

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**MÔN: KỸ THUẬT LẬP TRÌNH**

**ĐỀ TÀI: CON TRỎ**

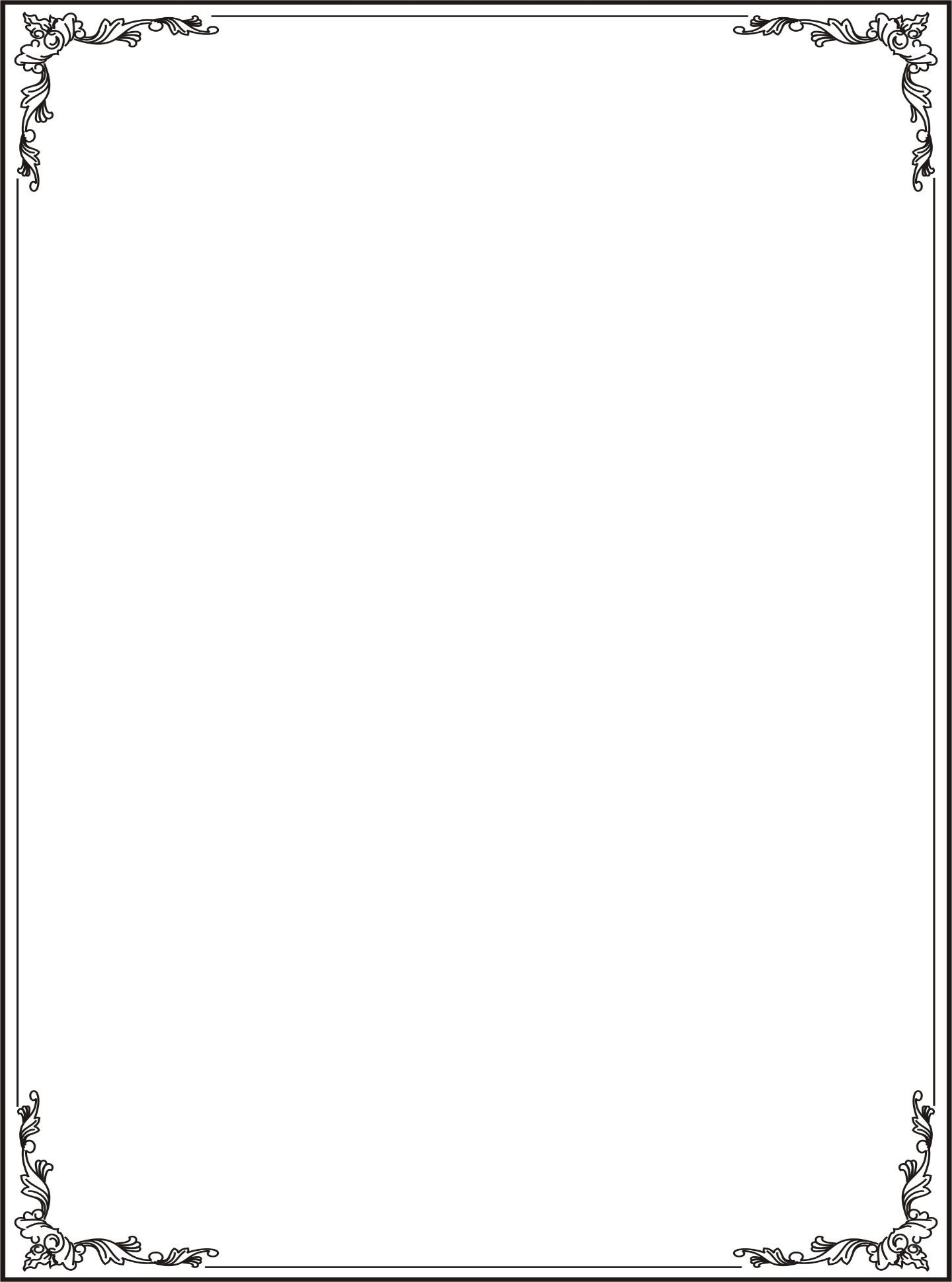
Giảng viên: TRẦN PHONG NHÃ

Sinh viên thực hiện: LÊ QUỐC ĐẠT

Lớp: CQ.65.CNTT

Khóa: 65

Tp. Hồ Chí Minh, tháng 5 năm 2025



**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**MÔN: KỸ THUẬT LẬP TRÌNH**

**ĐỀ TÀI: CON TRỎ**

Giảng viên: TRẦN PHONG NHÃ

Sinh viên thực hiện: LÊ QUỐC ĐẠT

Lớp: CQ.65.CNTT

Khóa: 65

Tp. Hồ Chí Minh, tháng 5 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 “**Con trỏ**”. 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.

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**

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

|  |  |
| --- | --- |
|  | **Giảng viên hướng dẫn** |
|  |  |
|  | **Trần Phong Nhã** |

......................................................................................................................................

# MỤC LỤC

[LỜI CẢM ƠN i](#_Toc198970352)

[NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN ii](#_Toc198970353)

[MỤC LỤC iii](#_Toc198970354)

[DANH MỤC CHỮ VIẾT TẮT vi](#_Toc198970355)

[DANH MỤC HÌNH ẢNH vii](#_Toc198970356)

[CHƯƠNG I: MỞ ĐẦU 1](#_Toc198970357)

[1.1 Tổng quan về đề tài : 1](#_Toc198970358)

[1.2 Mục tiêu nghiên cứu : 1](#_Toc198970359)

[1.3 Đối tượng và phạm vi nghiên cứu : 1](#_Toc198970360)

[1.4 Cấu trúc báo cáo : 2](#_Toc198970361)

[1.4.1 Chương 1: Mở đầu 2](#_Toc198970362)

[1.4.2 Chương 2: Tìm hiểu Con trỏ 2](#_Toc198970363)

[1.4.3 Chương 3: Con trỏ mảng 2](#_Toc198970364)

[1.4.4 Chương 4: Mảng con trỏ 2](#_Toc198970365)

[1.4.5 Chương 5: Cấp phát động 2](#_Toc198970366)

[1.4.6 Chương 6: Danh sách liên kết 2](#_Toc198970367)

[1.4.7 Chương 7: Ứng dụng 2](#_Toc198970368)

[CHƯƠNG II: TÌM HIỂU CON TRỎ 2](#_Toc198970369)

[2.1 Khái niệm con trỏ: 2](#_Toc198970370)

[2.2 Ký hiệu và cách sử dụng: 3](#_Toc198970371)

[2.3 Phân loại con trỏ: 4](#_Toc198970372)

[2.4 Ứng dụng của con trỏ: 4](#_Toc198970373)

[2.5 Lỗi phổ biến khi dùng con trỏ: 6](#_Toc198970374)

[CHƯƠNG III : CON TRỎ MẢNG 6](#_Toc198970375)

[3.1 Định nghĩa Con trỏ mảng: 6](#_Toc198970376)

[3.1.1 Arrays and Pointers: 6](#_Toc198970377)

[3.1.2 Pointer Arithmetic and Arrays: 7](#_Toc198970378)

[3.2 Ứng dụng con trỏ mảng: 8](#_Toc198970379)

[CHƯƠNG IV : MẢNG CON TRỎ 9](#_Toc198970380)

[4.1 Định nghĩa mảng con trỏ : 9](#_Toc198970381)

[4.2 Ứng dụng : 9](#_Toc198970382)

[CHƯƠNG V : CẤP PHÁT ĐỘNG 11](#_Toc198970383)

[5.1 Cấp phát động : 12](#_Toc198970384)

[5.1.1 Malloc : 12](#_Toc198970385)

[5.1.2 Calloc : 13](#_Toc198970386)

[5.1.3 Realloc : 13](#_Toc198970387)

[5.1.4 Free : 14](#_Toc198970388)

[5.2 Ứng dụng cấp phát động : 15](#_Toc198970389)

[CHƯƠNG VI : DANH SÁCH LIÊN KẾT 16](#_Toc198970390)

[6.1 Node : 16](#_Toc198970391)

[6.1.1 Tạo Node : 17](#_Toc198970392)

[6.1.2 Thêm Node đầu : 18](#_Toc198970393)

