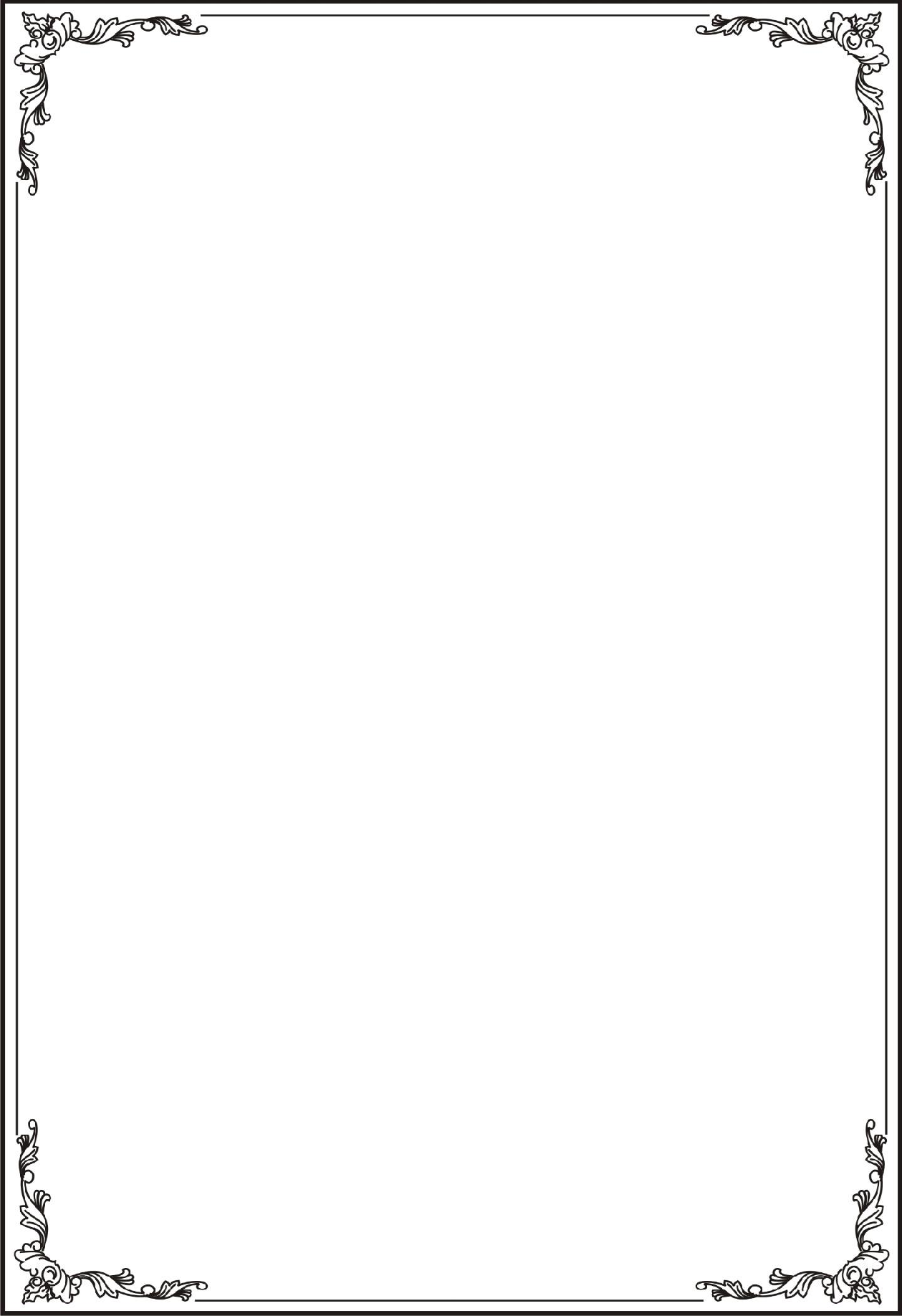


|  |
| --- |
| **TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI** |
| **PHÂN HIỆU TẠI TP. HỒ CHÍ MINH** |
| **BỘ MÔN CÔNG NGHỆ THÔNG TIN** |
|  |
| **BÁO CÁO** |
| **MÔN: KĨ THUẬT LẬP TRÌNH**  **BÀI TẬP LỚN** |

|  |  |
| --- | --- |
| Giảng viên hướng dẫn: | ThS. TRẦN PHONG NHÃ |
| Sinh viên thực hiện: | PHẠM THỊ MỸ HẰNG |
| Lớp : | CQ.CNTT.K65 |
| Khóa: | 65 |
|  |  |

|  |
| --- |
| TP.Hồ Chí Minh, ngày 7 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: KĨ THUẬT LẬP TRÌNH**  **BÀI TẬP LỚN** |

|  |  |
| --- | --- |
| Giảng viên hướng dẫn: | ThS. TRẦN PHONG NHÃ |
| Sinh viên thực hiện: | PHẠM THỊ MỸ HẰNG |
| Lớp : | CQ.CNTT.K65 |
| Khóa: | 65 |
|  |  |

|  |
| --- |
| TP.Hồ Chí Minh, ngày 7 tháng 5 năm 2025 |

# **LỜI CẢM ƠN.**

# **NHẬN XÉT CỦA GIẢNG VIÊN.**

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

# **MỤC LỤC**

[LỜI CẢM ƠN. i](#_Toc198602664)

[NHẬN XÉT CỦA GIẢNG VIÊN. ii](#_Toc198602665)

[MỤC LỤC iv](#_Toc198602666)

[DANH MỤC VIẾT TẮT. vi](#_Toc198602667)

[DANH MỤC HÌNH ẢNH vi](#_Toc198602668)

[PHẦN I: LÝ THUYẾT 1](#_Toc198602669)

[1. Hàm. 1](#_Toc198602670)

[1.1 Các khái niệm 1](#_Toc198602671)

[1.2 Xây dựng hàm 2](#_Toc198602672)

[1.3 Sử dụng hàm 3](#_Toc198602673)

[1.4 Nguyên tắc hoạt động của hàm 3](#_Toc198602674)

[2. Con trỏ. 3](#_Toc198602675)

[2.1 Con trỏ và địa chỉ. 3](#_Toc198602676)

[2.2 Quy tắc sử dụng con trỏ 4](#_Toc198602677)

[2.3 Các phép toán trên con trỏ. 4](#_Toc198602678)

[3. Con trỏ mảng. 5](#_Toc198602679)

[3.1 Con trỏ và mảng một chiều. 5](#_Toc198602680)

[3.2 Con trỏ và mảng nhiều chiều. 6](#_Toc198602681)

[3.3 Áp dụng bộ nhớ động cho con trỏ mảng. 7](#_Toc198602682)

[4. Mảng con trỏ. 8](#_Toc198602683)

[5. Con trỏ hàm. 8](#_Toc198602684)

[6. Cấp phát động. 9](#_Toc198602685)

[6.1 Cấp phát động( Dynamic memory allocation). 9](#_Toc198602686)

[6.2 malloc() 10](#_Toc198602687)

[6.3 Hàm calloc(). 11](#_Toc198602688)

[6.4 Hàm realloc() 11](#_Toc198602689)

[6.5 Hàm free() 12](#_Toc198602690)

[7. Xử lý tệp. 12](#_Toc198602691)

[7.1 Tệp tin kiểu nhị phân và tệp tin kiểu văn bản. 12](#_Toc198602692)

[7.2 Các thao tác trên tệp tin 13](#_Toc198602693)

[8. Kiểu cấu trúc. 17](#_Toc198602694)

[8.1 Định nghĩa 17](#_Toc198602695)

[8.2 Cách dùng 17](#_Toc198602696)

[8.3 Con trỏ struct 18](#_Toc198602697)

[8.4 Struct và trường dữ liệu có liên quan đến struct 19](#_Toc198602698)

[9. Danh sách liên kết. 19](#_Toc198602699)

[9.1 Cấu trúc tự trỏ. 19](#_Toc198602700)

[9.2 Đặc điểm của danh sách liên kết 20](#_Toc198602701)

[9.3 Cách sử dụng và ứng dụng danh sách liên kết 21](#_Toc198602702)

[9.4 So sánh giữa danh sách liên kết và mảng. 23](#_Toc198602703)

[PHẦN II: ỨNG DỤNG. 24](#_Toc198602704)

[TÀI LIỆU THAM KHẢO. 24](#_Toc198602705)

# **DANH MỤC VIẾT TẮT**.

# **DANH MỤC HÌNH ẢNH**

[Hình 1:Nguyên mẫu hàm. 2](#_Toc199029470)

[Hình 2: Hàm CheckMon. 3](#_Toc199029471)

