BÀI 6
HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

MỤC TIÊU

Người học sau khi học xong bài 6 sẽ có các khái niệm cơ bản về các vấn đề sau:

- Tổ chức chương trình thành các hàm.
- Xây dựng và sử dụng hàm.
- Con trỏ địa chỉ và con trỏ mảng.
- Mảng con trỏ, con trỏ trỏ tới hàm.
- Đệ quy.

KIẾN THỨC CẦN CÓ

Các kiến thức cần thiết:

- Học xong bài 5
- Có kiến thức về tư duy lập trình
- Phân tích chương trình thành module
- Thực hiện các bài toán đệ quy
- Khuyến nghị học môn Tin học cơ bản.

NỘI DUNG

1. Tổ chức chương trình thành các hàm
2. Xây dựng hàm và sử dụng hàm
3. Con trỏ và địa chỉ
4. Con trỏ và mảng một chiều
5. Con trỏ và mảng nhiều chiều
6. Kiểu con trỏ, kiểu địa chỉ
7. Mảng con trỏ
8. Con trỏ trỏ tới hàm
9. Đệ quy.

6.1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM

- Trong quá trình thực hiện chương trình, không nhất thiết phải khai báo hàm đúng như mẫu:
  - Nếu khai báo khác mẫu, có thể giúp cho quá trình code nhanh hơn
  - Khai báo đúng mẫu sẽ dễ hiểu, phân tích, gỡ rối và sửa lỗi.
- Quy chuẩn cho các hàm:
  - Mỗi hàm có 1 tên, tên phải theo đúng quy tắc
  - Hàm có thể có nhiều đối số hoặc không có đối số nào
  - Hàm thường trả về các giá trị, nó có kiểu trả về được khai báo trước Khi không có giá trị trả về thì kiểu trả về là void.

6.1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM (tiếp theo)

Quy tắc xây dựng hàm:

- Dòng tiêu đề: chứa các thông tin về kiểu hàm, tên hàm, kiểu và tên mỗi đối số.

Ví Dụ:

```
float max3s(float a, float b, float c);
```

- Thân hàm: Bắt đầu bằng { và kết thúc bằng }. Thân hàm có chứa các phép toán cần thiết với hàm và với các đối số, có trả kết quả ra màn hình hoặc trả về giá trị của hàm thông qua lệnh return.

```
{
/*Nội dung của thân hàm*/
return (biểu thức);
}
```

6.1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM (tiếp theo)

Quy tắc hoạt động của hàm:

- Chương trình gọi hàm:

```
max3s(6,8,12);
```

- Cấp phát bộ nhớ cho các biến cục bộ
- Gán giá trị của tham số cho các đối tượng
- Thực hiện các câu lệnh trong thân hàm
- Khi gặp câu lệnh return hoặc } thì thực hiện xóa các đối số, biến cục bộ và thoát khỏi hàm.
- Nếu lệnh return có biểu thức đi kèm, thì giá trị của biểu thức đó sẽ được gán cho giá trị của hàm.

6.1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM (tiếp theo)

- Cấu trúc tổng quát của chương trình:
  - Các hàm thư viện #include
  - Các hằng được khai báo #define
  - Các biến, mảng, cấu trúc được khai báo
  - Các hàm nguyên mẫu
  - Hàm main
  - Thông tin chi tiết về các hàm nguyên mẫu.

  Chú ý: Hàm main có thể đặt trước, hoặc sau các hàm nguyên mẫu.
- Thông tin chi tiết về các hàm nguyên mẫu cần phải được thể hiện hoặc khai báo trước khi được sử dụng lần đầu tiên.

6.2. XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM

- Để có thể xây dựng được 1 hàm, chúng ta cần chú ý các vấn đề sau:
  - Tên hàm.
  - Kiểu giá trị trả về của hàm.
  - Đối số hay tham số hình thức.
  - Thân hàm.
  - Khai báo hàm.
  - Lời gọi hàm.
  - Tham số thực hoặc đối số thực.
- Quá trình xây dựng hàm và sử dụng hàm phải được thực hiện tuần tự, 2 giai đoạn này độc lập và tách rời nhau.

