

Lập trình nâng cao

(Lập trình C nâng cao)

Mã học phần: CPM215.3

Nội dung chính

- ▶ NHẮC LẠI MỘT SỐ KIẾN THỨC CƠ BẢN VỀ NGÔN NGỮ C
- ▶ HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH
- ▶ CẤP PHÁT ĐỘNG BỘ NHỚ VÀ DANH SÁCH LIÊN KẾT
- ▶ THAO TÁC TRÊN CÁC TẬP TIN
- ▶ XỬ LÝ NGẮT

Lịch trình giảng dạy

► Lý thuyết:

- Từ ngày Đến ngày : tiết ... thứ ...
- Giảng viên: Họ tên - email - đt
- Địa điểm:

► Bài tập:

- Từ ngày Đến ngày : tiết ... thứ ...
- Giảng viên: Họ tên - email - đt

► Thực hành:

- Từ ngày Đến ngày : tiết ... thứ ...
- Giảng viên: Họ tên - email - đt
- Địa điểm: phòng máy A4

Đánh giá học phần

- ▶ Điểm thành phần (30%)

- Chuyên cần
- Điểm kiểm tra
- Điểm thưởng

- ▶ Điểm thi (70%)

Hình thức thi: Code trên máy tính, thời gian 60 phút

- ▶ Điểm học phần = $0.3 \times \text{Điểm thành phần} + 0.7 \times \text{Điểm thi}$

Tài liệu

- ▶ Giáo trình Lập trình nâng cao - Bộ môn CNPM
~150 trang, thư viện trường
- ▶ Kỹ thuật lập trình C cơ sở và nâng cao, PGS.TS Phạm Văn Ất
~ 40-50K
- ▶ Ebook về kỹ thuật (ngôn ngữ) lập trình C
- ▶ IDE: (download)
 - ▶ DevC ++ 4.9.xx trở lên (<http://www.bloodshed.net/>)
 - ▶ Free C
 - ▶ Turbo C ++ 3.0
 - ▶ Borland C
 - ▶ Visual C++ (Visual Studio)
- ▶ Ref: các trang lập trình online: laptrinhonline.clup, spoj.com,....

Bộ môn & Giảng viên

- ▶ Bộ môn Công nghệ phần mềm - P310 A9
- ▶ Giảng viên
 - ▶ TS. Nguyễn Hiếu Cường - cuonggt@gmail.com
 - ▶ TS. Cao Thị Luyện - luyenct@gmail.com
 - ▶ ThS. Nguyễn Đức Dư - nducdu@utc.edu.vn
 - ▶ ThS. Phạm Xuân Tích - tichpx@gmail.com

Q & A

► Q1- What is C Programming Language?

C programming language is a computer programming language that was developed to do system programming for the operating system UNIX and is an imperative programming language.

C was developed in **1972** by **Dennis Ritchie** at **bell Telephone Laboratories**. It is an outgrowth of an earlier language called B, which was also developed at Bell Laboratories.

C was largely confined to use within **Bell Laboratories** until **1978**, when Brian Kernighan and Ritchie Published a definitive description of the language.

Q & A

► Q2- Basic structure of C program

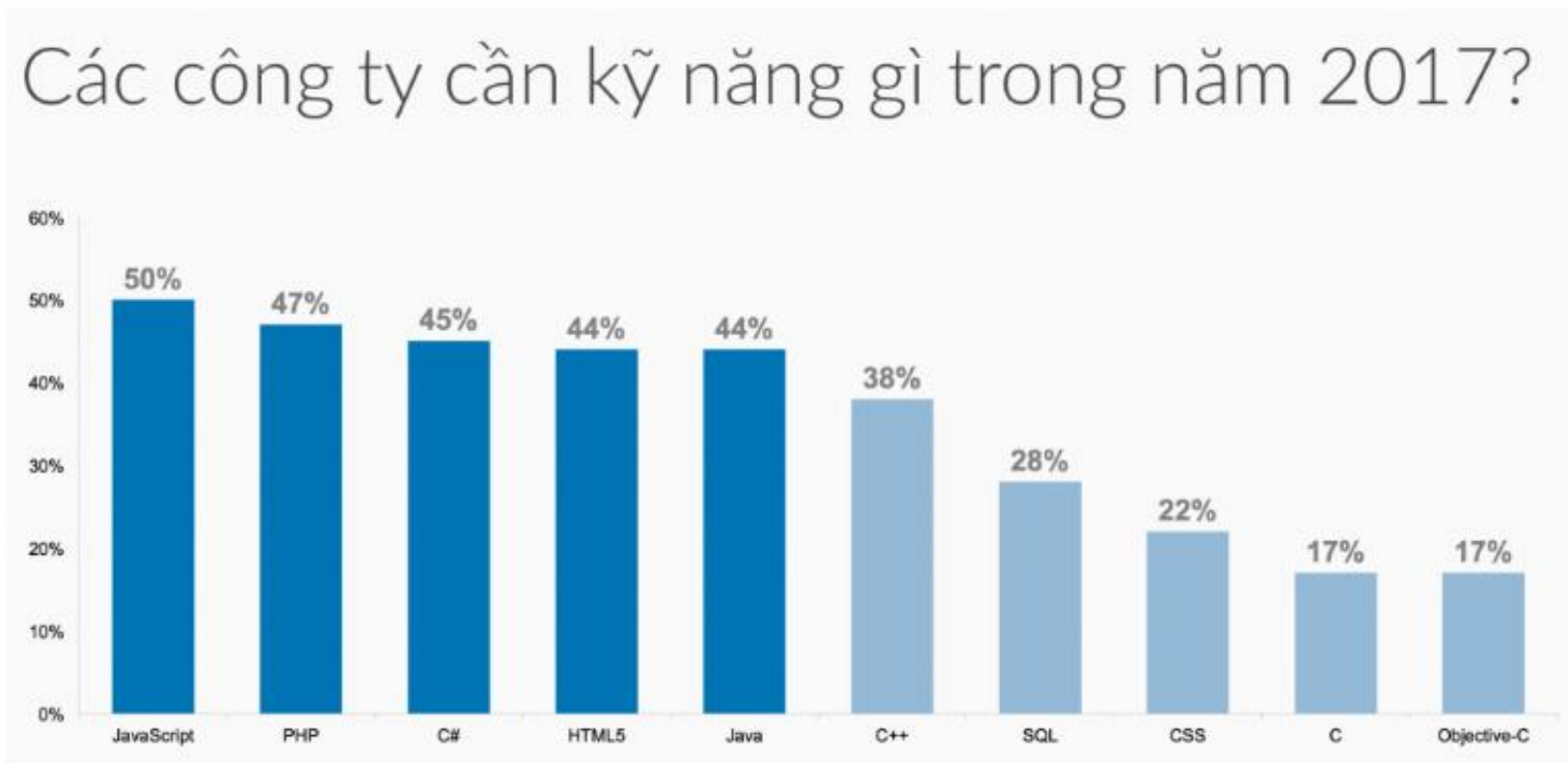
Structure of C program is defined by set of rules called protocol, to be followed by programmer while writing C program, every c programs are having these sections which are mentioned below:

- + Documentation section - This is the first part of C program.
- + Link - Header files that are required to execute a C program.
- + Definition - In definition section, variables are defined and values are set to these variables
- + Global Declaration - When a variable is to be used throughout the program.
- + Function prototype declaration
- + Main function - Function prototype gives many information about a function like return type, parameter names used inside the function.
- + User defined function - User can define their own functions which perform particular task as per the user requirement.

Q & A

- ▶ <https://www.indiabix.com/c-programming/questions-and-answers/>
- ▶ https://www.tutorialspoint.com/cprogramming/cprogramming_interview_questions.htm
- ▶ <http://www.gdatamart.com/interview/c-programming-interview-questions.aspx>

Tỷ lệ sử dụng các ngôn ngữ lập trình



Tỷ lệ sử dụng các ngôn ngữ lập trình

Thứ hạng	2018	2017
1	JavaScript	JavaScript
2	Java	Java
3	Python	Python
4	PHP	PHP
5	C#	C#,C++
6	C++	
7	CSS	CSS, Ruby
8	Ruby	
9	C	C
10	Swift	Objective-C

Bảng xếp hạng ngôn ngữ lập trình RedMonk

Worldwide, Aug 2018 compared to a year ago:

Rank	Change	Language	Share	Trend
1	↑	Python	24.21 %	+5.7 %
2	↓	Java	22.27 %	-0.7 %
3	↑	Javascript	8.45 %	+0.1 %
4	↓	PHP	7.88 %	-1.5 %
5		C#	7.74 %	-0.4 %
6		C/C++	6.19 %	-0.8 %
7	↑	R	4.15 %	-0.1 %
8	↓	Objective-C	3.33 %	-1.0 %
9		Swift	2.64 %	-0.9 %
10		Matlab	2.08 %	-0.4 %

Top ngôn ngữ lập trình phổ biến 08/2018 – chỉ số PYPL

I.MỘT SỐ KHÁI NIỆM CƠ BẢN TRONG NGÔN NGỮ C

- ▶ TẬP KÝ TỰ
- ▶ TỪ KHÓA
- ▶ TÊN
- ▶ CÂU LỆNH VÀ KHÓI LỆNH
- ▶ KIỂU DỮ LIỆU
- ▶ HẰNG, BIẾN, MẢNG
- ▶ CÁC TOÁN TỬ VÀ CÁC PHÉP TOÁN
- ▶ NHẬP XUẤT DỮ LIỆU
- ▶ CÁC HÀM VÀO RA TRÊN MÀN HÌNH, BÀN PHÍM
- ▶ KỸ THUẬT BẮT PHÍM TỔNG QUÁT
- ▶ CÁC CÂU LỆNH ĐIỀU KHIỂN

CÁC KIỂU DỮ LIỆU CƠ SỞ

- ▶ Kiểu ký tự (char)
char (1 byte), unsigned char (1 byte)
- ▶ Kiểu số nguyên (int)
int (2 byte), unsigned int (2 byte), long (4 byte), unsigned long (4 byte)
- ▶ Kiểu số thực (float, double)
float (4 byte), double (8 byte), long double (10 byte)
- ▶ Kiểu void

HẰNG

Các loại:

- ▶ Hằng số: hằng nguyên, hằng thực
- ▶ Hằng ký tự: 'ký tự'
 - + mã ASCII
 - + '\n', '\t', '\0', '\b', '\'', '\\'
- ▶ Hằng chuỗi: "các ký tự" = các ký tự + '\0'

Cách định nghĩa:

```
#define tên_hằng giá_trị
```

Các phép toán

- ▶ Các phép toán số học: *Cộng, Trừ, Nhân, Chia, Mod*
- ▶ Các phép thao tác bit: cho phép xử lý từng bit của một số nguyên (không dùng cho các kiểu số thực)
 - + AND (&), OR (|), XOR (^),
 - + Dịch trái (<<), Dịch phải (>>): $a \ll n = a * 2^n$ $a \gg n = a / 2^n$
 - + Bù bit (~): $\sim 1 = 0$ $\sim 0 = 1$
- ▶ Các phép toán quan hệ và logic
 - ▶ Quan hệ: ==, !=, >, >=, <, <= kết quả 0 (sai), đúng (1)
 - ▶ Logic: !, &&, ||
- ▶ Chuyển đổi kiểu dữ liệu (ép kiểu)
- ▶ Các phép toán tăng, giảm
- ▶ Câu lệnh gán
- ▶ Toán tử dãy phẩy
- ▶ Toán tử ?

