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

ĐỀ TÀI: LÝ THUYẾT NGÔN NGỮ C

Giảng viên hướng dẫn: Trần Phong Nhã

Sinh viên thực hiện: HÀ TIÊN TIẾN

Lớp : CQ.65.CNTT

Khoá : 65

Tp. Hồ Chí Minh, tháng 05 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

ĐỀ TÀI: LÝ THUYẾT NGÔN NGỮ C

Giảng viên hướng dẫn: Trần Phong Nhã

Sinh viên thực hiện: HÀ TIÊN TIẾN

Lớp : CQ.65.CNTT

Khoá : 65

Tp. Hồ Chí Minh, tháng 05 năm 2025

LỜI CẢM ƠN

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

Em xin chân thành cảm ơn quý thầy cô đã giúp đỡ tạo điều kiện để em hoàn thành báo cáo với đề tài "**Lý thuyết ngôn ngữ C**". Đặc biệt em xin cảm ơn thầy Trần Phong Nhã đã nhiệt tình giúp đỡ, hướng dẫn cho em kiến thức, định hướng và kỹ năng để có thể hoàn thành bài 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, em xin gửi lời chúc tới Quý Thầy Cô Bộ môn Công nghệ thông tin và hơn hết là thầy Trần Phong Nhã có thật nhiều sức khỏe, có nhiều thành công trong công việc. Em xin chân thành cảm ơn!

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

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

Giáo viên hướng dẫn

Trần Phong Nhã

MỤC LỤC

CHƯƠNG 1: MỞ ĐẦU

1.1 Giới thiệu sơ lược về ngôn ngữ C

-Ngôn ngữ lập trình C là một ngôn ngữ lập trình cấp trung, được phát triển bởi Dennis Ritchie vào năm 1972 tại Bell Labs. Ban đầu, C được thiết kế để viết hệ điều hành UNIX, và sau đó đã nhanh chóng trở thành một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới. C được đánh giá cao nhờ vào tính linh hoạt, hiệu năng cao và khả năng làm việc gần gũi với phần cứng, điều này giúp nó trở thành lựa chọn lý tưởng cho phát triển hệ điều hành, phần mềm hệ thống và ứng dụng yêu cầu tối ưu hóa về hiệu suất.

-Một trong những điểm mạnh của C là khả năng kiểm soát bộ nhớ qua con trỏ (pointer), cho phép lập trình viên thao tác trực tiếp với vùng nhớ của máy tính. Cũng nhờ vào cú pháp đơn giản và dễ hiểu, C dễ học và sử dụng, mặc dù nó yêu cầu lập trình viên phải chú ý nhiều đến các chi tiết nhỏ, như việc quản lý bộ nhớ và xử lý lỗi.

-Cũng chính vì tính gần gũi với phần cứng và khả năng tối ưu hóa mã nguồn, C được sử dụng rộng rãi trong các lĩnh vực như lập trình hệ thống nhúng, phát triển các phần mềm cần hiệu suất cao như game, máy chủ web, và nhiều ứng dụng khác. Ngoài ra, C còn là nền tảng để phát triển các ngôn ngữ lập trình hiện đại khác như C++, Java và C#.

-Nhờ những ưu điểm vượt trội, ngôn ngữ C không chỉ giữ được vị thế quan trọng trong ngành công nghiệp phần mềm mà còn là một phần không thể thiếu trong chương trình học của nhiều sinh viên ngành công nghệ thông tin.

1.2 Mục tiêu của báo cáo.

-Báo cáo này nhằm tìm hiểu và phân tích các khái niệm quan trọng trong ngôn ngữ lập trình C, đặc biệt là những tính năng nâng cao như con trỏ, cấp phát động, danh sách liên kết, và các cấu trúc dữ liệu cơ bản. Mục đích của báo cáo là cung cấp một cái nhìn toàn diện về cách sử dụng các tính năng này trong lập trình, từ đó giúp người học phát triển kỹ năng lập trình và ứng dụng các kiến thức đã học vào các bài toán thực tế.

-Cụ thể, báo cáo sẽ làm rõ cách thức hoạt động của con trỏ trong C, bao gồm việc sử dụng toán tử `*` và `&`, cũng như sự khác biệt giữa con trỏ mảng và mảng con trỏ. Ngoài ra, báo cáo sẽ trình bày chi tiết về các kỹ thuật cấp phát động với các hàm như `malloc`, `calloc`, `realloc`, và `free`, giúp quản lý bộ nhớ hiệu quả hơn trong các ứng dụng phức

tạp. Một phần quan trọng của báo cáo là danh sách liên kết, cấu trúc dữ liệu phổ biến giúp lưu trữ và quản lý dữ liệu linh hoạt, dễ dàng mở rộng mà không cần tái cấp phát bộ nhớ.

-Mục tiêu cuối cùng là giúp người học nắm vững các khái niệm này để có thể áp dụng chúng vào các chương trình thực tế, từ việc xử lý dữ liệu đến tối ưu hiệu suất của các ứng dụng. Các khái niệm sẽ được minh họa qua ví dụ cụ thể, đồng thời phân tích các lỗi thường gặp khi sử dụng các tính năng này trong lập trình.

1.3 Vai trò của ngôn ngữ C

-Ngôn ngữ lập trình C đóng vai trò nền tảng trong lĩnh vực khoa học máy tính và kỹ thuật phần mềm. Với tính linh hoạt, khả năng truy cập bộ nhớ trực tiếp và hiệu năng cao, C là lựa chọn ưu tiên cho các ứng dụng cần tối ưu hóa như hệ điều hành, phần mềm nhúng và trình điều khiển thiết bị. Vì vậy, C thường được xem là "ngôn ngữ của hệ thống".