6.2. XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM (tiếp theo)

Xây dựng hàm nhằm mục đích xác định các thông tin cơ bản về hàm.

Cấu trúc sau:

```
kiểu tên_hàm(khai báo các đối số)
{
khai báo các biến cục bộ .
các lệnh.
[return [biểu thức]]
}
```

Ví dụ:

```
void max3s(float a, float b, float c)
{
      float x;
      x = a;
      if (x<b) x = b;
      if (x<c) x = c;
      printf("\n Max =%0.2f",x);
}
```

6.2. XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM (tiếp theo)

- Sử dụng hàm: Sau khi khai báo và xây dựng hàm xong, thì có thể sử dụng hàm thông qua các câu lệnh gọi hàm:

```
tên_hàm ([danh sách các đối số thực])
```

Ví Dụ: max3s(23,45,332);

Chú ý:

- Danh sách các đối số thực có thể không có
- Số tham số thực phải bằng số đối số thực
- Kiểu của tham số thực và kiểu của đối số phải trùng nhau
- Với hàm không có kiểu trả về, thường có các câu lệnh printf kết quả tính toán của hàm.

6.2. XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM (tiếp theo)

Nguyên tắc hoạt động của hàm

Ví Dụ sai :

- Hàm hoanvi đổi vị trí của x và y nhưng khong có giá trị trả về nên kết quả không được ghi nhận.
- Chương trình chính gán x = 4, y = 8 sau đó gọi hàm, nhưng hàm không hoạt động nên khi in, kết quả vẫn cho x = 4 và y = 8 thay vì x = 8 và y = 4.

```
void hoanvi(int x,int y)
{
    int z;
    z = x; x = y; y = z;
}
void main()
{
    int x,y;
    x = 4; y = 8;
    hoanvi(x,y);
    print("x = %d y = %d",x,y);
}
```

6.3. CON TRỎ VÀ ĐỊA CHỈ

- Địa chỉ cho phép xác định các thông số liên quan đến biến:
  - tên biến : x
  - kiểu biến : float
  - giá trị : 30.5
  - kích thước : 4byte cho 1 biến float.

```
float x = 30.5;
```

- Con trỏ là 1 loại biến dùng để chứa địa chỉ. Có nhiều loại địa chỉ và nhiều con trỏ tương ứng. Cú pháp khai báo con trỏ như sau:
  - kiểu: kiểu của con trỏ.
  - *: cú pháp con trỏ.
  - tên_con_trỏ: tên biến dùng làm con trỏ.

```
kiểu * tên_con_trỏ;
```

Ví Dụ:

```
int* x,y,z;
```

6.3. CON TRỎ VÀ ĐỊA CHỈ (tiếp theo)

Quy tắc sử dụng con trỏ trong các biểu thức:

- Con trỏ cũng là 1 biến nên khi nó xuất hiện trong biểu thức thì giá trị của nó sẽ được sử dụng trong biểu thức.

Ví Dụ:

```
float a, *p, *q;
p = &a;
q = p;
```

  - ➢ Gán địa chỉ biến a cho con trỏ p.
  - ➢ Sau đó gán giá trị p cho q.
- Khi biết địa chỉ của 1 biến, thì có thể dùng con trỏ để dùng giá trị của nó hoặc làm thay đổi nội dung của giá trị. Điều này cho phép nhận kết quả của hàm thông qua đối số.

6.3. CON TRỎ VÀ ĐỊA CHỈ (tiếp theo)

Hàm có đối con trỏ: Đối số nằm trong hàm là con trỏ sẽ cho phép hàm gán giá trị mới cho nó thông qua các câu lệnh thích hợp:

Ví Dụ:

- Hàm hoanvi thực hiện đổi chỗ 2 giá trị x, y. Trong đó cả x và y đều được đổi chỗ trông qua địa chỉ của chúng.
- Chương trình chính khai báo a =8 và b =9. Sau khi thực hiện hoán vị thì a = 9 va b = 8

```
void hoanvi(float *x, float*y)
{
float z;
z = *x;
*x = *y;
*y = z;
}
#include “stdio.h”
main()
{
float a = 8, b=9;
hoanvi(&a,&b);
printf(“\n a = %0.2f b=%0.2f”,
a,b);
}
```

6.4. CON TRỎ VÀ MẢNG MỘT CHIỀU

- Sau khi khai báo mảng 1 chiều, ta có thể lấy địa chỉ của các phần tử trong mảng 1 chiều thông qua câu lệnh sau:

```
double b[20];
&b[i];
```

  - Mảng b gồm 20 phần tử.
  - Địa chỉ &b[i] với 0<=i <=19 sẽ cho giá trị của các phần tử nằm trong mảng b.

