**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN HỌC:**

**KĨ THUẬT LẬP TRÌNH**

**Giảng viên hướng dẫn: ThS. TRẦN PHONG NHÃ**

**Sinh viên thực hiện: ĐINH AN BANG**

**Lớp: CQ.65.CNTT**

**Khoá: 65**

**Mssv: 6551071006**

Tp. Hồ Chí Minh, tháng 5 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 HỌC:**

**KĨ THUẬT LẬP TRÌNH**

**Giảng viên hướng dẫn: ThS. TRẦN PHONG NHÃ**

**Sinh viên thực hiện: ĐINH AN BANG**

**Lớp: CQ.65.CNTT**

**Khoá: 65**

**Mssv: 6551071006**

Tp. Hồ Chí Minh, tháng 5 năm 2025

# LỜI CẢM ƠN

Em xin chân thành cảm ơn thầy và các bạn sinh viên đã hỗ trợ và đồng hành trong quá trình thực hiện bài báo cáo tiểu luận này.

Đặc biệt, em xin gửi lời cảm ơn sâu sắc đến các giảng viên và người hướng dẫn đã tận tình truyền đạt kiến thức và định hướng giúp em hoàn thành tốt nội dung báo cáo.

Em cũng trân trọng cảm ơn gia đình và bạn bè vì đã luôn động viên và tạo điều kiện thuận lợi cho em trong suốt quá trình học tập và làm bài.

Cuối cùng, em cảm ơn thầy cô đặc biệt là thầyTRẦN PHONG NHÃ dành thời gian đọc và góp ý cho báo cáo. Những phản hồi quý báu sẽ là động lực để em tiếp tục hoàn thiện bản thân trong tương lai.

Xin chân thành cảm ơn!

# NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

|  |
| --- |
| ***Tp. Hồ Chí Minh, ngày …tháng… năm …*** |
| **Giảng viên hướng dẫn** |
| **ThS. TRẦN PHONG NHÃ** |

# MỤC LỤC

[LỜI CẢM ƠN i](#_Toc198758667)

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN ii](#_Toc198758668)

[MỤC LỤC iii](#_Toc198758669)

[DANH MỤC HÌNH ẢNH vi](#_Toc198758670)

[CHƯƠNG 1: GIỚI THIỆU VỀ NGÔN NGỮ LẬP TRÌNH C 1](#_Toc198758671)

[1. Khái quát cơ bản và lý thuyết cần nắm của ngôn ngữ lập trình c 1](#_Toc198758672)

[1.1: Tổng quan 1](#_Toc198758673)

[1.2: Ứng dụng 1](#_Toc198758674)

[1.3: Nội dung lý thuyết chúng ta sẽ tìm hiểu 1](#_Toc198758675)

[CHƯƠNG 2: LÝ THUYẾT 2](#_Toc198758676)

[1: HÀM 2](#_Toc198758677)

[1.1: Khái niệm của hàm trong ngôn ngữ lập trình c 2](#_Toc198758678)

[1.2: Cú pháp chung của 1 hàm trong ngôn ngữ lập trình c 2](#_Toc198758679)

[1.3: Các loại hàm trong c 2](#_Toc198758680)

[1.4: Tầm quan trọng của hàm trong c 5](#_Toc198758681)

[2. CON TRỎ 6](#_Toc198758682)

[2.1: Tổng quát con trỏ trong ngôn ngữ lập trình c 6](#_Toc198758683)

[2.2: Tại sao con trỏ lại quan trọng 6](#_Toc198758684)

[2.3: Cách khai báo và khởi tạo một con trỏ (pointer) 7](#_Toc198758685)

[2.4: Toán tử & và \* trong con trỏ 7](#_Toc198758686)

[3. CON TRỎ MẢNG 8](#_Toc198758687)

[3.1: Khái niệm con trỏ mảng 8](#_Toc198758688)

[3.2: Ví dụ minh họa 8](#_Toc198758689)

[4. MẢNG CON TRỎ 9](#_Toc198758690)

[4.1: Khái niệm con trỏ hàm 9](#_Toc198758691)

[4.2: Lưu ý khi dùng Mảng con trỏ 9](#_Toc198758692)

[4.3: Cú pháp 9](#_Toc198758693)

[4.4: Ví dụ minh họa 10](#_Toc198758694)

[5. CON TRỎ HÀM 11](#_Toc198758695)

[5.1:Tổng quan và khái niệm con trỏ hàm 11](#_Toc198758696)

[5.2: Lưu ý khi sử dụng con trỏ hàm 12](#_Toc198758697)

[5.3: Ví dụ minh họa 12](#_Toc198758698)

[6. CẤP PHÁT ĐỘNG 13](#_Toc198758699)

[6.1 : Khái niệm về cấp phát động trong ngôn ngữ c 13](#_Toc198758700)

[6.2: Ví dụ minh họa 14](#_Toc198758701)

[6.3 : Lợi ích khi dùng cấp phát động với thư viện #include <stdlib.h> 14](#_Toc198758702)

[7. XỬ LÍ TỆP 15](#_Toc198758703)

[7.1: Khái niệm về xử lí tệp trong c 15](#_Toc198758704)

[7.2: Ưu điểm của xử lí tệp trong ngôn ngữ c 16](#_Toc198758705)

[7.3: Ví dụ minh họa 17](#_Toc198758706)

[8. KIỂU CẤU TRÚC 18](#_Toc198758707)

[8.1: Khái niệm kiểu cấu trúc 18](#_Toc198758708)

[8.2: Cú pháp kiểu cấu trúc 18](#_Toc198758709)

[8.3: Ví dụ minh họa 18](#_Toc198758710)

[9. DANH SÁCH LIÊN KẾT 20](#_Toc198758711)

[9.1: Khái niệm danh sách liên kết 20](#_Toc198758712)

[9.2 Tiện ích khi sử dụng danh sách liên kết: 20](#_Toc198758713)

[9.3: Liệu danh sách liên kết có tối ưu hóa hơn mảng không? 21](#_Toc198758714)

[9.4: Ví dụ minh họa 22](#_Toc198758715)

[CHƯƠNG 3: ỨNG DỤNG 24](#_Toc198758716)

[1. Ý TƯỞNG 24](#_Toc198758717)

[2. TÓM TẮT CHƯƠNG TRÌNH CHÍNH 24](#_Toc198758718)

[TÀI LIỆU THAM KHẢO 27](#_Toc198758719)

# DANH MỤC HÌNH ẢNH

[Hình 1.1: Cú pháp chung của 1 hàm 2](#_Toc198759002)

[Hình 1.2: Hàm không có tham số và không trả về giá trị 3](#_Toc198759003)

[Hình 1.3: Hàm không có tham số nhưng trả về giá trị 3](#_Toc198759004)

[Hình 1.4: Hàm có tham số và có trả về giá trị 4](#_Toc198759005)

[Hình 1.5: Hàm có tham số nhưng không trả về giá trị 4](#_Toc198759006)

[Hình 2.1: Ví dụ con trỏ 6](#_Toc198759007)

[Hình 2.2: Ví dụ khai báo con trỏ 7](#_Toc198759008)

[Hình 2.3: Lưu ý về gán giá trị NULL cho con trỏ 7](#_Toc198759009)

[Hình 2.4: Sử dụng toán tử trong con trỏ 7](#_Toc198759010)

[Hình 3.1: Con trỏ mảng 8](#_Toc198759011)

[Hình 4.1: Ví dụ mảng con trỏ 9](#_Toc198759012)

[Hình 4.2: Mảng con trỏ 10](#_Toc198759013)

[Hình 5.1: ví dụ về con trỏ hàm. 11](#_Toc198759014)

[Hình 5.2: Ví dụ minh họa về con trỏ hàm 12](#_Toc198759015)

[Hình 6.1: ví dụ minh họa cấp phát động 14](#_Toc198759016)

[Hình 7.1: cú pháp mở tệp 15](#_Toc198759017)

[Hình 7.2: ảnh ngôn ngữ c về xử lí file 16](#_Toc198759018)

[Hình 7.3: Ví dụ minh họa xử lí ghi file 17](#_Toc198759019)

[Hình 7.4: Ví dụ minh họa về mở file để đọc 17](#_Toc198759020)

[Hình 8.1: Cú pháp của kiểu cấu trúc 18](#_Toc198759021)

[Hình 8.2: Ví dụ minh họa về struct sinh viên 19](#_Toc198759022)

[Hình 9.1: Minh họa các node cảu một danh sách liên kết. 20](#_Toc198759023)

[Hình 9.2: Cách hoạt động cảu một node 21](#_Toc198759024)

[Hình 9.3: Ví dụ minh họa danh sách liên kết đơn 23](#_Toc198759025)

[Hình 1.1: Ảnh convenience store 24](#_Toc198759026)

[Hình 2.1 : Ý tưởng ứng dụng danh sách liên kết 24](#_Toc198759027)

[Hình 2.2: Hàm chính của Product 25](#_Toc198759028)

[Hình 2.3: Hàm chính của Customer 26](#_Toc198759029)

[Hình 2.4: Hàm chính của Common 26](#_Toc198759030)

# CHƯƠNG 1: GIỚI THIỆU VỀ NGÔN NGỮ LẬP TRÌNH C

**1. Khái quát cơ bản và lý thuyết cần nắm của ngôn ngữ lập trình c**

### 1.1: Tổng quan

**C là một ngôn ngữ lập trình có cấu trúc.** Nó được xem là một ngôn ngữ bậc cao vì cho phép lập trình viên tập trung vào vấn đề cần giải quyết thay vì phải lo lắng về máy tính mà chương trình sẽ chạy trên đó. Đây cũng là một lý do khác khiến ngôn ngữ C được các nhà phát triển phần mềm sử dụng, đặc biệt khi ứng dụng của họ cần chạy trên nhiều nền tảng phần cứng khác nhau.

### 1.2: Ứng dụng

Ngôn ngữ lập trình C được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau nhờ tính hiệu quả và linh hoạt. Nó được dùng để phát triển hệ điều hành (như UNIX, Linux), lập trình nhúng trong các thiết bị như máy in, điều hòa, và thiết bị IoT. Ngoài ra, C còn được sử dụng để viết trình biên dịch, hệ thống cơ sở dữ liệu (như MySQL), phần mềm điều khiển phần cứng, và một số trò chơi yêu cầu hiệu suất cao. Với cú pháp rõ ràng và gần với ngôn ngữ máy, C cũng là ngôn ngữ phổ biến trong giảng dạy lập trình căn bản tại các trường học.