[Hình 3: Sử dụng hàm CheckMon. 3](#_Toc199029472)

[Hình 4: Phép gán con trỏ. 5](#_Toc199029473)

[Hình 5:Sự tương ứng giữa khai báo kiểu hàm và return. 10](#_Toc199029474)

[Hình 6 Hàm CheckMon. 11](#_Toc199029475)

[Hình 7: Hàm CheckMap. 11](#_Toc199029476)

[Hình 8 Sử dụng hàm làm đối số. 11](#_Toc199029477)

[Hình 9:Cung cấp bộ nhớ cho con trỏ bằng malloc. 13](#_Toc199029478)

[Hình 10:Hàm malloc cung cấp bộ nhớ cho mảng so. 14](#_Toc199029479)

[Hình 11: Hàm reaaloc cấp phát bộ nhớ cho mảng a. 15](#_Toc199029480)

[Hình 12: Giair phóng vùng nhớ bằng hàm free. 16](#_Toc199029481)

[Hình 13: Hàm fopen. 17](#_Toc199029482)

[Hình 14 Hàm fprintf 18](#_Toc199029483)

[Hình 15 Hàm fscanf. 18](#_Toc199029484)

[Hình 16 Hàm fgets. 19](#_Toc199029485)

[Hình 17 Cấu trúc tự trỏ Food. 24](#_Toc199029486)

[Hình 18 Hàm Thêm node vào đầu Link List 27](#_Toc199029487)

[Hình 19 Hàm thêm vào node cuoi LinkList. 28](#_Toc199029488)

[Hình 20:n Hàm xoa node đầu Link List. 28](#_Toc199029489)

# **PHẦN I: LÝ THUYẾT**

## **Hàm.**

### **Các khái niệm**

* Tên hàm:

Mỗi hàm được đặt tên ngay từ dòng đầu tiên trong bước xây dựng hàm. Tên hàm được đặt theo các quy tắc như đặt tên cho biến. Mỗi hàm có một tên riêng. Tên hàm được đặt tùy ý nhưng nên đặt tên phù hợp với chức năng của hàm

* Kiểu giá trị của hàm:

Hàm thường cho một giá trị nào đó, giá trị của hàm có thể có kiểu int, float, double,....hoặc không trả về giá trị nào, thực hiện hàm với nhiệm vụ thực hiện các hoạt động thủ tục, lúc này hàm có giá trị là void.Giá trị của hàm được trả về thông qua câu lệnh return dưới dạng return[ biểu thức]. Trong thân hàm có thể sử dụng một câu lệnh return, có thể sử dụng nhiều câu lệnh return ở những vị rí khác nhau và cũng có thể không sử dụng câu lệnh này( trong trường hợp giá trị của hàm là void).

* Đối hay tham số hình thức

Đối hay còn gọi là tham số hình thức. Hàm thường có một vài đối hoặc cũng có thể không có đối nào. Đối của hàm có thể là tham chiếu hoặc tham trị, việc dùng tham chiếu hay tham trị phụ thuộc vào mục đích của hàm thay. Tham trị truyền giá trị sao chép, không làm thay đổi biến gốc. Tham chiếu truyền địa chỉ gốc thay đổi trong hàm có thể làm thay đổi giá trị của biến gốc. Sử dụng tham chiếu trong C liên quan mật thiết đến con trỏ.

* Thân hàm

Thân hàm nằm ngay sau dòng tiêu đề. Thân hàm là nội dung chính của hàm bắt đầu bằng dấu { và kết thúc bằng dấu }. Trong thân hàm chứa các câu lệnh cần thiết để thực hiện một yêu cầu nào đó đã đề ra cho hàm.

Trong thân hàm sẽ có một số biến được khai báo nhằm phục vụ cho chức năng của hàm, biến đó được gọi là biến cục bộ. Vòng đời của biến cục bộ chỉ kéo dài trong khoảng thời gian hoạt động của hàm, tức là biến cục bộ chỉ có tác dụng trong hàm và không có bất cứ mối liên hệ nào với các biến của các hàm khác. Do đó tên của các biến này có thể bị trùng so với các biến của hàm khác nhưng không được trùng với tên của đối.

* Khai báo hàm( nguyên mẫu hàm- prototype)

Nguyên mẫu hàm có dạng:

Kieu\_du\_lieu\_cua\_ham Ten\_ham(kieu\_du\_lieu (ten\_doi),....);

Nguyên mẫu của hàm thực chất là dòng đầu của hàm và thêm vào dấu chấm phẩy.

Không nhất thiết phải khai báo nguyên mẫu hàm. Nhưng nên có, vì nó cho phép chương trình biên dịch phát hiện lỗi khi gọi hàm, một số lỗi thường gặp là tham số đối và tham só hình thức không khớp vs nhau về kiểu số lượng hoặc không khớp về kiểu dữ liệu). Đồng thời khai báo nguyên mẫu hàm trong khi người viết mã viết hàm main hoặc hàm lớn hơn trước khi xây dựng hàm.

* Lời gọi hàm

Hàm được sử dụng thông qua lời gọi hàm. Lời gọi hàm có dạng:

Ten\_ham([danh sách các tham số thực]);

Cần chú ý: số lượng và kiểu dữ liệu của tham số thực phải tương ứng với tham số đối. Nếu hàm trả về một giá trị nhất định (khác void) thì cần phải được gán vào một biến tương ứng.

* Tham số thực

Tham số thực có thể được xem là mối kiên can chặt chẽ nhất của hàm và hàm chứa nó. Gía trị của tham số thực được truyền vào sao cho phù hợp với tham số đối tương ứng, cần phải tương thức về liểu dữ liệu của tham số thực và tham số hình thức. VD: tham số đối là kiểu con trỏ, thì tham số thực tương ứng là địa chỉ.

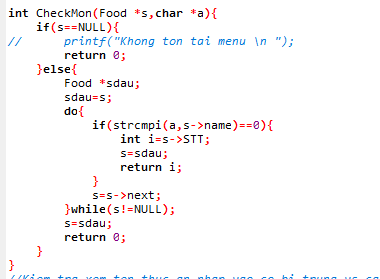
Vd:

Lời khai báo hàm CheckMon:



Hình :Nguyên mẫu hàm.

Hàm được xây dựng như sau:



Hình : Hàm CheckMon.

Và hàm được sử dụng như sau:



Hình : Sử dụng hàm CheckMon.

Trong hàm mẹ chứa hàm CheckMon , Hàm CheckMon được gọi với tham số thực là s và a. Hàm có nhiệm vụ kiểm tra xem trong danh sách liên kết có node nao có duwxx liệu trùngvới chuỗi a hay không, nếu có hàm trả về thứ tự của node, nếu không hàm trả về 0.

### **Xây dựng hàm**

Hàm có thể xem là một đơn vị đọc lập của chương trình. Vai trò của các hàm là ngang nhau. Vì vậy không thể xây dựng hàm này bên trong một hàm khác

Xây dựng hàm bao gồm:

+Dòng tiêu đề: Bao gồm khai báo kiểu giá trị của hàm, đặt tên hàm,khai báo các đối.

+Thân hàm: Bao gồm đưa ra các giá trị cần thiết để thực hiện nhiệm vụ đề ra cho hàm.

### **Sử dụng hàm**

Hàm được sử dụng thông qua lời gọi hàm.

### **Nguyên tắc hoạt động của hàm**

Khi gặp một lời gọi hàm ở một chỗ nào đó của chương trình, thì máy sẽ dừng ngay tại vị trí đó và chuyển đến hàm tương ứng và lần lượt thự hiện các bước trình tự:

* Cấp phát bộ nhớ cho các tham số và các biến cục bộ.
* Gán giá trị của tham số hực cho các đối tương ứng.
* Thực hiện các lệnh trong thân hàm.
* Khi gặp câu lệnh return hàm sẽ trả về giá trị tương ứng và gán cho hàm, hoặc gặp dấu } cuối cùng của hàm, máy sẽ giải phóng bộ nhớ của các đối, các biến cục bộ và kết thúc hàm quay lại hàm chính.

Chính nguyên tắc nay ảnh hưởng đến cách dùng tham trị, tham chiếu cần nắm rõ để có thể sử dụng chính xác.

## **Con trỏ.**

### **Con trỏ và địa chỉ**.

Khi khai báo một biến, máy sẽ cấp phát khoảng nhớ tương ứng với kiểu dữ liệu của nó. Và số thứ tự của byte đầu tiên trong dãy các byte tạo ra cho khoảng nhớ ấy được gọi là địa chỉ.

