**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**ĐỀ TÀI: LÝ THUYẾT CƠ BẢN NGÔN NGỮ LẬP TRÌNH C………**

**………………………...................…………………………………..…**

**.........................................................................................................**

Giảng viên hướng dẫn: TRẦN PHONG NHÃ

Sinh viên thực hiện: PHẠM NGỌC TÚ

MSSV: 6551071092

Lớp : 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**



**ĐỀ TÀI: LÝ THUYẾT CƠ BẢN NGÔN NGỮ LẬP TRÌNH C………**

**………………………...................…………………………………..…**

**.........................................................................................................**

Giảng viên hướng dẫn: TRẦN PHONG NHÃ

Sinh viên thực hiện: PHẠM NGỌC TÚ

MSSV: 6551071092

Lớp : CNTT

Khoá :65

Tp. Hồ Chí Minh, năm 2025

**LỜI CẢM ƠN**

Lời nói đầu tiên, em xin gửi lời cảm ơn chân thành đến 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 biết ơn sâu sắc nhất.

Em xin chân thành cảm ơn quý thầy cô đã giúp đỡ tạo điều kiện để em hoàn thành báo cáo. 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 GIẢNG VIÊN HƯỚNG DẪN**

|  |
| --- |
| Tp. Hồ Chí Minh, ngày 05 tháng 11 năm 2025 |
| Giảng viên hướng dẫn |
|  |
| ThS. Trần Phong Nhã |

Contents

LỜI CẢM ƠN i

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN ii

1) HÀM (FUNCTION) 1

2) Con trỏ (Pointer) 3

3) Con trỏ mảng (Pointer to Array) 6

4) Mảng con trỏ (Array of Pointers) 8

5) Con trỏ hàm (Function Pointer) 10

6) Cấp phát động (Dynamic Memory Allocation) 12

7) Xử lý tệp trong C (File Handling) 15

8) Kiểu cấu trúc (struct) 17

9) Danh sách liên kết (Linked List) 19

# HÀM (FUNCTION)

## Khái niệm:

* Hàm là một khối mã (code block) thực hiện một công việc cụ thể.
* Mỗi chương trình C đều bắt đầu từ hàm main().
* Hàm có thể được gọi lại nhiều lần, không cần viết lại cùng một đoạn mã.
* Giúp chia nhỏ chương trình thành các phần logic riêng biệt, dễ quản lý và bảo trì.

**Cấu trúc:**

A computer screen with text

AI-generated content may be incorrect.

**Ví dụ:**

A computer screen with numbers and symbols

AI-generated content may be incorrect.

**Kết quả:**

A black screen with white text

AI-generated content may be incorrect.

## Ưu điểm:

* Tái sử dụng mã: Viết một lần, dùng nhiều lần.
* Giảm độ phức tạp của chương trình: Mỗi hàm đảm nhận một chức năng → chương trình dễ hiểu hơn.
* Thuận tiện kiểm thử (test) và sửa lỗi: Kiểm tra từng hàm riêng biệt.
* Dễ bảo trì: Thay đổi trong một hàm không ảnh hưởng đến phần còn lại (nếu tổ chức tốt).
* Tăng tính tổ chức: Chia chương trình thành các phần rõ ràng, logic.

## Nhược điểm:

* Tốn thời gian ban đầu để thiết kế hệ thống hàm hợp lý.
* Nếu lạm dụng hàm không hợp lý → chương trình bị chia nhỏ quá mức, gây rối.
* Gọi hàm nhiều lần có thể làm tăng chi phí về hiệu năng (thời gian, bộ nhớ) trong một số trường hợp.

## Ứng dụng:

* Dùng để chia nhỏ các tác vụ như: nhập dữ liệu, xử lý, hiển thị kết quả.
* Xây dựng thư viện hàm dùng chung cho nhiều chương trình.
* Viết các thuật toán phức tạp: mỗi bước là một hàm riêng biệt.
* Giúp tổ chức chương trình có cấu trúc tốt hơn – cơ sở cho lập trình hướng đối tượng sau này.