### 1.3: Nội dung lý thuyết chúng ta sẽ tìm hiểu

1. Hàm

2. Con trỏ

3. Con trỏ mảng

4. Mảng con trỏ

5. Con trỏ hàm

6. Cấp phát động

7. Xử lý tệp

8. Kiểu cấu trúc

9. Danh sách liên kết

# CHƯƠNG 2: LÝ THUYẾT

## 1: HÀM

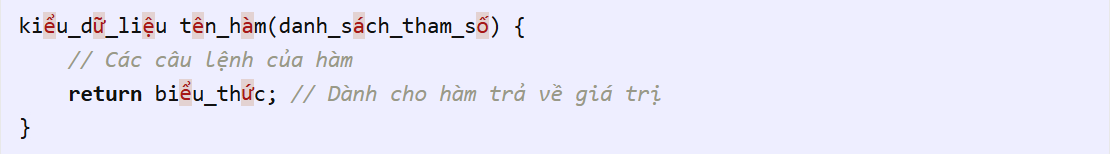
### 1.1: Khái niệm của hàm trong ngôn ngữ lập trình c

-Đầu tiên ta nên tìm hiểu hàm là gì:

+ Trong lập trình, **hàm** là một khối mã có thể tái sử dụng, thực hiện một nhiệm vụ nhất định. Hàm giúp chia nhỏ các phần của chương trình thành các khối chức năng độc lập, làm cho mã nguồn trở nên dễ đọc, dễ bảo trì và dễ kiểm soát hơn. Trong ngôn ngữ lập trình C, hàm là một phần quan trọng trong việc tổ chức và cấu trúc mã, giúp tối ưu hóa hiệu suất và giảm thiểu lỗi.

+ Hàm trong C là một khối mã thực hiện một nhiệm vụ hoặc tính toán cụ thể. Hàm giúp tổ chức mã nguồn một cách logic, giảm thiểu sự lặp lại mã và tăng tính tái sử dụng. Một hàm có thể nhận các tham số đầu vào và trả về một giá trị.

### 1.2: Cú pháp chung của 1 hàm trong ngôn ngữ lập trình c



Hình 1.1: Cú pháp chung của 1 hàm

+ kiểu\_dữ\_liệu : Xác định kiểu trả về của hàm (ví dụ: int, float, void,…).

+ tên\_hàm: bạn sẽ đặt tên cho hàm, và gọi nó ra để sử dụng.

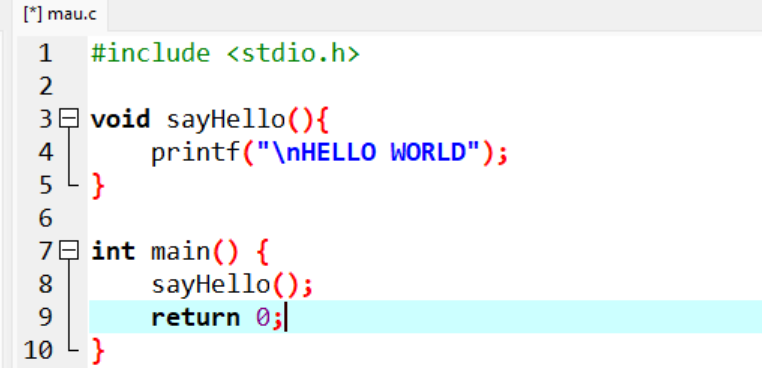
+ danh\_sách\_tham\_số: các biến nhận giá trị đầu vào, giúp hàm có thể hoạt động với các giá trị khác nhau.

### 1.3: Các loại hàm trong c

* Hàm không có tham số và không trả về giá trị
* Hàm không có tham số nhưng trả về giá trị
* Hàm có tham số và trả về giá trị
* Hàm có tham số nhưng không trả về giá trị

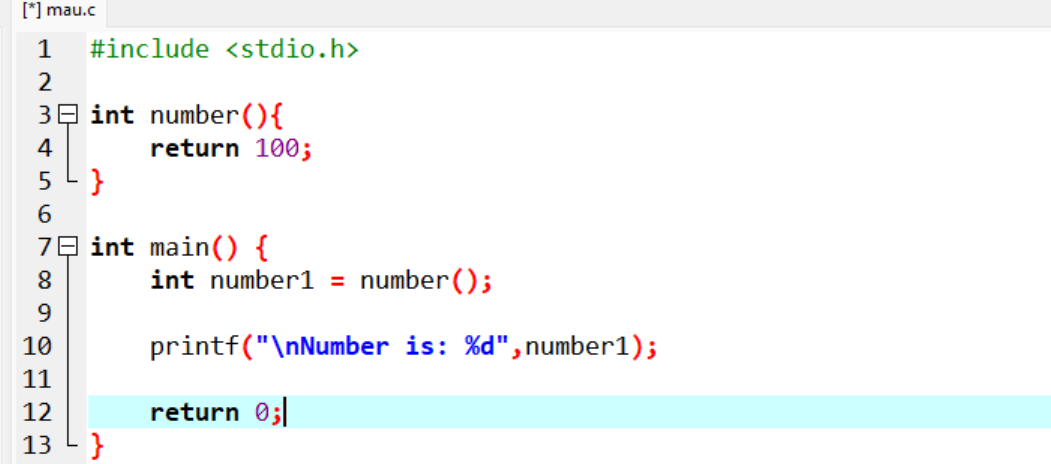
- ví dụ cho từng loại hàm:

+ Hàm không có tham số và không trả về giá trị



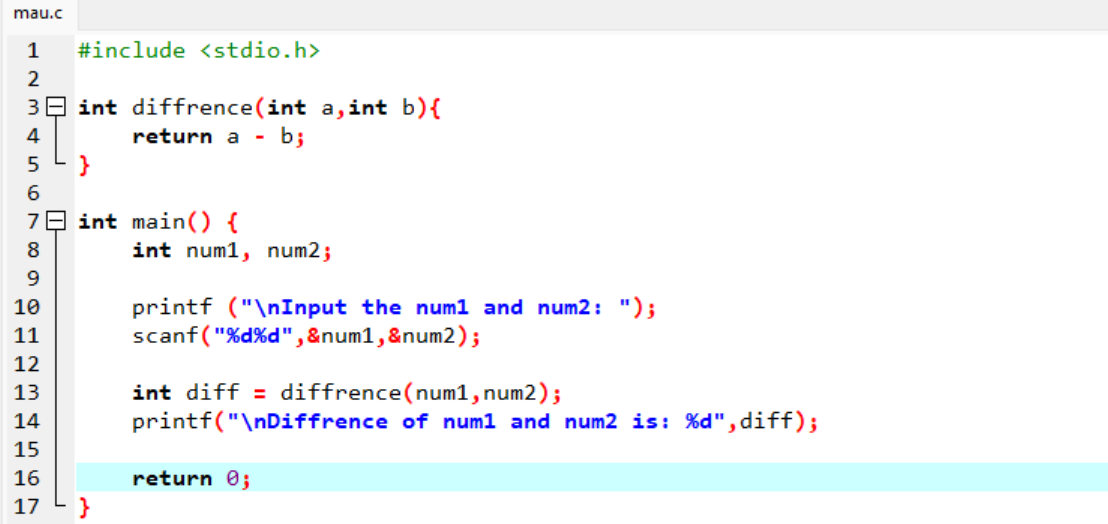
Hình 1.2: Hàm không có tham số và không trả về giá trị

+ Hàm không có tham số nhưng trả về giá trị



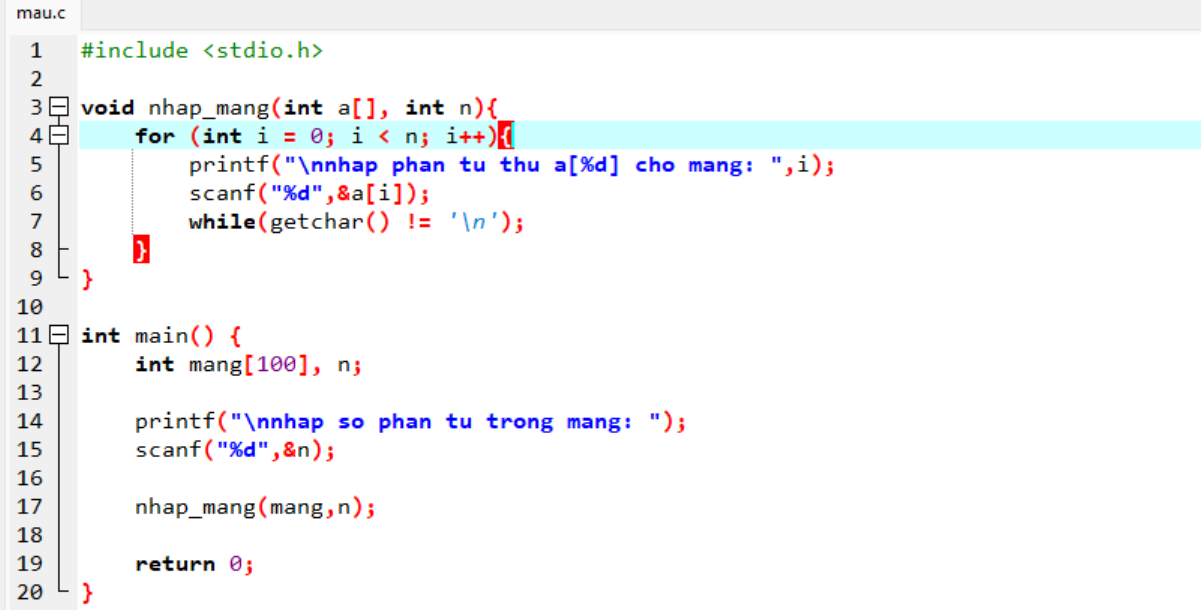
Hình 1.3: Hàm không có tham số nhưng trả về giá trị

+ Hàm có tham số và trả về giá trị



Hình 1.4: Hàm có tham số và có trả về giá trị

+ Hàm có tham số nhưng không trả về giá trị



Hình 1.5: Hàm có tham số nhưng không trả về giá trị

### 1.4: Tầm quan trọng của hàm trong c

Hàm trong C mang lại rất nhiều lợi ích cho quá trình lập trình, bao gồm:

* Tái sử dụng mã: Hàm có thể được gọi nhiều lần trong chương trình, giảm thiểu sự lặp lại mã.
* Tổ chức mã: Hàm giúp tổ chức mã nguồn thành các khối chức năng, giúp mã dễ đọc và dễ bảo trì hơn.
* Giảm lỗi: Khi một phần mã được viết thành hàm, việc kiểm tra và sửa lỗi dễ dàng hơn.

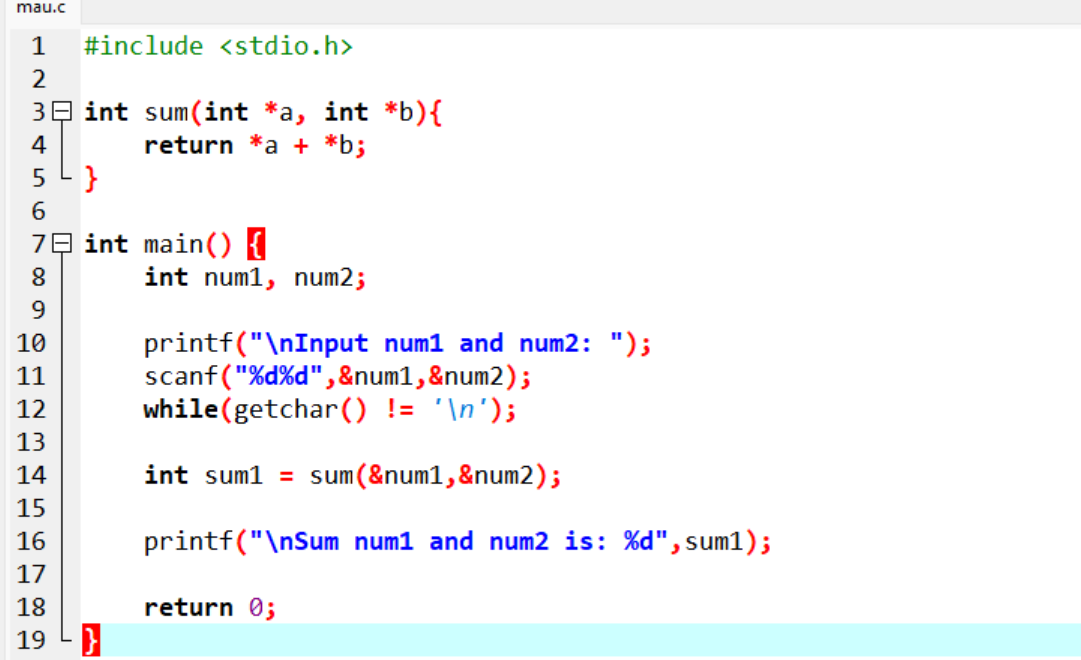
## 2. CON TRỎ

### 2.1: Tổng quát con trỏ trong ngôn ngữ lập trình c

Trong ngôn ngữ lập trình C, con trỏ (pointer) là một trong những khái niệm quan trọng nhất mà bất kỳ lập trình viên nào cũng cần nắm vững. Con trỏ không chỉ giúp quản lý bộ nhớ hiệu quả mà còn mở ra các khả năng xử lý dữ liệu linh hoạt hơn.