-C cũng là bước đệm quan trọng để tiếp cận các ngôn ngữ hiện đại như C++, Java hay Python, do nhiều nguyên lý lập trình trong các ngôn ngữ này đều kế thừa từ C. Việc học và nắm vững C giúp người học phát triển tư duy thuật toán, hiểu cơ chế hoạt động của máy tính, từ đó viết mã hiệu quả và tối ưu tài nguyên.

-Trong giáo dục, C thường là ngôn ngữ đầu tiên được giảng dạy nhằm xây dựng nền tảng lập trình vững chắc. Nhờ đó, sinh viên có thể dễ dàng tiếp cận các chủ đề phức tạp như cấu trúc dữ liệu, thuật toán và lập trình hệ thống.

CHƯƠNG 2: CÁC KHÁI NIỆM CƠ BẢN TRONGNGÔN NGỮ C

2.1 Hàm trong C

- Một hàm là một nhóm các lệnh đi cùng nhau để thực hiện một nhiệm vụ. Mỗi chương trình C có ít nhất một hàm là hàm **main()**, và tất cả hầu hết các chương trình bình thường đều định nghĩa thêm các hàm.

- Bạn có thể chia đoạn code của bạn thành những hàm riêng biệt. Cách bạn chia đoạn code của bạn thành các hàm khác nhau phụ thuộc vào bạn, nhưng theo tính logic, một hàm thường có một nhiệm vụ nhất định.

- Một sự **khai báo** hàm thông báo với bộ biên dịch về tên của hàm, kiểu trả về và tham số. Một **định nghĩa** hàm cung cấp phần thân của một hàm.

- Các thư viện tiêu chuẩn của ngôn ngữ C cung cấp rất nhiều hàm có sẵn để chương trình của bạn có thể gọi. Ví dụ, hàm **strcat()** có thể nối hai đoạn chuỗi, hàm **memcpy()** dùng để copy một vùng nhớ đến một vùng nhớ khác và rất nhiều hàm khác nữa.

- Một hàm được biết đến với các tên khác nhau như một phương thức, một tuyến phụ hoặc một thủ tục.

- Cú pháp:

```
<kiểu_trả_về> <tên_hàm>(<danh_sách_tham_số>) {
    // Các câu lệnh trong thân hàm
```

-Ví dụ:

```
// Hàm tính tổng có giá trị trả về, nhận 2 tham số a và b
int tinhTong1(int a, int b) {
```

```
// Hàm tính tổng không trả về giá trị và không nhận tham số
void tinhTong2() {
    int a, b;  // Khai báo hai biến nguyên a và b
    printf("Nhap a: "); scanf("%d", &a);
    printf("Nhap b: "); scanf("%d", &b);
    printf("Tong hai so a va b la: %d", a + b);
```

2.2 Con trỏ.

2.2.1 Địa chỉ

- Trong ngôn ngữ C, bộ nhớ được chia thành các ô nhớ, mỗi ô lưu trữ một giá trị (có thể là một số nguyên, số thực, ký tự, hay các kiểu dữ liệu khác). Địa chỉ là vị trí của một ô nhớ trong bộ nhớ. Mỗi biến trong C có một địa chỉ bộ nhớ riêng biệt mà giá trị của biến đó được lưu trữ.
- Khi bạn khai báo một biến trong C, hệ thống sẽ cấp phát bộ nhớ cho biến đó, và bạn có thể lấy địa chỉ của biến bằng cách sử dụng toán tử &.

Ví dụ:

```
int a = 10; // Khai báo một biến int
printf("Giá trị của a: %d\n", a);          // In giá trị của a
printf("Địa chỉ của a: %p\n", &a);      // In địa chỉ của a (lưu ý: %p dùng để in địa chỉ)
```

->Trong ví dụ trên, a là tên của biến, và &a là địa chỉ bộ nhớ nơi giá trị của a được lưu trữ. Lưu ý rằng %p trong printf là định dạng dùng để in địa chỉ bộ nhớ.

2.2.2 Toán tử **&:**

Toán tử & trong C có hai mục đích sử dụng:

- *Lấy địa chỉ của một biến*: Khi bạn đặt & trước tên một biến, bạn sẽ nhận được địa chỉ bộ nhớ của biến đó.

- Ví dụ:

```
int x = 5;
int *ptr = &x; // ptr sẽ chứa địa chỉ của biến x
```

->Trong ví dụ trên, &x lấy địa chỉ của biến x, và gán nó cho con trỏ ptr.

- *Lấy địa chỉ của đối tượng trong bộ nhớ*: Đây là cách để bạn làm việc với con trỏ, cho phép bạn gián tiếp truy cập và thay đổi giá trị của các biến thông qua địa chỉ của chúng.

- Ví dụ:

```
int a = 10;
int *ptr = &a; // ptr trỏ tới địa chỉ của a
```

->Ở đây, &a trả về địa chỉ của biến a, và gán địa chỉ đó cho con trỏ ptr. Khi con trỏ ptr chứa địa chỉ của a, bạn có thể thay đổi giá trị của a thông qua con trỏ.

2.2.3 Toán tử *

a) Khai báo con trỏ (Pointer Declaration):

- Khi khai báo một con trỏ, bạn sử dụng toán tử * để chỉ ra rằng biến đó sẽ là một con trỏ, tức là nó sẽ lưu trữ địa chỉ của một biến thay vì giá trị trực tiếp.