NHẬP XUẤT DỮ LIỆU

- ▶ Các hàm nhập xuất trong stdio.h

- ▶ Hàm printf
- ▶ Hàm scanf
- ▶ Hàm gets
- ▶ Hàm getchar(void)

Các hàm trên đều nhận dữ liệu từ stdin (dòng nhập chuẩn, có thể hiểu như một vùng nhớ đặc biệt)

- ▶ Hàm putchar(int ch)
- ▶ Hàm puts

NHẬP XUẤT DỮ LIỆU

- ▶ Các hàm nhập xuất trong conio.h

- ▶ Hàm getch(void)

- ▶ Hàm getche(void)

Hai hàm này chờ nhận 1 ký tự trực tiếp từ bộ đệm bàn phím. Nếu bộ đệm rỗng thì chờ. Khi 1 phím được ấn thì nhận ngay ký tự đó mà không cần phải Enter như các hàm nhập từ stdin.

Hàm getche cho hiện ký tự lên màn hình còn hàm getch thì không.

- ▶ Hàm cprintf: giống như hàm printf nhưng có màu của nội dung được quy định bởi hàm textcolor

CÁC CẤU TRÚC ĐIỀU KHIỂN

- ▶ if, switch
- ▶ for, while, do.. while
- ▶ break, continue
- ▶ goto

Mảng (Array) - Review

- Mảng là một tập hợp nhiều biến có cùng kiểu dữ liệu và cùng tên. Các phần tử được truy xuất thông qua chỉ số

- Biến, mảng nội (địa phương, cục bộ)

Là các biến (mảng) khai báo bên trong thân của một hàm.

- + Thời gian tồn tại: từ lúc máy làm việc với hàm cho đến khi hàm đó kết thúc. Các biến, mảng trong hàm main tồn tại trong suốt thời gian làm việc của chương trình.

- + Phạm vi sử dụng: chỉ sử dụng bên trong hàm mà nó được khai báo

- + Khởi đầu giá trị: dùng toán tử gán

Ví dụ

```
#include "stdio.h"

main()
{ int a[3] = {10, 20, 30};
  printf(" a[0] = %d, a[1]=%d a[2]= %d", a[0], a[1], a[2]);
}
```

```
#include "stdio.h"

main()
{ int a[2][3] = {{10, 20, 30}, {11, 21, 31}};
  int i, j;
  for( i=0;i<2;i++)
    { for(j=0;j<3;j++) printf(" %5d", a[i][j]);
      printf("\n");
    }
}
```

Mảng (Array) - Review

- Biến, mảng ngoài (toàn cục, toàn bộ)

Là biến, mảng khai báo bên ngoài các hàm

+ Thời gian tồn tại: tồn tại trong suốt thời gian làm việc của chương trình

+ Phạm vi sử dụng: từ vị trí khai báo đến cuối chương trình

+ Khởi đầu giá trị

a. khởi đầu như biến, mảng cục bộ

Ví dụ:

```
#include "stdio.h"
int a=100;
float x[6] = {3.2, 5.0, 4.5 , 9.0, 6.8, 7.3};
int y[3][2] = {{1,1}, {2,2}, {3,3}};
main()
{
...
}
```

Mảng (Array) - Review...

+ Khởi đầu giá trị

a. khởi đầu như biến, mảng cục bộ

b. không cần chỉ ra kích thước

Ví dụ:

```
float a[] = {5.6, 7.8, 3};
```

```
int t[][5] = {{1,2,3,4},5, {6,7,8,9,10}};
```

Mảng (Array) - Review...

+ Khởi đầu giá trị

a. khởi đầu như biến, mảng cục bộ

b. không cần chỉ ra kích thước

c. *khi chỉ rõ kích thước của mảng, kích thước cần không nhỏ hơn số lượng giá trị khởi đầu*

Ví dụ:

```
float a[10] = {5.6, 7.8, 3};
```

```
int t[8][5] = {{1,2,3,4,5}, {6,7,8,9,10}};
```


Mảng (Array) - Review...

+ Khởi đầu giá trị

- a. khởi đầu như biến, mảng cục bộ
- b. không cần chỉ ra kích thước
- c. khi chỉ rõ kích thước của mảng, kích thước cần không nhỏ hơn số lượng giá trị khởi đầu
- d. *Với mảng 2 chiều, số lượng giá trị khởi đầu cho chiều thứ 1 có thể khác nhau:*

Ví dụ:

```
float a[][3] = {{5}, {1.5, 2.5, 3.5}, {7, 5}};
```

```
int b[10][4] = {{6}, {5,6,7,8}, {5,8}, {3,9,7,5}};
```

Mảng (Array) - Review.

+ Khởi đầu giá trị

- a. khởi đầu như biến, mảng cục bộ
- b. không cần chỉ ra kích thước
- c. khi chỉ rõ kích thước của mảng, kích thước cần không nhỏ hơn số lượng giá trị khởi đầu
- d. Với mảng 2 chiều, số lượng giá trị khởi đầu cho chiều thứ 1 có thể khác nhau:
- e. *Bộ khởi đầu của một mảng char có thể:*
 - * hoặc là danh sách các hằng ký tự
 - * hoặc là một hằng xâu ký tự

Ví dụ:

```
char name[] = { 'c', 'n', 't', 't', '\0'};  
char name[] = "cntt";  
char name[30] = { 'c', 'n', 't', 't', '\0'};  
char name[30] = "cntt";
```

Xâu (chuỗi) ký tự - Review

- Là một dãy các ký tự đặt trong hai dấu nháy kép “ và ”
- Xâu rỗng: “” - hai dấu nháy kép liền nhau
- Khi gặp 1 xâu, máy cấp phát một khoảng nhớ cho một mảng kiểu char đủ lớn để chứa các ký tự của xâu và chứa thêm ký tự ‘\0’ là ký tự kết thúc xâu.
- Xâu ký tự là một trường hợp riêng của mảng 1 chiều khi mỗi thành phần của mảng là ký ký tự. Tuy nhiên, khác với mảng, xâu ký tự được kết thúc bằng một ký hiệu đặc biệt NULL có mã ASCII bằng 0 (ký hiệu ‘\0’). => `char s[20];` chỉ có thể chứa được nhiều nhất 19 ký tự.
- ‘A’ : là ký tự A được mã hóa bằng 1 byte
 - “A”: là một xâu chứa ký tự A, được mã hóa bằng 2 byte (1 byte cho A và 1 byte cho ‘\0’)

Xâu (chuỗi) ký tự - Review.

- Trong C, không tồn tại các phép toán so sánh hay gán nội dung giữa 2 xâu. => sử dụng các hàm trong thư viện “string.h”

+ *int strlen(char s[]) - độ dài của xâu*

+ *strcpy(char dest[], char sour[]) - sao chép (gán) nội dung xâu “sour” vào xâu “dest”*

+ *strncpy(char dest[], char sour[], int n) - sao chép n ký tự của xâu “sour” vào xâu “dest”*

+ *strcat(char s1[], char s2[]) - nối (ghép) xâu s2 vào cuối xâu s1; tương tự có strncat*

+ *int strcmp(char s1[], char s2[]) - so sánh 2 xâu s1 và s2. Nguyên tắc so sánh theo thứ tự từ điển (=0 nếu s1=s2, >0 nếu s1>s2 và <0 nếu s1<s2). Tương tự có strcmp*

+ *char *strchr(char s[], char c) - tìm lần xuất hiện đầu tiên của ‘c’ trong “s” , trả về địa chỉ của ký tự đó*

.....

II. HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM

XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM

CON TRỎ VÀ ĐỊA CHỈ

CON TRỎ VÀ MẢNG MỘT CHIỀU

CON TRỎ VÀ MẢNG NHIỀU CHIỀU

Kiểu CON TRỎ, Kiểu ĐỊA CHỈ, CÁC PHÉP TOÁN TRÊN CON TRỎ

MẢNG CON TRỎ

CON TRỎ TỚI HÀM

ĐỆ QUY

ĐỐI DÒNG LỆNH

Hàm và chương trình

- Chương trình (program):

+ Một chương trình bao gồm một hoặc nhiều hàm

+ Hàm `main()` là hàm bắt buộc của chương trình

+ Chương trình luôn bắt đầu thực hiện từ câu lệnh đầu tiên của hàm `main()` và thường kết thúc khi gặp dấu `}` cuối cùng của nó.

- Hàm (Function): là một đoạn chương trình độc lập thực hiện trọn vẹn một công việc nhất định, nó thường trả về cho chương trình gọi nó một giá trị.

Note: Không cho phép xây dựng một hàm bên trong một hàm khác.

Tổ chức chương trình

.....

hàm 1

.....

hàm 2

.....

hàm n

1-: #include, #define, định nghĩa các kiểu dữ liệu, khai báo biến ngoài

2- Việc truyền dữ liệu từ hàm này sang hàm khác?

=>

- Sử dụng biến toàn bộ, biến ngoài

- Sử dụng đối của hàm

Xây dựng hàm

Để viết một hàm, trước hết ta phải xác định mục đích của hàm là dùng để làm gì, từ đó xác định các thành phần:

- Nguyên mẫu hàm
- Kiểu giá trị trả về của hàm (kiểu hàm)
- Tên hàm
- Các tham số (đối số) của hàm
- Nội dung hàm

Xây dựng hàm – Nguyên mẫu hàm (Prototype)

*< kiểu dữ liệu của hàm > < **tên hàm**(danh sách các tham số) >;*

- Có thể ghi hoặc không ghi nguyên mẫu
- Các nguyên mẫu hàm (nếu có) cần đặt trước hàm main()

Ví dụ:

double exp(double x);

double pow(double x, double y);

*char *strcat(char *des, const char *source);*

Xây dựng hàm — Kiểu giá trị của hàm

- Giá trị của hàm được xác định dựa vào mục đích của hàm
- Trong thân hàm trả về (câu lệnh return) phải trả về đúng kiểu hàm đã định
- Các hàm không trả về giá trị ta sử dụng kiểu hàm là: void

Xây dựng hàm – Tên hàm

- Theo quy định đặt tên (định danh – identifier)
- Nên ngắn gọn và phản ánh phần nào mục đích của hàm
- Tên hàm trong nguyên mẫu, khai báo và sử dụng phải giống nhau

Xây dựng hàm – Tham số của hàm

- Các tham số mà ta ghi trong nguyên mẫu hay lúc khai báo hàm gọi là ***tham số hình thức***
- Các giá trị, biến mà ta ghi sau tên hàm được gọi để thực hiện hàm đó gọi là ***tham số thực***
- Tham số của một hàm có hai công dụng:
 - + *Cung cấp các giá trị cho hàm khi ta gọi nó thực hiện*
 - + *Lưu các kết quả tính toán được trong quá trình hàm hoạt động*

Như vậy ta có thể chia thành:

- + Tham số vào: cung cấp giá trị cho hàm
- + Tham số ra: Lưu kết quả tính toán được trong hàm
- + Tham số vừa vào, vừa ra.