+ **Con trỏ** là một biến đặc biệt trong C dùng để lưu trữ địa chỉ của một biến khác. Con trỏ không lưu giá trị cụ thể của biến mà chỉ lưu địa chỉ của biến đó trong bộ nhớ. Để khai báo con trỏ, chúng ta sử dụng dấu \*.

+ Ví dụ:



Hình 2.1: Ví dụ con trỏ

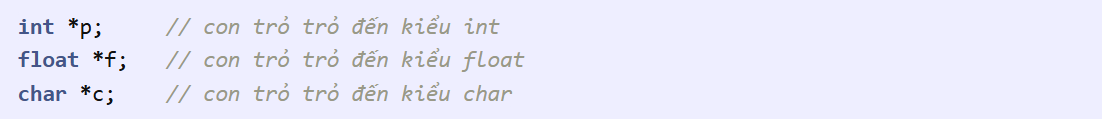
### 2.2: Tại sao con trỏ lại quan trọng

Con trỏ rất quan trọng vì chúng giúp thực hiện các thao tác như:

* **Quản lý bộ nhớ động**: Sử dụng hàm **malloc** và **free**để cấp phát và giải phóng bộ nhớ trong thời gian chạy.
* **Truyền tham chiếu**: Con trỏ giúp truyền tham chiếu cho hàm, giúp hàm có thể thay đổi giá trị của biến gốc.
* **Xử lý cấu trúc dữ liệu phức tạp**: Như danh sách liên kết, cây nhị phân, đồ thị.
* **Tối ưu hóa hiệu suất**: Giảm bớt sao chép dữ liệu trong các chương trình lớn.

### 2.3: Cách khai báo và khởi tạo một con trỏ (pointer)

**-** Để khai báo một con trỏ thứ bạn phải quan tâm và xác định đầu tiên là kiểu dữ liệu mà con trỏ bạn quản lý trỏ tới. Dưới đây là ví dụ:



Hình 2.2: Ví dụ khai báo con trỏ

- Lưu ý một chút: một con trỏ khi khởi tọa nó sẽ mang một giá trị ngẫu nhiên và vô định. Vậy nên, gán giá trị **NULL** cho con trỏ để nó không trỏ đến một gái trị nào cả:

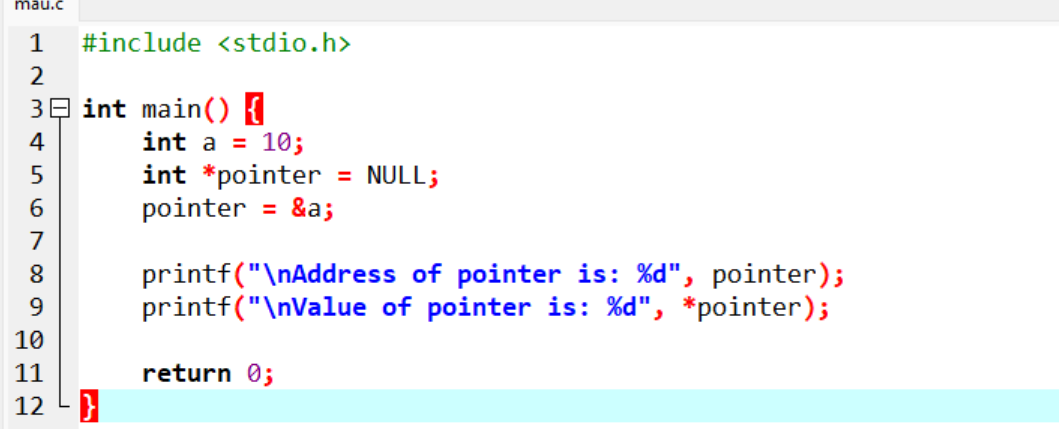


Hình 2.3: Lưu ý về gán giá trị NULL cho con trỏ

### 2.4: Toán tử & và \* trong con trỏ

* Toán tử **&**: Dùng để lấy địa chỉ của một biến.
* Toán tử \*: Dùng để truy xuất giá trị tại địa chỉ mà con trỏ đang trỏ tới (được gọi là dereferencing).

- Ví dụ:



Hình 2.4: Sử dụng toán tử trong con trỏ

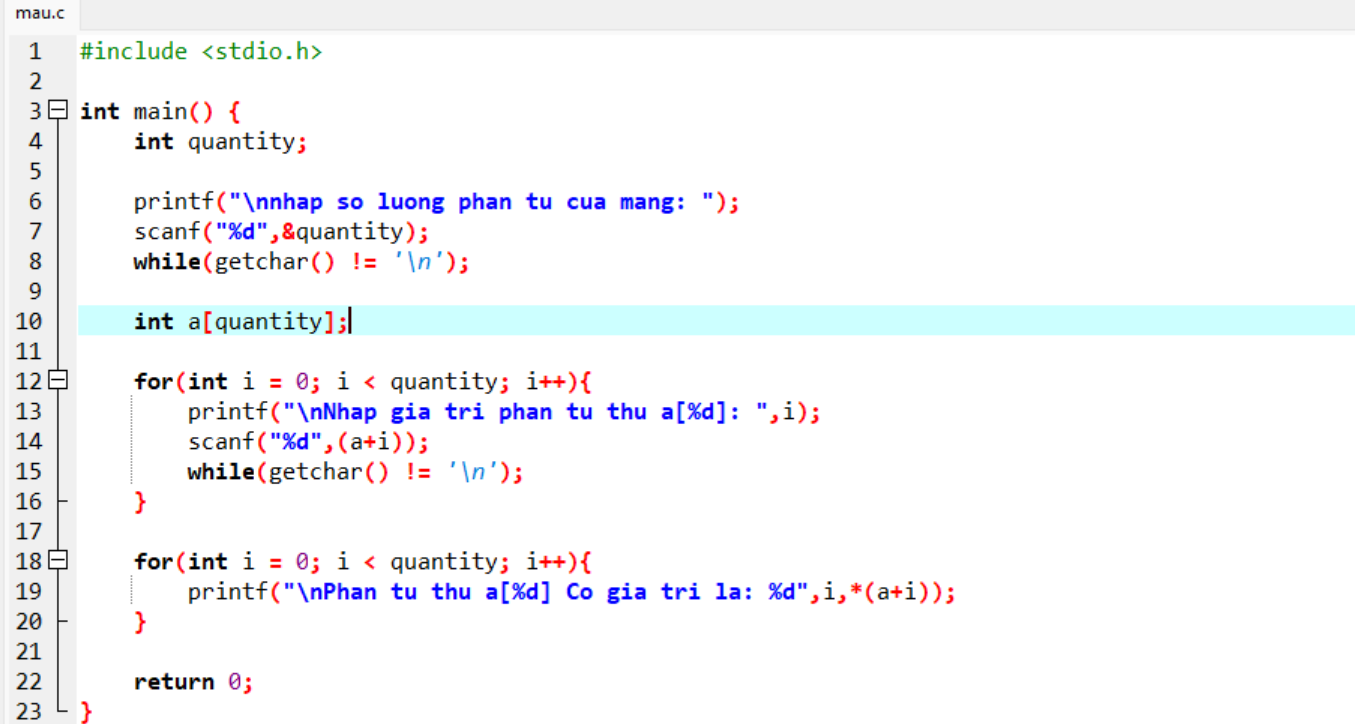
## 3. CON TRỎ MẢNG

### 3.1: Khái niệm con trỏ mảng

+ Con trỏ mảng là một con trỏ được dùng để trỏ đến một mảng hoặc địa chỉ phần tử đầu tiên của mảng. Trong C, tên mảng thực chất chính là một hằng con trỏ trỏ đến phần tử đầu tiên của mảng. Tuy nhiên, ta cũng có thể khai báo một con trỏ riêng biệt để truy cập và thao tác với các phần tử trong mảng.

+ Lưu ý thêm: Tham chiếu đến phần tử của mảng giúp bạn có thể sử dụng con trỏ để truy cập đến các phần tử của mảng bằng cách sử dụng toán tử chỉ mục [] hoặc toán tử pointer\*.

### 3.2: Ví dụ minh họa



Hình 3.1: Con trỏ mảng

- ở đây: (a+i) nhận công việc là tham chiếu từng đối tượng trong mảng và \*(a+i) là giải tham chiếu trừng đối tượng trong mảng bắt đầu từ a[0].

Đoạn mã thực hiện:

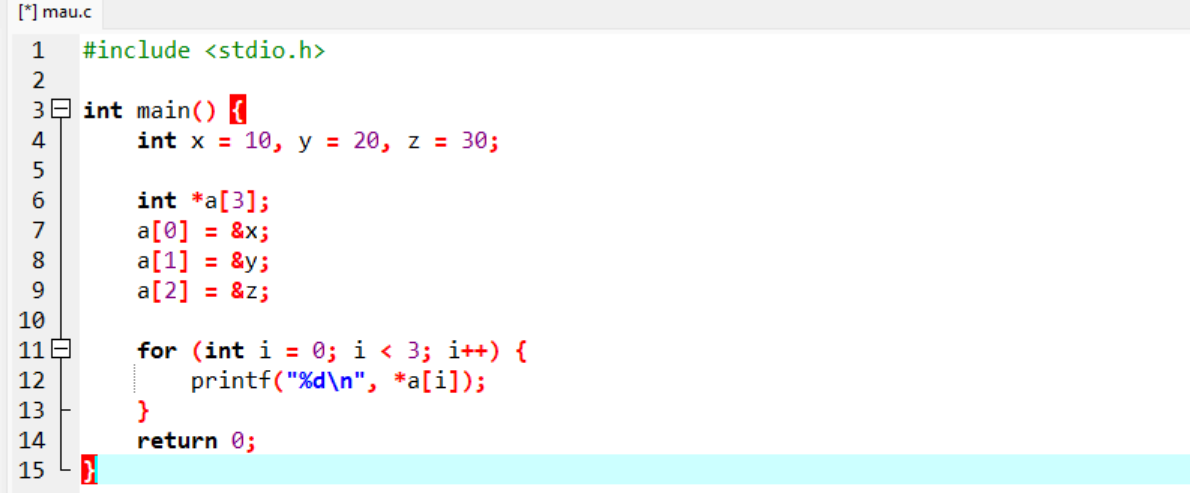
- Nhập số lượng phần tử → nhập giá trị từng phần tử → in từng phần tử.