- Ví dụ:

```
int *ptr; // ptr là một con trỏ kiểu int
```

->Trong trường hợp này, * cho biết rằng ptr không phải là một biến kiểu int, mà là **một con trỏ trỏ tới kiểu dữ liệu int.**

b) Dereferencing (Truy cập giá trị qua con trỏ):

- Khi con trỏ đã được gán một địa chỉ hợp lệ (chẳng hạn như địa chỉ của một biến), bạn có thể sử dụng toán tử * để **truy cập giá trị** tại địa chỉ đó. Đây gọi là **dereferencing** con trỏ.

-Ví dụ:

```
int x = 20;
int *ptr = &x; // ptr chứa địa chỉ của x
printf("Giá trị của x qua con trỏ ptr: %d\n", *ptr); // In giá trị
của x thông qua ptr
```

->Trong ví dụ này:

- ptr là con trỏ chứa địa chỉ của x.
- *ptr lấy **giá trị tại địa chỉ** mà ptr trỏ tới, tức là giá trị của x.
- Nếu không sử dụng toán tử *, con trỏ sẽ chỉ chứa địa chỉ của x, thay vì giá trị của x.

c) Ví dụ kết hợp **&** và *****

*- Một ví dụ thực tế về việc sử dụng cả & và * trong C là:*

```
#include <stdio.h>
void increment(int *p) {
   (*p)++; // Tăng giá trị tại địa chỉ mà con trỏ p trỏ tới
}
int main() {
   int a = 5;
   printf("Trước khi gọi hàm: a = %d\n", a); // In giá trị của a trước khi thay đổi
   increment(&a); // Gửi địa chỉ của a vào hàm increment
   printf("Sau khi gọi hàm: a = %d\n", a); // In giá trị của a sau khi thay đổi
   return 0;
}
```

->Trong ví dụ này:

- Trong hàm increment, p là con trỏ kiểu int *. Khi gọi increment(&a), chúng ta truyền địa chỉ của a cho con trỏ p.
- Sau đó, trong hàm, toán tử *p dùng để truy cập giá trị của a thông qua con trỏ p, và (*p)++ làm tăng giá trị của a.

2.3 Con trỏ mảng

- Khái niệm: Con trỏ mảng là một con trỏ dùng để trỏ tới toàn bộ mảng, tức là nó giữ địa chỉ của mảng chứ không phải chỉ địa chỉ phần tử đầu tiên.

- Cú pháp:

<Kiểu dữ liệu> `(*` **<tên biến con trỏ>** `)[<kích thước mảng>];`

-> Trong đó, p là con trỏ trỏ tới mảng có n phần tử kiểu int.

- Cách khai báo và sử dụng:

- Khai báo:

```
int a[5] = {1, 2, 3, 4, 5};
int (*p)[5] = &a;
```

- Truy cập phần tử:

```
printf("%d", (*p)[2]);  // In ra 3
```

- Duyệt mảng bằng con trỏ:

```
for (int i = 0; i < 5; i++) {
    printf("%d ", (*p)[i]);
}
```

- *Công dụng*:

- Dùng để truyền mảng 2 chiều vào hàm:

```
void inMaTran(int (*a)[3], int hang) {
    for (int i = 0; i < hang; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}
```

- Giữ đúng kích thước mảng khi làm việc trong hàm, tránh lỗi vượt giới hạn.
- Quản lý dữ liệu mảng lớn rõ ràng, đặc biệt khi cần xử lý toàn bộ mảng thay vì từng phần tử.

- *Lưu ý:*

- Phải dùng dấu () trong khai báo: int (*p)[n] (nếu không sẽ trở thành mảng con trỏ).
- Con trỏ mảng chỉ phù hợp khi kích thước mảng cố định, biết trước.

2.4 Mảng con trỏ

2.4.1 Khái niệm, cú pháp, cách khai báo và sử dụng

- *Khái niệm*:Mảng con trỏ là một mảng mà mỗi phần tử của nó là một con trỏ. Điều này cho phép mỗi phần tử trong mảng trỏ đến một vùng nhớ riêng biệt, thường được sử dụng để lưu trữ danh sách các chuỗi hoặc mảng có độ dài khác nhau.

- *Cú pháp:*

```
<kiểu dữ liệu> *<tên_mảng>[kích_thước];
```

- *Khai báo:*

```
char *dsTen[3]; // Mảng gồm 3 con trỏ kiểu char*
```

- *Sử dụng:*

- Lưu trữ danh sách các chuỗi: Mỗi con trỏ trong mảng có thể trỏ đến một chuỗi khác nhau.
- Truy cập phần tử: Sử dụng chỉ số mảng để truy cập từng con trỏ và sau đó dereference để lấy giá trị.
- Ví dụ:

```
#include <stdio.h>
int main() {
    char *dsTen[] = {"Nam", "Lan", "Huy"};
    for (int i = 0; i < 3; i++) {
        printf("Tên %d: %s\n", i + 1, dsTen[i]);
    }
    return 0;
}
```

- *Lưu ý:*

- Cần đảm bảo mỗi con trỏ trong mảng được gán địa chỉ hợp lệ trước khi sử dụng.
- Mảng con trỏ rất hữu ích khi làm việc với danh sách các chuỗi có độ dài khác nhau, giúp tiết kiệm bộ nhớ và linh hoạt trong xử lý dữ liệu.