Và con trỏ là biến đẻ lưu giá trị địa chỉ. Kiểu dữ liệu của con trỏ sẽ phụ thuộc vào kiểu dữ liệu của địa chỉ của biến và do đó cũng phụ thuộc vào kiểu dữ liệu của biến. Phép ép kiểu dữ liệ của con trỏ cũng giống như phép ép kiểu dữ liệu của biến: (kiểu dữ liệu\*).

Tùy vào cách sử dụng mà có thể dùng con trỏ cấp 1, con trỏ cấp 2, con trỏ cấp 3.

Đối với một biến thông thường cách lấy địa chỏ của nó là sử dụng phép toán “&”.

### **Quy tắc sử dụng con trỏ**

Trong khai báo, tên của con trỏ dựa theo quy định chung của biến. Biến thường, con trỏ, mảng có cùng chung kiểu dữ liệu có thể được khai báo trên cùng một câu lệnh khai báo.

Trong khi sử dụng con trỏ, ta tập trung vào hai cách sử dụng chủ yếu:

* Sử dụng tên con trỏ: Con trỏ cũng có thể được xem là một biến đặt biệt, và giá trị của nó là một địa chỉ. Tên con trỏ có thể được vận dụng trng các trường hợp tăng giảm vị trí phần tử của mảng, hoặc dùng cho phép gán.
* Sử dụng dạng khai báo của con trỏ: Nghĩa là bổ sung phép truy xuất ”\*”. Từ con trỏ truy cập vào địa chỉ ( giá trị mà con trỏ chứa ) , rồi lại truy cập vào giá trị mà tại địa chỉ đó lưu giữ. Nghĩa là khi biết được địa chỉ của một biến thì chẳng những chúng ta có thể sử dụng giá trị của nó mà còn có thể truy cập thay đổi, thực hiện các phép toán như: gán, cộng, trừ, nhân chia,...

Cách sử dụng con trỏ liên quan mật thiết đến đối của hàm với tên gọi là tham chiếu.

### **Các phép toán trên con trỏ.**

* Phép gán: Phép gán chỉ có thể thực hiện trên các dữ liệu cùng kiểu, nếu hai kiểu dữ liệu khác khiểu nhau phải dùng phép ép kiểu dữ liệu.

Vd:



Hình : Phép gán con trỏ.

\*s và sdau là hai kiểu con trỏ Food , cùng kiểu dữ liệu nêu có thể thực hiện phép gán.

Đối với việc bên phải là tên con trỏ thì bên phải dấu bằng dữ liệu phải là một địa chỉ. Hai biến con trỏ cùng bậc , cùng kiểu dữ liệu có thể thực hiệ phép gán.

VD: float \*c,a,\*p,f;

c=&a;

p=c;

* Từ đây, \*c và a là tương đương nhau.
* Phép tăng giảm địa chỉ: chức năng này giành cho mảng. Mỗi lần tăng lên i đơn vị trong biểu thức tăng vị trí địa chỉ của con trỏ sẽ tăng lên i\*sizeof(kiểu dữ liệu tương ứng với con trỏ). Được minh họa rõ nhất trong các bài tập truy cập mảng.

Vd: Float \*pa, a;

pa=&a;

pa++;

=>lúc này bị trí của con trỏ tăng lên 4 byte so với ban đầu;

## **Con trỏ mảng.**

Trong C, mối quan hệ giữa mảng và con trỏ vô cùng chặt chẽ. Các phần tử của mảng có thể được xác định, thay đổi thhoong qua chỉ số hoặc thông qua con trỏ mảng.

### **Con trỏ và mảng một chiều.**

* **Phép toán lấy địa chỉ.**

Đối với mảng một chiều, phép lấy địa chỉ vẫn được áp dụng và sử dụng nhanh.

VD: float a[10];

scanf (“%f”,&a[1]) // &a[1] cho ta địa chỉ của xủa phần tử ở vị trí thứ 1.

* **Phép cộng địa chỉ mảng một chiều.**

Tên mảng là một hằng địa chỉ, nó là địa chỉ của phần tử đầu tiên của mảng.

Khi khai báo một mảng float a[10]; máy sẽ cấp phát cho mảng một khoảng nhớ 10 sizeof (float) liên tiếp. Vì bộ nhớ liên tiếp nên chỉ cần biết được địa chỉ của một phần tử của mảng ta được phép dùng các phép toán tăng giảm địa chỉ để lấy dữ liệu của các phần tử còn lại.

Với tên mảng ta đã biết được địa chỉ của phần tử đầu tiên. Như vậy với mọi hoàn cảnh, ta có:

+ a tương đương với &a[0];

+ a+i tương đương với &a[i].

Đối với con trỏ mảng cũng giống như con trỏ đã nói đến ở mục 3. Sử dụng toán tử \* để truy cập vào giá trị mà tại địa chỉ đó đang chứa.

\*(a+i) tương đương với a[i].

Để quản lý một mảng bằng con trỏ, ngoài cách dùng chính tên của con trỏ ta cũng có thể dùng con trỏ cùng kiểu dữ liệu để quản lý mảng.

VD:

float a[10];

float \*b;

b=a;// Cho con trỏ b trỏ vào địa chỉ đầu tiên của mảng a

int i;

i=1;

printf (“Phan tu thu i: %2.2f \n ”,i,b[i]);// Hoặc có thể thay thế b[i] bằng a[i] hoặc \* (a+i) hoặc \*(b+i). Vì bây giờ bốn cách viết là như nhau.

Đồng thời, nếu con trỏ b trỏ vào một phần tử a[k] với k>0 và k<11 thì:

+ pa+i tương đương với &a[k+i];

+pa-i tương đương với &a[k-i];

+ \*(pa+i) tương đương với a[k+i];

+\*(pa-i) tương đương với a[k-i];

* **Con trỏ mảng là tham số hàm**.

Mảng và con trỏ của mảng được sử dụng trong hàm. Khi sử dụng vẫn sẽ tuân theo quy luật đối tương ứng giữa tham số hình thức và tham số thực, Nếu tham số thực là một mảng a, có thể biểu diễn bằng tên mảng (như đã nói, tên mảng là địa chỉ đầu tiên của mảng) hoặc tên một con trỏ trỏ vào đầu mảng a. Vậy thì tham số hình thức tương ứng phải là địa chỉ tương ứng, được lấy thông qua con trỏ.Địa chỉ tương ứng phải cùng kiểu dữ liệu với tham số thực, nghĩa là mảng a thuộc kiểu int (float, char,... ) thì tham số hình thức của hàm phải là địa chỉ tương ứng kiểu int\* (float\*, char \*,...).

### **Con trỏ và mảng nhiều chiều.**

Việc xử lý mảng nhiều chiều phức tạp hơn mảng một chiều. Không phải quy tắc nào áp dụng cho mảng một chiều đều có thể áp dụng cho mảng nhiều chiều.

* Phép toán lấy địa chỉ.

Phép toán lấy địa chỉ (&) không dùng được với mảng nhiều chiều (ngoại trừ mảng 2 chiều kiểu số nguyên).

VD: float a[10][5]

&a[i][j] là không hợp lệ;

* **Phép cộng địa chỉ trong mảng hai chiều.**

Bộ nhớ trong máy tính luôn là một khoảng nhớ hàng ngang. Vì vậy khi khai báo một mảng có nhiều chiều tùy vào khoảng mà người biên dịch yêu cầu mà tạo ra một khaongr nhớ riêng.

Vd: Tạo một mảng hai chiều a[2][3] sẽ tạo ra 6 ô nhớ sizeof và xắp sếp nó theo hàng ngang: a[0][0] a[0][1] a[0][2] a[1][0] a[1][1] a[1][2].

Và ngôn ngữ lập trình C sẽ không hiểu ngôn ngữ ma trận, mà C hiểu là mảng a sẽ bao gồm 1 mảng một chiều chứa hai phần tử, mà mỗi thành phần của mảng a là một mảng một chiều bao gồm 3 phần tử.

Vậy nên a là tên mảng nên nó sẽ là địa chỉ đầu tiên của mảng tức là địa chỉ của a[0][0], a+1 sẽ trỏ tới thành phần thứ hai của mảng là mảng 3 phần tử sau mảng 3 phần tử ban đầu, tức là trỏ tơi a[1][0].

Tương tự như vậy từ 1 đến i.

* **Con trỏ và mảng hai chiều**

Sử dụng con trỏ ta vẫn có thể truy cập vào các phần tử của mảng hai chiều.

VD: float \*a, b[2][3];

a=(float\*)b;

Khi đó: a trỏ tới b[0][0]

a+1 trỏ tới b[0][1]

....