- Dùng con trỏ (pointer) để thao tác trên mảng (thay vì chỉ dùng chỉ số thông thường).

## 4. MẢNG CON TRỎ

### 4.1: Khái niệm con trỏ hàm

+ **Mảng con trỏ** là một mảng mà **các phần tử của nó là các con trỏ**. Nói cách khác, đây là một **mảng chứa địa chỉ** thay vì giá trị trực tiếp. Các con trỏ trong mảng này thường trỏ tới các biến, các mảng khác, hoặc các chuỗi. Ví dụ:



Hình 4.1: Ví dụ mảng con trỏ

### 4.2: Lưu ý khi dùng Mảng con trỏ

- Không cấp phát bộ nhớ khi khai báo kiểu char \*arr[] = {...}

- Các chuỗi được lưu trong bộ nhớ chỉ đọc (read-only).

- Nếu bạn cần thay đổi nội dung chuỗi, hãy dùng cấp phát động (malloc) hoặc mảng hai chiều ( đặc biệt sử dụng malloc cực kì an toàn khi thao tác với chuỗi).

- mảng thông thường lưu trữ các giá trị cùng kiểu dữ liệu liên tiếp trong bộ nhớ và có thể truy cập trực tiếp qua chỉ số. Mảng con trỏ chứa các con trỏ mỗi con trỏ trỏ đến một địa chỉ khác nhau, và cần sử dụng \* để truy cập giá trị..

### 4.3: Cú pháp

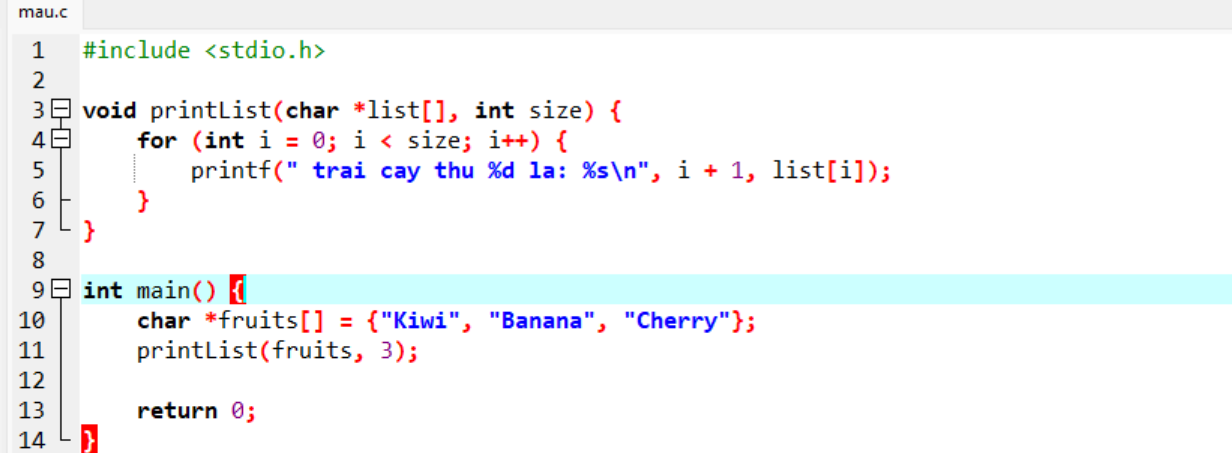
**type \*arrayName[size];**

**+ type**: là kiểu dữ liệu của các phần tử, mà mỗi phần tử của mảng con trỏ sẽ trỏ đến.

+ **arrayName:** tên con trỏ sẽ trỏ tới.

+ **size**: là số lượng phần tử trong mảng con trỏ.

### 4.4: Ví dụ minh họa



Hình 4.2: Mảng con trỏ

+ Mảng con trỏ **fruits** được truyền vào hàm **printList** như một mảng bình thường.

+ Hàm có thể xử lý danh sách chuỗi một cách linh hoạt, có thể tùy chỉnh độ dài.

## 5. CON TRỎ HÀM

### 5.1:Tổng quan và khái niệm con trỏ hàm

Con trỏ hàm trong ngôn ngữ lập trình C là một khái niệm quan trọng giúp lập trình viên có thể truyền một hàm như một đối số vào các hàm khác, hoặc lưu trữ địa chỉ của hàm để có thể gọi lại sau này. Điều này mang lại sự linh hoạt và hiệu quả cao trong việc thiết kế các chương trình phức tạp.

- Con trỏ hàm là một biến dùng để lưu trữ địa chỉ của một hàm. Cú pháp khai báo con trỏ hàm trong C như sau:

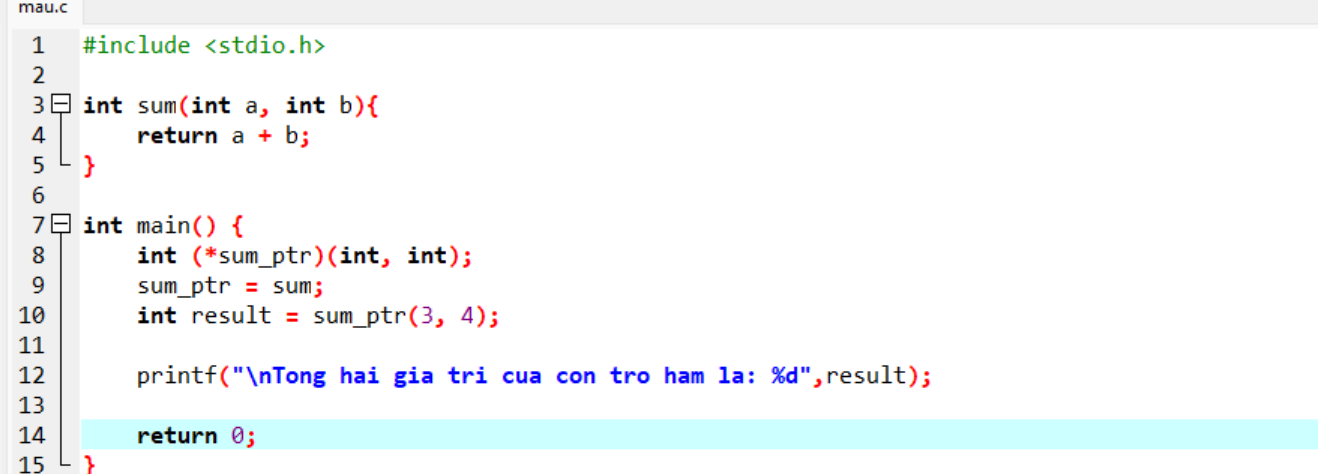
**return\_type (\*pointer\_name)(parameter\_list);**

**+ return\_type:** kiểu dữ liệu mà con trỏ sẽ trả về.

+ **(\*pointer\_name) :** Khai báo một con trỏ có tên là pointer\_name trỏ tới một hàm.

+ **parameter\_list:** Danh sách tham số của hàm mà con trỏ trỏ tới (giống định nghĩa hàm).

- Ví dụ:

****

Hình 5.1: ví dụ về con trỏ hàm.

+ Ở đây, **(\*sum\_ptr)** có nhiệm vụ làm con trỏ trỏ tới hàm nhận 2 giá trị int và trả về giá trị int.

+ **sum\_ptr** **= sum**: có nhiệm vụ gán địa chỉ hàm sum cho con trỏ.

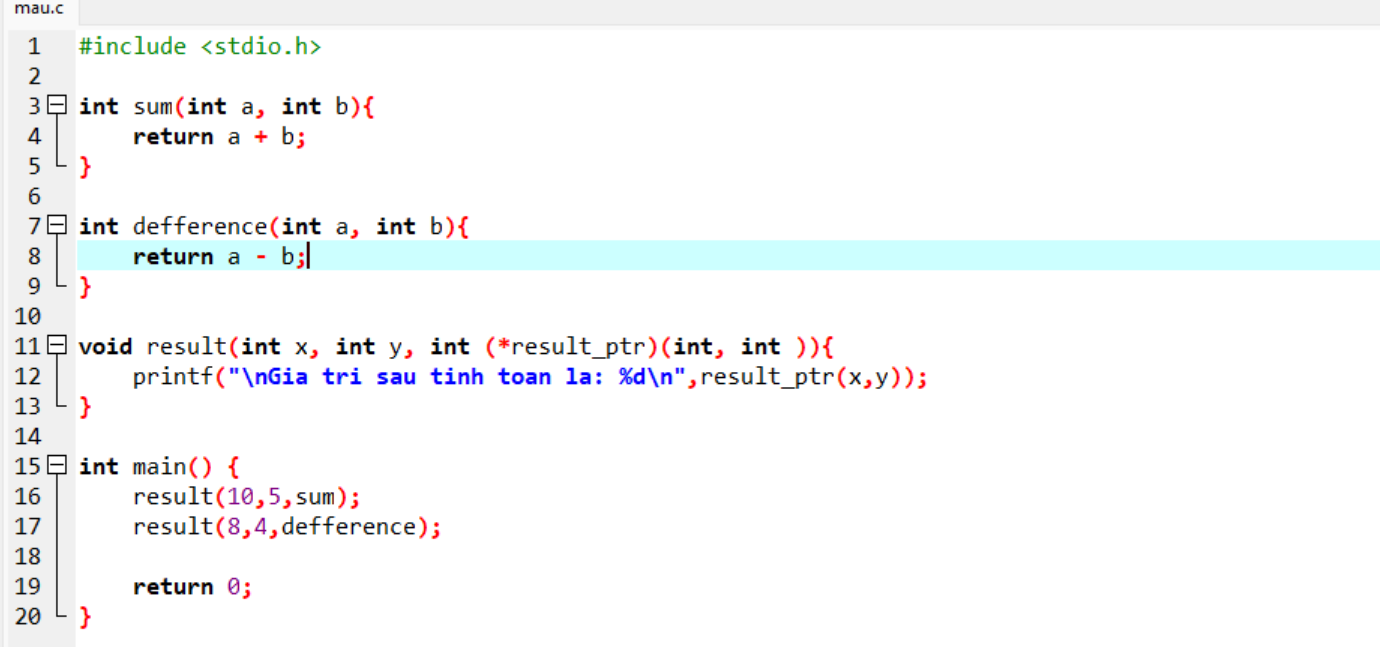
+ cuối cùng là dùng **result** để gọi hàm thực hiện.

### 5.2: Lưu ý khi sử dụng con trỏ hàm

- Con trỏ hàm thường được sử dụng trong các tình huống sau:

* Truyền hàm như một đối số cho các hàm khác.
* Gọi lại các hàm tại thời điểm chạy chương trình, đặc biệt hữu ích trong các tình huống như xử lý sự kiện hoặc callback.
* Giảm thiểu mã trùng lặp, tăng tính linh hoạt và tái sử dụng mã nguồn.