2.4.2 Sự khác nhau giữa mảng con trỏ và con trỏ mảng

- *Định nghĩa:*
  - Con trỏ mảng: là con trỏ trỏ đến toàn bộ mảng.
  - Mảng con trỏ: là mảng mà mỗi phần tử là một con trỏ.
- *Cú pháp:*
  - Con trỏ mảng: <kiểu dữ liệu> (*<tên biến>)[kích thước];
  - Mảng con trỏ: <kiểu dữ liệu>* <tên mảng>[kích thước];
- *Ví dụ khai báo:*
  - Con trỏ mảng: int (*p)[5];
  - Mảng con trỏ: int *a[5];
- *Gán giá trị:*
  - Con trỏ mảng: p = &arr; // với arr là mảng int[5]
  - Mảng con trỏ: a[0] = &x; a[1] = &y; ...
- *Công dụng:*
  - Con trỏ mảng: dùng khi xử lý nguyên cả mảng cố định, truyền mảng vào hàm.
  - Mảng con trỏ: dùng khi cần quản lý nhiều vùng nhớ riêng biệt, như danh sách chuỗi.
- *Truy cập phần tử:*
  - Con trỏ mảng: (*p)[i]
  - Mảng con trỏ: *a[i] hoặc a[i][j]

2.5 Con trỏ hàm

- *Khái niệm*: Con trỏ hàm là một loại con trỏ dùng để lưu trữ địa chỉ của hàm. Khi một hàm được gọi thông qua con trỏ hàm, chương trình sẽ thực thi hàm mà con trỏ trỏ tới. Con trỏ hàm cho phép chương trình linh hoạt hơn trong việc gọi các hàm khác nhau tại thời điểm chạy, điều này rất hữu ích trong các tình huống như callback, xử lý sự kiện, hoặc khi cần thay đổi hành vi của một phần mềm.
- *Cú pháp:*

```
<kiểu trả về> (*<tên con trỏ>)(<danh sách tham số>);
```

- *Cách gán hàm cho con trỏ:*

```
<con trỏ hàm> = <tên hàm>;
```

- *Gọi hàm thông qua con trỏ:*

<con trỏ hàm>(<danh sách tham số>) || (*<con trỏ hàm>)(<danh sách tham số>);

- Ví dụ:

```
#include <stdio.h>
int cong(int a, int b) {
return a + b;
}
int tru(int a, int b) {
return a - b;
}
int main() {
    int (*thaoTac)(int, int);
        thaoTac = cong;
    printf("Cộng: %d\n", thaoTac(5, 3));
        thaoTac = tru;
    printf("Trừ: %d\n", thaoTac(5, 3));
    return 0;
```

->Trong ví dụ trên:

- int(*ptrHam)(int,int);
  → Khai báo con trỏ hàm ptrHam trỏ đến hàm có 2 tham số kiểu int, trả về int.
- ptrHam=cong;
  → Gán địa chỉ hàm cong cho con trỏ ptrHam.
- ptrHam(5,3);
  → Gọi hàm cong thông qua con trỏ.

=> Ưu điểm: Có thể thay đổi linh hoạt hàm thực thi bằng cách gán hàm khác vào con trỏ, không cần thay đổi mã gọi.

2.6 Cấp phát động

- *Khái niệm:* Cấp phát động là quá trình xin cấp phát bộ nhớ trong lúc chương trình

đang chạy (runtime) thay vì xác định trước kích thước bộ nhớ tại thời điểm biên dịch (compile-time).

- Khi nào cần cấp phát động:

- Khi bạn không biết trước số lượng phần tử cần lưu trữ.
- Khi muốn tiết kiệm bộ nhớ bằng cách chỉ cấp phát đúng khi cần thiết.
- Khi cần cấu trúc dữ liệu linh hoạt như: danh sách liên kết, cây, đồ thị,...

- Bộ nhớ được cấp phát nằm ở **vùng Heap** – là vùng bộ nhớ dùng để quản lý dữ liệu có vòng đời linh hoạt hơn so với stack.

- Cú pháp sử dụng:

- **Malloc()**: Cấp phát size byte bộ nhớ, chưa khởi tạo giá trị.

Cú pháp: ptr = (type*)malloc(size)

- **Calloc()**: Cấp phát bộ nhớ cho mảng n phần tử, mỗi phần tử size byte, được

Cú pháp: ptr = (type*)calloc(n, size)

  khởi tạo về 0**.**

- **Realloc():** Thay đổi kích thước vùng nhớ đã cấp phát (có thể đổi địa chỉ).

Cú pháp: ptr = (type*)realloc(ptr, new_size)

- **free():** Giải phóng bộ nhớ đã cấp phát, tránh rò rỉ bộ nhớ.

Cú pháp: free(ptr)

- Đối tượng áp dụng:

- **Mảng động**: khi kích thước mảng được nhập từ người dùng.
- **Con trỏ đến kiểu dữ liệu**: int *p = malloc(sizeof(int));
- **Cấu trúc (struct)**: tạo một struct động.

  Ví dụ: struct SV *s = malloc(sizeof(struct SV));

- **Cấu trúc dữ liệu động**: danh sách liên kết, cây, hàng đợi, ngăn xếp,...