=> Khi viết hàm, ta phải xác định xem có bao nhiêu tham số? Và là tham số gì?

Xây dựng hàm – Thân hàm

- Thân hàm là nội dung chính của hàm bắt đầu bằng dấu { và kết thúc bởi 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 có thể sử dụng một câu lệnh return, có thể dùng nhiều câu lệnh return ở những chỗ khác nhau và cũng có thể không sử dụng câu lệnh này. Dạng tổng quát của nó là:

`return [biểu thức];`

Giá trị của biểu thức trong câu lệnh return sẽ được gán cho hàm.

Xây dựng hàm (tổng kết).

[kiểu_hàm tên_hàm (khai báo các đối – tham số hình thức);]

kiểu_hàm tên_hàm (khai báo các đối – tham số hình thức)

{

khai báo các biến cục bộ

các câu lệnh

[return [biểu thức];]

}

Sử dụng hàm

Hàm được sử dụng thông qua lời gọi tới nó. Cách viết một lời gọi hàm như sau:

`tên_hàm ([danh sách tham số thực])`

Một điều cần nhớ khi viết lời gọi hàm là:

+ số tham số thực phải bằng số tham số hình thức (đối)

+ mỗi tham số thực phải có cùng kiểu giá trị như kiểu giá trị của đối tượng ứng với nó.

Ví dụ:

`sqrt(100)`

`sqrt(3*x)`

`pow(d, 1.0/k)`

Quy tắc hoạt động của hàm

Khi gặp một lời gọi hàm thì hàm bắt đầu được thực hiện. Nói cách khác, khi máy gặp một lời gọi hàm ở một chỗ nào đó của chương trình, thì máy sẽ tạm rời chỗ đó và chuyển đến hàm tương ứng.

Quá trình thực hiện hàm sẽ diễn ra theo trình tự 4 bước như sau:

a/ Cấp phát bộ nhớ cho các đối và các biến cục bộ.

b/ Gán giá trị của các tham số thực cho các đối tượng ứng.

c/ Thực hiện các câu lệnh trong thân hàm.

d/ Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xóa các đối, các biến cục bộ (giải phóng bộ nhớ của các đối, biến cục bộ) và thoát khỏi hàm.

Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

Một số lưu ý về biến cục bộ (biến trong hàm) và đối

- Do đối và biến cục bộ đều có phạm vi hoạt động trong cùng một hàm nên đối và biến cục bộ cần có tên khác nhau.
- Đối và biến cục bộ đều là các biến tự động. Chúng được cấp bộ nhớ khi hàm được xét đến và chúng sẽ lập tức bị xoá trước khi ra khỏi hàm. Như vậy, không thể mang giá trị của đối ra khỏi hàm.
- Đối hoặc biến cục bộ có thể trùng tên với bất kỳ đại lượng nào ở ngoài hàm mà không gây ra một nhầm lẫn nào cả

Ví dụ: xây dựng, sử dụng hàm, tổ chức chương trình

Bài 1: Xây dựng hàm tính $n!$ và sử dụng tính một số giai thừa

Bài 2: Xây dựng hàm nhận vào hai số nguyên dương a và b , nhập một trong các phép toán pt: $+$, $-$, $*$, $/$, $^$. Hàm trả ra một số là kết quả của biểu thức a pt b .

Sử dụng hàm trong chương trình, cho phép nhập 2 số nguyên bất kỳ cùng với 1 phép toán, tính và in ra màn hình giá trị của phép toán giữa a và b .

Bài 3: Xây dựng hàm tính giá trị lớn nhất của 3 số thực? Trong chương trình chính sử dụng hàm để in ra giá trị lớn nhất của 3 số thực bất kỳ?

Bài 4: Xây dựng hàm tính khoảng cách giữa 2 điểm trong xOy . Sử dụng hàm trong chương trình chính để tính khoảng cách giữa 2 điểm bất kỳ?

Bài 5: Xây dựng hàm tính diện tích của tam giác khi biết tọa độ 3 đỉnh dựa trên hàm tính khoảng cách ở bài 3. Sử dụng trong chương trình để tính diện tích tam giác cho 3 điểm?

Bài 6: Xây dựng hàm giải phương trình bậc 2? Sử dụng để giải phương trình bậc 2 bất kỳ?

Ví dụ:

Bài 7: Xây dựng hàm kiểm tra một số n có là số nguyên tố hay không? Sử dụng trong chương trình

Bài 8: Viết hàm kiểm tra số hoàn hảo.

Nhập vào một dãy n số và thông báo có phải số nguyên tố, số hoàn hảo hay không?

Bài 9: Xây dựng hàm tính số Fibonacci

Bài 10: Viết các hàm cho các [bài tập 1](#).

Bài 11: Xây dựng hàm tính căn bậc 2

Dùng dẫn hướng #define để định nghĩa hàm đơn giản

Ví dụ 1:

```
#include <stdio.h>
```

```
#define tong(x, y) x + y
```

```
main()  
{  
    int a = 5, b = 8;  
    printf("%d + %d = %d", a, b, tong(a, b));  
}
```

Dùng dẫn hướng #define để định nghĩa hàm đơn giản

Ví dụ 2:

```
#include <stdio.h>
```

```
#define lamtron(x) (x < (int)x + 0.5 ? (int)x : (int)x + 1 )
```

```
main()
```

```
{
```

```
    float a = 2.0/3;
```

```
    printf(" %f ",lamtron(a));
```

```
}
```

Con trỏ (Pointer)

Địa chỉ của biến: Liên quan đến một biến ta đã có khái niệm:

- Tên biến
- Kiểu biến
- Giá trị của biến

Ví dụ:

```
float alpha = 30.5;
```

xác định một biến có tên là alpha có kiểu float và có giá trị 30.5.

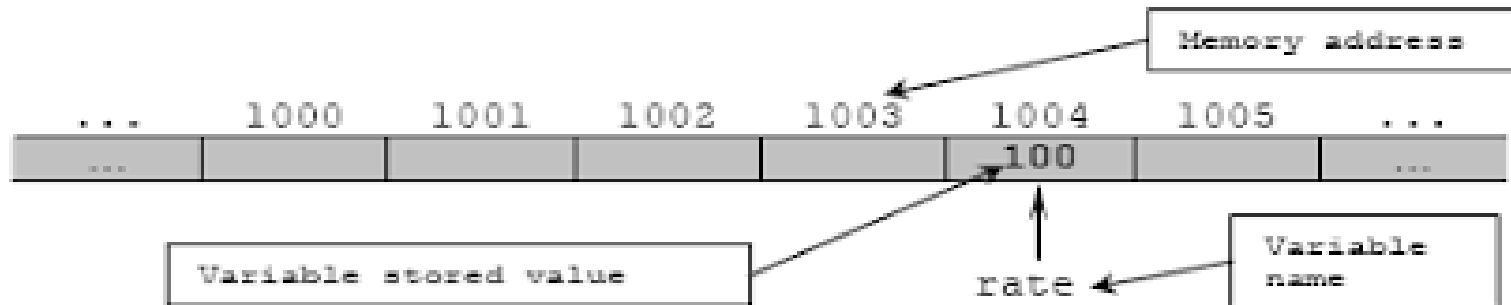
theo khai báo trên, máy sẽ cấp phát cho biến alpha một khoảng nhớ gồm 4 byte liên tiếp

=> Địa chỉ của biến là số thứ tự của byte đầu tiên trong một dãy các byte liên tiếp mà máy dành cho biến (các byte được đánh số từ 0).

Con trỏ - Địa chỉ của biến

Note:

- Địa chỉ của biến là một số nguyên nhưng không được đánh đồng nó với các số nguyên thông thường dùng trong các phép tính.
- Các kiểu địa chỉ khác nhau: Địa chỉ kiểu int, kiểu float, kiểu double,...
- Phép toán lấy địa chỉ: &tên_biến



Con trỏ - Khái niệm

- Con trỏ là **một biến** dùng để **chứa địa chỉ**.
- Mỗi loại địa chỉ sẽ (cần) có một kiểu con trỏ tương ứng (phụ thuộc vào loại dữ liệu lưu trữ trong địa chỉ đó):
 - + con trỏ kiểu int dùng để chứa địa chỉ các biến kiểu int.
 - + con trỏ kiểu float, kiểu double,...

Con trỏ - Khai báo

Cũng như đối với bất kỳ một biến nào khác, một con trỏ cần khai báo trước khi sử dụng

Việc khai báo biến con trỏ được thực hiện theo mẫu sau:

kiểu_dữ_liệu *tên_con_trỏ;

Ví dụ:

`int x,y,*px,*c; // khai báo hai biến kiểu int là x,y và hai con trỏ kiểu int là px và c.`

`float *t, *d; //khai báo hai con trỏ kiểu float là t và d.`

Con trỏ

Qui tắc sử dụng con trỏ trong các biểu thức

+ Sử dụng tên con trỏ

Con trỏ cũng là một biến nên khi tên của nó xuất hiện trong một biểu thức thì giá trị của nó sẽ được sử dụng trong biểu thức này

Note: *Giá trị của một con trỏ là địa chỉ của một biến nào đó*

Ví dụ: (với các khai báo ở slide trước)

`c = &y; // gán địa chỉ của y cho con trỏ c`

`px = &x; // gán địa chỉ của biến x cho con trỏ px`

=> trong con trỏ c chứa địa chỉ của biến y và trong con trỏ px chứa địa chỉ của biến x.

▲ Khi con trỏ px chứa địa chỉ của biến x thì ta nói px trỏ tới x

`t = &y; // wrong`

Con trỏ

Qui tắc sử dụng con trỏ trong các biểu thức...

+ Sử dụng dạng khai báo của con trỏ (= * + tên con trỏ)

```
float x, y, z, *px, *py;
```

```
px = &x;
```

```
py = &y; // px trỏ tới x, py trỏ tới y
```

=> Khi con trỏ px trỏ tới biến x thì 2 cách viết : x và *px

là tương đương trong mọi ngữ cảnh.