[6.1.3 Thêm Node vào giữa : 18](#_Toc198970394)

[6.1.4 Thêm Node vào cuối : 19](#_Toc198970395)

[6.1.5 Xóa Node : 19](#_Toc198970396)

[6.2 Kết luận : 20](#_Toc198970397)

[CHƯƠNG VII : ỨNG DỤNG 21](#_Toc198970398)

[CHƯƠNG VIII : KẾT QUẢ VÀ KIẾN NGHỊ 21](#_Toc198970399)

[8.1 Kết quả đạt được : 21](#_Toc198970400)

[8.2 Kiến nghị : 22](#_Toc198970401)

[TÀI LIỆU THAM KHẢO 22](#_Toc198970402)

# **DANH MỤC CHỮ VIẾT TẮT**

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Mô tả | Ý nghĩa | Ghi chú |
| 1 |  |  | \* |
| 2 |  |  | \*\* |
| 3 |  |  | \*\* |

# **DANH MỤC HÌNH ẢNH**

Hình 2.1: Khái niệm con trỏ 2

Hình 2.2: Cách sử dụng con trỏ 3

Hình 2.3: Gộp khai báo và gán 3

Hình 2.4: Kiểu dữ liệu 4

Hình 2.5: Vấn đề hoán đổi biến bằng hàm nếu không dùng con trỏ 5

Hình 2.6: Hàm hoán đổi sau khi dùng con trỏ 5

Hình 3.1: Tên mảng là con trỏ chỉ phần tử đầu tiên 6

Hình 3.2: Ví dụ tên mảng tương đương con trỏ chỉ phần tử đầu tiên 7

Hình 3.3: Con trỏ chỉ đến tên mảng 7

Hình 3.4: Con trỏ số học và mảng 7

Hình 3.5: Hai cách viết con trỏ số học tương đương 8

Hình 3.6: Đảo ngược một dãy số bằng con trỏ mảng 8,9

Hình 4.1: Ví dụ cấp phát động mảng 2 chiều sử dụng mảng con trỏ 10

Hình 5.1: Cách nhìn tổng quan về bộ nhớ 11

Hình 5.2: Các hàm quản lý bộ nhớ 12

Hình 5.3: Malloc() 12

Hình 5.4: Calloc() 13

Hình 5.5: Realloc() 13

Hình 5.6: Free() 14

Hình 5.7: Ứng dụng malloc 15

Hình 5.8: Ứng dụng realloc 15

Hình 5.9: Ứng dụng malloc trong struct 15

Hình 5.10: Ứng dụng trong danh sách liên kết 16

Hình 6.1: Linked list 16

Hình 6.2: Cấu trúc node danh sách liên kết 17

Hình 6.3: Định nghĩa cấu trúc Node 17

Hình 6.4: Thêm một Node vào đầu danh sách 18

Hình 6.5: Thêm một Node vào giữa danh sách 18

Hình 6.6: Thêm một Node vào cuối danh sách 19

Hình 6.7: Xóa một Node trong danh sách 19,20

# **CHƯƠNG I: MỞ ĐẦU**

## **1.1 Tổng quan về đề tài :**

Trong lập trình C, con trỏ (pointer) là một trong những khái niệm mạnh mẽ và quan trọng nhất. Việc sử dụng con trỏ cho phép lập trình viên thao tác trực tiếp với địa chỉ bộ nhớ, làm tăng hiệu suất xử lý và mở rộng khả năng của chương trình. Tuy nhiên, con trỏ cũng là chủ đề gây khó khăn cho người học do tính trừu tượng và dễ gây lỗi nếu dùng sai cách.

## **1.2 Mục tiêu nghiên cứu :**

* Hiểu rõ khái niệm và cách hoạt động của con trỏ trong bộ nhớ.
* Biết cách khai báo, khởi tạo và sử dụng con trỏ trong các trường hợp khác nhau.
* Vận dụng con trỏ để xử lý mảng, chuỗi, hàm, struct và cấp phát động.
* Phân tích các lỗi thường gặp khi dùng con trỏ và cách khắc phục.
* Ứng dụng: Xây dựng ứng dụng cho bài toán cụ thể với đầy đủ tính năng

cần thiết

## **1.3 Đối tượng và phạm vi nghiên cứu :**

* Con trỏ trong ngôn ngữ lập trình C, bao gồm:
  + Khái niệm và cách khai báo, sử dụng con trỏ.
  + Các loại con trỏ: con trỏ đơn, con trỏ hàm, con trỏ mảng, mảng con trỏ, con trỏ struct,...
  + Các thao tác cơ bản với con trỏ như lấy địa chỉ, truy cập giá trị, truyền tham chiếu, cấp phát động,...
* Cách con trỏ hoạt động trong bộ nhớ máy tính.
* Mối quan hệ giữa con trỏ với các cấu trúc dữ liệu như mảng, chuỗi, hàm và struct.
* Ứng dụng của con trỏ trong việc xây dựng các cấu trúc dữ liệu động như danh sách liên kết, cây, đồ thị,…
  + - Nghiên cứu giới hạn trong phạm vi ngôn ngữ lập trình C.
    - Tập trung vào cách sử dụng con trỏ để thao tác với:
  + Biến đơn và mảng (1 chiều).
  + Chuỗi ký tự.
  + Hàm (truyền tham chiếu, con trỏ hàm).
  + Cấu trúc (struct) và mảng struct.
  + Cấp phát động và giải phóng bộ nhớ động.
  + Cấu trúc dữ liệu động đơn giản như danh sách liên kết đơn.

## **1.4 Cấu trúc báo cáo :**

### 1.4.1 Chương 1: Mở đầu

### 1.4.2 Chương 2: Tìm hiểu Con trỏ

### 1.4.3 Chương 3: Con trỏ mảng

### 1.4.4 Chương 4: Mảng con trỏ

### 1.4.5 Chương 5: Cấp phát động

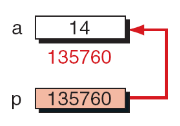
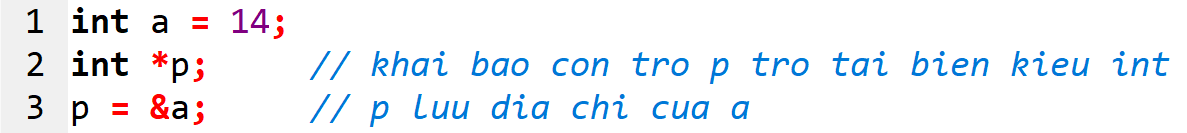
### 1.4.6 Chương 6: Danh sách liên kết

### 1.4.7 Chương 7: Ứng dụng

# **CHƯƠNG II: TÌM HIỂU CON TRỎ**

**2.1 Khái niệm con trỏ:**

* Con trỏ (pointer) là một biến lưu địa chỉ của một biến khác.
* Con trỏ không lưu giá trị trực tiếp, mà lưu vị trí (address) của ô nhớ nơi lưu giá trị đó.



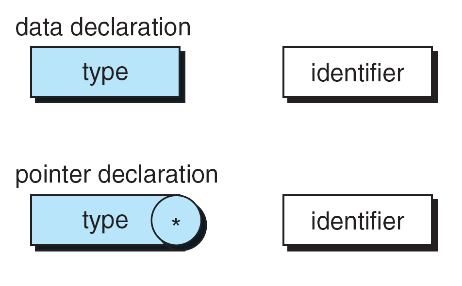
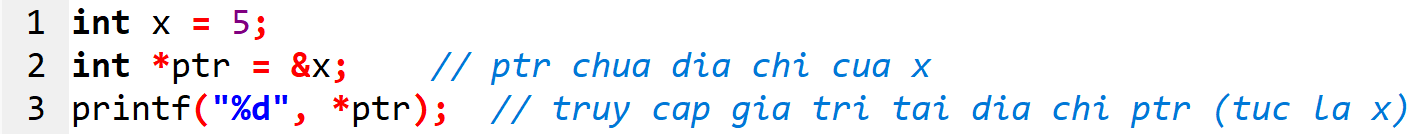
**Hình 2.1: Khái niệm con trỏ**