- Nếu tên mảng là 1 hằng địa chỉ thì tọa độ các biến trong mảng liên tiếp nhau. Do đó, khi biết 1 địa chỉ của 1 phần tử trong mảng, sẽ có thể suy ra được địa chỉ các phần tử còn lại.

Ví Dụ:

  - Mảng float a[8] có 8 phần tử. Nếu x là con trỏ trỏ tới vị trí a[3]

    thì x+2 sẽ trỏ tới vị trí a[3+2] = a[5]

6.4. CON TRỎ VÀ MẢNG MỘT CHIỀU (tiếp theo)

Ví Dụ: Khai báo 1 mảng a gồm 4 phần tử, sau đó nhập dữ liệu mẫu vào mảng rồi in ra màn hình giá trị của mảng:

- 4 giá trị của mảng a[i] được Nhập vào từ bàn phím
- Tổng giá trị của 4 phần tử này được tính và chuyển vào giá trị
- Màn hình sẽ in ra giá trị của s.

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

6.4. CON TRỎ VÀ MẢNG MỘT CHIỀU (tiếp theo)

- Nếu đối số của mảng là 1 chiều thì ta hoàn toàn có thể thực hiện khai báo dạng biến con trỏ thay vì khai báo mảng:

```
int *a;
float *b;
double *c;
```

```
int a[];
float b[];
double c[];
```

- Với chuỗi ký tự, chương trình C cung cấp 1 mảng bộ nhớ để lưu thông tin và chèn thêm ký tự \0 để báo hiệu kết thúc chuỗi. Ta có thể dùng biến con trỏ đế lấy thông tin của chuỗi ký tự:

```
char *s;
s = "Nguyen Van A"';
```

```
puts(s);
puts("Nguyen Van A");
```

- Quá trình hiển thị chuỗi hoặc con trỏ cho cùng 1 kết quả như nhau.

6.5. CON TRỎ VÀ MẢNG NHIỀU CHIỀU

- Mảng nhiều chiều thường có vài tham số thay đổi, cho phép định nghĩa nhiều phần tử và đa dạng hơn so với mảng 1 chiều.
- Các quy tắc áp dụng cho mảng 1 chiều có thể sai đối với mảng nhiều chiều:

Ví Dụ:

```
a[i][j];
```

  - Phần tử i, j thuộc mảng a. Khai báo thông thường không sai
  - Nhưng khi lấy địa chỉ của chúng thì lại là sai :

```
&a[i][j];
```

- Phép toán lấy địa chỉ của mảng nhiều chiều không dùng được
- Quá trình đánh địa chỉ cho mảng nhiều chiều được bố trí theo thứ tự các hàng.

Ví Dụ: mảng a[2][3] gồm 6 phần tử thì địa chỉ của chúng như sau:

a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
1	2	3	4	5	6

6.5. CON TRỎ VÀ MẢNG NHIỀU CHIỀU (tiếp theo)

- Có thể kết hợp con trỏ và mảng 2 chiều để trỏ các thông tin của mảng.

Ví Dụ: Với khai báo float *p, a[2][3]:

- p trỏ tới a[0][0] .
- p +1 trỏ tới a[0][1] .
- p +2 trỏ tới a[0][2] .
- p +3 trỏ tới a[1][0] .
- p +4 trỏ tới a[1][1] .
- p +5 trỏ tới a[1][2] .

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

- Chương trình trên mô tả các dùng con trỏ để vào số liệu cho mảng 2 chiều.

6.5. CON TRỎ VÀ MẢNG NHIỀU CHIỀU (tiếp theo)

- Ngoài ra, ta cũng có thể dùng biến trung gian để vào số liệu cho mảng 2 chiều.