-Ví dụ:

```
int *a;
int n;
printf("Nhap so luong: ");
scanf("%d", &n);
a = (int*)malloc(n * sizeof(int)); // cấp phát mảng động
for(int i = 0; i < n; i++) {
   printf("a[%d] = ", i);
   scanf("%d", &a[i]);
}
free(a); // giải phóng bộ nhớ
```

- Cấu trúc động:

```
typedef struct {
    char ten[30];
    float diem;
} SV;
SV *p = (SV*)malloc(sizeof(SV));
strcpy(p->ten, "Nguyen Van A");
p->diem = 8.5;
free(p);
```

- Một số lưu ý:

- Luôn kiểm tra con trỏ sau khi cấp phát:
- Nhớ giải phóng bộ nhớ bằng free() khi không còn dùng nữa → tránh memory leak.
- Không truy cập vào vùng nhớ đã free → gây lỗi nghiêm trọng.
- malloc() và realloc() không khởi tạo giá trị, còn calloc() khởi tạo toàn bộ bằng 0.
- realloc() có thể trả về địa chỉ khác → nên gán lại cho con trỏ.

2.7 Xử lí file

2.7.1 Khái quát về tệp trong C và cách khai báo, mở tệp

- Tệp (file) là nơi lưu trữ dữ liệu lâu dài trên đĩa cứng (HDD/SSD), không mất dữ liệu sau khi chương trình kết thúc.Trong C, để xử lý tệp, bạn sử dụng **con trỏ tệp** thuộc kiểu FILE * và các hàm có sẵn trong thư viện <stdio.h>.

- Khai báo và mở tệp:

- Khai báo đồng thời mở tệp:

```
<FILE *> <tên con trỏ> = fopen("<tên tệp>", "<chế độ>");
```

Ví dụ:

```
FILE *f = fopen("data.txt", "r")
```

- Khai báo và mở tệp riêng:

```
<FILE *> <tên_con_trỏ>;
<tên con trỏ> = = fopen("<tên tệp>", "<chế độ>");
```

->Kết luận:

- Khai báo đồng thời mở tệp: Nhanh gọn nhưng thiếu linh hoạt, dùng khi mở tệp đơn giản rõ ràng.
- Khai báo và mở tệp riêng: Linh hoạt, rõ ràng nhưng tốn dòng code hơn, dùng khi cần mở tệp theo điều kiện, dùng trong nhiều hàm.

2.7.2 Các mode khi mở file

-Mode " r ":

- Công dụng: Mở file để đọc.
- Lưu ý: Nếu file không tồn tại thì hàm fopen() trả về con trỏ NULL

- Mode " rb ":

- Công dụng: Mở file để đọc theo kiểu file nhị phân.
- Lưu ý: Nếu file không tồn tại thì hàm fopen() trả về con trỏ NULL

- Mode " w ":

- Công dụng: Mở file để ghi

- Lưu ý: Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới

- Mode " wb ":

- Công dụng: Mở file để ghi theo kiểu file nhị phân
- Lưu ý: Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới

- Mode " a ":

- Công dụng: Mở file text lên để ghi tiếp vào cuối file mà không xóa nội dung cũ trong file
- Lưu ý: Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới

- Mode " ab ":

- Công dụng: Mở file nhị phân lên để ghi tiếp vào cuối file mà không xóa nội dung cũ trong file
- Lưu ý: Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới

- Mode " r+ ":

- Công dụng: Mở file để vừa đọc vừa ghi
- Lưu ý: Nếu file không tồn tại thì hàm fopen() trả về con trỏ NULL

- Mode " rb+ ":

- Công dụng: Mở file để vừa đọc vừa ghi theo kiểu nhị phân
- Lưu ý: Nếu file không tồn tại thì hàm fopen() trả về con trỏ NULL

- Mode " w+ ":

- Công dụng: Mở file để vừa đọc vừa ghi
- Lưu ý: Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới

- Mode " wb+ ":

- Công dụng: Mở file để vừa đọc vừa ghi theo kiểu nhị phân

- Lưu ý: Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới

- *Mode " a+ ":*

- Công dụng: Mở file lên vừa để đọc và ghi vào cuối file
- Lưu ý: Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới

- *Mode " ab+ ":*

- Công dụng: Mở file để vừa đọc vừa ghi vào cuối file theo kiểu nhị phân
- Lưu ý: Nếu file đã tồn tại thì sẽ làm việc với file đó, nếu file chưa tồn tại sẽ tạo 1 file mới

2.7.3 Ghi dữ liệu vào tệp và đọc dữ liệu trong tệp

- Ghi dữ liệu vào tệp:

- Cú pháp:

```
fprintf(<tệp>, "<định dạng>", <giá trị>);
fputs("<chuỗi>", <tệp>);
fwrite(<con trỏ dữ liệu>, <kích thước phần tử>, <số phần tử>, <tệp>);
```

- Ví dụ:

```
fprintf(f, "Diem: %.2f\n", diem);
fputs("Hello\n", f);
fwrite(&sv, sizeof(SinhVien), 1, f);
```

- *Đọc dữ liệu từ tệp đã có:*

- Cú pháp:

```
fscanf(<tệp>, "<định dạng>", &<biến>);
fgets(<chuỗi>, <kích thước>, <tệp>);
fread(<con trỏ lưu dữ liệu>, <kích thước phần tử>, <số phần tử>, <tệp>);
```

- Ví dụ:

```
fprintf(f, "Diem: %.2f\n", diem);
fputs("Hello\n", f);
fwrite(&sv, sizeof(SinhVien), 1, f);
```