## **2.2 Ký hiệu và cách sử dụng:**

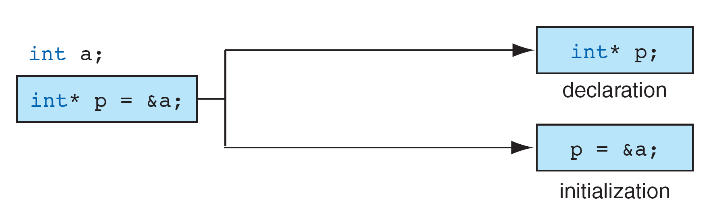
|  |  |
| --- | --- |
| * \* | Dùng để khai báo hoặc giải tham chiếu con trỏ |

|  |  |
| --- | --- |
| * & | Lấy địa chỉ của một biến |

* Khai báo con trỏ: int \*p;
* Gán địa chỉ: p = &a;
* Lấy giá trị tại địa chỉ: \*p
* Gán giá trị qua con trỏ: \*p = 5;



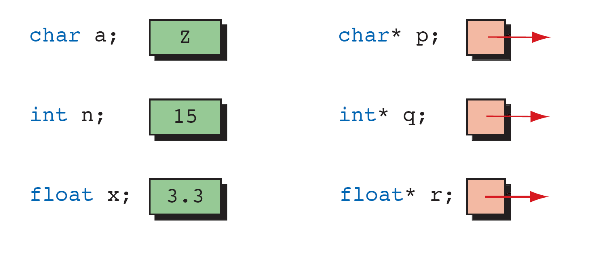
**Hình 2.2: Cách sử dụng con trỏ**



**Hình 2.3: Gộp khai báo và gán**

## **2.3 Phân loại con trỏ:**

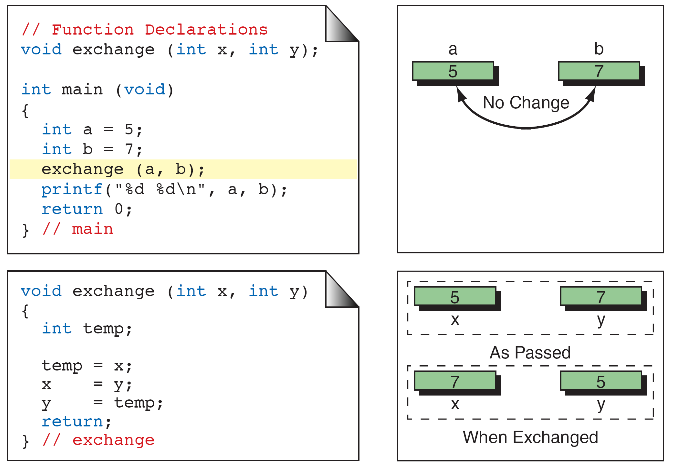
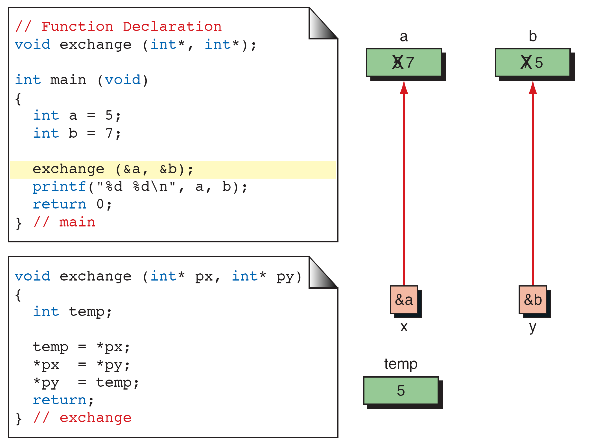
* int \*p: con trỏ tới số nguyên
* float \*p: con trỏ tới số thực
* char \*p: con trỏ tới ký tự hoặc chuỗi
* void \*p: con trỏ không kiểu – dùng linh hoạt, thường thấy trong thư viện

 **Hình 2.4: Kiểu dữ liệu**

## **2.4 Ứng dụng của con trỏ:**

|  |  |
| --- | --- |
| Ứng dụng | Mô tả |
| Truyền tham chiếu | Truyền địa chỉ biến vào hàm, thay đổi được giá trị gốc |
| Quản lý mảng động | Sử dụng malloc, calloc, free để cấp phát/giải phóng bộ nhớ |
| Làm việc với chuỗi | char \* để thao tác chuỗi dễ dàng hơn |
| Cấu trúc dữ liệu động | Danh sách liên kết, cây, ngăn xếp, hàng đợi... |
| Con trỏ hàm | Cho phép truyền hàm như biến, ứng dụng trong callback |
| … | … |

* 1 Ứng dụng nổi bật là hoán đổi giá trị biến:

****

**Hình 2.6: Hàm hoán đổi sau khi dùng con trỏ**

**Hình 2.5: Vấn đề hoán đổi biến bằng hàm nếu không dùng con trỏ**

## **2.5 Lỗi phổ biến khi dùng con trỏ:**

* Dangling Pointer: trỏ tới vùng nhớ đã giải phóng.
* Memory Leak: cấp phát mà không free.
* Uninitialized Pointer: dùng con trỏ trước khi gán địa chỉ hợp lệ.
* Pointer Arithmetic sai cách: thao tác sai dẫn đến lỗi logic.

# **CHƯƠNG III : CON TRỎ MẢNG**

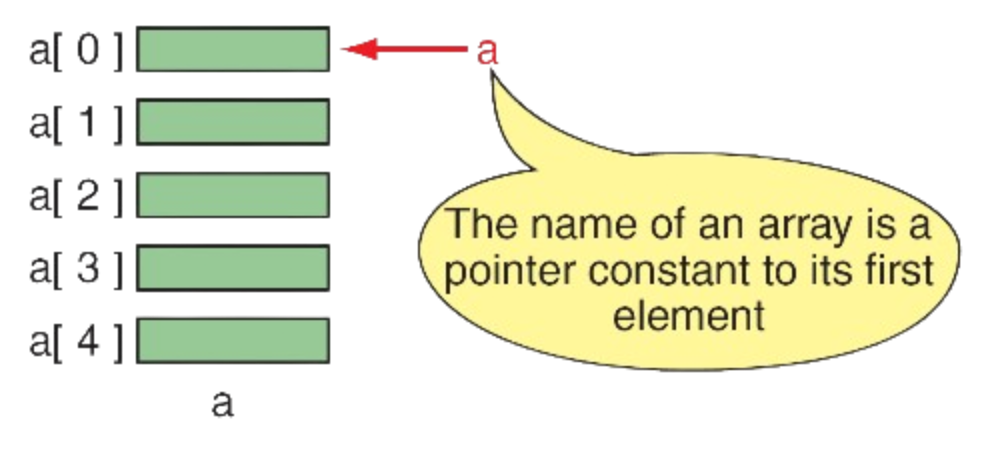
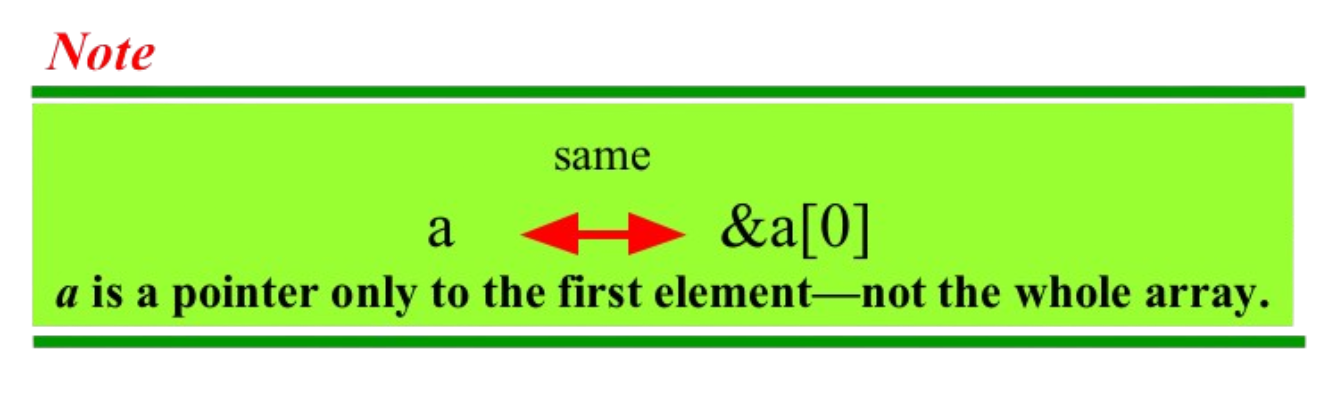
## **Định nghĩa Con trỏ mảng:**

Nó là một con trỏ trỏ tới một mảng (hoặc một phần tử trong mảng), và có thể được sử dụng để thao tác với các phần tử trong mảng thông qua việc truy cập các ô nhớ mà mảng lưu trữ.