Khi sử dụng con trỏ để truy cập vào phần tử của mảng hai chiều, phía sau phép tử toán tử gán ‘=’ cần dùng phép ép kiểu dữ liệu để đưa mảng về cùng kiểu dữ liệu con trỏ.

* **Sử dụng biến con trỏ trung gian.**

Sử dụng biến con trỏ trung gian là cách đơn giản hơn để vào số liệu cho mảng nhiều chiều.

Đọc một giá trị và chứa tạm vào một biến trung gian, sau đó ta gán biến này cho phần tử mảng.

### **Áp dụng bộ nhớ động cho con trỏ mảng.**

Đối với mảng một chiều hay mảng nhiều chiều ta đều có thể dùng bộ nhớ động để tạo bộ nhớ cho nó. Có thể dùng malloc, calloc, hay realloc, hoặc free.

Nhưng cách tối ưu nhất đó là cho nó một khoảng nhớ ngay từ bang đầu theo dự kiến, sau đó thực hiện các thao tác nhập dữ liệu, đến khi số phần tử của mảng đã bằng với khoảng mà bộ nhớ đề ra ta dùng reaalloc để bổ sung bộ nhớ (thường sẽ tạo ra khoảng nhớ gấp đôi bộ nhớ ban đầu). Như vậy sẽ đảm báo cho việc đủ bộ nhớ dùng đồng thời cũng sẽ tối ưu không cần cho quá nhiều khoảng nhớ gây lãng phí tài nguyên, đảm bảo thời gian tối ưu không cần lặp đi lặp lại nhiều lần copy paste dữ liệu của hàm realloc.

## **Mảng con trỏ.**

Trong mục trên, ta đã đề cập đến định nghĩa mảng trong mảng, và đi sâu vào bản chất của nó chính là mảng con trỏ, một khái niệm mở rộng của con trỏ. Mảng con trỏ là một mảng mà mỗi phần tử của nó có thể chứa được một địa chỉ nào đó.

Tính chất của mảng con trỏ vẫn tuân theo tính chất của mảng, phần tử của mảng con trỏ có cùng dữ liệu với nhau, và là kiểu dữ liệu của toàn mảng.

Cú pháp khai báo: type \*<tên\_mảng>[N];

Khi máy gặp câu lệnh khai báo trên máy sẽ cấp phát N khoảng nhớ liên tiếp

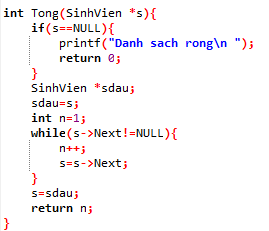
Cách dùng của mảng con trỏ cũng giống như cách dùng của biến thông thường. Có thể dùng một con trỏ bên ngoài trỏ vào một phần tử của mảng. Nhưng mảng con trỏ không thể lưu trữ dữ liệu mà chỉ lưu trữ địa chỉ nơi dữ liệu được lưu.

## **Con trỏ hàm.**

#### **Khai báo con trỏ hàm:**

type \*<tên\_ham>(type1,type2,...);

VD:



Hình :Sự tương ứng giữa khai báo kiểu hàm và return.

Khai báo là hàm kiểu int, vì vậy khi trả về hàm lag một giá trị của biến n kiểu dữ liệu int.

khi sử dụng là con trỏ hàm thì hàm đó sẽ return về giá trị tương ứng

Ta cũng có thể mở rộng khái niệm thành con trỏ khai báo mảng hàm.

VD: type \*<tên\_hàm>[n]( type1,type2,...);

float \*mg[30](float i, int n);

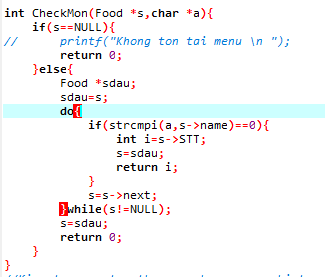
#### **Tác dụng của con trỏ hàm.**

Con trỏ hàm dùng để chứa địa chỉ của hàm, Muốn vậy ta phải thực hiện phép gán tên hàm cho con trỏ hà. Để phép gán có nghĩa thì kiểu hàm và kiểu con trỏ phải tương thích.

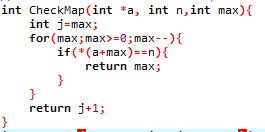
Đồng thời con trỏ hàm cũng có thể là tham số hình thức trong lời gọi hàm. Và tham số hình thưc trong hàm phải là con trỏ hàm cùng kiểu dữ liệu.

VD:

Hàm CheckMap và hàm CheckMon đều là 2 hai hàm riêng biệt được xây dựng với mục đích khác nhau trong chương trình quản lý quán ăn:



Hình Hàm CheckMon.



Hình : Hàm CheckMap.

Nhưng trong hàm ChonMon, hàm CheckMon lại trở thành tham số cho hàm CheckMap, với tham số hình thức và kiểu dữ liêu của hàm CheckMon tương ứng là int.



Hình Sử dụng hàm làm đối số.

Ngoài ra Ta có thể thực hiện phép gán tên hàm cho con trỏ hàm. Để phép gán con trỏ hàm có nghĩa thì kiểu hàm và kiểu con trỏ phải tương ứng, đối số của con trỏ và đối số của hàm phải tương ứng. Sau phép gán ta có thể dùng tên con trỏ hàm thay cho tên hàm.

## **Cấp phát động.**

### **Cấp phát động ( Dynamic memory allocation).**

Chức năng của biến là lưu trữ dữ liệu, để thực hiện chức năng đó máy tính cần cung cấp bộ nhớ cho biến. Việc khai báo biến trước khi sử dụng biến chính là việc gửi yêu cầu cho máy tính cấp phát bộ nhớ cho biến. Và cách khai báo biến đơn thuần này được gọi là cấp phát tĩnh.

VD: int n; //cấp phát bộ nhớ 4 byte cho biến n;

Int \*a;

/\*con trỏ a cs thể là địa chỉ của một biến hoặc là địa chỉ của một mảng nhưng khi khai báo con trỏ thì bước đầu máy tính luôn luôn luôn cung cấp một khoảng nhớ sizeof (int) để lưu dữ liệu của con trỏ.\*/

Tuy nhiên, không phải khi nào số lượng và kích thước của biến cũng được xác định ngay từ khi biên dịch, chẳng hạn nó có thể phụ thuộc vào các thông tin người dùng cung cấp khi thực hiện chương trình. Và cấp phát động là một giải pháp để xin cấp phát bộ nhớ một cách linh hoạt thay vì phải dùng phương pháp cấp phát tĩnh.

Cấp phát động thường dùng cho việc cấp phát bộ nhớ mảng động hoặc các cấu trúc dữ liệu, ứng dụng vào các cấu trúc dữ liệu quan trọng như: mảng động,link list, cây nhị phân,...

Bộ nhớ của việc cấp phát mảng động là khoảng nhớ cung cấp trong vùng nhớ head, điều này khác với cấp phát tĩnh nằm trong vùng nhớ stack( vùng nhớ nhỏ, tự động thu hồi khi hàm kết thúc ), vùng nhớ rộng nhưng thu hồi bằng phương pháp thủ công, nếu phương pháp thu hồi không hiệu quả hoặc người biên dịch khong kiểm soát tốt dữ liệu dễ xảy ra tràn bộ nhớ, dẫn đến lỗi strack chương trình, mất dữ liệu.

Trong ngôn ngữ lập trình C, các hoạt động chủ yếu trong kỹ thuật cấp phát động bao gồm: malloc(), calloc(), realloc(), free() ( thuộc thư mục stdlib.h).

### **malloc()**

malloc( memory allocation). Hàm malloc() dùng để xin một vùng nhớ theo số byte mà người biên dịch yêu cầu.

Hàm malloc() tạo ra một vùng nhớ và giá trị trong vùng nớ đó sẽ là giá trị rác. Đồng thời khi hàm cấp phát bộ nhớ thành công hàm sẽ trả về giá trị kiểu void , đây là lí do ta phải ép kiểu dữ liệu cho câu lệnh. Và ngược lại nếu cấp phát vùng nhớ thất bại hàm sẽ trả về NULL. Do đó mỗi khi sử dụng hàm malloc cần kiểm soát chặt chẽ việc máy tính có cung cấp đủ bộ nhớ hay không tránh cho việc chương trình bị crack, mất dữ liệu.

Hàm malloc cung cấp bộ nhớ hoàn toàn mới và sẽ không dữ lại dữ liệu cũ trong con trỏ. Có thể nói là cấp lại một vùng nhớ hoàn toàn mới ban đầu cho biến.