# Con trỏ (Pointer)

## Khái niệm:

* Con trỏ là một biến lưu địa chỉ của biến khác trong bộ nhớ.
* Cho phép truy cập và thao tác dữ liệu gián tiếp thông qua địa chỉ mà nó trỏ đến.
* Là một phần quan trọng trong ngôn ngữ C, đặc biệt khi làm việc với mảng, chuỗi, hàm và cấu trúc dữ liệu động.

## Ký hiệu:

* \* : Dùng để khai báo biến con trỏ và truy cập giá trị tại địa chỉ mà con trỏ trỏ đến.
  + Ví dụ: int \*p;, \*p = 10;
* &: Dùng để lấy địa chỉ của một biến.
  + Ví dụ: p = &x;
* -> : Dùng để truy cập thành phần của struct thông qua con trỏ.
  + Ví dụ: sv->ten tương đương với (\*sv).ten

A computer screen shot of a program code

AI-generated content may be incorrect.

**Ví dụ và kết quả:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

## Ưu điểm:

* Truy cập trực tiếp vào bộ nhớ, giúp tối ưu hiệu năng trong nhiều trường hợp.
* Cho phép truyền tham chiếu cho hàm → tránh sao chép dữ liệu.
* Quản lý bộ nhớ động (dùng với malloc, free, realloc...).
* Linh hoạt khi thao tác với cấu trúc dữ liệu phức tạp như danh sách liên kết, cây, đồ thị...

## Nhược điểm:

* Dễ gây lỗi truy cập bộ nhớ nếu dùng sai (vd: trỏ đến vùng nhớ chưa cấp phát, hoặc đã bị giải phóng).
* Gây khó khăn cho người mới học, dễ nhầm giữa địa chỉ, giá trị và cách dùng toán tử.
* Nếu không quản lý tốt bộ nhớ cấp phát → rò rỉ bộ nhớ (memory leak).

## Ứng dụng:

* Truyền tham chiếu để thay đổi giá trị biến trong hàm.
* Quản lý mảng và chuỗi hiệu quả.
* Làm việc với cấp phát động (dynamic memory allocation).
* Xây dựng cấu trúc dữ liệu động như danh sách liên kết, cây, đồ thị.
* Truy cập thành phần của struct hoặc dữ liệu phức tạp một cách linh hoạt.

# Con trỏ mảng (Pointer to Array)

## Khái niệm:

* Con trỏ mảng là con trỏ trỏ đến phần tử đầu tiên của mảng.
* Có thể sử dụng con trỏ để duyệt mảng tương tự như dùng chỉ số ([]).
* Khi truyền mảng vào hàm, thực chất là truyền địa chỉ phần tử đầu tiên → chính là con trỏ.

**Lưu ý:**

A black background with white text

AI-generated content may be incorrect.

**Ví dụ và kết quả:**

**A computer screen with text and numbers

AI-generated content may be incorrect.**

## Ưu điểm:

* Tối ưu truy cập mảng, đặc biệt trong các vòng lặp.
* Có thể dễ dàng truyền mảng cho hàm thông qua con trỏ.
* Hỗ trợ thao tác với mảng động (cấp phát bằng malloc).
* Cho phép xử lý linh hoạt các mảng 1 chiều, 2 chiều, thậm chí mảng các con trỏ.

## Nhược điểm:

* Khó đọc và khó hiểu đối với người mới học (do cú pháp phức tạp).
* Dễ gây lỗi khi vượt quá giới hạn mảng (out of bounds).
* Thiếu thông tin về kích thước mảng, vì con trỏ chỉ biết địa chỉ, không biết độ dài mảng.

## Ứng dụng:

* Duyệt và xử lý mảng bằng cách tăng con trỏ (ptr++).
* Truyền mảng vào hàm để tính toán, xử lý, sắp xếp, tìm kiếm...
* Quản lý mảng động trong các chương trình lớn.
* Làm việc với mảng ký tự (chuỗi) trong C.