### **3.1.1 Arrays and Pointers:**

Tên của một mảng là một con trỏ hằng chỉ tới phần tử đầu tiên. Vì tên mảng là một con trỏ hằng, giá trị của nó không thể thay đổi. Và vì tên mảng là một con trỏ hằng chỉ tới phần tử đầu tiên, địa chỉ của phần tử đầu tiên và tên mảng đều đại diện cho cùng một vị trí trong bộ nhớ.

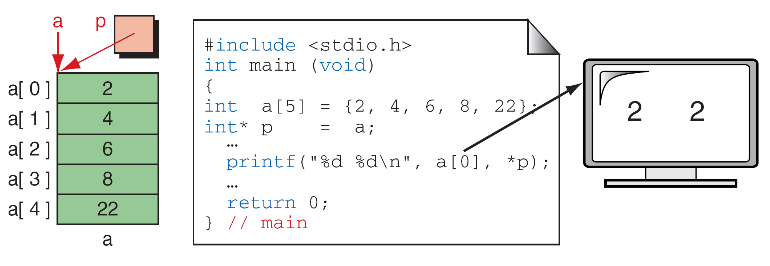
**Hình 3.1: Tên mảng là con trỏ chỉ phần tử đầu tiên**



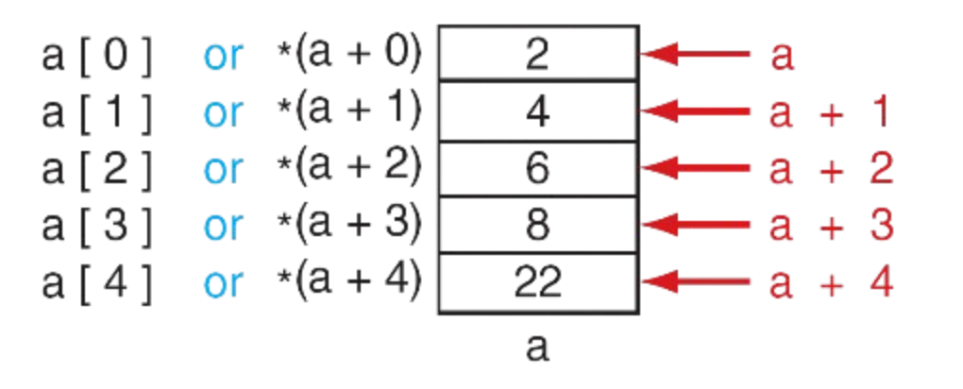


**Hình 3.2: Ví dụ tên mảng tương đương con trỏ chỉ phần tử đầu tiên**

**Hình 3.3: Con trỏ chỉ đến tên mảng**

****

### **3.1.2 Pointer Arithmetic and Arrays:**

* Cạnh việc sử dụng chỉ số, lập trình viên còn dùng một phương pháp mạnh mẽ khác để duyệt qua mảng: phép toán trên con trỏ. Phép toán trên con trỏ cung cấp một tập hợp giới hạn các toán tử số học để thao tác với địa chỉ trong con trỏ.
* Cho một con trỏ p, thì p ± n là con trỏ trỏ đến giá trị cách p n phần tử.

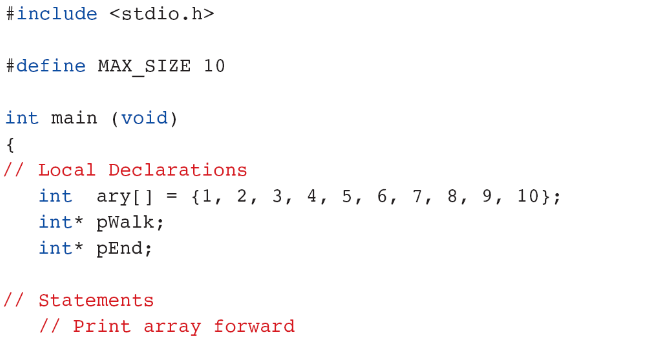
**Hình 3.4: Con trỏ số học và mảng**

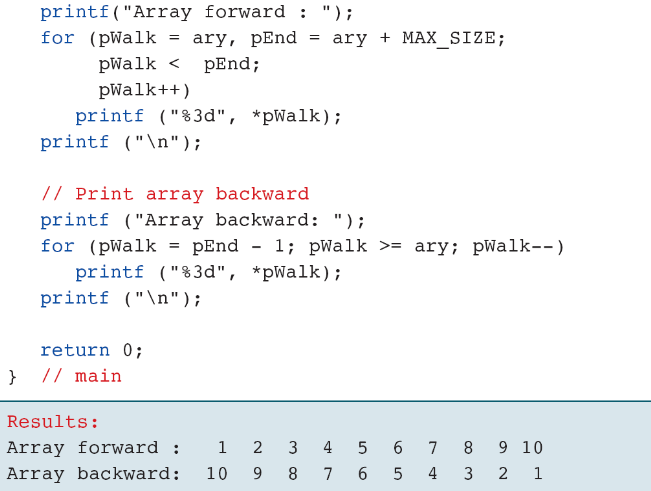
# 

## **3.2** **Ứng dụng con trỏ mảng:**

**Hình 3.5: Hai cách viết con trỏ số học tương đương**

|  |  |
| --- | --- |
| Ứng dụng | Mô tả |
| Duyệt mảng bằng con trỏ | Truy cập các phần tử thông qua phép toán địa chỉ thay vì chỉ số. |
| Cấp phát động mảng | Sử dụng malloc, calloc, realloc để tạo mảng có kích thước linh hoạt. |
| Truyền mảng vào hàm | Dễ dàng truyền mảng bằng con trỏ để tránh sao chép dữ liệu lớn. |
| Làm việc với mảng chuỗi (mảng con trỏ) | Quản lý danh sách chuỗi như tên, mô tả, từ điển... |
| Truy xuất và thao tác với mảng struct thông qua con trỏ | Dùng để xử lý danh sách struct như sinh viên, phân số, sản phẩm... |
|  |  |
|  |  |
|  |  |

* 1 ứng dụng nổi bật là đảo ngược 1 dãy số:



**Hình 3.6: Đảo ngược 1 dãy số bằng con trỏ mảng**

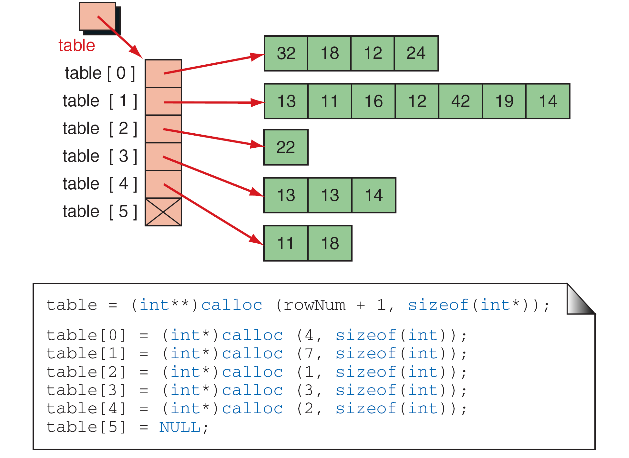
# **CHƯƠNG IV : MẢNG CON TRỎ**

## **4.1 Định nghĩa mảng con trỏ :**

* Một cấu trúc hữu ích khác sử dụng mảng và con trỏ là mảng các con trỏ.  
  Cấu trúc này đặc biệt hữu ích khi số lượng phần tử trong mảng là thay đổi.
* Kiểu dữ liệu: type \*arrayName[size];
* Truy cập phần tử: Dùng chỉ số như mảng bình thường: array[i], nhưng mỗi array[i] là một con trỏ.
* Cấp phát động: Có thể cấp phát từng phần tử của mảng con trỏ bằng malloc.