Cú pháp của hàm : (void\*) malloc(size\_byte);

Vd: char \*a;

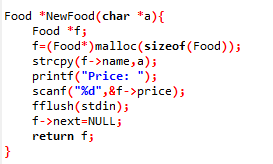
a=(char\*)malloc(10\*sizeof(char));

Trong đó: a là tên con trỏ(địa chỉ của biến cần cung cấp bộ nhớ);

char\* là phép ép kiểu dữ liệu của bộ nhớ cần cung cấp, cùng kiểu dữ liệu của a;

10\*sizeof(char) là cách phổ biến nhất để tính dễ dàng số byte cần cung cấp thông qua hàm sizeof(kieu\_du\_lieu);

Trong hàm NewFood dưới đây, ta vận dụng hàm malloc để gán bộ nhớ cho con trỏ kiểu Food f. Chỉ khi đó cá lệnh tiếp theo mới có thể hoạt động hiệu quả.



Hình :Cung cấp bộ nhớ cho con trỏ bằng malloc.

### **Hàm calloc().**

Callloc( contiguous allocation), hàm có chức năng tương tự như hàm malloc(). Nhưng khác ở chỗ giá trị ban đầu được gán vào khoảng nhớ cung cấp có giá trị là 0. Việc này sẽ có lợi cho một số bài toán nhất định, như mảng đếm, tạo mảng 0, tạo ma trận 0, bản khới đầu, hoặc bộ đệm,...

Cú pháp: void\* calloc(size\_t nitems, size\_t size)

VD: char \*a;

a=(char\*) calloc(10, sizeof(char));

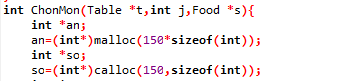
Trong đó: a là con trỏ( địa chỉ) đầu của khỏng nhớ cần được bổ sung;

(char\*) phép ép kiểu dữ liệu từ void sang kiểu dữ liệu của a;

10 số khoảng nhớ có độ rộng sizeof(kieu\_du\_lieu) của a;

VD:

Để dễ dàng tính toán.ta cũng có thể khởi tạo tất cả phần tử trong mảng có tên so đều có giá trị bằng 0, thông qua hàm malloc.



Hình :Hàm malloc cung cấp bộ nhớ cho mảng so.

### **Hàm realloc()**

Realloc( real allocation – cấp phát lại).

Khác với hàm malloc() và hàm calloc(), hàm realloc() sẽ coppy lại dữ liệu cũ của biến sau đó đến một vùng nhớ cung cấp sao lưu lại dữ liệu ấy, sau đó cung cấp thêm vùng nhớ sao cho vừa đủ với yêu cầu của người biên dịch.

Hàm realloc() cũng giống hàm calloc() và malloc() đều trả về giá trị là void và cần phải ép kiểu dữ liệu cho phù hợp với biến. Thất bại cung cấp lại bộ nhớ trả về NULL.

Trong khi dùng realloc() nhằm tránh để mất hoàn toàn dữ liệu khi tràn bộ nhớ cần lưu dữ liệu vào một biến tạm thời để tránh mất dữ liệu

Đồng thời dữ liệu mà vùng nhớ cung cấp thêm sẽ là dữ liệu rác.

Với chức năng trên, hàm realloc() thường được dùng để bổ sung thêm bộ nhớ cho biến mà cần giữ lại dữ liệu cũ, quen thuộc nhất là cung cấp dữ liệu cho mảng,link list sau khi dùng malloc(), calloc().

Cần cân nhăc kỹ trước khi dùng cấp phát động để đưa ra phương pháp hợp lý, vì mỗi lần cấp phát lại bộ nhớ hàm realloc sẽ phải thực hiện chức nawg coppy and pase trước, nếu sử dụng hàm realloc() không cần thiết gây mất thời goan hap phí tài nguyên của máy.

Cú pháp: void \*realloc(void \*pa, size\_t size)

Vd: int a[10];

//Bây giờ cần 15 phần tử int

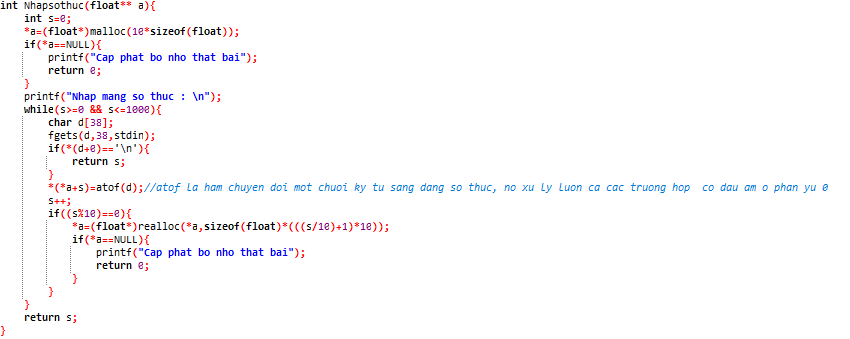
a=(int\*)realloc(a,15):

Trong đó: a là con trỏ địa chỉ của mảng;

(int\*) là phép ép kiểu dữ liệu tuef void sang kiểu dữ liệu của a.

15 là độ lớn mới của mảng, đã bao gồm độ dài mảng cũ.

Đưa vào thực tế hàm như sau:



Hình : Hàm reaaloc cấp phát bộ nhớ cho mảng a.

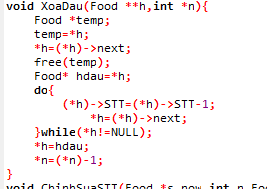
### **Hàm free()**

Bởi vì các hoạt động cấp phát động đều hoạt động trên vùng nhớ head, nên sẽ không tự mất đi khi chương trình hàm kết thúc, vòng đời của nó chỉ kết thúc khi chương trình kết thúc. Vì vậy nhằm tránh cho tình trạng crack do tràn bộ nhớ xảy ra gây lỗi chương trình, mất dữ liệu người biên dịch cần giải phóng nó, kiểm soát vùng nhớ hiệu quả. Đây là nhiệm vụ của lệnh free(). Và nếu giá trị của biến là NULL thì lệnh free() sẽ không có ảnh hưởng gì.

Cú pháp : free(ten\_bien);

VD: Hàm xóa node đầu tiên của danh sách liên kết như sau:

Để tránh cho việc mất hoặc trộn lẫn dữ liệu, ta gán địa chỉ đầu tiên cho con trỏ cùng kiểu dữ liệu temp, sau đó cho con trỏ h tiến tới một bước và đặt node đó là node đầu tiên. Nếu như lúc này ta không giải phóng biến temp, rất dễ xảy ra việc laanx lộn dữ liệu, và đẻ giải phóng temp ta dùng hàm free().



Hình : Giair phóng vùng nhớ bằng hàm free.

## **Xử lý tệp.**

### **Tệp tin kiểu nhị phân và tệp tin kiểu văn bản.**

* Kiểu nhị phân

Trong quá trình nhập xuất, dữ liệu không bị biến đổi. Dữ liệu ghi trên tệp theo các byte nhị phân trong bộ nhớ.

Khi kết thúc tệp hàm trả về giá trị mã EOF (-1) và hàm feof cho giá trị khác 0.

* Kiểu văn bản

Kiểu nhập xuất văn bản chỉ khac kiểu nhị phân khi xử lý kí tự chuyển dòng ( mã 10 ) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc như nhau.

Khi ghi, một ký tự mã 10 (\n) được lưu dưới dạng hai ký tự khác nhau bao gồm ký tự mã 12 (\0) và ký tự mã 10 (\n). Khi đọc, hai ký tự liên tiếp (\0) và (\n) liên tiếp nhau đọc thành ký tự (\n).

Trong tệp văn bản, khi gặp ký tự mã 26 hoặc cuối tệp, máy tính đều trả về mã EOF (-1) và trả về hàm feof giá trị khác không (-1).

Ta thực hiện các thao tác với tệp tin nhị phân và văn bản bằng các hàm thao tác tệp tin. Trong C, các hàm này được chia thành 2 nhóm: cấp 1 và cấp 2.

### **Các thao tác trên tệp tin**

#### **Tạo File**

Để có thể làm việc với file, bắt buộc file đã được mở và địa chỉ của file phải được gán cho một con trỏ để chương trình có thể làm việc với tệp.

Con trỏ để trỏ về file có dạng khai báo: FiLE \*<tên\_con\_trỏ>;

VD: FILE \*f;

Trong C, hàm mở tệp tin là hàm fopen. Cú pháp: f=fopen(“tên\_tệp”,”kiểu”).

Tham số thứ nhất trong hàm fopen là tên file muốn mở với điều kiện tệp đó phải cùng thư mục với tệp chương trình, nếu tệp tin nằm khác thư mục tham số là dãy đường dẫn tuyệt đối.