=> 03 câu lệnh sau có giá trị tương đương

```
y = 3*x + z;
```

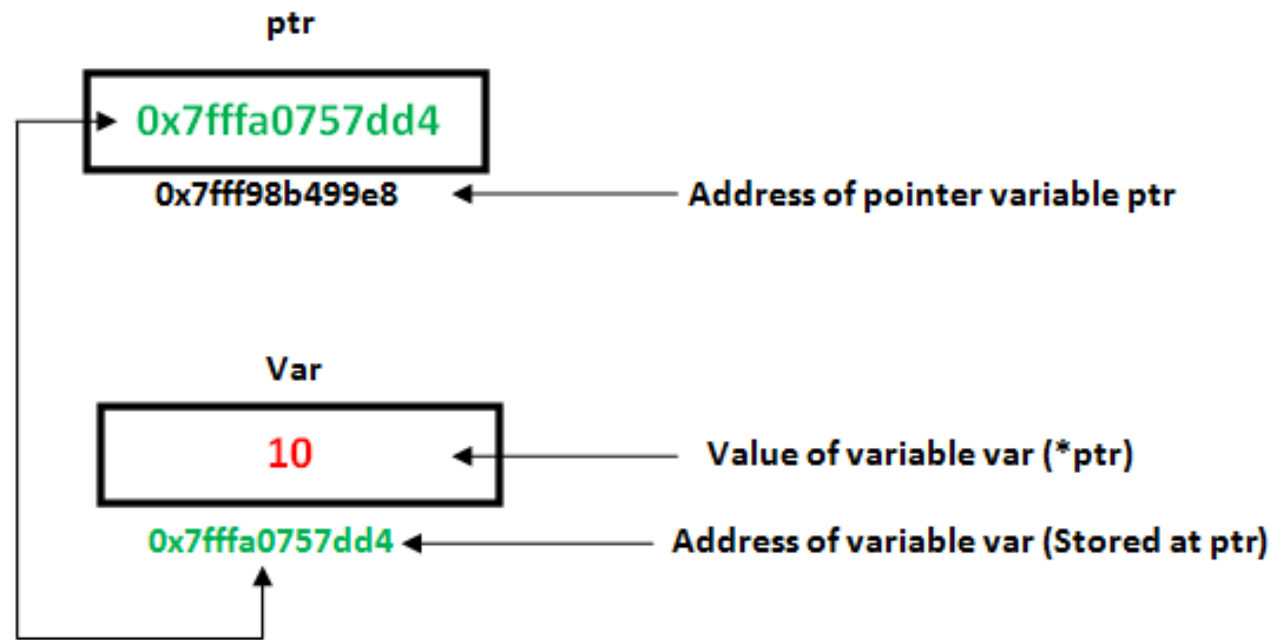
```
*py = 3*x + z;
```

```
*py = 3*(*px) + z;
```

*=> 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 gán cho nó một giá trị mới (làm thay đổi nội dung của nó).
- Điều này sẽ được áp dụng như một phương pháp chủ yếu để nhận kết quả của hàm thông qua đối*

Con trỏ

Qui tắc sử dụng con trỏ trong các biểu thức.



Quy tắc về kiểu giá trị trong khai báo

Mọi thành phần của cùng một khai báo (biến, phần tử mảng, hàm, con trỏ) khi xuất hiện trong biểu thức đều cho cùng một kiểu giá trị.

Ví dụ:

Giả sử ta có khai báo

```
float a, b[7], *px;
```

khi đó mọi thành phần của nó đều cho giá trị kiểu float. Nói cách khác, khi:

```
a, b[i], *px
```

xuất hiện trong một biểu thức thì chúng luôn luôn cho một giá trị kiểu float

Con trỏ

Hàm có đối con trỏ

Nếu đối của hàm là con trỏ kiểu int (float, double,...) thì tham số thực tương ứng phải là địa chỉ của biến hoặc địa chỉ của phần tử mảng kiểu int (float, double,...).

Khi đó địa chỉ của biến được truyền cho đối con trỏ tương ứng.

Ta có thể gán cho nó các giá trị mới bằng cách sử dụng các câu lệnh thích hợp trong thân hàm do đã biết địa chỉ của biến

Ví dụ:

- Hàm hoán đổi giá trị của 2 biến
- + Phiên bản không dùng đối con trỏ
- + Phiên bản dùng đối con trỏ

Con trỏ

Khi nào sử dụng đối con trỏ

Trong số các đối của hàm, ta có thể chia ra làm hai loại.

- + Loại thứ nhất gồm các đối dùng để chứa các giá trị đã biết, ta gọi chúng là các đối vào.
- + Loại thứ hai gồm các đối dùng để chứa các kết quả mới nhận được sau khi thực hiện hàm, gọi là các đối ra.

=> Các đối ra phải là con trỏ

Ví dụ:

- Hàm giải phương trình bậc 2
- Hàm giải hệ phương trình bậc nhất 2 ẩn
- Hàm hoán đổi giá trị của 2 biến

Con trỏ

Các phép toán

- Phép **gán** địa chỉ cho con trỏ là hợp lệ khi và chỉ khi kiểu địa chỉ phù hợp với nó

Ví dụ:

```
float a[20][30], *pa, *pm[30];
```

+ pa là con trỏ kiểu float

+ pm là con trỏ kiểu float[30]

+ a là địa chỉ kiểu float[30]

=> pa = a; // SAI

a = pa; // SAI

pm = a; // ĐÚNG

Con trỏ

Các phép toán...

- Phép toán **truy nhập nội dung** của một đối tượng con trỏ: *

Ví dụ:

```
int x=1, y=2, z[10], *pi;
```

```
pi=&x;    // gán địa chỉ của biến x cho con trỏ pi hay pi trỏ tới x
```

```
y=*pi;    // gán giá trị của vùng nhớ do pi trỏ tới cho biến y, tức y có giá trị bằng 1
```

```
*pi=0;    //thay đổi nội dung (giá trị) vùng nhớ do pi đang trỏ tới, tức x có giá trị bằng 0
```

```
*pi = *pi + 10; // ⇔ x=x +10;
```

```
*pi+=1;    // ⇔ x+=1; ⇔ ++*pi; ⇔ (*pi)++; - chú ý dấu ()
```

Con trỏ

Các phép toán

- Phép toán **truy nhập nội dung** (**Nguyên tắc truy nhập bộ nhớ**)

Ví dụ:

```
float *pf;
```

```
int *pi;
```

```
char *pc;
```

Khi đó:

- + Nếu pf trỏ đến byte thứ 10001, thì *pf biểu thị vùng nhớ 4 byte liên tiếp từ byte 10001 đến byte 10004.
- + Nếu pi trỏ đến byte 10001 thì *pi biểu thị vùng nhớ 2 byte gồm các byte 10001 và 10002.
- + Nếu pc trỏ đến byte 10001 thì *pc biểu thị vùng nhớ 1 byte là byte 10001.

Con trỏ

Các phép toán...

- Phép **tăng giảm** địa chỉ

Ví dụ:

1- float x[30], *px;

px = &x[10]; // px trỏ đến phần tử thứ 11 – x[10] của mảng x

px + i //trỏ đến phần tử x[10+i]

px - i // trỏ đến phần tử x[10-i]

2- float b[40][50];

b là mảng gồm các dòng 50 phần tử thực.

Kiểu địa chỉ của b là $50 \times 4 = 200$ byte. Ta có:

b trỏ tới đầu dòng thứ nhất (phần tử b[0][0])

b+1 trỏ tới đầu dòng thứ hai (phần tử b[1][0])

b+5 trỏ tới đầu dòng thứ sáu (phần tử b[5][0])

.....

Con trỏ

Các phép toán.

- Phép **so sánh**

Trong C cho phép so sánh các con trỏ cùng kiểu

```
float *p1, *p2;
```

+ $p1 < p2$ nếu địa chỉ p1 trỏ tới **thấp hơn** địa chỉ p2 trỏ tới

+ $p1 = p2$ nếu địa chỉ p1 trỏ tới **bằng** địa chỉ p2 trỏ tới

+ $p1 > p2$ nếu địa chỉ p1 trỏ tới **cao hơn** địa chỉ p2 trỏ tới

Ví dụ:

```
float a[100], *p, *pcuoi, s=0.0;
```

```
int n;
```

```
pcuoi = a+n-1; /* Địa chỉ cuối dãy */
```

```
for (p=a; p<=pcuoi; ++p) s += *p;
```

Con trỏ

Con trỏ kiểu void

Con trỏ void được khai báo như sau:

```
void *tên_con_trỏ;
```

Đây là con trỏ đặc biệt (con trỏ không kiểu), nó có thể nhận bất kỳ địa chỉ kiểu nào. Chẳng hạn các câu lệnh sau là hợp lệ

```
void *pa;
```

```
float a[20][30];
```

```
pa = a;
```

Chú ý: Các phép tăng giảm địa chỉ, truy nhập bộ nhớ và so sánh không dùng trên con trỏ void.

Cách dùng: Con trỏ void thường dùng làm đối để nhận bất kỳ địa chỉ kiểu nào từ tham số thực. Trong thân hàm phải dùng phép ép kiểu để chuyển sang dạng địa chỉ cần xử lý. Ví dụ 2 dưới đây minh họa điều này.

Con trỏ và mảng 1 chiều

- Trong C có mối quan hệ chặt chẽ giữa con trỏ và mảng: Các phần tử của mảng có thể được xác định nhờ chỉ số hoặc thông qua con trỏ
- Tên mảng là một hằng địa chỉ, nó chính là địa chỉ của phần tử đầu tiên của mảng

Như vậy với khai báo:

```
float a[10];
```

máy sẽ bố trí cho mảng a mười khoảng nhớ liên tiếp (mỗi khoảng nhớ 4 byte)

a tương đương với &a[0]

a+i tương đương với &a[i]

*(a+i) tương đương với a[i]

Con trỏ và mảng 1 chiều...

Nếu con trỏ pa trỏ tới một phần tử $a[k]$ nào đó thì:

$pa+i$ trỏ tới phần tử thứ i sau $a[k]$, tức là $a[k+i]$

$pa-i$ trỏ tới phần tử thứ i trước $a[k]$, tức là $a[k-i]$

$*(pa+i)$ tương đương với $pa[i]$

Như vậy sau hai câu lệnh

```
float a[30], *p;
```

```
p = a;
```

thì bốn cách viết sau có tác dụng như nhau:

```
a[i]  *(a+i)  *(p+i)  p[i]
```

Lưu ý:

Ta có thể gán giá trị cho con trỏ p nhưng **KHÔNG THỂ** gán giá trị mới cho a vì nó là một **HẲNG** địa chỉ

Con trỏ và mảng 1 chiều...

Ví dụ: Nhập từ bàn phím giá trị các phần tử của một mảng và tính tổng của chúng

```
#include "stdio.h"
main()
{
float a[4],s;
int i;
for (i=0;i<4; ++i )
{
printf("\n a[%d] = ",i);
scanf("%f",&a[i]);
}
s = 0;
for (i=0; i<4; ++i )
    s += a[i];
printf("\n tong = %8.2f",s);
}
```

Phương án 1

```
#include "stdio.h"
main()
{
float a[4],s;
int i;
for (i=0; i<4; ++i )
{
printf("\n a[%d] = ",i);
scanf("%f", a+i );
}
s = 0;
for(i=0; i<4; ++i )
    s += a[i];
printf("\n tong = %8.2f",s);
}
```

Phương án 2

Con trỏ và mảng 1 chiều.