## **4.2 Ứng dụng :**

|  |  |
| --- | --- |
| Loại mảng con trỏ | Ứng dụng thực tế |
| char \*str[] | Danh sách tên, câu, chuỗi. |
| int \*arr[] | Mảng chứa các mảng động – tạo mảng 2 chiều linh hoạt. |
| struct Student \*list[] | Mảng chứa các con trỏ đến struct sinh viên. |

* Cấp phát bộ nhớ mảng 2 chiều không đồng đều: A Ragged Table

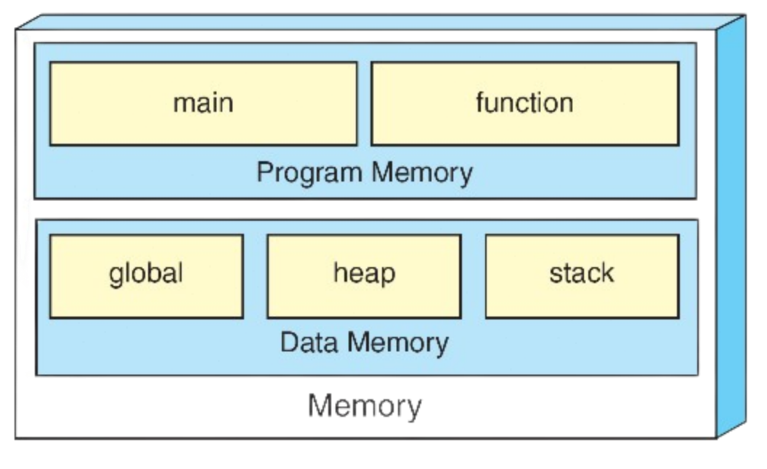
**Hình 4.1: Ví dụ cấp phát động mảng 2 chiều sử dụng mảng con trỏ**

* Giải thích:

|  |  |
| --- | --- |
| int \*\*table | Mảng con trỏ hai chiều không đều (Ragged array). |
| table[i] | Là int\*, trỏ đến mảng động từng hàng. |
| calloc | Dùng để cấp phát vùng nhớ động đã khởi tạo = 0. |
| NULL ở cuối | Dễ kiểm tra điểm kết thúc, giống chuỗi '\0'. |

# **CHƯƠNG V : CẤP PHÁT ĐỘNG**

Ngôn ngữ C cung cấp cho chúng ta hai lựa chọn khi muốn cấp phát vùng nhớ cho một đối tượng: cấp phát tĩnh và cấp phát động.



**Hình 5.1: Cách nhìn tổng quan về bộ nhớ**

* Vùng Data memory:

|  |  |
| --- | --- |
| Vùng | Mô tả |
| global | Dành cho biến toàn cục (global variables) và biến tĩnh (static) – tồn tại trong suốt thời gian chạy chương trình. |
| heap | Dành cho cấp phát động (dynamic allocation) qua malloc, calloc, realloc, free. Người lập trình quản lý thủ công. |
| stack | Dành cho biến cục bộ, tham số hàm, và quản lý gọi hàm lồng nhau (call stack). Tự động cấp phát/thu hồi khi vào/thoát hàm. |

* Liên hệ với cấp phát bộ nhớ:

|  |  |  |
| --- | --- | --- |
| Loại cấp phát | Vùng nhớ | Cách sử dụng |
| Tĩnh (static) | Global / Data | Khai báo biến toàn cục, hoặc static trong hàm. |
| Động (dynamic) | Heap | Dùng con trỏ + malloc, calloc, free. |
| Tự động (auto) | Stack | Biến trong hàm, tự giải phóng khi hàm kết thúc. |

* Chúng ta chỉ có thể truy xuất bộ nhớ được cấp phát trên heap thông qua một con trỏ.

## **5.1 Cấp phát động :**

* Cấp phát động trong C (Dynamic Memory Allocation in C) là quá trình cấp phát bộ nhớ trong thời gian chạy (runtime) thay vì trong thời gian biên dịch (compile time). Điều này cho phép chương trình linh hoạt hơn khi làm việc với dữ liệu có kích thước không xác định trước**.**

**Hình 5.2: Các hàm quản lý bộ nhớ**

### **5.1.1 Malloc :**

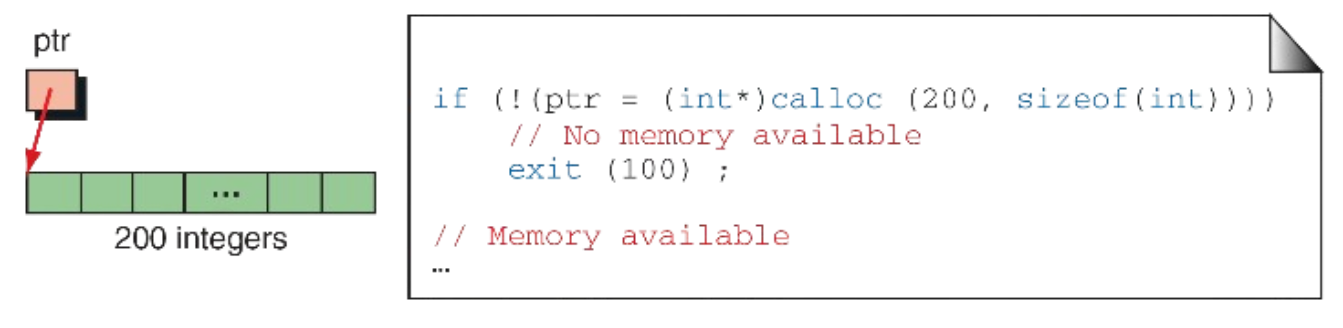
**Hình 5.3: Malloc()**

|  |
| --- |
| * Cấp phát một vùng nhớ có kích thước **‘**size**’** byte. Nội dung không được khởi tạo. |

**void\* malloc(size\_t size);**

* Lưu ý: Ép kiểu khi cấp phát bộ nhớ:
* Trước C99, việc ép kiểu con trỏ được trả về từ hàm cấp phát bộ nhớ là bắt buộc. Mặc dù hiện nay (từ C99 trở đi) điều đó không còn cần thiết, nhưng nếu ép kiểu đúng, nó cũng không gây hại gì.
* Cú pháp ép kiểu sẽ như sau:pointer = (type\*) malloc(size);
* Trong C++, việc ép kiểu là bắt buộc vì C++ không tự động chuyển void\* sang kiểu con trỏ khác như C.

**5.1.2 Calloc :**

* Cấp phát một mảng gồm n phần tử, mỗi phần tử có kích thước size. Nội dung được khởi tạo về 0. 

**Hình 5.4: Calloc()**

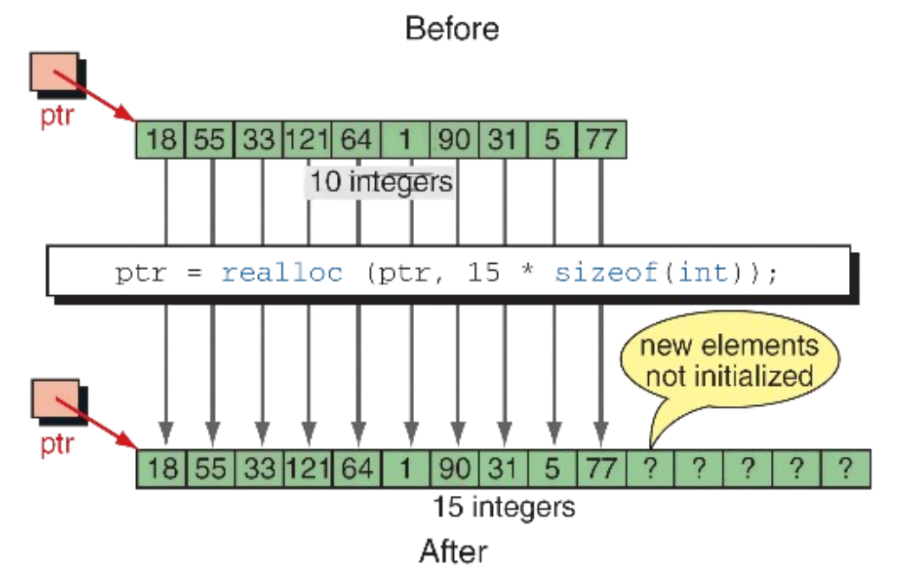
**void\* calloc(size\_t num, size\_t size);**

* num: số lượng phần tử cần cấp phát.
* size: kích thước của mỗi phần tử (tính bằng byte).
* Hàm trả về một con trỏ kiểu void\* trỏ đến vùng nhớ được cấp phát.