Tham số thứ hai là kiểu tệp, mục đích để mở tệp

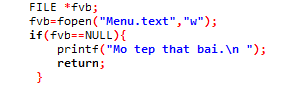
VD: FILE \*f;

f=fopen(“Bài tập lớn”,”r”);

Với: “Bài tập lớn”: là tên tệp; “r ” kiểu mode để mở một tệp để đọc văn bản. Tệp cần tồn tại nếu không sẽ bị lỗi.

Hàm fopen mở tệp thành công nó sẽ trả về con trỏ trỏ tới tệp đã mở. Nếu thất bạo hàm trả về con trỏ giá trị NULL.

VD:



Hình : Hàm fopen.

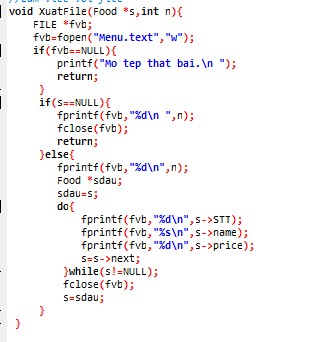
Ta dùng hàm fopen để mở tệp “Menu.text”, với con trỏ tệp FILE \*fvb, với ký hiệu định dạng là “w” dùng để mở tệp để ghi, nếu tệp đã tồn tại nó bị xóa. Nhằm tránh mất kiểm soát dùng lệnh if để kiểm tra tệp có mở được không, nếu không thì dừng và báo lỗi tránh việc mất kiểm soát chương trình.

#### **Thao tác trên tệp tin văn bản**

* Hàm fprintf: hoạt động giống như hàm printf, dùng để ghi dữ liệu theo khuôn dạng lên tệp. Nếu thành công hàm trả về số byte ghi lên tệp,trả về giá trị kiểu int, nếu thất bại hàm cho mã EOF.

Cú pháp: fprintf (FILE \*fp,char format, đối....);

VD:

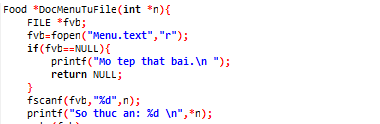


Hình Hàm fprintf

* Hàm fprintf là công cụ chính trong thao tác ghi dữ liệu vào tệp
* Hàm fscanf: hàm hoạt động như hàm scanf , dùng để đọc dữ liệu từ tệp, biến đổi theo khuôn dạng và lưu kết quả vào đối. Hàm trả về một giá trị bằng số trường được đọc.

Cú pháp : fscanf (FILE \*f, format,danh sách đối,...);

VD:



Hình Hàm fscanf.

* Hàm fputs: hoạt động giống như hàm puts(), ghi chuỗi lên tệp. Khi thành công hàm trả về số byte ghi lên tệp, nếu thất bài hàm trả về EOF.

Cú pháp: fputs (const char \*s,FILE \*f).

* Hàm fgets : hoạt động giống như hàm get(), nhưng an toàn hơn so với gets().

Công dụng: Đọc một dãy ký tự từ tệp f vào vùng nhớ s, việc đọc kết thúc khi:

+ Hoặc đã đọc n-1 ký tự.

+ Hoặc gặp dấu xuống dòng (cặp mã 13,10). Khi đó mã 10 được đưa vào xâu kết quả.

+Hoặc kết thúc tệp

Hàm fgets có thể nhận hoặc không nhận ký tự xuống dòng \n. Và xâu ký tự được bổ sung thêm ký tự kết thúc xâu \0.

Hàm trả về địa chỉ của vùng nhận kết quả. Khi thất bại hàm trảm trả về giá trị NULL.

Cú pháp: fgets(char \*s, int n,FILE \*fp);

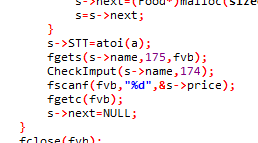
Trong đó:

Char \*s: là con trỏ s nơi giữ liệu đọc được dược lưu vào.

Int n: dự đoán số ký tự được nhập vào, trở thành 1 trong những điều kiện dừng.

FILE \*fp: tên tệp nguồn, đã đc mở.

VD:



Hình Hàm fgets.

#### **Các thao tác trên tệp tin nhị phân:**

Tệp tin nhị phân có những thuận lợi hơn so với tệp văn bản, do đó đây là kiểu tệp tin thường dùng:

+ Truy cập tệp một cách ngẫu nhiên dễ dàng.

+ Dữ liệu có thể đọc ghi theo từng khối (block).

+ Hầu hết các hàm dùng cho tệp văn bản đều dùng được cho hàm nhị phân, ngoại trừ các hàm fgets, fputs,fprintf, fscanf.

Ngoài ra, đối với tệp nhị phân, C còn cung cấp một ssos hàm đặc biệt chỉ dùng cho tệp nhị phân.

* Hàm putw: dùng để ghi một số nguyên (2 byte) lên tệp.

Cú pháp: putw (int \*n,FILE\*f);

* Hàm getw: dùng để đọc một số nguyên từ tệp.

Cú pháp: get (FILE \*f);

* Hàm fwrite: dùng để ghi n mẫu tin kích thước size từ vùng nhớ trỏ bởi ptr lên tệp fp, hàm trả về số mẫu tin thực sự ghi.

Cú pháp: fwrite (void \*ptr, int size, int n, FILE \*fp);

Trong đó:

ptr là con trỏ mà trỏ mà ở đó lưu trữ dữ liệu gốc, dùng dữ liệu đó đưa vào tệp nhị phân.

Size: Khích thước kiểu dữ liệu đọc, thường nhập một số byte cụ thể hoặc tính byte thông qua hàm sizeof.

N: Số ký tự đọc được từ file dự kiến.

Fp: Con trỏ quản lý tệp.

* Hàm fread: dùng để đọc n mẫu tin kích thước size từ tệp fp lên vùng nhớ trỏ bởi ptr, hàm trả về số mẫu tin thực sự được đọc.

Cú pháp: fread(void \*ptr,int sze, int n, FILE \*fp);

Trong đó:

ptr là con trỏ mà sau khi đọc được từ tệp tin sẽ lưu dữ liệu vào vùng nhớ mà con trỏ quản lý.

Size: Khích thước kiểu dữ liệu đọc, thường nhập một số byte cụ thể hoặc tính byte thông qua hàm sizeof.

N: Số ký tự đọc được từ file dự kiến.

Fp: Con trỏ quản lý tệp.

#### **Các hàm khác**

Ngoài các hàm được tách riêng dành cho tệp nhị phân và tệp văn bản, hai dạng tệp vẫn sử dụng các hàm cấp 1 chung :

* Hàm fflush: Dạng hàm int, làm sạch vùng nhớ đệm của tệp. Nếu hàm thành công hàm cho giá trị 0, ngược lại cho EOF.

Cú pháp: fflush(FILE \*fp);

VD:

* Hàm fflushall: Dạng hàm int, làm sạch vùng nhớ đệm của tất cả các tệp đang mở, Nếu thành công hàm trả về giá trị nguyên là số tệp đang mở, ngược lại trả về EOF.

Cú pháp: fflushall();

* Hàm feof: Dạng hàm int, Hàm dùng để kiểm tra cuối tệp. Hàm trả về giá trị nguyên khác 0 nếu gặp cuối tệp, ngược lại trả về EOF.

Cú pháp: feof(FILE\* fp);

VD:

* Hàm rewind: Dạng hàm void, chuyển con trỏ về vị trí đầu tệp.

Cú pháp: rewind(FILE \*fp);

* Hàm fseek: Dạng hàm int , hàm di chuyển con trỏ chỉ vị trí của tệp từ vị trí xác định ban đầu qua một số sb byte, chiều di chuển về phía tệp. Khi thành công hàm trả về giá tri 0, ngược lại trả về giá trị khác 0.

Cú pháp: fseek(FILE \*fp,long sb,int xp);

Xp có thể nhận các giá trị sau :

+xp=0=SEEK\_SET: đầu tệp.

+xp=1=SEEK\_CUR: vị trí hiện tại.

+xp=0=SEEK\_SET: cuối tệp.

VD:

## **Kiểu cấu trúc.**

Cấu trúc hay còn gọi là struct là một kiểu dữ liệu mà người dùng tự định nghĩa ra để giải quyết các bào toán thực tế.

### **Định nghĩa**

Cấu trúc (struct) có thể xem như một sự mở rộng của các khái niệm biến, và mảng, nó cho phép lưu trữ và xử lý các dạng thông tin có độ phức tạp lớp hơn bao gồm nhiều kiểu dữ liệu. Cấu trúc là một tập hợp các biến, các mảng và được biểu thị bằng một cái tên duy nhất.