2.7.4 Đóng file

- *Cú pháp:* `fclose(<tên tệp>);`

- Ví dụ: `fclose(f);`

2.8 Kiểu cấu trúc

2.8.1 Khái niệm và cú pháp khai báo kiểu cấu trúc

- *Khái niệm:* **Cấu trúc** (struct) trong C là một kiểu dữ liệu cho phép kết hợp nhiều kiểu dữ liệu khác nhau lại với nhau dưới một tên duy nhất. Các thành phần trong một cấu trúc được gọi là **trường** (fields) hoặc **thành viên** (members)**.**

- *Cú pháp khai báo:*

```
//Không có từ khóa typedef
struct <tên cấu trúc> {
    <kiểu dữ liệu thành viên1>;
    <kiểu dữ liệu thành viên2>;
    ...
}<Danh sách các biến>;
```

Trong đó:

- **<tên cấu trúc>**: Tên của cấu trúc.
- **Các thành viên**: Các biến với kiểu dữ liệu khác nhau, ví dụ như int, float, char, ...
- **Lưu ý:** Khai báo như cách trên thì khi khai báo thêm biến thì cần từ khóa struct ở trước.

```
//Khai báo có từ khóa typedef
typedef struct <tên cấu trúc> {
    <kiểu dữ liệu thành viên1>;
    <kiểu dữ liệu thành viên2>;
    ...
}<Tên mới cho kiểu cấu trúc>;
```

Trong đó:

- **<tên cấu trúc>**: Tên của cấu trúc.
- **Các thành viên**: Các biến với kiểu dữ liệu khác nhau, ví dụ như int, float, char, ...
- **Lưu ý**: Khi khai báo có từ khóa typedef thì lúc khai báo thêm biến chỉ cần dùng tên mới của cấu trúc để khai báo mà không cần từ khóa struct.

2.8.2 Khai báo và sử dụng kiểu cấu trúc

- Khai báo:

- **Cấu trúc có thể được khai báo như một kiểu dữ liệu**: Sau khi khai báo cấu trúc, bạn có thể tạo nhiều biến kiểu cấu trúc đó.

```
typedef struct Student{
    char name[50];
    int age;
    float gpa;
} St;
int main(){
St s1, s2, s3;  //hoặc struct Student s1, s2, s3;
Return 0;
}
```

- **Truy cập các thành viên**: Sử dụng dấu chấm (.) để truy cập các thành viên của cấu trúc.

```
    s1 = {"Nguyen Van A", 20, 8.5};
    printf("Ho ten: %s\n", s1.name);
    printf("Tuoi: %d\n", s1.age);
    printf("Diem trung binh: %.2f\n", s1.gpa);
```

2.8.3 Khai báo và sử dụng con trỏ cấu trúc

- Khai báo:

```
Struct Student *ptr;  //hoặc  St *ptr;
```

- Sử dụng:

- Bạn có thể sử dụng con trỏ để tham chiếu tới cấu trúc. Khi đó, thay vì dấu chấm (.), bạn sẽ dùng dấu mũi tên (->) để truy cập thành viên

```
ptr = &s1;      // Trỏ tới biến s1
   // In thông tin thông qua con trỏ
   printf("Ho ten: %s\n", ptr->name);
   printf("Tuoi: %d\n", ptr->age);
   printf("Diem TB: %.2f\n", ptr->gpa);
```

- **Lý do sử dụng con trỏ**: Con trỏ giúp bạn dễ dàng thao tác với các cấu trúc trong các hàm hoặc truyền cấu trúc qua các hàm mà không cần sao chép toàn bộ dữ liệu.

2.8.4 Cấu trúc lồng nhau

- Một cấu trúc có thể chứa cấu trúc khác bên trong nó, được gọi là cấu trúc lồng nhau.

Ví dụ:

```
#include <stdio.h>
typedef struct Date {
   int day, year, month;
}date;
typedef struct Person {
   char hoten[50];
   date birthdate;  // Cấu trúc lồng nhau
}Person;
int main() {
   Person p1;
   strcpy(p1.name, "Alice");
   p1.birthdate.day = 15;
   p1.birthdate.month = 7;
   p1.birthdate.year = 1990;
   return 0;
}
```

2.8.5 Cấu trúc và cấp phát động

- Bạn có thể sử dụng malloc() để cấp phát bộ nhớ cho một cấu trúc động.

Ví dụ:

```
struct Person *ptr;
ptr = (struct Person*) malloc(sizeof(struct Person));
ptr->age = 25;
strcpy(ptr->name, "Bob");
free(ptr);
```

2.9 Danh sách liên kết

- *Khái niệm:* **Danh sách liên kết đơn (**singly linked list**)** là một cấu trúc dữ liệu động bao gồm nhiều node liên kết nhau thông qua con trỏ, trong đó mỗi node chứa: dữ liệu (Data), con trỏ tới node kế tiếp (next).

- *Cú pháp khai báo và cấp phát:*

```
//Cấu trúc Node
typedef struct Node {
    int data;
    struct Node *next;
} Node;
//Cách cấp phát một node mới
Node* taoNode(int x) {
    Node* p = (Node*)malloc(sizeof(Node));
    if (p == NULL) {
        printf("Khong du bo nho!\n");
        exit(1);
    }
    p->data = x;
    p->next = NULL;
    return p;
}
```

-Đặc điểm:

- Không giới hạn số lượng phần tử như mảng.
- Thêm, xóa phần tử linh hoạt, không cần dịch chuyển phần tử như mảng.
- Được quản lý hoàn toàn bằng **cấp phát động** (heap memory).