**Khác biệt giữa malloc và calloc:**

|  |  |  |
| --- | --- | --- |
| Đặc điểm | malloc | calloc |
| Khởi tạo dữ liệu | Không khởi tạo | Khởi tạo tất cả phần tử về 0 |
| Tham số | 1 (tổng số byte) | 2 (số phần tử và kích thước mỗi phần tử) |
| Cú pháp | malloc(n \* sizeof(T)) | calloc(n, sizeof(T)) |

**5.1.3 Realloc :**

* Thay đổi kích thước vùng nhớ được trỏ bởi ptr.

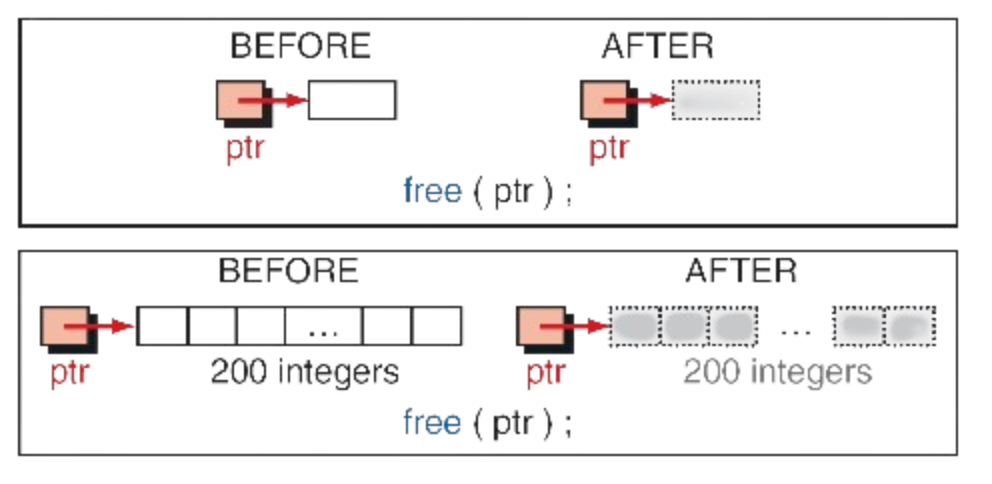
**Hình 5.5: Realloc()**

**void\* realloc(void\* ptr, size\_t new\_size);**

* ptr: con trỏ trỏ đến vùng nhớ đã được cấp phát trước đó (bằng malloc, calloc hoặc realloc).
* new\_size: kích thước mới (tính bằng byte).
* Trả về:
* Con trỏ mới đến vùng nhớ đã được thay đổi kích thước.
* Trả về NULL nếu không đủ bộ nhớ (và vùng nhớ cũ vẫn còn, không bị mất).
* Chức năng chính:
* Dùng để mở rộng hoặc thu hẹp vùng nhớ đã cấp phát trước đó.
* Nếu mở rộng, vùng mở rộng thêm có thể chứa giá trị rác.
* Nếu thu hẹp, dữ liệu vượt quá new\_size sẽ bị mất.
* Lưu ý:

|  |  |
| --- | --- |
| Tình huống | Hành động |
| ptr == NULL | Tương đương với malloc(new\_size) |
| new\_size == 0 | Tương đương với free(ptr) và trả về NULL |
| realloc thất bại | Trả về NULL, vùng nhớ cũ vẫn còn và phải được free() nếu không còn dùng |

**5.1.4 Free :**

* Giải phóng vùng nhớ đã cấp phát

**Hình 5.6: Free()**

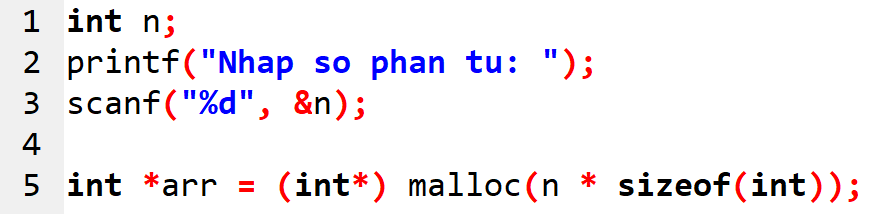
**free(pointer);**

* Chức năng:
* Giải phóng vùng nhớ trên heap mà con trỏ đang trỏ tới.
* Tránh gây rò rỉ bộ nhớ (memory leak) nếu chương trình cấp phát nhiều lần nhưng không giải phóng.
* Lưu ý:

|  |  |
| --- | --- |
| Điều cần tránh | Ghi chú |
| Không free() một vùng chưa được malloc() | Dẫn đến lỗi không xác định |
| Không free() cùng một vùng nhiều lần | Gây lỗi chương trình (double free) |
| Sau khi free(), nên gán con trỏ về NULL | Tránh dùng nhầm vùng đã giải phóng |

## **5.2 Ứng dụng cấp phát động :**

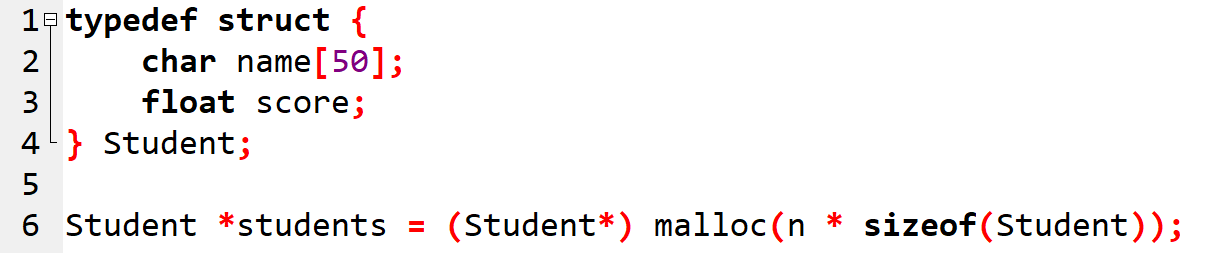
* + - Làm việc với mảng có kích thước không xác định trước :

Khi bạn không biết trước số lượng phần tử cần lưu (do người dùng nhập vào). 

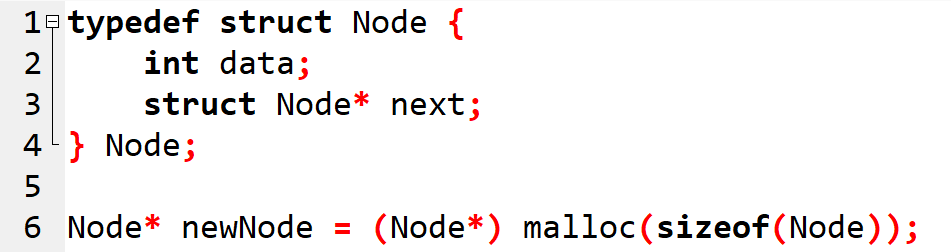
**Hình 5.7: Ứng dụng malloc**

* + - Mở rộng hoặc thu hẹp mảng bằng realloc :

**Hình 5.8: Ứng dụng realloc**

* + - **Quản lý danh sách sinh viên, danh sách sách... với struct :**

**Hình 5.9: Ứng dụng malloc trong struct**

* Tạo danh sách liên kết (linked list), cây (tree), đồ thị (graph) :

**Hình 5.10: Ứng dụng trong danh sách liên kết**

# **CHƯƠNG VI : DANH SÁCH LIÊN KẾT**

* Ngôn ngữ C không cung cấp sẵn bất kỳ cấu trúc hay cách triển khai danh sách nào. Khi cần sử dụng, chúng ta phải tự xây dựng các cấu trúc dữ liệu và hàm xử lý cho chúng. Theo truyền thống, hai kiểu dữ liệu thường được sử dụng để triển khai danh sách là mảng và con trỏ. 