Các phần tử dữ liệu được gọi là các trường(field).

Cấu trúc struct là giải pháp quen thuộc cho các bài toán thực tế, mà đối tượng cần lưu trữ rất nhiều thông tin.

Vd: Cấu trúc struct chứa thông tin của sinh viên bao gồm họ tên, lớp, điểm,....

Cấu trúc struct chứa thông tin của quán ăn bao gồm tên món ăn, giá món ăn,...

### **Cách dùng**

* Khai báo cấu trúc

Để sử dụng cấu trúc struct trong chương trình phải hoàn thành hai bước: định nghĩa, và khai báo struct.

Để định nghĩa cấu trúc struct ta có thể sử dụng hai cách sau:

+struct ten\_struct{

Khai báo thành phần;

};

( Với cách này khi khai báo thì khi khai báo phải kèm theo từ khóa struct,

struct ten\_kieustruct ten\_bien).

Nếu muốn không cần phải kèm theo struct khi khai báo thì cần kèm theo typedef :

typedef struct ten\_struct = ten\_di\_kem ( Có thể giống nhau ).

+typedef struct ten\_struct{

Khai báo thành phần;

}ten\_kieu\_du\_lieu;

Với cách định nghĩa này khi khai báo trong hàm chỉ cần khai báo:

ten\_cau\_truc ten\_bien;

Thành phần của struct có thể là các kiểu dữ liệu cơ bản như char, float, int,...còn có thể chứa các kiểu struct lồng nhau.

* Biến trong hàm

Sau khi khai báo các struct, có thể dùng biến struct như một biến bình thường.

Tuy nhiên,các thành phần chính của struct vẫn là các kiểu dữ liệu cơ bản, nên việc sử lý dữ liệu của struct bao giờ cũng thực hiện trên các kiểu dữ liệu cơ bản nhất của máy tính. Và để truy nhập tới thành phần của cấu trúc cần thông qua tên của biến cấu trúc theo phép truy cập: bien\_cau\_truc.ten\_thanh\_phan.

* Đối của hàm

Biến struct có thể trở thành tham chiếu hoặc tham trị tùy thuộc vào mục đích người biên dịch.

* Trả hàm

Đồng thời khi cần thiết hàm có thể trả về một struct, và trả kết quả trong hàm main().

### **Con trỏ struct**

Sau khi định nghĩa struct và khai báo struct, tên struct có thể xem là một biến đặt biệt, vì vậy kiểu dữ liệu struct vẫn có cấu trúc mảng struct và con trỏ struct.

Con trỏ kiểu cấu trúc được ứng dụng khi bạn xây dựng các cấu trúc kiểu dữ liệu như danh sách liên kết hay cây nhị phân.

Con trỏ kiểu cấu truct dùng để lưu giữ địa chỉ của biến cấu trúc và mảng cấu trúc.

Khai báo con trỏ cấu trúc cũng tương tự như khai báo các loại con trỏ khác:

VD: struct <tên\_cấu\_trúc>\* <tên\_biến\_con\_trỏ>;

Hoặc: <tên\_cấu\_trúc>\* <tên\_biến\_con\_trỏ>;

Khi khai báo con trỏ cấu trúc, biến con trỏ chưa chứa địa chỉ cụ thể, lúc này nó chỉ được cấp phát 2 bytes để lưu trữ địa chỉ nhưng chưa chỉ đến một giá trị cụ thể nào.Do đó khi muốn thao tác bằng con trỏ cấu trúc ta cần:

+ Khai báo con trỏ:

Khai báo con trỏ cấu trúc cũng tương tự như khai báo các loại con trỏ khác:

VD: struct <tên\_cấu\_trúc>\* <tên\_biến\_con\_trỏ>;

Hoặc: <tên\_cấu\_trúc>\* <tên\_biến\_con\_trỏ>;

+ Cấp phát bộ nhớ cho con trỏ cấu trúc:

Thông thường người ta cấp phát bộ nhớ cho con trỏ cấu trúc bằng hàm malloc.

<tên\_cấu \_trúc> \*<tên \_con\_trỏ>;

<tên\_con\_trỏ>=( tên\_cấu \_trúc\*)malloc(n\*sizeof(tên \_con\_trỏ));

Đối với con trỏ (địa chỉ không xác định) dùng phép giải ->

Đối với một biến đã xác định,hoặc một mảng có độ dài nhất định dùng phép giải “.”.

Đối với một hàm cần thay đổi nội dung phải dùng con trỏ struct và dùng tham số phép giải “->”.

### **Struct và trường dữ liệu có liên quan đến struct**

Trong trường hợp thành phần của struct là một chuỗi dữ liệu, thông thường sẽ được khai báo bằng mảng cố định ten\_cau\_truc(độ dài) nhưng cũng có thể là con trỏ. Khi đó, khi bước đầu khai báo, máy tính chỉ cung cấp bộ nhớ để lưu con trỏ ban đầu, do vậy ta cần dùng cấp phát động tạo vùng nhớ cho thành phần đó, haowcj tạo ra một vùng nhớ gián tiếp sau đó gán con trỏ chỉ vào vùng nhớ đó.

Trong trường hợp là mảng, ta cần truy cập vào thành phần mảng thứ i, thông thường sẽ dùng phép truy cập ten\_truong\_du\_lieu[i].

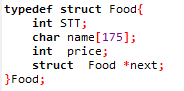
## **Danh sách liên kết.**

### **Cấu trúc tự trỏ.**

Cấu trúc tự trỏ là một phần của danh sách liên kết, tạo thành một node trong danh sách liên kết.

Cấu trúc tự trỏ là kiểu cấu trúc struct, mà trong đó có ít nhất một thành phần là con trỏ kiểu cấu trúc đang định nghĩa.

VD:



Hình Cấu trúc tự trỏ Food.

Cấu trúc Food là một cấu truca tự trỏ với ba thành phần int, char int và chính Food.

Cấu trúc tự trỏ thể được định nghĩa theo nhiều cách :

Cách 1:

Typedef struct <tên\_cấu\_trúc\_tự\_trỏ>{

....//Khai báo các thành phần khác của cấu trúc, thuộc các dạng kiểu dữ liệu cơ bản int, char, float,... hoặc các cấu trúc truct thông thường khác

Struct < tên\_cấu\_trúc\_tự\_trỏ > \* next;

}<tên\_kiểu\_dữ\_liệu\_được\_typedef\_định\_nghĩa\_lại>;

Cach 2:

Typedef struct <tên\_cấu\_trúc\_tự\_trỏ> <tên\_định\_nghĩa\_typedef>;

Struct <tên\_cấu\_trúc\_tự\_trỏ> {

....//Khai báo các thành phần khác của cấu trúc.

<tên\_định\_nghĩa\_typedef> \*next;

};

Cách 3:

Struct <tên\_cấu\_trúc\_tự\_trỏ> {

....//Khai báo các thành phần khác của cấu trúc.

Struct <tên\_cấu\_trúc\_tự\_trỏ> \*next;

};

Typedef <tên\_cấu\_trúc\_tự\_trỏ> <tên\_định\_nghĩa\_typedef>;

Như vậy mỗi một node do cấu trúc tự trỏ tạo ra bao giờ cũng mang theo một phần dữ liệu và một hoặc nhiều con trỏ struct khác, mà các con trỏ này sẽ là một node của danh sách liên kết, các node trongdanh sách liên kết sẽ kết nối với nhau qua cá con trỏ tự trỏ, và tạo thành các mắc xích liên kết, hay còn gọi là danh sách liên kết.

Tính chất của các node cấu trúc tự trỏ quy định nội dung, cách thức hoạt động của danh sách liên kết.

### **Đặc điểm của danh sách liên kết**

Thừa hưởng tính chất của các node thành phần, một danh sách liên kết cần đảm bảo 3 đặc điểm cơ bản:

+ Biết địa chỉ cấu trúc đầu, và địa chỉ đó được lưu vào một con trỏ nào đó( thường sẽ là \*pdau,\*sdau).

+ Trong mỗi node ( trừ node cuối cùng ) chứa địa của cấu trúc tiếp của danh sách link list.

+Node cuối cùng của danh sách liên kết chỉ vào NULL.

Danh sách liên kết có thể có một hoặc nhiều chiều. Các danh sách liên kết như trên là kiểu danh sách liên kết loại 1, với chiều thuận. Ngoài ra còn có danh sách liên kết ngược hoặc danh sách liên kết hai chiều.

Danh sách liên kết ngược:

+ Biết địa chỉ cấu trúc cuối.

+ Trong mỗi node ( trừ node cuối cùng ) chứa địa của cấu trúc trước đó của node trong danh sách link list.