# Mảng con trỏ (Array of Pointers)

## Khái niệm:

* Là một mảng mà mỗi phần tử là một con trỏ.
* Mỗi con trỏ trong mảng có thể trỏ đến một vùng nhớ khác nhau.
* Thường dùng với char \* để lưu nhiều chuỗi khác nhau, mỗi chuỗi là một dãy ký tự được quản lý qua con trỏ.

**Ví dụ và kết quả:**

**A computer screen with text and symbols

AI-generated content may be incorrect.**

**Giải thích 1 số ký hiệu:**

|  |  |
| --- | --- |
| Biểu thức | Ý nghĩa |
| ten[i] | Là cách viết ngắn gọn của \*(ten + i) |
| ten | Là con trỏ trỏ đến phần tử đầu tiên trong mảng con trỏ |
| \*(ten + i) | Lấy con trỏ thứ i (→ là chuỗi) |
| \*(\*(ten + i)) | Lấy ký tự đầu tiên của chuỗi i |

**Ví dụ và kết quả cho \*(\*(ten + i):**

A computer screen with text and symbols

AI-generated content may be incorrect.

A black screen with a black background

AI-generated content may be incorrect.

## Ưu điểm:

* Tiết kiệm bộ nhớ khi làm việc với các chuỗi có độ dài khác nhau.
* Linh hoạt: mỗi phần tử có thể trỏ đến vị trí khác nhau, không cần liên tiếp trong bộ nhớ.
* Phù hợp khi lưu danh sách từ khóa, chuỗi nhập vào, thông báo...

## Nhược điểm:

* Cần quản lý bộ nhớ cẩn thận, nhất là khi cấp phát động (malloc, free).
* Nếu thao tác sai với từng con trỏ → dễ bị lỗi truy cập bộ nhớ.
* Gây khó hiểu khi lồng với các con trỏ hàm hoặc mảng nhiều chiều.

## Ứng dụng:

* Quản lý danh sách chuỗi (vd: danh sách tên sinh viên, danh sách câu lệnh).
* Dùng trong argv[] của hàm main(int argc, char \*argv[]).
* Tạo bảng từ khóa, menu hoặc tập hợp các chuỗi hằng số.

# Con trỏ hàm (Function Pointer)

## Khái niệm:

* Là con trỏ có thể trỏ đến một hàm, tức là lưu địa chỉ của một hàm.
* Cho phép gọi hàm thông qua con trỏ, hoặc truyền hàm như một tham số cho hàm khác.

**Cú pháp:**

A black screen with white text

AI-generated content may be incorrect.

**Ví dụ và kết quả:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Ví dụ 2 – đùng mảng con trỏ để tạo Menu:**

A computer screen shot of text

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

## Ưu điểm:

* Tăng tính linh hoạt: Có thể thay đổi hàm được gọi tại runtime.
* Giúp viết mã tổng quát: Truyền hàm làm tham số → viết hàm xử lý chung.
* Là công cụ cơ bản để xây dựng callback - tức là truyền hàm vào một hàm khác để thực thi tùy ý - hoặc mô phỏng hành vi hướng đối tượng.

## Nhược điểm:

* Cú pháp phức tạp, dễ gây nhầm lẫn, nhất là khi lồng nhiều cấp.
* Khó đọc, khó debug hơn so với cách gọi hàm thông thường.
* Nếu trỏ sai (con trỏ không hợp lệ) → dễ gây lỗi runtime nghiêm trọng.

## Ứng dụng:

* Truyền hàm làm tham số (ví dụ: hàm so sánh trong qsort()).
* Tạo mảng các con trỏ hàm để xây dựng menu chương trình hoặc các hành động tương ứng.
* Callback function trong lập trình giao tiếp thiết bị, lập trình sự kiện hoặc hệ thống.