**Hình 6.1: Linked list**

* Danh sách tuyến tính tổng quát là một danh sách cho phép thực hiện các thao tác như truy xuất, chèn, thay đổi và xóa ở bất kỳ vị trí nào trong danh sách, tức là ở đầu, giữa hoặc cuối danh sách.

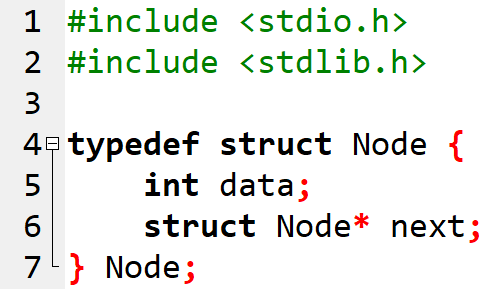
## **6.1 Node :**

* Trong ngôn ngữ lập trình C, node trong danh sách liên kết (linked list) là một cấu trúc (struct) dùng để lưu trữ một phần tử của danh sách, cùng với con trỏ trỏ đến phần tử kế tiếp trong danh sách. Đây là thành phần cơ bản để tạo ra các danh sách liên kết đơn, đôi hoặc vòng.

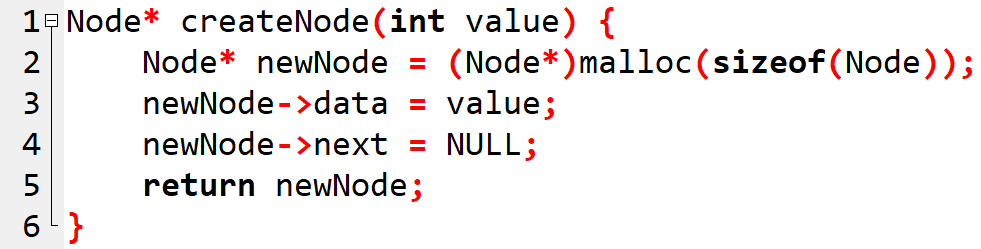
### **6.1.1 Tạo Node :**

**Hình 6.2: Cấu trúc node danh sách liên kết**

* Khai báo cấu trúc Node bằng struct:



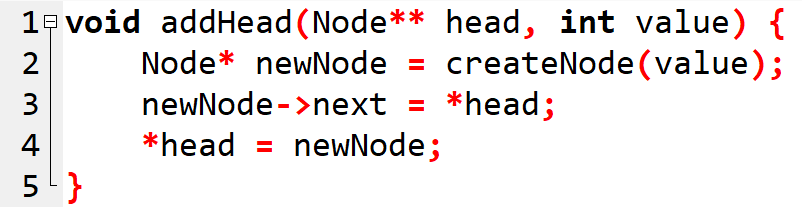
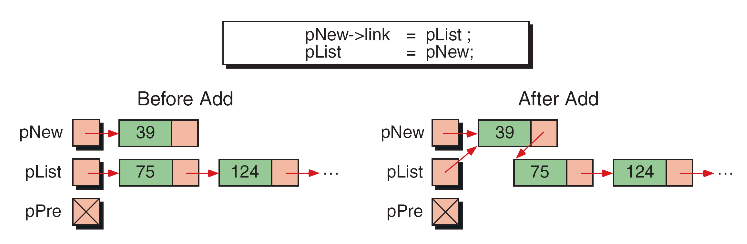
**Hình 6.3: Định nghĩa cấu trúc Node**

* Cấp phát bộ nhớ động và tạo một node mới:

**Hình 6.4: Hàm tạo một Node mới**

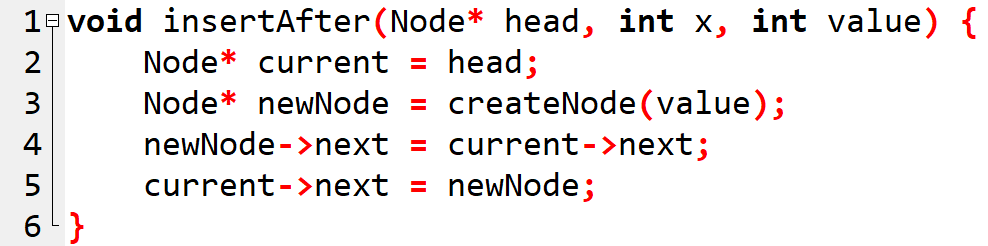
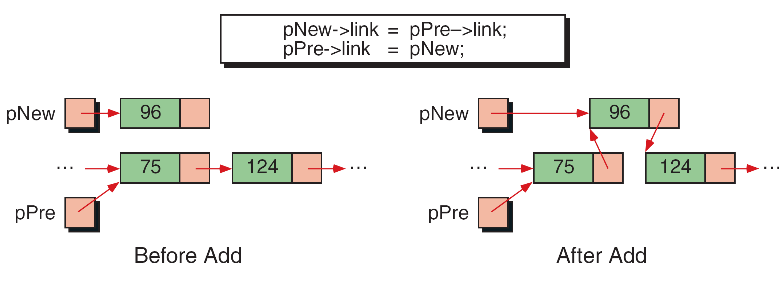
### **6.1.2 Thêm Node đầu :**

**Hình 6.4: Thêm một Node vào đầu danh sách**



### **6.1.3 Thêm Node vào giữa :**

* Để thực hiện ta sẽ tạo một Node (current) có nhiệm vụ quét danh sách (walk, traversal, …) dừng lại khi tìm thấy Node cần tìm, với mục đích đẩy phần đó xuống cuối Node mới link tới và Node mới sẽ thêm vào cuối danh sách.

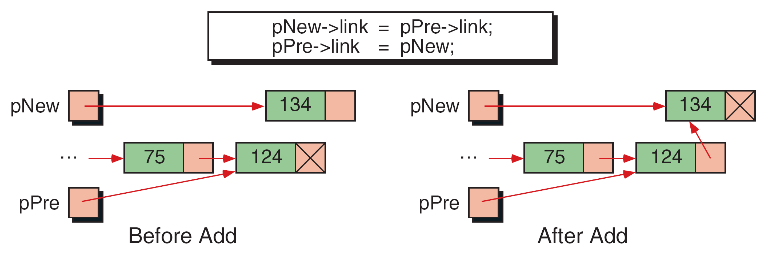
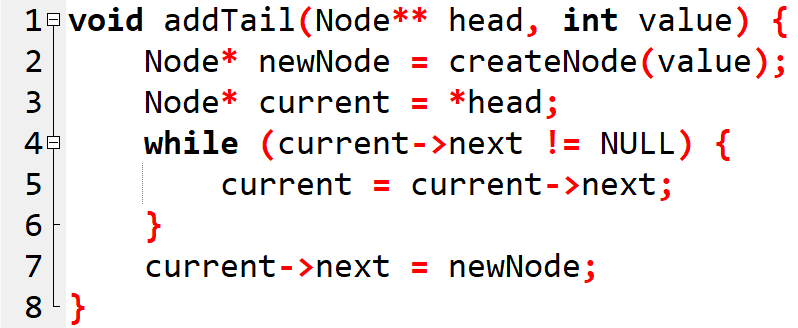