### 5.3: Ví dụ minh họa

****

Hình 5.2: Ví dụ minh họa về con trỏ hàm

+ Ở đây, trong ngôn ngữ C thì tên hàm được coi như địa chỉ của hàm nên khi gọi hàm **result** ra để tính toán thì chắc chắn rằng con trỏ **result\_ptr** đã trỏ tới địa chỉ của hàm **sum** và **defference** rồi.

+ **result** đóng vai trò là một hàm chung, tính toán dựa vào tham chiếu của hàm nào được truyền vào.

## 6. CẤP PHÁT ĐỘNG

### 6.1 : Khái niệm về cấp phát động trong ngôn ngữ c

Cấp phát động trong ngôn ngữ lập trình C là một kỹ thuật quan trọng giúp quản lý bộ nhớ hiệu quả. Kỹ thuật này cho phép chương trình cấp phát và giải phóng bộ nhớ theo yêu cầu trong thời gian chạy, giúp tối ưu hóa việc sử dụng tài nguyên hệ thống.

- Các hàm sử dụng nhiều khi cấp phát động trong thư viện **#include <stdlib.h>:**

**+** Hàm **malloc() (**memory allocation): là hàm được sử dụng để cấp phát một vùng nhớ với kích thước xác định. Ví dụ:

**int\* ptr = (int\*) malloc(sizeof(int));**

**+** Hàm **calloc()** (contiguous allocation): là hàm cấp phát bộ nhớ cho một mảng gồm nhiều phần tử, mỗi phần tử có kích thước cố định. Ví dụ:

**int\* ptr = (int\*) calloc(sizeof(int));**

+ Hàm **realloc()** (reallocation): là hàm thay đổi kích thước của một vùng nhớ đã được cấp phát trước đó. Ví dụ:

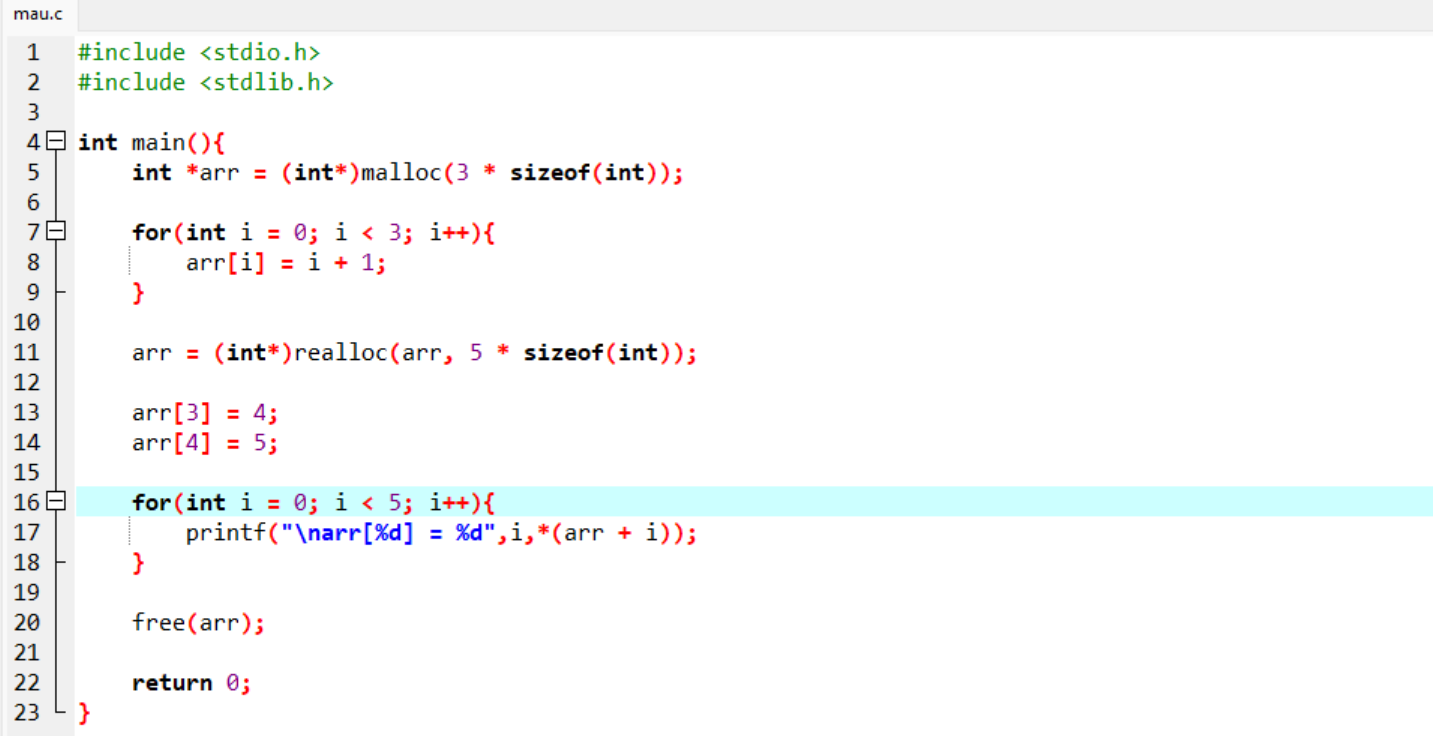
**arr = (int\*) realloc(arr, 5 \* sizeof(int));**

+ Hàm **Free()** : Sau khi hoàn thành việc sử dụng bộ nhớ, bạn cần giải phóng bộ nhớ để tránh lãng phí tài nguyên. Để giải phóng bộ nhớ đã được cấp phát bằng hàm **malloc()**, chúng ta sử dụng hàm **free()**. Ví dụ:

**Free(arr);**

- **Lưu ý**: Việc sử dụng cấp phát động cho phép tạo ra các cấu trúc dữ liệu linh hoạt như mảng động, danh sách liên kết, cây nhị phân hoặc đồ thị. Nhờ đó, chương trình có thể xử lý dữ liệu có kích thước thay đổi một cách hiệu quả. Tuy nhiên, việc quản lý bộ nhớ động khá phức tạp và nếu không cẩn thận, có thể dẫn đến các lỗi như rò rỉ bộ nhớ hoặc truy cập sai vùng nhớ. Do đó, lập trình viên cần sử dụng cấp phát động một cách có kiểm soát và luôn đảm bảo giải phóng bộ nhớ sau khi sử dụng.

### 6.2: Ví dụ minh họa

****

Hình 6.1: ví dụ minh họa cấp phát động

- Mục tiêu của code:

* Cấp phát mảng động chứa 3 số nguyên.
* Khởi tạo giá trị cho 3 phần tử đầu với hàm malloc.
* Dùng realloc để mở rộng mảng lên 5 phần tử.
* Gán thêm 2 giá trị nữa.
* In ra toàn bộ mảng.
* Dùng free() để Giải phóng bộ nhớ.

### 6.3 : Lợi ích khi dùng cấp phát động với thư viện #include <stdlib.h>

- Linh hoạt về kích thước. Bạn có thể:

* Cấp phát bao nhiêu bộ nhớ tùy ý ngay khi chương trình đang chạy.
* Phản ứng với đầu vào từ người dùng hoặc dữ liệu từ file.

- Thay đổi kích thước vùng nhớ chỉ với hàm realloc:

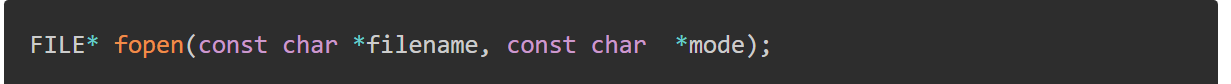
* Bạn có thể **mở rộng hoặc thu hẹp** vùng nhớ đã cấp phát bằng realloc.

## 7. XỬ LÍ TỆP

### 7.1: Khái niệm về xử lí tệp trong c

- Trong thế giới lập trình, quản lý tập tin là kỹ năng cơ bản mà mọi nhà phát triển phải thành thạo. Khi nói đến ngôn ngữ C, việc xử lý tệp trở nên quan trọng hơn vì nó tập trung vào việc thao tác trực tiếp các tệp trên hệ điều hành.

- Xử lý tệp trong ngôn ngữ C đề cập đến khả năng đọc và ghi dữ liệu vào tệp từ chương trình được viết bằng ngôn ngữ này. Thông qua quản lý tệp, bạn có thể tạo, mở, đóng, đọc và ghi vào tệp, cho phép bạn lưu trữ và truy xuất thông tin một cách liên tục. Ví dụ:



Hình 7.1: cú pháp mở tệp

- Các hàm quan trọng trong công việc xử lí file ở ngôn ngữ c:

* Hàm **fopen()**: là bước đầu tiên khi làm việc với tệp. Bạn cần chỉ định chế độ mở tệp (đọc, ghi, thêm mới, v.v.) để hệ thống biết bạn định làm gì với tệp đó. Cú pháp:

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

+ các chế độ mở file:

* r: mở file để đọc.
* w: mở file để ghi.
* a: mở file để viết thêm vào file.
* r+: mở file để đọc và ghi.
* w+: tạo file mới để đọc và ghi.
* a+f: viết thêm vào file hoặc tạo thêm file mới để đọc và ghi.
* Hàm **fclose():** Đóng file sau khi xử lý xong. Cú pháp:

**fclose(f);**

* Hàm **fprintf():** Ghi dữ liệu dạng văn bản (giống printf, nhưng ghi vào file). Ví dụ:

**fprintf(f, "Name: %s, Age: %d\n", name, age);**

* Hàm **fscanf() :** Đọc dữ liệu định dạng (giống scanf). Ví dụ:

**fscanf(f, "%s %d", name, &age);**

### 7.2: Ưu điểm của xử lí tệp trong ngôn ngữ c

- Việc xử lý tệp trong ngôn ngữ C mang lại một số lợi thế quan trọng:

* Tính bền vững của dữ liệu: Lưu trữ cho phép dữ liệu tồn tại lâu hơn thời gian tồn tại của một chương trình, nghĩa là dữ liệu có thể được truy cập trong các lần thực thi sau.
* Lưu trữ dữ liệu lớn: Việc xử lý tệp đặc biệt hữu ích khi làm việc với khối lượng thông tin lớn không thể chứa trong bộ nhớ chính.



Hình 7.2: ảnh ngôn ngữ c về xử lí file

* Chia sẻ thông tin giữa các chương trình: Việc lưu trữ dữ liệu trong các tệp giúp trao đổi thông tin giữa các chương trình khác nhau dễ dàng hơn, ngay cả khi được viết bằng các ngôn ngữ khác nhau.
* Tổ chức dữ liệu: Tệp cho phép dữ liệu được sắp xếp thành các cấu trúc cụ thể, giúp việc đọc và ghi dễ dàng hơn.
* Phục hồi dữ liệu: Trong trường hợp chương trình bị lỗi hoặc gián đoạn, dữ liệu được lưu trữ trong các tệp có thể được khôi phục.