# Cấp phát động (Dynamic Memory Allocation)

## Khái niệm:

* Là quá trình cấp phát bộ nhớ trong khi chương trình đang chạy, giúp quản lý vùng nhớ linh hoạt, không cần biết trước kích thước chính xác.
* Thường dùng trong các tình huống: mảng không cố định kích thước, danh sách liên kết, cấu trúc dữ liệu động...
* Thư viện <stdlib.h> cung cấp các hàm cấp phát động.

**Ví dụ và kết quả:**

A computer screen shot of a program code

AI-generated content may be incorrect.

**Lưu ý:** Cần kiểm tra cấp phát và giải phóng bộ nhớ sau khi dùng.

**Kiểm tra sau khi cấp phát**:

A black screen with white text

AI-generated content may be incorrect.

**Ví dụ giải phóng toàn bộ danh sách liên kết đơn:**

A computer screen with white text

AI-generated content may be incorrect.

**Các hàm cấp phát động:**

|  |  |
| --- | --- |
| Hàm | Công dụng |
| malloc(size) | Cấp phát một vùng nhớ có kích thước size byte, **không khởi tạo giá trị**. |
| calloc(n, size) | Cấp phát vùng nhớ cho n phần tử, mỗi phần tử size byte, **được khởi tạo về 0**. |
| realloc(ptr, new\_size) | Thay đổi kích thước vùng nhớ đã cấp phát bởi malloc hoặc calloc. Giữ dữ liệu cũ nếu có thể. |
| free(ptr) | Giải phóng vùng nhớ được cấp phát động, tránh rò rỉ bộ nhớ. |

**Lưu ý**: Các hàm này **trả về con trỏ kiểu void\***, nên cần ép kiểu phù hợp khi dùng trong C (vd: (int \*)malloc(...)).

**So sánh malloc() và calloc():**

|  |  |  |
| --- | --- | --- |
| Tiêu chí | malloc() | calloc() |
| Cách gọi | malloc(size) | calloc(num\_elements, size\_of\_element) |
| Khởi tạo giá trị | Không khởi tạo (giá trị rác) | Tự động khởi tạo tất cả về 0 |
| Hiệu suất | Nhanh hơn do không cần khởi tạo | Chậm hơn vì phải khởi tạo 0 |
| Dễ gây lỗi khi quên khởi tạo | Có | Ít hơn |
| Dùng khi nào | Khi không cần giá trị mặc định | Khi cần vùng nhớ sạch (gán = 0 ban đầu) |

## Ưu điểm:

* Tối ưu sử dụng bộ nhớ: chỉ cấp phát khi cần, giải phóng khi xong.
* Linh hoạt mở rộng hoặc thu hẹp vùng nhớ tùy theo nhu cầu tại runtime.
* Phù hợp với cấu trúc dữ liệu danh sách liên kết, cây, đồ thị, nơi kích thước không cố định.

## Nhược điểm:

* Cần quản lý bộ nhớ thủ công → dễ gây rò rỉ bộ nhớ nếu quên free().
* Nếu sử dụng sai → lỗi nghiêm trọng (vd: truy cập vùng nhớ chưa cấp phát hoặc đã giải phóng).
* Gây khó khăn cho người mới học vì cần thao tác trực tiếp với con trỏ.

## Ứng dụng:

* Quản lý mảng động, danh sách, chuỗi nhập từ người dùng.
* Xây dựng cấu trúc dữ liệu phức tạp trong các ứng dụng thực tế.
* Phù hợp với các bài toán cần cấp phát nhiều dữ liệu linh hoạt theo thời gian chạy.

## Lưu ý quan trọng khi sử dụng cấp phát động

### Buffer Overflow – Lỗi nghiêm trọng nhất

**Nguyên nhân:** Truy cập vượt quá vùng nhớ đã cấp phát

Ví dụ lỗi:

A screen shot of a computer code

AI-generated content may be incorrect.