Ví dụ: Nhập từ bàn phím giá trị các phần tử của một mảng và tính tổng của chúng

```
#include "stdio.h"
main()
{
float a[4],s,*pa;
int i;
pa = a;
for (i = 0; i<4; ++i )
{
printf("\n a[%d] = ",i);
scanf("%f",&pa[i] );
}
s = 0;
for(i=0; i<4; ++i )
    s += pa[i];
printf("\n tong = %8.2f",s);
}
```

Phương án 3

```
#include "stdio.h"
main()
{
float a[4], s, *pa;
int i;
pa = a;
for (i=0; i<4; ++i)
{
printf("\n a[%d] = ",i );
scanf("%f", pa+i );
}
s = 0;
for (i=0; i<4; ++i )
    s += *(pa+i);
printf("\n tong = %8.2f",s);
}
```

Phương án 4

Con trỏ và mảng 1 chiều

Sử dụng trong đối của hàm

Nếu tham số thực là tên mảng a (một chiều) kiểu int (float double,...) thì đối (tham số hình thức) pa tương ứng cần phải là một con trỏ kiểu int (float, double,...). Đối pa có thể khai báo kiểu con trỏ:

```
int *pa;  
float *pa;  
double *pa;  
...
```

hoặc cũng có thể khai báo như một mảng hình thức:

```
int pa[];  
float pa[];  
double pa[];  
...
```

Chú ý: Hai cách khai báo trên là tương đương.

Con trỏ và mảng 1 chiều

Sử dụng trong đối của hàm...

Khi gọi hàm (hàm bắt đầu làm việc) thì giá trị của `a` được truyền cho `pa`.

Vì `a` là hằng địa chỉ biểu diễn (lưu giữ) địa chỉ đầu của mảng, nên con trỏ `pa` sẽ chứa địa chỉ phần tử đầu tiên của mảng.

Như vậy, trong thân hàm khi muốn truy nhập đến phần tử `a[i]` ta có thể dùng một trong hai cách viết sau:

`*(pa+i)` hoặc `pa[i]`

Con trỏ và mảng 1 chiều

Sử dụng trong đối của hàm...

Ví dụ: xây dựng một hàm để tính tổng các phần tử của một dãy số kiểu double

<pre>double sum1(double *a, int n) /* Phương án 1 */ { double s = 0; int i; for (i=0; i<n; ++i) s += *(a+i); return s; }</pre>	<pre>double sum2(double *a,int n) /* Phương án 2 */ { double s = 0; int i; for (i=0; i<n; ++i) s += a[i]; return s; }</pre>
<pre>double sum3(double *a,int n) /* Phương án 3 */ { double s = 0; while (n--) s += *a++; return s; }</pre>	<pre>double sum4(double a[],int n) /* Phương án 4 */ { double s = 0; int i; for (i=0; i<n; ++i) s += *(a+i); return s; }</pre>

Con trỏ và mảng 1 chiều

Sử dụng trong đối của hàm.

```
double sum5(double a[],int n) /* Phương án 5 */
{
    double s = 0;
    int i;
    for (i=0; i<n; ++i) s += a[i];
    return s;
}
```

```
double sum6(double a[],int n) /* Phương án 6 */
{
    double s = 0;
    while (n--)
        s += *a++;
    return s;
}
```

```
main()
{
    double b[4];
    b[0] = 465.2;
    b[1] = 1256.7;
    b[2] = 2.35e3;
    b[3] = 45e-2;
    printf("\n sum1 = %5.2 sum2 = %5.2", sum1(b,4), sum2(b,4));
    printf("\n sum3 = %5.2 sum4 = %5.2", sum3(b,4), sum4(b,4));
    printf("\n sum5 = %5.2 sum6 = %5.2", sum5(b,4), sum6(b,4));
}
```

Con trỏ và mảng 1 chiều

Bài tập.

- Nhập mảng 1 chiều, tính tổng các phần tử
- Nhập mảng 1 chiều n phần tử, kiểm tra mảng có đối xứng không
- Xây dựng lại các hàm có đối là con trỏ:
 - + Bài 2, 3 Mục Bài tập chương 1 trang 46 Giáo trình Lập trình nâng cao
 - + Bài 1, bài 2, bài 4 Mục Bài tập chương 2 trang 95 Giáo trình Lập trình nâng cao
- Xây dựng các hàm sắp xếp dãy số tăng dần theo các phương pháp: bubble sort, select sort, insert sort, chèn nhị phân.
- Nhập tọa độ 2 véc tơ 5 chiều. Tính module (tích vô hướng) và tổng 2 véc tơ đó.

Con trỏ và chuỗi ký tự

- Cũng giống như tên mảng, chuỗi ký tự là một hằng địa chỉ biểu thị địa chỉ đầu của mảng chứa nó

Vì vậy nếu ta khai báo ch như một con trỏ kiểu char:

```
char *ch;
```

thì phép gán

```
ch = "IT-UTC";
```

hoàn toàn có nghĩa. Sau khi thực hiện câu lệnh này trong con trỏ ch sẽ có địa chỉ đầu của mảng (kiểu char) đang chứa chuỗi ký tự bên phải. Khi đó các câu lệnh

```
puts("IT-UTC");
```

và

```
puts(ch);
```

sẽ có cùng tác dụng là cho hiện lên màn hình dòng chữ: IT-UTC

Con trỏ và chuỗi ký tự...

- Mảng kiểu char thường dùng để chứa một dãy ký tự đọc vào bộ nhớ

Ví dụ: để nhập từ bàn phím tên của một người ta cần dùng một mảng kiểu char với độ dài 25.

```
char t[25];
```

```
printf("\n Ho ten: "); gets(t);
```

- Sự khác nhau giữa mảng kiểu char và con trỏ kiểu char:

Nếu `char *ch, t[15];` thì 2 câu lệnh sau là **KHÔNG HỢP LỆ**

```
t = "Pham Van Anh";
```

//Sai, lý do: t là một hằng địa chỉ và ta không thể gán một hằng địa chỉ này cho một hằng địa chỉ khác

```
gets(ch);
```

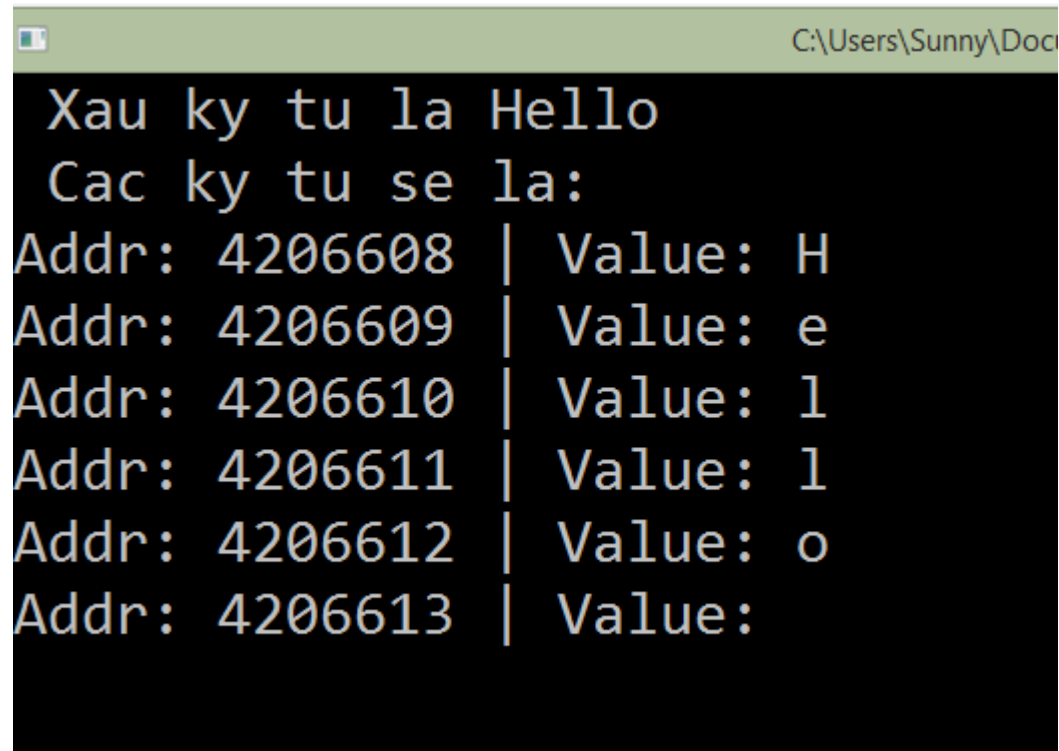
//Đúng về ngữ pháp nhưng nó không thể thực hiện được. Vì ch chưa có vùng nhớ

Nếu ch đã trỏ tới một vùng nhớ nào đó thì câu lệnh gets hoàn toàn có nghĩa. Ví dụ:

nếu `ch = t;` thì `gets(t);` và `gets(ch);` có tác dụng như nhau

Con trỏ và chuỗi ký tự. Ví dụ

```
char xau[] = "Hello"
main()
{ char *ptr;
  ptr = xau;
  printf(" Xau ky tu la %s \n", xau);
  printf(" Cac ky tu se la:\n");
  printf( Addr: %d | Value: %c \n", ptr, *ptr);
  printf( Addr: %d | Value: %c \n", ptr+1, *(ptr+1));
  printf( Addr: %d | Value: %c \n", ptr+2, *(ptr+2));
  printf( Addr: %d | Value: %c \n", ptr+3, *(ptr+3));
  printf( Addr: %d | Value: %c \n", ptr+4, *(ptr+4));
  printf( Addr: %d | Value: %c \n", ptr+5, *(ptr+5));
}
```



```
Xau ky tu la Hello
Cac ky tu se la:
Addr: 4206608 | Value: H
Addr: 4206609 | Value: e
Addr: 4206610 | Value: l
Addr: 4206611 | Value: l
Addr: 4206612 | Value: o
Addr: 4206613 | Value:
```

Con trỏ và Xâu ký tự

Bài tập

- Đếm số lần xuất hiện của 1 ký tự trong 1 xâu ký tự
- Nhập một số nguyên từ 1 đến 7. Hiện lên màn hình tên ngày trong tuần theo số nhập
- Bài 5 bài tập chương 1 trang 46 Giáo trình
- Bài 3 bài tập chương 2 trang 96 Giáo trình
- Chạy một dòng quảng cáo từ phải sang trái của màn hình
- Cộng, trừ hai số nguyên lớn (có nhiều chữ số)
-

Con trỏ và mảng nhiều chiều

Việc xử lý mảng nhiều chiều phức tạp hơn so với mảng một chiều.

Không phải mọi qui tắc đúng với mảng một chiều đều có thể đem ra áp dụng đối với mảng nhiều chiều.

1- Phép toán lấy địa chỉ không dùng được đối với các phần tử của mảng nhiều chiều. Nói cách khác, trong nhiều trường hợp câu lệnh:

&a[i][j]

là không hợp lệ và gây ra lỗi. (mảng hai chiều nguyên có thể dùng phép &a[i][j]).