Ví Dụ: Tính tích 2 ma trận a và b, sau đó đưa vào ma trận c:

```
for(i = 0. i<3. i++)
for (j = 0. i<2. j++)
{
printf("\n a[%d] [%d] = ",i,j);
scanf("%f",&x);
a[i][j] = x;
}
```

```
for(i = 0. i<3. i++)
for (j = 0. i<4. j++)
{
c[i][j] =0;
for(k = 0. k<2.k++)
c[i][j] += a[i][k]* b[k][j];
}
```

- Nhập biến vào địa chỉ x, sau đó mới gán vào giá trị a[i][j]

6.5. CON TRỎ VÀ MẢNG NHIỀU CHIỀU (tiếp theo)

Khi mảng nhiều chiều là tham số thực, để có thể dùng tên mảng trong lời gọi hàm ta có thể thông qua một số cách sau:

- Dùng đối con trỏ:

```
float a[50][60];
```

```
float (*pa)[50];
float pa[][50];
```

- Dùng 2 đối số:

```
float *pa; /*biểu thị địa chỉ đầu mảng a*/
int N; /* biểu thị số cột*/
```

6.6. KIỂU CON TRỎ, KIỂU ĐỊA CHỈ

Con trỏ dùng để lưu trữ địa chỉ, mỗi kiểu địa chỉ cần có kiểu con trỏ tương ứng. Phép gán chỉ được thực hiện khi 2 kiểu này trùng nhau.

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

- pa: con trỏ kiểu float
- pm: con trỏ kiểu float[30]
- a: địa chỉ kiểu float[30]
- phép gán : pa = a sẽ bị cảnh báo
- phép gán : pm= a hợp lệ.

6.6. KIỂU CON TRỎ, KIỂU ĐỊA CHỈ (tiếp theo)

Các phép toán trên con trỏ điển hình:

- Phép gán: hai bên phải cùng kiểu hoặc được ép kiểu:

```
int x;
char *pc;
pc = (char *) (&x); /*ép kiểu*/
```

- Phép tăng, giảm địa chỉ:

```
float x[30], *px;
px = &x[10]; /*Khi đó px + i sẽ trỏ tới x[10 +i]*/
```

- Phép truy cập bộ nhớ: Mỗi loại con trỏ sẽ có kích thước truy cập bộ nhớ khác nhau, con trỏ float, int, char có kích thước bộ nhớ lần lượt là 4,2,1 byte.
- Phép so sánh: dùng để so sánh các con trỏ có cùng kiểu với nhau.

6.6. KIỂU CON TRỎ, KIỂU ĐỊA CHỈ (tiếp theo)

Trường hợp đặc biệt khi ta khai báo con trỏ mà không muốn xác định kiểu cho nó thì có thể dùng kiểu void thay thế.

Cú pháp:

```
void *tên_con_trỏ;
```

Ví Dụ:

```
void *p;
float a[20][30];
p = a;
```

- lúc đầu p chưa có kiểu.
- sau phép gán, kiểu của p sẽ là float.

6.7. MẢNG CON TRỎ

- Mảng con trỏ là một khái niệm mở rộng của con trỏ.
- Mảng con trỏ là một mảng mà mỗi phần tử có thể chứa một địa chỉ nào đó.
- Mảng con trỏ có thể có nhiều kiểu khác nhau.
- Cú pháp khai báo:

```
kiểu  *tên_mảng[N];
```

  - Kiểu có thể là int, float, double, char…
  - tên_mảng: tuân theo quy tắc đặt tên biến thông thường.
  - N: hằng số nguyên xác định độ lớn của mảng.