### 7.3: Ví dụ minh họa

**-** ví dụ về ghi file:

****

Hình 7.3: Ví dụ minh họa xử lí ghi file

- **Mục tiêu của đoạn code này**:

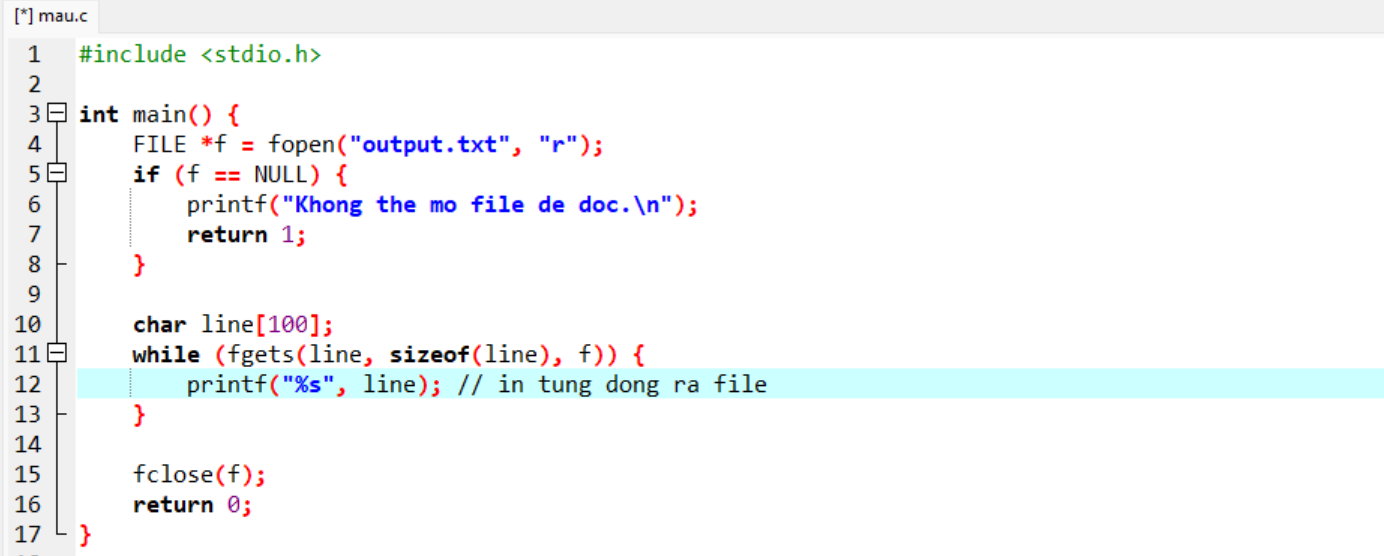
+ Minh họa cách ghi dữ liệu vào một file văn bản trong ngôn ngữ C.

+ xử dụng chế độ **‘w”** trong hàm **fopen** mở file ra để ghi.

+ kết hợp kiểm tra file bằng lệnh **if ( f = NULL)** nhằm kiểm tra lỗi mở file.

+ Dùng **fprintf()** để ghi văn bản vào file.

- Tương tự ta cũng có ví dụ với mở file để đọc:



Hình 7.4: Ví dụ minh họa về mở file để đọc

## 8. KIỂU CẤU TRÚC

### 8.1: Khái niệm kiểu cấu trúc

- **Struct** là giải pháp khi bạn cần giải quyết các bài toán thực tế khi mà đối tượng bạn cần lưu lại trong chương trình cần rất nhiều thông tin.

- Ví dụ khi bạn muốn lưu thông tin của một sinh viên thì không có kiểu dữ liệu nào trong C phù hợp cả, mà bạn cần tập hợp nhiều kiểu dữ liệu vào làm một.

- Struct hay cấu trúc là một kiểu dữ liệu mà tự bạn định nghĩa ra bằng cách gộp nhiều kiểu dữ liệu có sẵn lại nhằm mục đích có thể mô tả nhiều trường thông tin của đối tượng bạn cần lưu.

### 8.2: Cú pháp kiểu cấu trúc

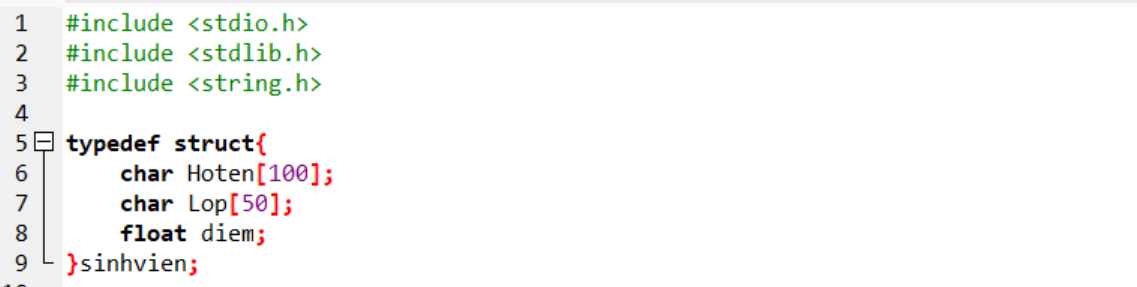


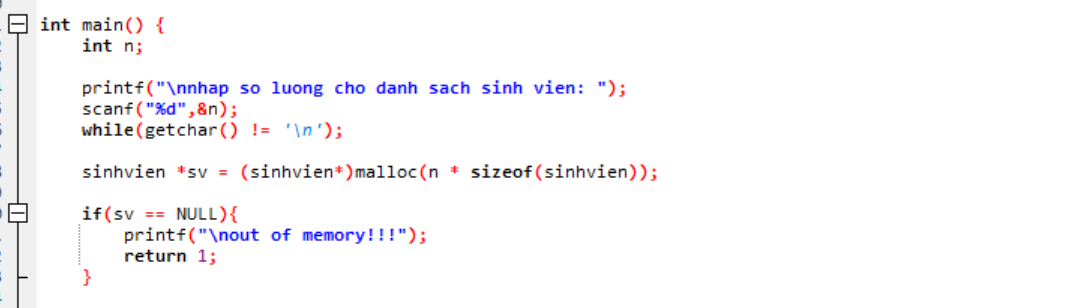
Hình 8.1: Cú pháp của kiểu cấu trúc

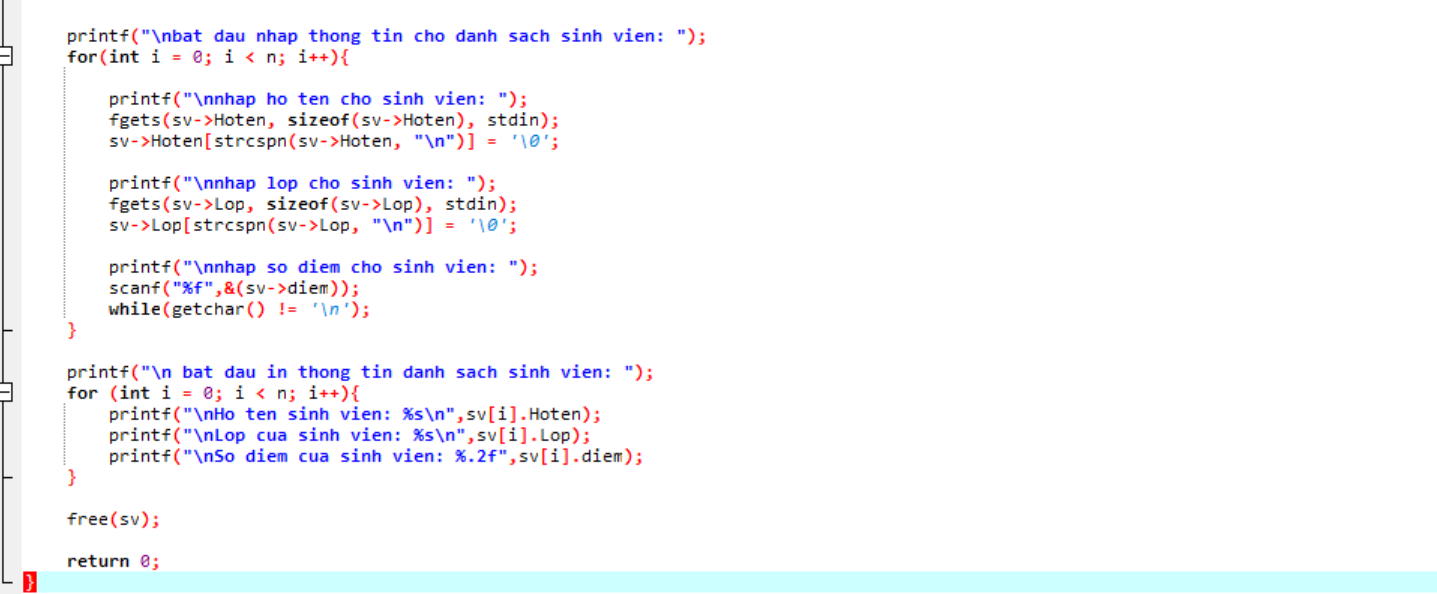
- Ở đây:

* **Struct** : từ khóa để khai báo một cấu trúc.
* **ten\_struct** : tên do người dùng đặt cho cái struct này.
* **data\_typeX**: kiểu dữ liệu mà cấu trúc này sẽ mang (int, float,….).
* **data\_fieldX** : tên biến thành phần (trường) trong câu.

### 8.3: Ví dụ minh họa

****

****

****

Hình 8.2: Ví dụ minh họa về struct sinh viên

**- Tổng quan chương trình**

Chương trình quản lý danh sách sinh viên gồm:

* Nhập số lượng sinh viên: sau khi nhập n thì dùng **malloc** để cấp phát động số lượng sinh viên tương ứng với mỗi **n** được nhập.
* Nhập thông tin từng sinh viên (họ tên, lớp, điểm): sử dụng các hàm như **fgets()** và **strcspn()** đồng thời còn cả **input buffer clearing** để tối ưu hóa việc nhập.
* In ra danh sách sinh viên vừa nhập: sử dụng **vòng lặp** để chạy từng sinh viên trong danh sách ứng với mỗi i.
* Sau cùng, dùng **free()** để giải phóng bộ nhớ sv vừa cấp phát.

## 9. DANH SÁCH LIÊN KẾT

### 9.1: Khái niệm danh sách liên kết

**- Danh sách liên kết** là một cấu trúc dữ liệu dùng để lưu trữ và quản lý tập hợp các phần tử. Mỗi phần tử (gọi là nút) gồm hai phần:

* Dữ liệu: thông tin cần lưu (số, ký tự, hoặc kiểu dữ liệu phức tạp).
* Con trỏ tiếp theo: trỏ đến nút kế tiếp trong danh sách.

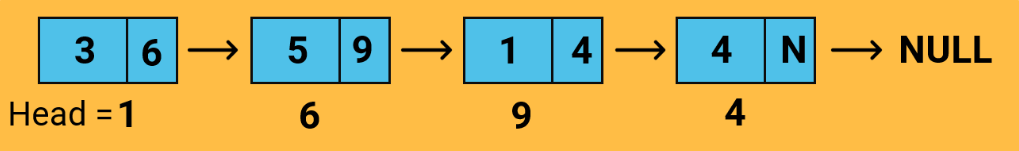
Nhờ sử dụng con trỏ, danh sách liên kết linh hoạt trong việc thêm hoặc xóa nút mà không cần thay đổi kích thước bộ nhớ như mảng.

Có hai loại chính:

* Danh sách liên kết đơn: mỗi nút chỉ trỏ đến nút kế tiếp.
* Danh sách liên kết đôi: mỗi nút trỏ đến cả nút trước và nút sau, cho phép duyệt hai chiều.