Con trỏ và mảng nhiều chiều...

2- Phép cộng địa chỉ trong mảng hai chiều: C quan niệm mảng hai chiều là mảng (một chiều) của mảng. Tên mảng biểu thị địa chỉ đầu tiên của mảng.

Như vậy với khai báo

```
float a[2][3];
```

thì a là mảng mà mỗi phần tử của nó là một dãy 3 số thực (một hàng của bảng).

Mảng hai chiều a[2][3] gồm sáu phần tử ứng với sáu địa chỉ liên tiếp trong bộ nhớ được xếp theo thứ tự sau (thứ tự hàng)

Phần tử	a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
---------	---------	---------	---------	---------	---------	---------

Địa chỉ	1	2	3	4	5	6
---------	---	---	---	---	---	---

Vì vậy:

a trỏ tới đầu hàng thứ nhất (phần tử a[0][0]),

a+1 trỏ tới đầu hàng thứ hai (phần tử a[1][0]).

...

Con trỏ và mảng nhiều chiều...

Để lần lượt duyệt trên các phần tử của mảng hai chiều ta vẫn có thể dùng con trỏ theo cách sau:

```
float *pa, a[2][3];
```

```
pa = (float*) a;
```

Khi đó:

pa trỏ tới a[0][0]

pa+1 trỏ tới a[0][1]

pa+2 trỏ tới a[0][2]

pa+3 trỏ tới a[1][0]

pa+4 trỏ tới a[1][1]

pa+5 trỏ tới a[1][2]

Chú ý: Câu lệnh gán

pa = a;

là SAI, vì kiểu của pa và a khác nhau:

pa là con trỏ float, còn a là địa chỉ kiểu float[3]

Ví dụ 1:

```
#include "stdio.h"
main()
{
    float a[2][3], *pa;
    int i;
    pa = (float*) a;
    for (i = 0; i<6; ++i)
        scanf("%f", pa+i);
}
```

Ví dụ 2:

```
#include "stdio.h"
main()
{
    float a[2][3];
    int i;
    for (i = 0; i<6; ++i)
        scanf("%f", (float*)a+i);
}
```

Con trỏ và mảng nhiều chiều...

Giả sử a là một mảng 2 chiều có kích thước cột tối đa là N (**a**[][**N**]).

Khi đó:

$$a_{ij} \Leftrightarrow a + i * N + j$$

Lưu ý: các tính chỉ số i, j

```
#include "stdio.h"
main()
{
    float a[50][50];
    int m,n,i,j;
    printf("\n Nhap m va n: "); scanf("%d%d",&m,&n);
    for (i=1; i<=m; ++i)
        for(j=1; j<=n; ++j)
        {
            printf("\n a[%d][%d]= ",i,j);
            scanf("%f", (float*)a + i*50 + j);
        }
}
```

Con trỏ và mảng nhiều chiều...

3- Tham số thực là tên mảng nhiều chiều

Giả sử a là mảng hai chiều: `float a[60][50];`

Làm thế nào để dùng tên mảng hai chiều a trong lời gọi hàm?

Cách 1:

+ Dùng đối con trỏ kiểu `float[50]`, khai báo theo 1 trong 2 mẫu sau:

```
float (*pa)[50]; // dùng để khai báo con trỏ kiểu float[50]
```

```
float pa[][50]; // chỉ dùng để khai báo đối
```

+ Trong thân hàm, để truy nhập đến phần tử `a[i][j]` ta dùng: `pa[i][j]`

+ **Chú ý:** Theo cách này, hàm chỉ dùng được đối với các mảng hai chiều có 50 cột (số hàng không quan trọng).

Con trỏ và mảng nhiều chiều...

3- Tham số thực là tên mảng nhiều chiều

Cách 1: Dùng đối con trỏ kiểu float[50]

```
#include "stdio.h"

void tong_h(float a[][50], int m, int n, int *th);

main()
{
    float b[30][50], h[30]; /* b phải có 50 cột, h chứa các tổng hàng */
    int i,j,m,n;
    printf("\n Nhap m va n: "); scanf("%d%d",&m,&n);
    for (i=1; i<=m; ++i)
        for(j=1; j<=n; ++j)
        { float x;
          printf("\n b[%d][%d]= ",i,j); scanf("%f",&x); b[i][j] = x;
        }
    tong_h(b, m, n, h); /* Tính các tổng hàng */
    /* In các tổng hàng */
    for (i=1; i<=m; ++i) printf("\n Tong hang %d = %0.2f",i,h[i]);
}
```

```
void tong_h(float a[][50], int m, int n, int *th)
{
    int i,j;
    for(i=1;i<=m;++i)
    {
        th[i]=0;
        for(j=1;j<=n;++j) th[i] += a[i][j];
    }
}
```


Con trỏ và mảng nhiều chiều...

3- Tham số thực là tên mảng nhiều chiều

Cách 2:

+ Dùng 2 đối:

`float *pa; /* biểu thị địa chỉ đầu của mảng a */`

`int N; /* biểu thị số cột của mảng a */`

+ Trong thân hàm, để truy nhập đến phần tử `a[i][j]` ta dùng công thức:

$$*(pa + i*N + j)$$

+ Mảng hai chiều được quy về mảng một chiều. Việc xử lý trong thân hàm sẽ phức tạp hơn so với cách thứ nhất, nhưng vì không cố định bao nhiêu cột nên có thể dùng hàm cho bất kỳ mảng hai chiều nào.

Con trỏ và mảng nhiều chiều...

3- Tham số thực là tên mảng nhiều chiều

Cách 2: Dùng 2 đối

```
include "stdio.h"
```

```
void vao_mt(float *a, int N, int m);
```

```
void ra_mt(float *a, int N, int m);
```

```
void nhan_mt(float *a, float *b, float *c, int N, int m);
```

```
void cong_mt(float *a, float *b, float *c, int N, int m);
```

```
main()
```

```
{
```

```
float a[20][20],b[20][20],c[20][20],d[20][20];
```

```
vao_mt ((float*)a,20,5); vao_mt ((float*)b,20,5);
```

```
nhan_mt((float*)a, (float*)b, (float*)c,20,5);
```

```
cong_mt((float*)a, (float*)b, (float*)d,20,5);
```

```
printf("\n Ma tran A\n ");
```

```
ra_mt((float*)a,20,5);
```

```
printf("\n Ma tran B\n ");
```

```
ra_mt((float*)b,20,5);
```

```
printf("\n Ma tran C\n ");
```

```
ra_mt((float*)c,20,5);
```

```
printf("\n Ma tran D\n ");
```

```
ra_mt((float*)d,20,5);
```

```
}
```

Con trỏ và mảng nhiều chiều.

3- Tham số thực là tên mảng nhiều chiều

Cách 2: Dùng 2 đối.

```
void vao_mt(float *a, int N, int m)
```

```
{  
    int i,j;  
    for(i=1; i<=m; ++i)  
        for(j=1; j<=m; ++j)  
        {  
            printf("\nphan tu (%d,%d) =  
                ",i,j);  
            scanf("%f", a + i*N + j);  
        }  
}
```

```
void ra_mt(float *a, int N, int m)
```

```
{  
    int i,j;  
    for(i=1; i<=m; ++i)  
    {  
        printf("\n");  
        for(j=1; j<=m; ++j)  
            printf("%8.2f", *(a + i*N + j) );  
    }  
}
```

```
void nhan_mt(float *a, float *b, float *c, int N, int m)
```

```
{  
    int i,j,k;  
    for(i=1; i<=m; ++i)  
        for(j=1; j<=m; ++j)  
        {  
            *(c + i*N + j) = 0;  
            for(k=1; k<=m; ++k)  
                *(c+i*N +j) +=  
                    (*(a+i*N+k))*(*(b+k*N+j));  
        }  
}
```

```
void cong_mt(float *a, float *b, float *c, int N, int m)
```

```
{  
    int i,j;  
    for(i=1; i<=m; ++i)  
        for(j=1; j<=m; ++j)  
            *(c+i*N +j) = *(a+i*N+j) +  
                *(b+i*N+j);  
}
```

Con trỏ và mảng nhiều chiều.

Bài tập

- Bài 4 Bài tập chương 1
- Các bài tập về ma trận khác:
 - + Tìm phần tử âm đầu tiên trong ma trận
 - + Xoay ma trận 5×5 ($n \times n$) một góc 90 độ theo chiều kim đồng hồ (dùng ma trận phụ, không dùng ma trận phụ)
 - + Sắp xếp một mảng 2 chiều $n \times n$ tăng dần theo cột và theo hàng ($a_{00} \leq a_{01} \leq \dots \leq a_{10} \dots \leq a_{nn}$)
 - + Biến đổi dấu của tất cả các số của 1 hàng hoặc 1 cột sao cho số lần biến đổi là ít nhất để được một ma trận có tổng mọi con số ở mọi hàng và mọi cột đều lớn hơn hoặc bằng 0
 - + Kiểm tra ma trận có đối xứng qua đường chéo chính không
 - + Sắp xếp ma trận 5×5 tăng dần theo hình xoắn ốc

MẢNG CON TRỎ - Con trỏ chỉ đến con trỏ

Mảng con trỏ là sự mở rộng khái niệm con trỏ

Mảng con trỏ là một mảng mà mỗi phần tử của nó có thể chứa được một địa chỉ nào đó

Mảng con trỏ được khai báo theo mẫu

```
type *namearr[N];
```

Ví dụ:

```
double *pa[100];
```

khai báo một mảng con trỏ kiểu double gồm 100 phần tử. **Mỗi phần tử pa[i] có thể dùng để lưu trữ một địa chỉ kiểu double**

Lưu ý: - Mảng con trỏ không thể dùng để lưu trữ số liệu

- Trước khi sử dụng một mảng con trỏ cần gán cho mỗi phần tử của nó một giá trị. Giá trị này phải là địa chỉ của một biến hoặc của một phần tử mảng.

MẢNG CON TRỎ - Con trỏ chỉ đến con trỏ

- Ta có thể khai báo một con trỏ chỉ đến một con trỏ chỉ đến 1 biến kiểu type như sau:

```
type  **ptr_ptr;
```

Ví dụ: `int **ptr_ptr;`

Với khai báo này ta có 1 biến con trỏ ptr và nếu lấy đối tượng (giá trị) của biến này ta sẽ được kết quả là một con trỏ trỏ đến (chỉ đến, lưu trữ địa chỉ) của một biến số nguyên.

- Khi sử dụng mảng con trỏ thì bản thân tên mảng là địa chỉ của con trỏ đầu tiên, và địa chỉ đó có thể được gán cho một con trỏ

MẢNG CON TRỎ

Ví dụ:

1-

```
char *monthname={ "January", "February", "March", "April", "May", "June", "July", "August",  
"September", "October", "November", "December"};
```

```
char **pp;
```

```
pp = monthname;
```

pp trỏ đến con trỏ đầu tiên của mảng monthname, con trỏ này chỉ đến chuỗi "January",

pp+1 thì trỏ đến "February",

2- trang 74, 75 giáo trình

CON TRỞ TỚI HÀM

Cách khai báo con trở hàm và mảng con trở hàm

Ta trình bày quy tắc khai báo thông qua các ví dụ.