- Các thao tác cơ bản:

- Chèn node vào cuối danh sách:

```
void chenCuoi(Node **head, int x) {
    Node *newNode = taoNode(x);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node *p = *head;
    while (p->next != NULL) p = p->next;
    p->next = newNode;
}
```

- Xóa toàn bộ danh sách:

```
void giaiPhong(Node *head) {
    Node *tmp;
    while (head != NULL) {
        tmp = head;
        head = head->next;
        free(tmp);
    }
}
```

- Chèn vào đầu danh sách:

```
void chenDau(Node **head, int x) {
    Node* newNode = taoNode(x);
    newNode->next = *head;
    *head = newNode;
}
```

- In danh sách:

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

CHƯƠNG 3: KẾT LUẬN

3.1 Tóm tắt nội dung

- Qua quá trình tìm hiểu và nghiên cứu, báo cáo đã trình bày tổng quan về ngôn ngữ lập trình C – một trong những ngôn ngữ lập trình lâu đời và có ảnh hưởng lớn nhất trong lĩnh vực công nghệ thông tin. Nội dung của báo cáo tập trung vào các phần lý thuyết cốt lõi bao gồm:

- **Lịch sử và sự phát triển**: Ngôn ngữ C ra đời vào đầu những năm 1970 bởi Dennis Ritchie, được thiết kế nhằm hỗ trợ lập trình hệ thống, đặc biệt là hệ điều hành UNIX. Cho đến nay, C vẫn là nền tảng của nhiều ngôn ngữ hiện đại như C++, C#, Java, v.v.
- **Cấu trúc chương trình**: Một chương trình C được tổ chức chặt chẽ với hàm `main()` là điểm bắt đầu. Ngoài ra, các thành phần khác như thư viện tiêu chuẩn (`#include`), khai báo biến, và định nghĩa hàm được sử dụng để xây dựng chương trình một cách có tổ chức.
- **Cú pháp cơ bản**: Người học được tiếp cận với các kiểu dữ liệu chuẩn như `int`, `float`, `char`, cùng các toán tử số học, logic, quan hệ và gán. Các câu lệnh điều khiển dòng lệnh như `if`, `switch`, `for`, `while` và `do...while` giúp kiểm soát luồng thực thi của chương trình.
- **Hàm, mảng và con trỏ**: Đây là những thành phần quan trọng trong lập trình C. Hàm giúp chia chương trình thành các phần nhỏ, dễ quản lý. Mảng cho phép xử lý nhiều giá trị cùng kiểu dữ liệu, trong khi con trỏ mang lại khả năng truy cập trực tiếp đến địa chỉ bộ nhớ – một đặc điểm nổi bật và cũng là thử thách lớn đối với người học.
- **Quản lý bộ nhớ và xử lý dữ liệu**: Báo cáo cũng đề cập đến cách sử dụng con trỏ, cấp phát động (`malloc`, `free`) và thao tác với dữ liệu thông qua hàm nhập/xuất (`scanf`, `printf`) – những kỹ năng không thể thiếu khi lập trình trong môi trường thực tế.

3.2 Nhận xét và đánh giá

3.2.1 Nhận xét và đánh giá về đặc điểm ngôn ngữ C

- Ngôn ngữ lập trình C tuy không còn mới mẻ, nhưng vẫn giữ vai trò rất quan trọng trong giảng dạy và ứng dụng thực tế. Việc học C giúp người học xây dựng nền tảng lập trình vững chắc, đặc biệt là về tư duy logic, cách tổ chức chương trình và khả năng hiểu sâu về hoạt động của máy tính.

- Một số đặc điểm nổi bật của C có thể kể đến như:

  - Cú pháp rõ ràng, hiệu suất cao, gần với ngôn ngữ máy, phù hợp cho các ứng dụng hệ thống hoặc cần tối ưu tài nguyên.
  - Khả năng thao tác bộ nhớ trực tiếp thông qua con trỏ – một ưu điểm vượt trội so với nhiều ngôn ngữ hiện đại khác.
  - Tính di động cao, nhờ vào chuẩn ANSI C, các chương trình viết bằng C có thể dễ dàng biên dịch và chạy trên nhiều nền tảng khác nhau.

- Tuy nhiên, việc lập trình với C cũng mang lại không ít khó khăn, đặc biệt với người mới bắt đầu:

  - Không có kiểm tra lỗi tự động như trong các ngôn ngữ cấp cao hơn, dễ dẫn đến lỗi sai bộ nhớ, lỗi logic hoặc tràn bộ đệm.
  - Khó khăn trong quản lý bộ nhớ nếu chưa quen với con trỏ hoặc thao tác cấp phát động.

-Mặc dù vậy, việc học C là hoàn toàn xứng đáng và cần thiết, vì nó giúp người học làm quen với cách tư duy của máy tính, từ đó dễ dàng tiếp cận các ngôn ngữ hiện đại hơn sau này.

3.2. Nhận xét và đánh giá về từng thành phần trong ngôn ngữ C

- *Hàm:*

  - **Vai trò**: Giúp chia nhỏ chương trình thành các phần độc lập, dễ quản lý, mỗi hàm thực hiện một nhiệm vụ cụ thể.
  - **Ý nghĩa**: Tăng tính tái sử dụng, giảm lỗi, dễ bảo trì mã nguồn. Là nền tảng để xây dựng chương trình có tổ chức và mở rộng.