+Cấu trúc đầu chứa hằng số NULL;

Danh sách liên kết hai chiều:

+ Biết địa chỉ của node đầu hoặc node cuối.

+ Trong mỗi node của danh sách liên kết chứa hai phần tử, một chứa địa chỉ của cấu trúc trước, một chứa địa chỉ của cấu trúc sau node.

+ Cấu trúc đầu và cấu trúc sau chứa hàng số NULL.

Với loại danh sách liên kết hai chiều, ta có thể truy cập từ thấp lên cao theo chiều thuận hoặc truy cập từ cao xuống thấp theo chiều ngược nhau

### **Cách sử dụng và ứng dụng danh sách liên kết**

* Cách sử dụng:

+ Giống như con trỏ struct,mỗi khi khai báo một con trỏ thuộc kiểu cấu trúc tự trỏ, máy tính chỉ cung cấp bộn nhớ 2 bytes để lưu trữ bộ nhớ, vì vậy để chỉnh sửa thông tin, tạo node mới trong danh sách liên kết cần phải cung cấp bộ nhớ cho node thông qua phương pháp cấp phát động malloc.

+ Có thể xem mỗi node trong danh sách liên kết là một con trỏ, vì ta truy cập dữ liệu của node đều thông qua địa chỉ được lưu trong con trỏ, hoặc trong con trỏ next của node trước hoặc sau nó.

+ Danh sách liên kết luôn được khai báo dưới dạng con trỏ, do đó ta thường dùng phép giải “->” để truy cập vào dữ liệu thành phần của node.

+ Danh sách liên kết có thể được coi là một mảng dữ liệu quản lí các con trỏ kiểu node.

Vì các đặc trưng trên, tùy vào mục đích mà sử dụng cách gọi con trỏ trong mảng, trong hàm, trả về hàm,...

Cách thức hay dùng trong bài báo cáo này là: khai báo con trỏ danh sách liên kết trong hàm main bằng con trỏ \*, tùy vào mục đích của hàm mà dùng con trỏ cấp 1 hoặc con trỏ cấp 2.

Khi sử dụng danh sách liên kết cần lưu ý về việc nắm giữ địa chỉ đầu hoặc cuối mảng nhằm tránh mất giữ liệu, mất kiểm soát chương trình,...

* Ứng dụng thực tế của danh sách liên kết:

Danh sách liên kết thường dùng cho việc ghi dữ liệu nhờ vào khả năng kích thước có thể mở rộng thường xuyên, việc chèn xóa một node dữ liệu thuận lợi hơn mảng.

Trong thực tế, link list có thể dùng trong: [1]

+ Triển khai các cấu trúc dữ liệu phức tạp: stack và queue, graph ( đồ thị ), hash table với chaining.

+ Quản lý bộ nhớ động.

+ Triển khai Undo/ Redo.

+ Trình phát nhạc, video.

+ Trình duyệt Web.

+ Thuật toán xử lý chuỗi lớn ( phổ biến nhất là các trình soạn thảo).

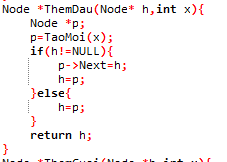
+ Hệ thống bộ nhớ phân mảnh.

+ Danh sách sinh viên trong hệ thống quản lý học sinh, sinh viên, danh sách món ăn được ứng dụng trong ứng dụng quản lý nhà hàng, quán ăn,...

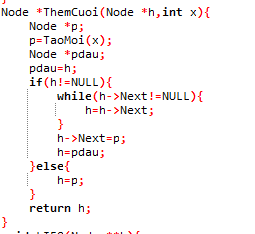
Các thao tác phổ biến nhất của link list bao gồm:

+Thêm đầu, cuối, giữa danh sách liên kết. [2]

VD: hàm thêm đầu, cuối giữa danh sách liên kết đơn.



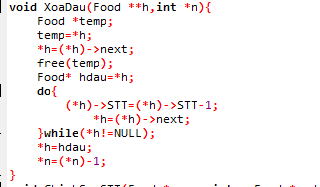
Hình Hàm Thêm node vào đầu Link List



Hình Hàm thêm vào node cuoi LinkList.

+ Xóa đàu, cuối,, giữa danh sách liên kết.

VD: hàm xóa node đầu, giữa va cuoi danh sách liên kết đơn.



Hình :n Hàm xoa node đầu Link List.

void XoaGiua(Food \*\*h,int n,int \*i){

Food \*hdau;

hdau=\*h;

Food \*hbefore;

while(n>1){

if((\*h)->next==NULL){

printf("Xac dinh vi tri mon an sai. \n");

\*h=hdau;

return;

}else{

hbefore=\*h;

\*h=(\*h)->next;

}

n--;

}

if((\*h)->next==NULL){

Food \*temp;

temp=\*h;

hbefore->next=NULL;

free(temp);

\*h=hdau;

\*i=(\*i)-1;

}else{

Food \*temp;

temp=\*h;

hbefore->next=(\*h)->next;

free(temp);

\*h=hbefore->next;

do{

(\*h)->STT=(\*h)->STT-1;

\*h=(\*h)->next;

}while(\*h!=NULL);

\*h=hdau;

\*i=(\*i)-1;

return;

}

}

### **So sánh giữa danh sách liên kết và mảng.**

#### **Ưu điểm**

+ Mở rộng và thu hẹp một cách linh hoạt.

* Thuận lợi hơn so với mảng ( có kích thước cố định, có thể mở rộng nhưng phải dùng realloc, mất nhiều thời gian lãng phí tài nguyên).

+ Số lượng phần tử phụ thuộc vào vùng nhớ head.

* Thuận lợi hơn so với mảng ( có kích thước cố định không quá 1000 phần tử cho một mảng).

+Dễ dàng chèn và xóa các phần tử.

* Thuận lợi hơn so với mảng ( Phải sửa đổi chỉ số ở các phần tử nằm phía sau).

+Sử dụng bộ nhớ không liên tục, dễ phân mảnh.

* Thuận lợi hơn so với mảng trong việc cấp phát bộ nhớ, nhưng cũng sẽ khó khăn khi bộ nhớ không liên tục

#### **Nhược điểm**

+ Mức độ phức tạp hơn khó viết hơn mảng.

+So với mảng trong việc tìm kiếm không phải là ưu thế, chỉ có thể dùng phương pháp tìm kiếm tuyến tính, không thể tìm kiếm theo phương pháp nhị phân. Chỉ có thể triuiy cập vào một node khi đã biết được vị trí của nó hoặc vị trí của node trước hoặc sau node đó.

+ Thao tác sắp sếp của link list phức tạp hơn mảng.

+ Bắt buộc phải tồn tại bộ nhớ phụ để lưu giữ con trỏ.

#### **Tổng kết:**

Tùy thuộc vào yêu cầu dữ liệu mà sử dụng mảng và link list sao cho hợp lý. Thông thường sẽ dựa vào các tiêu chí: Số lượng có biết trước hay không, lớn hay nhỏ, hoạt động chèn và xóa có là mục tiêu của chức năng cần làm.

# **PHẦN II: ỨNG DỤNG.**

Ứng dụng được đề cập trong báo cáo bài tập lớn lần này nhằm quản lý một quán ăn.

Những chức năng cơ bản của ứng dụng sẽ phân thành 2 đối tượng:

* Chủ quán:

Muốn đang nhập vào ứng dụng với vai trò là người chủ quán thì người sử dụng phải có mật khẩu đăng nhập (ứng dụng sử dụng chức năng của các hàm tệp tin văn bản là chủ yếu.).

Với vai trò là người chủ có thể tạo và chỉnh sửa các thông tin của menu thực đơn bao gồm STT, món ăn, giá cả, chỉnh sửa thứ tự trên menu,cũng như xóa hoặc thêm một số món .., nhập thông tin về số bàn ăn của quán

* Với vai trò khách hoặc nhân viên, với tư cách này không cần phải đăng nhập bằng mật khảu, nhân viên quán hoặc khách đều có thể tương tác với các chức năng được, bao gồm chọn bàn ăn, chọn món ăn, chốt số tiền bữa ăn.

Ứng dụng chủ yếu vận dụng các kiến thức về cấu trúc tự trỏ, danh sách liên kết Link List, mảng , con trỏ, hàm,....

# **TÀI LIỆU THAM KHẢO**.

|  |  |
| --- | --- |
| [1] | OpenAI, “ChatGPT,” 20 5 2025. [Trực tuyến]. Available: https://chatgpt.com/. |
| [2] | 28tech, "28tech," 20 5 2025. [Online]. Available: https://blog.28tech.com.vn/dslk-don-them-node-vao-dau-dslk. |