+ Câu lệnh:

```
float (*f)(float), (*mf[50])(int);
```

là khai báo:

- f là con trở hàm kiểu float có đối float,
- mf là mảng con trở hàm kiểu float có đối int (mảng có 50 phần tử).

+ Câu lệnh:

```
double (*g)(int, double), (*mg[30])(double, float);
```

là khai báo:

- g là con trở hàm kiểu double có các đối int và double,
- mg là mảng con trở hàm kiểu double có các đối double và float (mảng có 30 phần tử).

CON TRỎ TỚI HÀM

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

Ví dụ 1: Vừa khai báo vừa gán.

```
double fmax(double x, double y) /* Hàm tính max */
{
    return (x>y?x:y);
}
/* Khai báo và gán tên hàm cho con trỏ hàm */
double (*pf)(double, double) = fmax;
/* Sử dụng con trỏ hàm trong hàm main */
main()
{
    printf("\nmax= %f",pf(4.5,78.9));
}
```

Ví dụ 2: Gán sau khi khai báo.

```
double fmax(double x, double y) /* Hàm tính max */
{
    return (x>y?x:y);
}
/* Khai báo con trỏ hàm */
double (*pf)(double, double);
main()
{
    /* Gán tên hàm cho con trỏ hàm */
    pf = fmax;
    /* Sử dụng con trỏ hàm */
    printf("\nmax= %f",pf(4.5,78.9));
}
```

CON TRỎ TỚI HÀM

Đổi con trỏ hàm

C cho phép thiết kế các hàm mà tham số thực trong lời gọi tới nó lại là tên của một hàm khác. Khi đó tham số hình thức tương ứng phải là một con trỏ hàm.

Nếu đổi được khai báo:

```
double (*f)(double, int)
```

thì trong thân hàm ta có thể dùng các cách viết sau để xác định giá trị của hàm (do con trỏ `f` trỏ tới):

```
f(x,m)  (f)(x,m)  (*f)(x,m)
```

ở đây `x` là biến `double`, `m` là biến `int`.

ĐỆ QUY

C không những cho phép từ hàm này gọi tới hàm khác, mà nó còn cho phép **từ một vị trí trong thân của một hàm gọi tới chính hàm đó**. Hàm như vậy gọi là hàm đệ quy.

Khi hàm gọi đệ quy đến chính nó thì mỗi lần gọi, máy sẽ tạo ra một tập các biến cục bộ mới hoàn toàn độc lập với các tập biến (cục bộ) đã được tạo ra trong các lần gọi trước.

Có bao nhiêu lần gọi tới hàm thì cũng có bấy nhiêu lần thoát ra khỏi hàm và cứ mỗi lần ra khỏi hàm thì một tập các biến cục bộ bị xóa.

Sự tương ứng giữa các lần gọi tới hàm và các lần ra khỏi hàm được thực hiện theo thứ tự ngược, nghĩa là: Lần ra đầu tiên ứng với lần vào cuối cùng và lần ra khỏi hàm cuối cùng ứng với lần đầu tiên gọi tới hàm.

ĐỆ QUY....

Ví dụ: Viết một chương trình tính n giai thừa.

Hàm không dùng phương pháp đệ quy như sau:

```
long gt(int n) /* tính n! với n>=0 */
{
    long int s=1; int i;
    for(i=1; i<=n; ++i)
        s *= i;
    return s;
}
```

$n!$ có thể tính theo công thức truy hồi như sau:

$n! = 1$ nếu $n = 0$

$n! = (n-1)! * n$ nếu $n > 0$

Dựa vào công thức trên ta có thể xây dựng hàm để tính $n!$ một cách đệ quy như sau:

```
long gtdq(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return (n*gtdq(n-1));
}
```

ĐỀ QUY....

```
include "stdio.h"

main()
{
    printf("\n 3! = %ld",gtdq(3));
}
```

Lần gọi đầu tiên tới hàm gtdq được thực hiện từ hàm main(). Máy sẽ tạo ra một tập các biến cục bộ của hàm gtdq. Tập này chỉ gồm đối n. Ta gọi đối n được tạo từ lần thứ nhất là n thứ nhất. Giá trị của tham số thực (số 3) được gán cho n thứ nhất. Lúc này biến n trong thân hàm được xem là n thứ nhất. Do n thứ nhất có giá trị 3, nên điều kiện trong câu lệnh if là sai, máy lựa chọn câu lệnh sau else. Theo câu lệnh này máy sẽ tính biểu thức

$n * \text{gtdq}(n-1)$

Để tính biểu thức trên cần gọi tới chính hàm gtdq. Lần gọi thứ hai được thực hiện. Máy sẽ tạo ra đối n mới, ta gọi nó là n thứ hai. Giá trị của n-1 ở đây vẫn hiểu là n thứ nhất, được truyền cho n thứ hai. Như vậy, n thứ hai có giá trị 2. Bây giờ trong thân hàm n được hiểu là n thứ hai. Điều kiện $n==1$ || $n==0$ vẫn chưa được thỏa mãn nên máy lại tính biểu thức

$n * \text{gtdq}(n-1)$

Cần phải gọi tới hàm gtdq lần thứ ba. Máy sẽ tạo ra đối n mới, ta gọi nó là n thứ ba. Giá trị n-1, ở đây n hiểu là n thứ hai được truyền cho n thứ ba. Vậy n thứ ba có giá trị 1. Trong thân hàm n được hiểu là n thứ ba. Điều kiện $n==1$ đúng và máy thực hiện câu lệnh `return 1`

Bắt đầu từ đây máy sẽ lần lượt thực hiện ba lần ra khỏi hàm gtdq. Lần ra thứ nhất ứng với lần vào thứ ba. Kết quả là: Đối n thứ ba được giải phóng, hàm cho giá trị $\text{gtdq}(1) = 1$, máy trở về để xét biểu thức

$n * \text{gtdq}(1)$

n hiểu là n thứ hai, nên $n=2$. Biểu thức trên có giá trị $2*1 = 2$. Theo câu lệnh `return`, máy sẽ thực hiện lần ra khỏi thứ hai. Kết quả là: Đối n thứ hai được giải phóng, hàm cho giá trị $\text{gtdq}(2)=2$, máy trở về với biểu thức

$n * \text{gtdq}(2)$

n hiểu là n thứ nhất, nên $n = 3$. Biểu thức trên có giá trị $3.2.1 = 6$. Câu lệnh `return` thực hiện lần ra thứ ba. Ta để ý rằng lần ra thứ ba ứng với lần vào thứ nhất. Máy sẽ giải phóng n thứ nhất, trở về hàm main() và cho giá trị $\text{gtdq}(3) = 6$. Giá trị này được sử dụng trong câu lệnh `printf` trong hàm main(). Do vậy trên màn hình hiện ra kết quả sau

3! = 6

một hàm đệ quy dùng nhiều vùng nhớ trên ngăn xếp và có thể dẫn đến tràn ngăn xếp. Vì vậy với một bài toán có cách giải lặp (không đệ quy) thì nên chọn cách này. Song tồn tại nhiều bài toán chỉ có thể giải bằng đệ quy.

ĐỆ QUY....

Bài toán nào có thể dùng đệ quy

Phương pháp đệ quy thường áp dụng cho các bài toán phụ thuộc tham số có hai đặc điểm sau:

- Bài toán dễ dàng giải quyết trong một số trường hợp riêng ứng với các giá trị đặc biệt của tham số. Ta gọi đây là trường hợp suy biến.
- Trong trường hợp tổng quát, bài toán có thể quy về một bài toán cùng dạng nhưng giá trị tham số thì bị thay đổi. Và sau một số hữu hạn bước biến đổi đệ quy, sẽ dẫn tới trường hợp suy biến.

ĐỆ QUY....

+ Cách xây dựng hàm đệ quy

Hàm đệ quy thường được viết theo thuật toán sau:

if (trường hợp suy biến)

{

trình bày cách giải bài toán

(giả định đã có cách giải)

}

else /* trường hợp tổng quát */

{

gọi đệ quy tới hàm (đang lập) với giá trị khác của tham số

}

ĐỀ QUY....

Ví dụ:

- UCLN

- Tháp Hà Nội

ĐỀ QUY

Bài tập

ĐỐI DÒNG LỆNH

C cho phép đưa vào hàm main hai đối: Đối thứ nhất là biến kiểu int, đối thứ hai là mảng con trỏ kiểu char. Tên các đối do người lập trình tự đặt. Dạng của hàm main có đối là

```
void main (int n, char *a[])
```

```
{
```

```
....
```

```
}
```

giá trị của n và a được nhận từ bàn phím khi khởi động chương trình

```
Ví dụ:#include <stdio.h>
#include <stdlib.h>
main(int pc, char *pv[])
{
float x, s=0.0;
int i;
if(pc > 1)
{
printf("\nKet qua");
for(i=1; i<pc; ++i)
{
```

```
x=atof(pv[i]); /* Đổi từ chuỗi sang float */
printf("\n%0.2f",x);
s += x;
}
}
printf("\nTong la: %0.2f\n",s);
}
```

Với thao tác

```
D:\TC>SUM 1.5 3.6 -97 5.1
```

(chụp ảnh màn hình)

IV. THAO TÁC TRÊN CÁC TỆP TIN

- KIỂU TỆP TIN NHỊ PHÂN VÀ VĂN BẢN
- CÁC THAO TÁC TRÊN TỆP TIN
- CÁC HÀM NHẬP XUẤT KIỂU TỆP TIN VĂN BẢN
- TRUY CẬP TỆP TIN NGẪU NHIÊN

KIỂU TỆP TIN NHỊ PHÂN VÀ VĂN BẢN

Một tệp tin (dù nó được xây dựng bằng cách nào) đơn giản chỉ là một dãy các byte (có giá trị từ 0 đến 255) ghi trên đĩa. Số byte của dãy chính là độ dài của tệp.

Kiểu tệp tin nhị phân

- Trong quá trình nhập xuất dữ liệu không bị biến đổi. Dữ liệu ghi trên tệp theo các byte nhị phân như trong bộ nhớ.
- Trong khi đọc nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (định nghĩa trong `stdio.h` bằng -1) và hàm `fEOF` cho giá trị khác không.

Kiểu tệp tin nhị phân và văn bản.

Kiểu tệp tin văn bản

- Nhập xuất tệp tin văn bản chỉ khác tệp tin nhị phân khi xử lý ký tự chuyển dòng (mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.
- Khi ghi, một ký tự LF (mã 10) được chuyển thành hai ký tự CR (mã 13) và LF. Khi đọc, hai ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF.
- Trong khi đọc nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (số -1) và hàm feof(fp) cho giá trị khác không (số 1). Điều này cốt để phù hợp với một số hệ soạn thảo (như C, Notepad,...). Tệp soạn thảo trong các hệ này kết thúc bởi mã 26.