**Hình 6.5: Thêm một Node vào giữa danh sách**

### **6.1.4 Thêm Node vào cuối :**

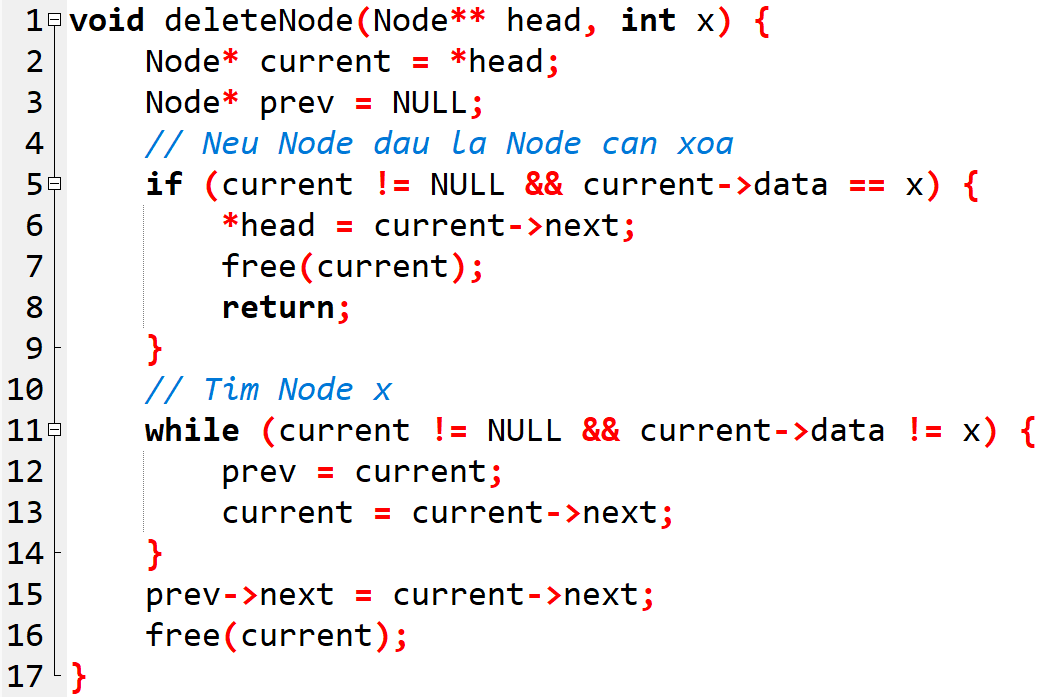
* Ta cũng tạo một Node current chạy đến Node cuối cùng (-> next = NULL) thì dừng và thêm Node mới vào.

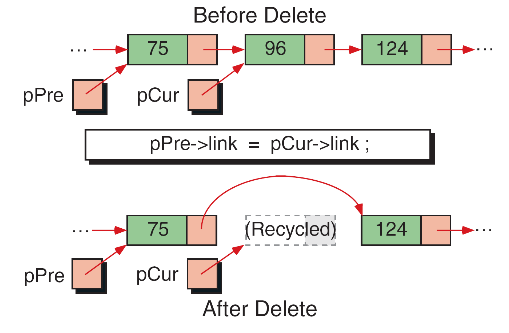
**Hình 6.6: Thêm một Node vào cuối danh sách**



### **6.1.5 Xóa Node :**

* Ta thêm 2 Node là current và previous, với mục đích prev luôn link tới cur như vậy khi xóa chỉ cần prev link tới cur->next.





**Hình 6.7: Xóa một Node trong danh sách**

## **6.2 Kết luận :**

* Danh sách liên kết là một trong những cấu trúc dữ liệu nền tảng trong khoa học máy tính. Nó không chỉ dùng riêng lẻ, mà còn là nền móng để xây dựng các cấu trúc phức tạp hơn như:
* Stack (ngăn xếp)
* Queue (hàng đợi)
* Hash table
* Cây (tree), đồ thị (graph)
* Trong nhiều ứng dụng thực tế như hệ điều hành, cơ sở dữ liệu, trình quản lý bộ nhớ, và hệ thống xử lý dữ liệu lớn, danh sách liên kết cho phép:
* Dễ dàng thêm/xóa dữ liệu mà không cần dịch chuyển
* Tối ưu hiệu năng trong điều kiện không biết trước kích thước dữ liệu

| * Khi nào dùng danh sách liên kết? |
| --- |
| * Dữ liệu thay đổi thường xuyên (thêm/xóa giữa, đầu/cuối). |
| * Không biết trước số lượng phần tử. |
| * Ưu tiên linh hoạt hơn tốc độ. |

# **CHƯƠNG VII : ỨNG DỤNG**

Áp dụng con trỏ, struct, danh sách liên kết, cấp phát động để tạo một menu chương trình quản lý bệnh nhân đặt lịch thăm khám với các lựa chọn:

* 1. Nhập số lượng bệnh nhân n với: Tên, tuổi, lý do khám bệnh, đặt giờ (24/24)
  2. Đặt lịch dựa vào danh sách bệnh nhân đã nhập từ 1.
  3. In danh sách lịch thăm khám.
  4. Tìm bệnh nhân theo tên.
  5. Hủy lịch khám theo tên.
  6. Thoát chương trình.

**Mã nguồn :** Github: <https://github.com/lequocdat8272/BaoCao>

# **CHƯƠNG VIII : KẾT QUẢ VÀ KIẾN NGHỊ**

## **8.1 Kết quả đạt được :**

Qua quá trình tìm hiểu và thực hành về con trỏ trong ngôn ngữ C, đặc biệt là các khái niệm nâng cao như mảng con trỏ, con trỏ mảng, cấp phát động và danh sách liên kết, bài báo cáo đã đạt được các kết quả sau:

* Nắm vững kiến thức cơ bản về con trỏ, bao gồm: khai báo, khởi tạo, truy cập địa chỉ và nội dung biến thông qua con trỏ.
* Phân biệt và sử dụng đúng mảng con trỏ và con trỏ mảng:
  + Biết cách khai báo và sử dụng char \*arr[] (mảng con trỏ) để lưu nhiều chuỗi ký tự.
  + Hiểu cách sử dụng int (\*arr)[n] (con trỏ mảng) để truy xuất và thao tác với ma trận.
* Thực hành cấp phát động bộ nhớ bằng malloc, calloc, realloc, và biết giải phóng bằng free để tránh rò rỉ bộ nhớ.
* Xây dựng và thao tác trên danh sách liên kết đơn, bao gồm:
  + Tạo node, thêm/xóa node ở đầu/cuối/dựa theo giá trị.
  + Duyệt, tìm kiếm và nối danh sách.
* Rèn luyện kỹ năng tư duy con trỏ và quản lý bộ nhớ thủ công trong lập trình C – nền tảng quan trọng để học các cấu trúc dữ liệu phức tạp hơn sau này.

## **8.2 Kiến nghị :**

Dựa trên quá trình học và làm báo cáo, em xin đưa ra một số kiến nghị để nâng cao hiệu quả học tập:

* Thực hành thường xuyên với nhiều dạng bài tập về con trỏ.
* Minh họa trực quan bằng sơ đồ khối, cây trong quá trình học.
* Đọc và thực hành nhiều loại sách về lập trình đặc biệt là sách tiếng Anh.
* Tham khảo thầy cô, bạn học hoặc trên những forum(Stack Overflow) lập trình.

# **TÀI LIỆU THAM KHẢO**

[1] Sách :” The C Programming Language 2nd Ed - Prentice Hall - Brian.W.Kernighan and Dennis.M.Ritchie” – Truy cập [5/2025]

[2] Sách :” Computer Science: A Structured Programming Approach Using C” – Bài giảng Eng từ thầy Trần Phong Nhã – Truy cập [5/2025]

[3] <https://blog.28tech.com.vn/c-ly-thuyet-con-tro> – [C]. Lý Thuyết Con Trỏ - Truy cập [5/2025]

[4] <https://blog.28tech.com.vn/c-con-tro-va-mang-1-chieu> – [C]. Con Trỏ Và Mảng 1 Chiều - Truy cập [5/2025]

[5] <https://blog.28tech.com.vn/c-malloc-calloc-free-va-realloc> – [C]. malloc(), calloc(), free() và realloc() - Truy cập [5/2025]

[6] <https://blog.28tech.com.vn/dslk-don-danh-sach-lien-ket-don-singly-linked-list> - [DSLK Đơn]. Danh Sách Liên Kết Đơn - Singly Linked List - Truy cập [5/2025]

[7] <https://vietjack.com/cau-truc-du-lieu-va-giai-thuat/danh-sach-lien-ket-trong-c.jsp> - Danh sách liên kết (Linked List) trong C - Truy cập [5/2025]

[8] <https://vietjack.com/thu-vien-c/stdlib-h-trong-c.jsp> - stdlib.h trong C - Truy cập [5/2025]