**Hậu quả:**

* Crash chương trình hoặc hành vi không dự đoán
* Ghi đè dữ liệu của biến khác
* Lỗ hổng bảo mật nghiêm trọng

### Use After Free - Sử dụng con trỏ chưa khởi tạo:

**Nguyên nhân:** Sử dụng con trỏ sau khi đã free()

**Ví dụ:**

A screen shot of a computer

AI-generated content may be incorrect.

**Tại sao nguy hiểm:**

* Vùng nhớ có thể được cấp phát lại cho mục đích khác
* Data corruption nghiêm trọng
* Lỗi xuất hiện ngẫu nhiên, rất khó debug

## Quy tắc vàng:

* Luôn kiểm tra kết quả của malloc()
* Mỗi malloc() phải có một free() tương ứng
* Gán NULL cho con trỏ sau khi free
* Không truy cập bộ nhớ đã free
* Không free cùng một con trỏ hai lần
* Free tất cả bộ nhớ trước khi thoát chương trình

# Xử lý tệp trong C (File Handling)

## Các bước xử lý tệp

A black screen with white text

AI-generated content may be incorrect.

**Lưu ý:** Nếu mở file thất bại, fopen() trả về NULL. Cần kiểm tra file trước khi xử lý.

## Phân biệt tệp văn bản & tệp nhị phân

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Tệp văn bản (.txt) | Tệp nhị phân (.bin) |
| Dữ liệu lưu | Dạng **ký tự có thể đọc được** | Dạng **byte raw (nhị phân)** |
| Hàm sử dụng chính | fprintf(), fscanf(), fgets()... | fread(), fwrite() |
| Ưu điểm | Dễ đọc, dễ debug | Nhanh, tiết kiệm bộ nhớ, phù hợp với struct |
| Nhược điểm | Không hiệu quả với struct/phức tạp | Không mở được bằng Notepad hoặc trình soạn thảo |

## Các chế độ mở file

|  |  |
| --- | --- |
| Chế độ | Ý nghĩa |
| "r" | Mở để đọc (file phải tồn tại) |
| "w" | Ghi đè nội dung mới |
| "a" | Ghi thêm cuối file |
| "rb" | Đọc file nhị phân |
| "wb" | Ghi file nhị phân |
| "r+" | Đọc & ghi (file phải tồn tại) |

## Một số hàm xử lý tệp phổ biến:

|  |  |
| --- | --- |
| Hàm | Mô tả |
| fgetc(FILE \*f) | Đọc 1 ký tự từ file |
| fputc(int c, FILE \*f) | Ghi 1 ký tự vào file |
| fgets(char \*str, int n, FILE \*f) | Đọc 1 dòng (tối đa n-1 ký tự) |
| fputs(const char \*str, FILE \*f) | Ghi 1 dòng vào file |
| feof(FILE \*f) | Kiểm tra đã kết thúc file chưa (true nếu đã tới cuối file) |
| fseek(FILE \*f, long offset, int origin) | Di chuyển con trỏ đọc/ghi tới vị trí nhất định |
| ftell(FILE \*f) | Trả về vị trí hiện tại của con trỏ trong file |
| rewind(FILE \*f) | Đưa con trỏ file về đầu (tương đương fseek(f, 0, SEEK\_SET)) |

## Ví dụ ghi và đọc tệp nhị phân

**Ví dụ struct + ghi nhị phân:**

A computer code on a black background

AI-generated content may be incorrect.



**Đọc lại từ tệp nhị phân:**

A screen shot of a computer

AI-generated content may be incorrect.

# Kiểu cấu trúc (struct)

## Khái niệm:

* Struct là kiểu dữ liệu do người dùng định nghĩa, gồm nhiều biến (có thể khác kiểu) gom lại thành một đơn vị.
* Dùng để mô tả các đối tượng phức tạp như sinh viên, nhân viên, sản phẩm...
* Giúp tổ chức dữ liệu có cấu trúc và xử lý logic hơn trong chương trình.