Các thao tác thường gặp gồm: thêm, xóa, tìm kiếm, và duyệt danh sách.

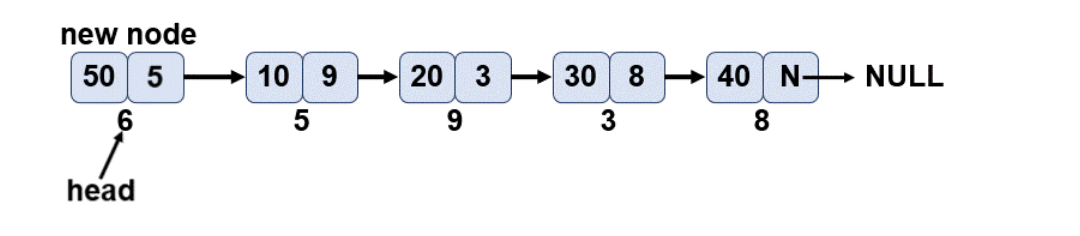
- Minh họa danh sách liên kết:



Hình 9.1: Minh họa các node cảu một danh sách liên kết.

### 9.2 Tiện ích khi sử dụng danh sách liên kết:

* Mở rộng và thu hẹp một cách linh hoạt
* Các phần tử trong DSLK gọi là node và được cấp phát động khi cần
* Số lượng phần tử trong DSLK phụ thuộc vào bộ nhớ heap
* Dễ dàng chèn và xóa phần tử
* Các phần tử trong DSLK không có thứ tự
* Truy cập phần tử trong DSLK cần truy cập tuần tự không thể truy cập qua chỉ số
* Mỗi node trong DSLK cần có thêm 1 con trỏ để lưu liên kết



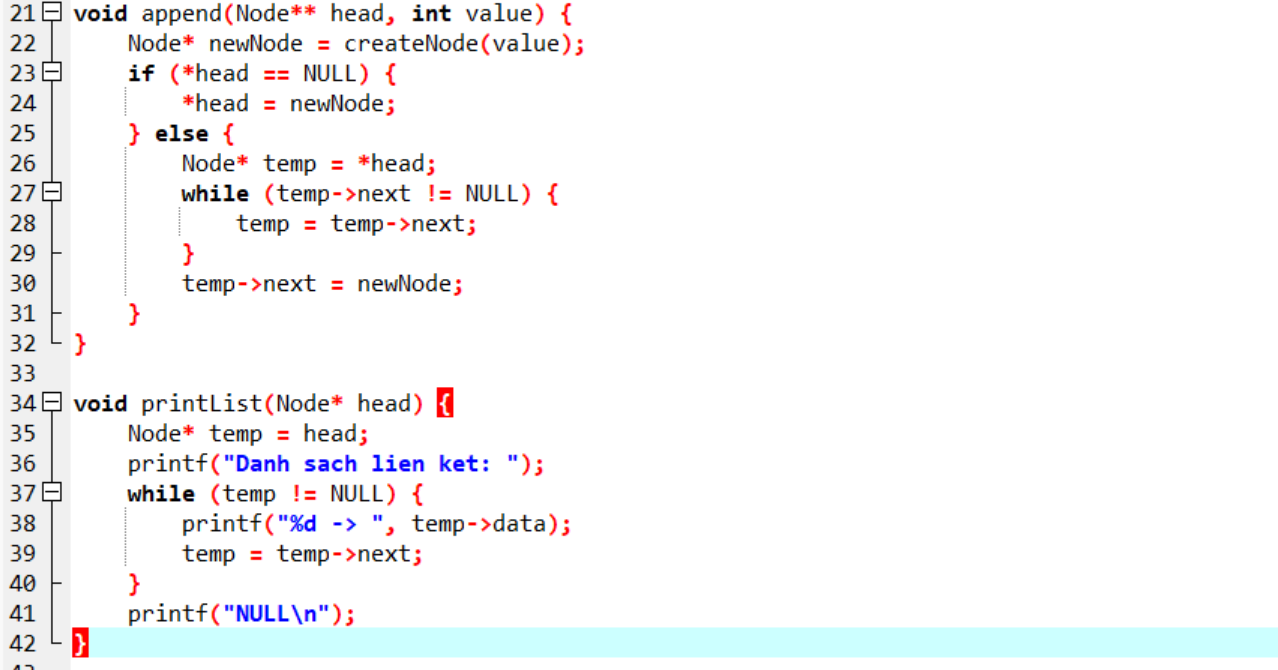
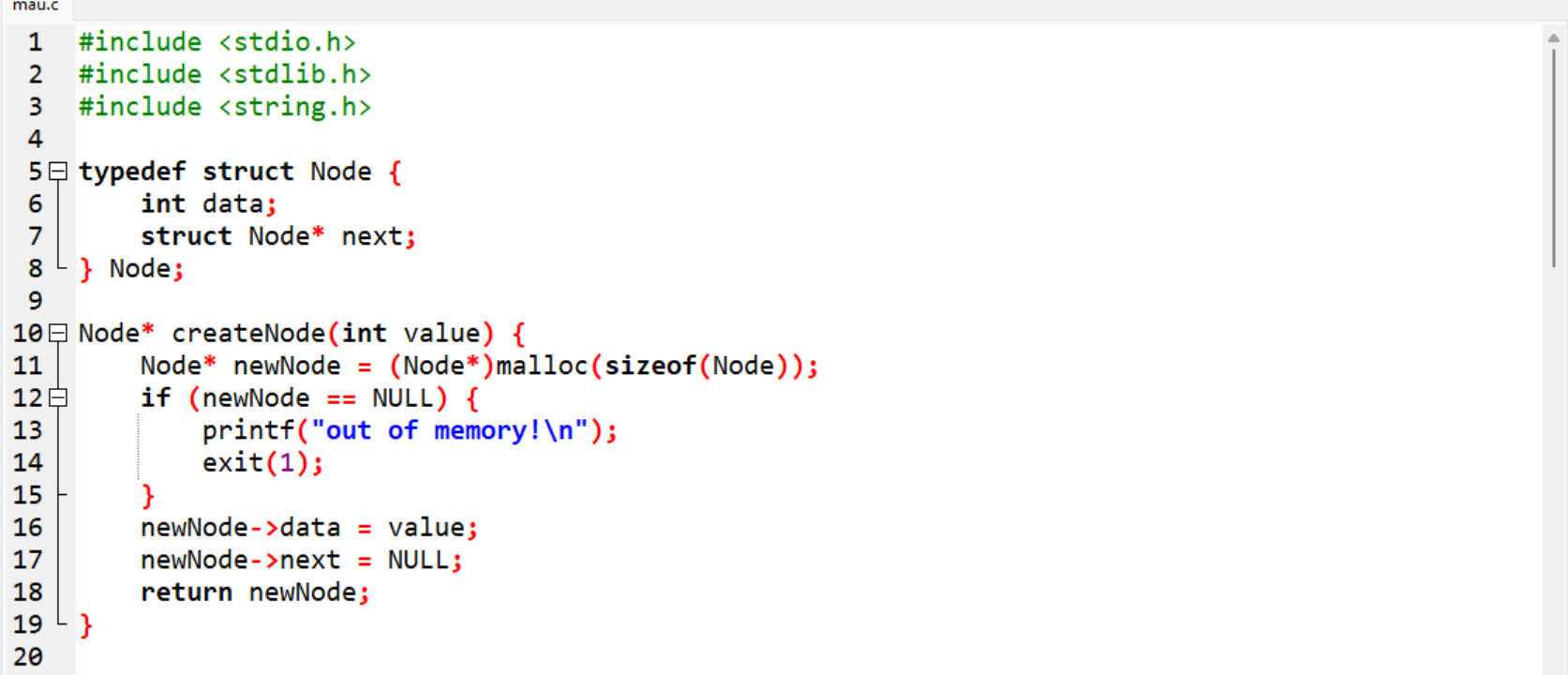
Hình 9.2: Cách hoạt động cảu một node

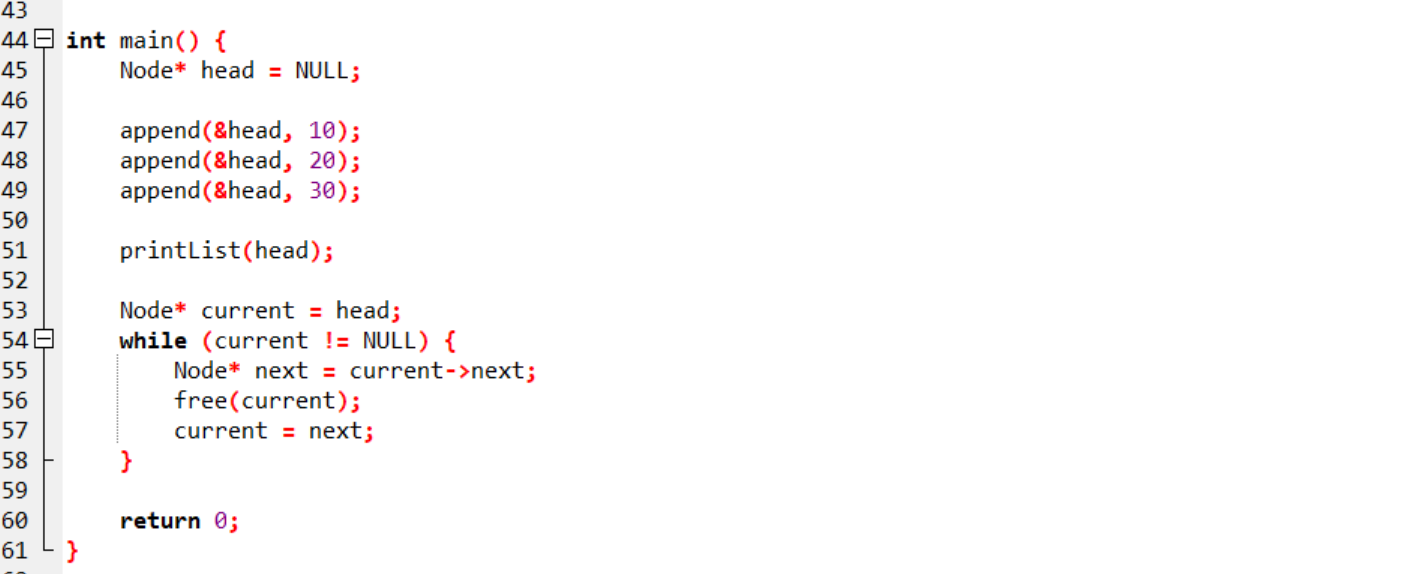
Mỗi phần tử trong DSLK được gọi là một node hay nút, node sẽ lưu thông tin dữ liệu (ví dụ như một số nguyên, 1 chuỗi ký tự, 1 sinh viên...) và ngoài ra cần có phần liên kết, phần liên kết này giúp các node có thể liên lạc với nhau. Mỗi node sẽ lưu thêm địa chỉ của node phía sau nó trong DSLK thông qua 1 thuộc tính con trỏ.