- *Con trỏ:*

- **Vai trò**: Lưu địa chỉ của biến khác, cho phép truy cập và thao tác trực tiếp tại vị trí bộ nhớ.
- **Ý nghĩa**: Là công cụ mạnh để tối ưu chương trình, giúp truyền tham chiếu, làm việc với cấu trúc dữ liệu động, và xử lý hiệu quả trong các thao tác cấp thấp.

- Con trỏ mảng:

- **Vai trò**: Lưu trữ địa chỉ của nhiều mảng hoặc chuỗi khác nhau.
- **Ý nghĩa**: Giúp quản lý linh hoạt các chuỗi (ví dụ: danh sách tên), mảng nhiều chiều, và tăng khả năng tổ chức dữ liệu khi số lượng phần tử thay đổi.

- Mảng con trỏ:

- **Vai trò**: Con trỏ trỏ đến toàn bộ mảng, cho phép duyệt hoặc thay đổi các phần tử trong mảng thông qua địa chỉ.
- **Ý nghĩa**: Tăng tính linh hoạt khi truyền mảng vào hàm, hoặc xử lý dữ liệu lớn trong các hàm chung, tránh sao chép dữ liệu không cần thiết.

- Con trỏ hàm (Function Pointer):

- **Vai trò**: Cho phép lưu địa chỉ của một hàm, gọi hàm thông qua con trỏ.
- **Ý nghĩa**: Cần thiết trong các tình huống cần gọi hàm linh hoạt (ví dụ: callback, menu động), hoặc lập trình hướng module. Gần giống khái niệm "delegate" trong các ngôn ngữ hiện đại.

- Cấp phát động (Dynamic Memory Allocation):

- **Vai trò**: Cấp phát vùng nhớ trong khi chương trình đang chạy bằng malloc, calloc, realloc, và giải phóng bằng free.
- **Ý nghĩa**: Cho phép xử lý dữ liệu có kích thước thay đổi, tiết kiệm bộ nhớ và tạo cấu trúc dữ liệu linh hoạt như danh sách liên kết, cây, v.v.

- Xử lý tệp (File I/O):

- **Vai trò**: Đọc và ghi dữ liệu từ/đến tệp thông qua các hàm fopen, fprintf, fread, fclose,...
- **Ý nghĩa**: Làm cho chương trình có thể lưu trữ và truy xuất dữ liệu lâu dài, cần thiết cho các ứng dụng thực tế như quản lý sinh viên, điểm, sản phẩm, v.v.

- Kiểu cấu trúc (Struct):

- **Vai trò**: Nhóm nhiều biến có kiểu khác nhau lại thành một đơn vị (struct), đại diện cho đối tượng phức tạp như sinh viên, nhân viên,...
- **Ý nghĩa**: Cho phép mô hình hóa dữ liệu có cấu trúc trong thực tế, giúp chương trình dễ hiểu và tổ chức logic hơn.

- Danh sách liên kết (Linked List):

- **Vai trò**: Cấu trúc dữ liệu động gồm các nút liên kết với nhau thông qua con trỏ.
- **Ý nghĩa**: Linh hoạt hơn mảng trong việc thêm, xóa phần tử (không cần dịch chuyển bộ nhớ). Là nền tảng cho các cấu trúc cao hơn như cây, đồ thị.

3.3 Hướng phát triển khi học ngôn ngữ lập trình C

- Việc nắm vững lý thuyết ngôn ngữ lập trình C là bước khởi đầu quan trọng trong hành trình trở thành lập trình viên. Sau khi hoàn thành phần lý thuyết, người học có thể phát triển kỹ năng theo một số hướng sau:

- **Thực hành lập trình với các bài toán thực tế**: Áp dụng kiến thức đã học để xây dựng các chương trình như quản lý sinh viên, quản lý điểm, máy tính đơn giản, xử lý chuỗi, v.v. Đây là cách tốt nhất để củng cố kiến thức và hình thành kỹ năng giải quyết vấn đề.
- **Nghiên cứu về cấu trúc dữ liệu và giải thuật**: Sử dụng C để triển khai các cấu trúc như danh sách liên kết, cây, ngăn xếp, hàng đợi, bảng băm... nhằm hiểu rõ cách tổ chức và xử lý dữ liệu hiệu quả.
- **Tìm hiểu lập trình hệ thống và nhúng**: Với đặc điểm gần với phần cứng, C rất phù hợp để học về lập trình hệ điều hành, trình điều khiển thiết bị hoặc lập trình vi điều khiển (như Arduino, STM32...).
- **Học lập trình tệp tin và xây dựng chương trình có lưu trữ dữ liệu**: Biết cách thao tác với file (fopen, fprintf, fread, fwrite...) để viết các ứng dụng đơn giản nhưng thực tế.
- **Tiếp cận các ngôn ngữ lập trình hiện đại**: Sau khi vững C, người học có thể chuyển sang C++, Java, Python, hay thậm chí là học lập trình hướng đối tượng, lập trình web, hoặc phát triển ứng dụng di động.

- Việc học ngôn ngữ lập trình C không chỉ dừng lại ở việc viết được một chương trình chạy được, mà còn là nền móng để phát triển sâu hơn trong ngành công nghệ thông tin.

TÀI LIỆU THAM KHẢO

[1]. https://200lab.io/blog/lap-trinh-c-co-ban(05/11/2025)"Lập trình C cơ bản – 200lab"

[2]. https://vietjack.com/lap_trinh_c/(05/11/2025)"Lập trình C cơ bản, nâng cao"

[3]. https://blog.28tech.com.vn/c%2005/11/2025)"Lập trình C 28Tech"