**Cú pháp khai báo:**

A black rectangular object with white text

AI-generated content may be incorrect.

**Ví dụ:**

A screenshot of a computer program

AI-generated content may be incorrect.

## Ưu điểm:

* Gom nhiều biến có liên quan thành một khối dữ liệu logic.
* Dễ quản lý và xử lý nhiều đối tượng cùng loại (danh sách struct).
* Tăng tính modular hóa chương trình, dễ mở rộng.
* Có thể kết hợp với con trỏ hoặc mảng, file, hàm để xây dựng các hệ thống lớn.

## Nhược điểm:

* Không có tính đóng gói (encapsulation) như class trong OOP.
* Không hỗ trợ các hàm thành viên bên trong struct (chỉ có dữ liệu).
* Có thể tốn bộ nhớ nếu khai báo dư thừa thành phần hoặc padding bộ nhớ.

## Ứng dụng:

* Lưu thông tin sinh viên, sản phẩm, nhân viên,...
* Xử lý danh sách liên kết, hàng đợi, cây, đồ thị (struct + con trỏ).
* Dùng kết hợp với file nhị phân để lưu trữ dữ liệu cấu trúc phức tạp.
* Làm việc với thư viện hoặc hệ thống C chuẩn.

# Danh sách liên kết (Linked List)

## Khái niệm:

* Danh sách liên kết là cấu trúc dữ liệu gồm các node, mỗi node chứa:
  + Dữ liệu.
  + Con trỏ trỏ đến node kế tiếp (hoặc cả trước và sau tùy loại DSLK).
* Khác với mảng, DSLK không yêu cầu bộ nhớ liên tục → cấp phát động linh hoạt.
* Duyệt danh sách phải tuần tự từ đầu, không truy cập trực tiếp theo chỉ số.

→ Các loại danh sách liên kết:

|  |  |  |
| --- | --- | --- |
| Loại danh sách | Cấu trúc node | Con trỏ |
| Đơn (Singly Linked List) | data + next | Chỉ trỏ tới **node kế tiếp** |
| Đôi (Doubly Linked List) | data + next + prev | Có thể đi **hai chiều (prev, next)** |
| Vòng (Circular Linked List) | Giống đơn/đôi | Node cuối **trỏ ngược lại** node đầu |

**Ví dụ danh sách liên kết đơn (Singly Linked List):**

A computer screen with white text

AI-generated content may be incorrect.

*Duyệt từ head → đi theo next → đến NULL.*

**Ví dụ danh sách liên kết đôi (Doubly Linked List):**

A computer screen with white text and green text

AI-generated content may be incorrect.

*Duyệt có thể đi từ đầu → cuối hoặc ngược lại.*

**Ví dụ danh sách liên kết vòng (Circular Linked List)**

A computer screen with text on it

AI-generated content may be incorrect.

*Node cuối trỏ ngược lại node đầu → có thể lặp vô hạn.*

## Ưu điểm:

* Không cần biết trước kích thước → cấp phát động linh hoạt.
* Thêm/xóa node dễ dàng tại bất kỳ vị trí nào (đặc biệt là đầu/cuối).
* Tốt cho các cấu trúc động như hàng đợi, ngăn xếp, cây.

## Nhược điểm:

* Không truy cập trực tiếp theo chỉ số → phải duyệt tuần tự.
* Tốn bộ nhớ cho con trỏ đi kèm mỗi node.
* Khó debug hơn mảng, dễ sai khi thao tác con trỏ.

## Ứng dụng:

* Cài đặt các cấu trúc dữ liệu động như:
  + Stack (ngăn xếp), Queue (hàng đợi), Deque (hàng đợi hai đầu).
* Tổ chức dữ liệu có kích thước thay đổi thường xuyên.
* Quản lý danh sách học sinh, sản phẩm, tiến trình chạy, chuỗi nhạc,...
* Dùng trong các giải thuật phức tạp: đồ thị, danh sách kề,...