### 9.3: Liệu danh sách liên kết có tối ưu hóa hơn mảng không?

* **Kích thước:**
* Danh sách liên kết: Kích thước động, có thể thay đổi trong quá trình thực thi.
* Mảng: Kích thước cố định, phải khai báo trước và không thể thay đổi sau khi cấp phát.
* **Thêm và xóa phần tử:**
* Danh sách liên kết: Thêm hoặc xóa phần tử nhanh chóng, không cần di chuyển các phần tử khác.
* Mảng: Thao tác thêm hoặc xóa phức tạp, cần di chuyển các phần tử còn lại để đảm bảo thứ tự.
* **Truy cập phần tử:**
* Danh sách liên kết: Truy cập chậm hơn do phải duyệt qua từng nút (độ phức tạp O(n)).
* Mảng: Truy cập nhanh nhờ chỉ số và bố trí liên tiếp trong bộ nhớ (độ phức tạp O(1)).
* **Tính linh hoạt:**
* Danh sách liên kết: Linh hoạt, có thể dễ dàng thêm/bớt phần tử bất cứ lúc nào.
* Mảng: Không linh hoạt, kích thước cố định nên khó mở rộng hoặc thu hẹp.
* **Quản lý bộ nhớ:**
* Danh sách liên kết: Tiết kiệm bộ nhớ hơn trong một số trường hợp vì cấp phát động từng nút.
* Mảng: Cấp phát bộ nhớ một lần cho toàn bộ mảng, có thể gây lãng phí nếu không dùng hết.

### **9.4: Ví dụ minh họa**



****

Hình 9.3: Ví dụ minh họa danh sách liên kết đơn

-**Tổng quan chương trình:**

* Khai báo cấu trúc nút: Mỗi nút gồm hai thành phần: data (dữ liệu kiểu int) và next (con trỏ trỏ đến nút tiếp theo).
* Tạo nút mới:Sử dụng hàm **createNode()** để cấp phát động bộ nhớ và khởi tạo giá trị cho một nút mới.
* Thêm nút vào cuối danh sách: Hàm **append()** kiểm tra nếu danh sách rỗng thì gán nút mới làm đầu danh sách, nếu không thì duyệt đến cuối và thêm vào.
* In danh sách liên kết: Hàm **printList()** sử dụng vòng lặp để duyệt từng nút và in ra giá trị các phần tử nối nhau bằng dấu **->.**
* Giải phóng bộ nhớ:Sau khi dùng xong, sử dụng vòng lặp **while** và hàm **free()** để giải phóng toàn bộ bộ nhớ đã cấp phát cho các nút trong danh sách.

# CHƯƠNG 3: ỨNG DỤNG

## **1. Ý TƯỞNG**

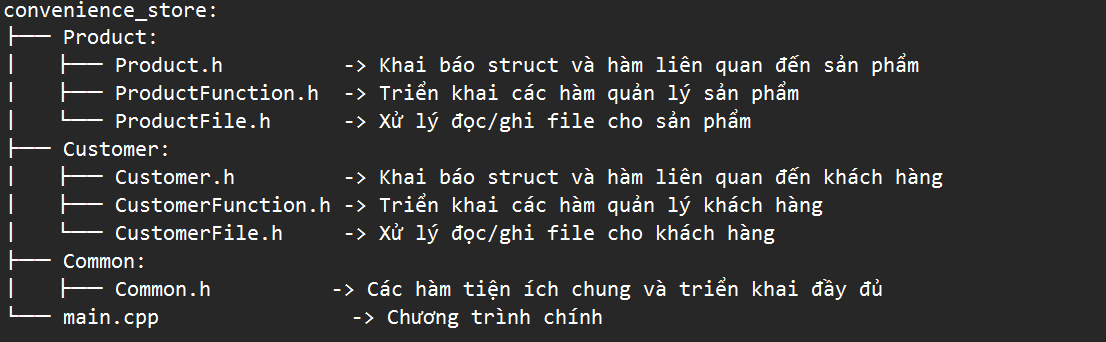
**-** Ý tưởng gốc từ chương trình quản lý sinh viên đơn giản và dựa vào những kiến thức về lý thuyết đã học của danh sách liên kết mà tạo nên 1 chương trình cửa hàng tiện lợi đơn giản ( convenience store ).



Hình 1.1: Ảnh convenience store

- Code xoay quanh các phần đã được học và thực hành trên lớp như: thêm phần tử, xóa phần tử, tìm kiếm phần tử,….. Và gộp lại tạo nên 1 khung chương trình cửa hàng tiện lợi đơn giản.

## **2. TÓM TẮT CHƯƠNG TRÌNH CHÍNH**

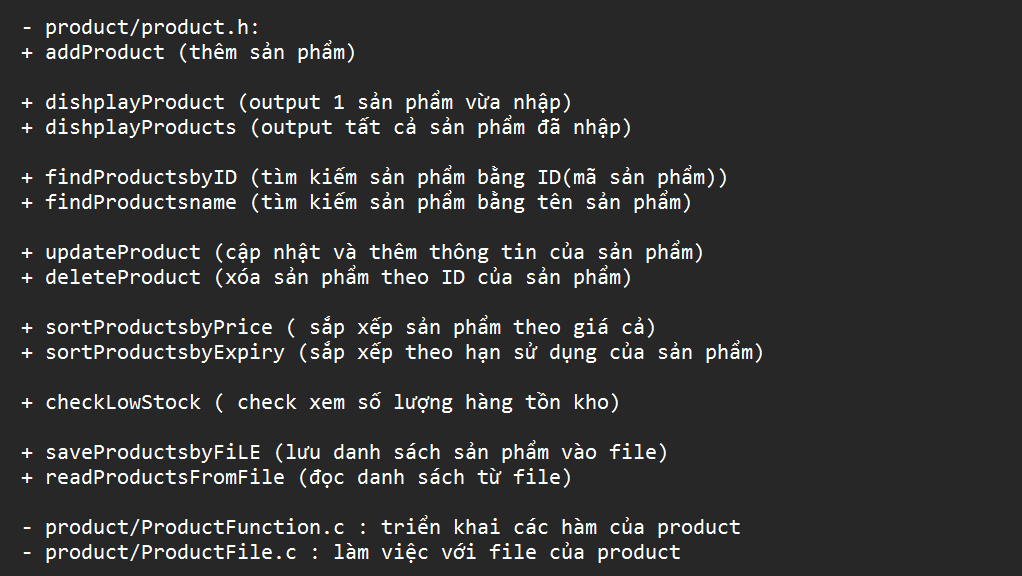
****

Hình 2.1 : Ý tưởng ứng dụng danh sách liên kết

- Trương trình được tập trung chính vào 3 phần là:

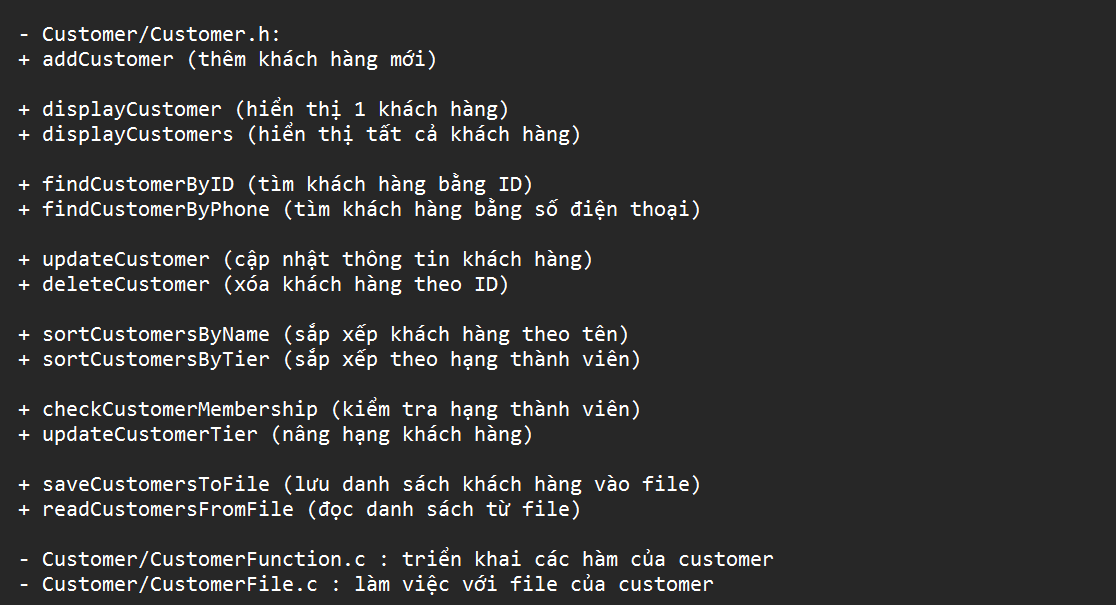
* **Sản phẩm (Product):** được chia làm 3 file chính là **Product.h** với nhiệm vụ file chính lưu các thư viện và khai báo các hàm và định nghĩa **struct product**, **ProductFunction.h** với nhiệm vụ triển khai các hàm đã được khai báo ở thư viện chính **Product.h** và cuối cùng là **ProductFile.h** có nhiệm vụ triển khai hàm đọc và ghi của thư viện chính **Product.h.**
* **Khách hàng (Customer):** Tương tự như Product được chia làm 3 file chính là **Customer.h** với nhiệm vụ file chính lưu các thư viện và khai báo các hàm và định nghĩa **struct Customer**, **CustomerFunction.h** với nhiệm vụ triển khai các hàm đã được khai báo ở thư viện chính **Customer.h** và cuối cùng là **CustomerFile.h** có nhiệm vụ triển khai hàm đọc và ghi của thư viện chính **Customer.h.**
* **Các hàm tiện ích chung(Common):** chủ yếu là thêm các hàm như input buffer, check nhập số điện thoại hợp lệ, check nhập email hợp lệ, check ngày tháng nhập,… file này chủ yếu là sửa chửa nâng cấp cho code tối ưu hơn.

- Các hàm chính trong nhánh Product:



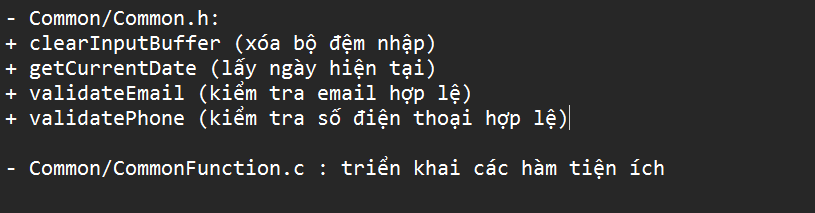
Hình 2.2: Hàm chính của Product

- Các hàm chính trong nhánh Customer:



Hình 2.3: Hàm chính của Customer

- Các hàm chính trong nhánh Common:



Hình 2.4: Hàm chính của Common

# TÀI LIỆU THAM KHẢO

[1] [[DSLK Đơn]. Thêm Node Vào Đầu DSLK](https://blog.28tech.com.vn/dslk-don-them-node-vao-dau-dslk), Tài liệu tham khảo về danh sách liên kết đơn.

[truy cập 20/05/2025].

[2] [Xử lý tệp trong ví dụ ngôn ngữ C: Hướng dẫn đầy đủ](https://informatecdigital.com/vi/x%E1%BB%AD-l%C3%BD-file-trong-ng%C3%B4n-ng%E1%BB%AF-c/), Xử lí file trong ngôn ngữ c.

[truy cập 20/05/2025].