CÁC THAO TÁC TRÊN TỆP TIN

Các thao tác tệp trong C được thực hiện nhờ các hàm thư viện. Các hàm này được chia thành hai nhóm: cấp 1 và cấp 2.

Mỗi hàm (dù cấp 1 hay cấp 2) đều có thể truy xuất theo cả hai kiểu nhị phân và văn bản.

Các hàm cấp 1 (còn gọi là các hàm nhập/xuất hệ thống) có các đặc trưng cơ bản là:

- + Không có dịch vụ nhập xuất riêng cho từng kiểu dữ liệu mà chỉ có dịch vụ đọc ghi một dãy các byte.
- + Mỗi tệp có một thẻ (handle) và các hàm cấp 1 làm việc với tệp thông qua số hiệu tệp.

CÁC THAO TÁC TRÊN TỆP TIN...

- Các hàm cấp 2 được xây dựng từ các hàm cấp 1 nên dễ sử dụng và có nhiều khả năng hơn:
- + Có dịch vụ truy xuất cho từng kiểu dữ liệu.
- + C tự động cung cấp một vùng đệm. Mỗi lần đọc/ghi thì thường tiến hành trên vùng đệm chứ không hẳn trên tệp. Chẳng hạn khi ghi một số nguyên thì số được đưa vào vùng đệm và khi nào đầy thì vùng đệm mới được đẩy lên đĩa. Khi đọc, thông tin được lấy từ vùng đệm, và chỉ khi vùng đệm đã trống rỗng thì máy mới lấy dữ liệu từ đĩa chứa vào vùng đệm. Việc sử dụng vùng đệm sẽ giảm số lần nhập xuất trên đĩa và nâng cao tốc độ làm việc. Tuy vậy trong một số trường hợp ta phải nhớ tới tác nghiệp vét vùng đệm để đề phòng mất mát thông tin.
- + Các hàm cấp 2 làm việc với tệp thông qua một biến con trỏ tệp

CÁC THAO TÁC TRÊN TỆP TIN...

- Các hàm cấp 1 ở mức độ sâu hơn, gần hệ điều hành hơn nên tốc độ truy nhập sẽ nhanh hơn. Tuy vậy, các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1, nên trong chương trình C các hàm cấp 2 được ưa chuộng hơn. Các hàm cấp 2 sẽ là nội dung của phần này

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM DÙNG CHUNG CHO CẢ HAI LOẠI TỆP

1- Hàm fopen

- + Dạng hàm:
- FILE *fopen(const char *tên_tệp, const char *kiểu);
- + Công dụng: Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp 2 sẽ làm việc với tệp thông qua con trỏ này. Nếu có lỗi hàm trả về giá trị NULL.
- + Các đối:
- Đối thứ nhất là xâu ký tự, đường dẫn đến tên tệp, đối thứ hai là kiểu truy nhập. Kiểu có thể có các giá trị sau:

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM DÙNG CHUNG CHO CẢ HAI LOẠI TỆP...

Kiểu	Ý nghĩa
"r" "rt"	Mở một tệp để đọc theo kiểu văn bản. Tệp cần tồn tại nếu không sẽ có lỗi.
"w" "wt"	Mở một tệp mới để ghi theo kiểu văn bản. Nếu tệp đã tồn tại nó bị xoá.
"a" "at"	Mở một tệp để ghi bổ sung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"rb"	Mở một tệp để đọc theo kiểu nhị phân. Tệp cần tồn tại nếu không sẽ có lỗi.
"wb"	Mở một tệp mới để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại nó bị xoá.
"ab"	Mở một tệp để ghi bổ sung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+" "r+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Tệp cần tồn tại nếu không sẽ có lỗi.
"w+" "w+t"	Mở một tệp mới để ghi/đọc theo kiểu văn bản. Nếu tệp đã tồn tại nó bị xoá.
"a+" "a+t"	Mở một tệp để đọc/ghi bổ sung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+b"	Mở một tệp để đọc/ghi theo kiểu nhị phân. Tệp cần tồn tại nếu không sẽ có lỗi.
"w+b"	Mở một tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại nó bị xoá.
"a+b"	Mở một tệp để đọc/ghi bổ sung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM DỪNG CHUNG CHO CẢ HAI LOẠI TỆP...

- Hàm **fclose**
- + Dạng hàm:
- `int fclose(FILE *fp);`
- + Công dụng: Hàm dùng để đóng tệp. Nội dung đóng tệp gồm: Đẩy dữ liệu còn trong vùng đệm lên đĩa (khi đang ghi), xoá vùng đệm (khi đang đọc) và giải phóng biến fp để nó có thể dùng cho tệp khác. Nếu thành công hàm trả về giá trị 0, trái lại hàm trả về giá trị EOF.
- + Đối: fp là con trỏ tương ứng với tệp cần đóng.

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM DỪNG CHUNG CHO CẢ HAI LOẠI TỆP...

- Hàm `fcloseall`
- + Dạng hàm:
- `int fcloseall(void);`
- + Công dụng: Hàm dùng để đóng tất cả các tệp đang mở. Nếu thành công hàm trả về giá trị nguyên bằng số tệp đóng được, trái lại hàm trả về giá trị EOF.

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM DỪNG CHUNG CHO CẢ HAI LOẠI TỆP...

- Hàm fflush
- + Dạng hàm:
- `int fflush(FILE *fp);`
- + Công dụng: Hàm dùng làm sạch vùng đệm của tệp do con trỏ fp trỏ tới. Nếu thành công hàm trả về giá trị 0, trái lại hàm trả về giá trị EOF.
- + Đối: fp là con trỏ tệp.

CÁC THAO TÁC TRÊN TẬP TIN...

CÁC HÀM DỪNG CHUNG CHO CẢ HAI LOẠI TẬP...

- Hàm fflush
- + Dạng hàm:
 - int fflush(void);
- + Công dụng: Hàm dùng làm sạch vùng đệm của các tệp đang mở. Nếu thành công hàm trả về giá trị nguyên bằng số tệp đang mở, trái lại hàm trả về EOF.

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM DỪNG CHUNG CHO CẢ HAI LOẠI TỆP...

- Hàm feof
- + Dạng hàm:
 - `int feof(FILE *fp);`
- + Công dụng: Hàm dùng để kiểm tra con trỏ tệp đang mở có trỏ cuối tệp hay không. Hàm trả về giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm trả về giá trị 0.
- + Đối: fp là con trỏ tệp.

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM CHỈ DÙNG CHO TỆP NHỊ PHÂN

- Các hàm chỉ dùng cho tệp nhị phân

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM CHỈ DÙNG CHO TỆP VĂN BẢN

CÁC HÀM NHẬP XUẤT KIỂU TỆP TIN VĂN BẢN

- **Hàm fprintf**

- + Dạng hàm:

- `int fprintf(FILE *fp, const char *dk, bt);`

- + Công dụng: Giá trị các biểu thức bt được ghi lên tệp fp theo khuôn dạng xác định trong chuỗi điều khiển dk. Nếu thành công, hàm trả về một giá trị nguyên bằng số byte ghi lên tệp. Khi có lỗi hàm cho EOF. Hàm làm việc giống như printf.

- + Đối:

- fp là con trỏ tệp,

- dk chứa địa chỉ của chuỗi điều khiển,

- bt là danh sách các biểu thức mà giá trị của chúng cần ghi lên tệp.

- Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm printf.

CÁC THAO TÁC TRÊN TỆP TIN

CÁC HÀM CHỈ DÙNG CHO TỆP VĂN BẢN...

- Ví dụ: Xét chương trình:
- `#include <stdio.h>`
- `main()`
- `{`
- `FILE *f;`
- `int i;`
- `f=fopen("text","wt");`
- `fprintf(f, "Cac dong");`
- `for(i=1; i<=2; ++i)`
- `fprintf(f, "\nDong%2d", i);`
- `fclose(f);`
- `}`

CÁC THAO TÁC TRÊN TỆP TIN...

CÁC HÀM CHỈ DÙNG CHO TỆP VĂN BẢN

- **Hàm fscanf**
- + Dạng hàm:
 - `int fscanf(FILE *fp, const char *dk, ...);`
- + Công dụng: Đọc dữ liệu từ tệp fp, biến đổi theo khuôn dạng (đặc tả) trong dk và lưu kết quả vào các đối. Hàm làm việc giống như scanf. Hàm trả về một giá trị bằng số trường được đọc.
- + Đối:
 - fp là con trỏ tệp,
 - dk chứa địa chỉ của chuỗi điều khiển,
 - ... là danh sách địa chỉ các đối chứa kết quả đọc được từ tệp.
 - Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm scanf.

CÁC THAO TÁC TRÊN TỆP TIN

CÁC HÀM CHỈ DÙNG CHO TỆP VĂN BẢN...

- **Hàm fputs**

- + Dạng hàm:

- `int fputs(const char *s, FILE *fp);`

- + Công dụng: Ghi chuỗi s lên tệp fp (dấu '\0' không ghi lên tệp). Khi thành công, hàm trả về ký tự cuối cùng được ghi lên tệp. Khi có lỗi hàm cho EOF.

- + Đối:

- s là con trỏ trỏ tới địa chỉ đầu của một chuỗi ký tự kết thúc bằng dấu '\0'

- fp là con trỏ tệp.

CÁC THAO TÁC TRÊN TỆP TIN

CÁC HÀM CHỈ DÙNG CHO TỆP VĂN BẢN...

- **Hàm fgetc**

- + Dạng hàm:

- `char *fgetc(char *s, int n, FILE *fp);`

- + Công dụng: Đọc một dãy ký tự từ tệp fp chứa 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.

- Xâu kết quả sẽ được bổ sung thêm dấu hiệu kết thúc chuỗi '\0'. Khi thành công, hàm trả địa chỉ vùng nhận kết quả. Khi có lỗi hoặc gặp cuối tệp, hàm cho giá trị NULL.

- + Đối:

- s là con trỏ (kiểu char)trỏ tới một vùng nhớ đủ lớn để chứa chuỗi ký tự đọc từ tệp.

- n là số nguyên xác định độ dài cực đại của dãy cần đọc.

- fp là con trỏ tệp.

TRUY CẬP TẬP TIN NGẪU NHIÊN

- Document

III. CẤP PHÁT ĐỘNG BỘ NHỚ VÀ DANH SÁCH LIÊN KẾT

-
- CẤP PHÁT ĐỘNG BỘ NHỚ
 - CẤU TRÚC
 - DANH SÁCH LIÊN KẾT

-
- Cấu trúc có ít nhất một thành phần là con trỏ kiểu cấu trúc đang định nghĩa được gọi là cấu trúc tự trỏ. Cấu trúc tự trỏ có thể định nghĩa theo một trong các cách sau đây.

V. XỬ LÝ NGẮT VÀ CÁC CHỈ THỊ TIỀN XỬ LÝ

- XỬ LÝ NGẮT
- GIAO DIỆN GIỮA C VÀ ASSEMBLER
- CÁC CHỈ THỊ TIỀN XỬ LÝ