Ví dụ:

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

6.7. MẢNG CON TRỎ (tiếp theo)

Ví Dụ:

- Danh sách của các thành viên được lập bởi mảng con trỏ tĩnh kiểu char.
- Tên các thành viên chính là các phần tử và được đưa vào theo cơ chế khởi đầu của mảng static.

```
static char *ds[] = {
"Ma sai",
"Nguyen Van A",
"Pham Thu Huong",
"Tran Thi Lien",
"Dang Binh Long",
"Nguyen Quang Thai",
"Do Thi Be",
"Vu Thi Binh",
}
```

6.8. CON TRỎ TỚI HÀM

Cú pháp khai báo con trỏ hàm và mảng con trỏ hàm như sau:

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

- f: là con trỏ hàm float có đối số cũng có kiểu float.
- mf: mảng con trỏ hàm kiểu float với đối số kiểu int.

```
double (*g)(int, double), (*mg[50](double,float);
```

- g: là hàm kiểu double với 2 đối số có kiểu là int và double.
- mg là mảng con trỏ hàm kiểu double, mg có 2 đối số là double và float (mảng có 50 phần tử).

6.8. CON TRỎ TỚI HÀM (tiếp theo)

Tác dụng:

- Con trỏ hàm để chứa địa chỉ của hàm thông qua phép gán tên hàm cho con trỏ hàm.
- Để phép gán có nghĩa thì kiểu con trỏ và kiểu hàm phải tương thích với nhau (cùng kiểu hoặc ép kiểu).
- Sau khi thực hiện phép gán, ta có thể thay thế tên hàm bởi tên con trỏ.

```
#include      "stdio.h"
#include      "conio.h"
int max(int x, int y)
{
int z;  z =
x;
if (x<y) z = y;
return z;
}
int (*tinhmax)(int, int) = max;

main()
{
clrscr();
printf("\n max = %d",tinhmax (9,6));
getch();
}
```

6.8. CON TRỎ TỚI HÀM (tiếp theo)

- Chương trình C cũng cho phép thiết kế các hàm mà tham số thực trong lời gọi nó lại là tên của hàm khác. Khi đó các tham số hình thức phải là tên của một con trỏ hàm.
  - Cú pháp khai báo đối số con trỏ hàm:
    - Con trỏ hàm f, có 2 đối số:
      - x kiểu double.
      - m kiểu int.

```
double (*f)(double, int);
f(x,m)
(f) (x,m)
(*f)(x,m)
```

    - Xác định giá trị của hàm f có thể thông qua 1 trong 3 cách sau:
      - f(x,m).
      - (f) (x,m).
      - (*f)(x,m).

6.9. ĐỆ QUY

- Khái niệm chung: Đệ quy là cách chương trình di chuyển từ 1 điểm trong thân hàm tới chính hàm của nó.
- Khi hàm gọi đệ quy đến chính nó, chương trình C sẽ tạo ra một tập các biến cục bộ mới hoàn toàn độc lập với các biến đã được tạo ra:

Ví Dụ:

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

```
long dq(int n) /*tính n! đệ quy*/
{
if(n =1 | n = 0)
   return 1;
else
   return(n*dq(n-1));
}
```

6.9. ĐỆ QUY (tiếp theo)

Cách dùng đệ quy thường được áp dụng cho các bài toán phụ thuộc vào tham số với các đặc điểm chủ yếu sau:

- Bài toán dễ dàng giải quyết trong 1 số trường hợp riêng ứng với các giá trị đặc biệt của tham số (trường hợp suy biến).
- Trong trường hợp tổng quát, bài toán có thể quy về bài toán cùng dạng nhưng đối số bị thay đổi. Quá trình này sẽ dẫn đến trường hợp suy biến.

```
if(trường hợp suy biến)
{
        cách giải đơn giản và cụ thể
}
else
{
        Gọi hàm đệ quy.
        Tính toán nhằm dẫn đến trường hợp suy biến
}
```

TÓM LƯỢC CUỐI BÀI

- Bài 6 khá quan trọng trong nội dung chương trình, nó cung cấp cho người học tổng quan về các kỹ thuật khó trong C.
- Việc phân tích chương trình thành từng module nhỏ cũng như các hàm nhỏ sẽ giúp cho quá trình lập trình thuận tiện hơn, mã chương trình thân thiện, dễ nâng cấp, chỉnh sửa và gỡ rối
- Với các cấu trúc tương đương, việc ứng dụng mảng để lưu thông tin và dùng con trỏ để xác định thông tin sẽ cho hiệu quả cao hơn nhiều so với các phương pháp khai báo sử dụng.