 3:
                ghiFile(ds, n, "sinhvien.txt");
                themGhiChu(&ghiChu, "Đã ghi danh sách ra file.");
                break;
            case 4:
                sapXep(ds, n, soSanhTen);
                hienThiDanhSach(ds, n);
                themGhiChu(&ghiChu, "Đã sắp xếp theo họ tên.");
                break;
            case 5:
```

```
            sapXep(ds, n, soSanhDTB);
            hienThiDanhSach(ds, n);
            themGhiChu(&ghiChu, "Đã sắp xếp theo điểm trung bình.");
            break;
        case 6:
            hienThiGhiChu(ghiChu);
            break;
        case 0:
            printf("Kết thúc chương trình.\n");
            break;
        default:
            printf("Lựa chọn không hợp lệ!\n");
    }
  } while (luaChon != 0);

  // Giải phóng bộ nhớ
  free(ds);
  while (ghiChu != NULL) {
    Node *tmp = ghiChu;
    ghiChu = ghiChu->next;
    free(tmp);
  }

  return 0;
}
```

KẾT LUẬN

Trong quá trình thực hiện, chương trình Quản lý Công việc được phát triển bằng ngôn ngữ C đã phần nào đáp ứng được những yêu cầu cơ bản của một ứng dụng quản lý công việc. Việc tích hợp danh sách liên kết cùng với các kỹ năng lập trình khác giúp chương trình thực hiện được các chức năng thiết yếu.

Do thời gian nghiên cứu và thực hành còn hạn chế nên một số chức năng nâng cao vẫn chưa được xây dựng và chương trình vẫn còn tồn tại một số hạn chế nhất định. Em sẽ cố gắng nghiên cứu, cải tiến và bổ sung thêm các tính năng để chương trình hoàn thiện hơn.

Em rất mong nhận được sự góp ý, đánh giá từ thầy cô và bạn bè để có thể hoàn thiện bài báo cáo cũng như chương trình một cách tốt nhất.