Chương 4: Hàm & kỹ thuật tổ chức chương trình

4.1 Khái niệm về Hàm

- Hàm (function) là một đoạn chương trình được xây dựng để thực hiện một chức năng nào đó. Đoạn chương trình này chỉ cần phải viết duy nhất một lần và có thể được sử dụng nhiều lần trong toàn bộ chương trình.

- Hàm trong C bao gồm luôn thủ tục (procedure) và còn gọi là chương trình con /module.

- Sử dụng hàm có thể làm giảm kích thước và chi phí của chương trình, đồng thời nâng cao tính dễ đọc và độ tin cậy.

- Các chương trình con (thường được tập trung thành các thư viện) là một cơ chế quan trọng cho việc chia sẻ và tái sử dụng mã nguồn, làm tăng đáng kể hiệu quả lập trình.

- Một hàm sẽ được xác định bởi tên hàm và các tham số đầu vào liên quan đến hàm đó. Hàm sẽ xử lý các thông tin đầu vào (nếu có) và trả về một kết quả nào đó *(ngoại trừ các hàm đơn giản có thể không có tham số vào).*

- *Với chương trình con không có giá trị trả về thì một số ngôn ngữ lập trình gọi là thủ tục – trong C vẫn gọi là hàm, lúc này kiểu trả về là* ***void****.*

- Hàm có thể được gọi từ nhiều nơi trong chương trình, thậm chí có thể gọi từ chính bên trong thân hàm đó (khi này gọi là **đệ qui**), mỗi lần gọi có thể truyền một bộ tham số vào khác nhau.

Cú pháp khai báo hàm:

```
<Kiểu dữ liệu trả về> <Tên Hàm> ( [<Danh sách tham số>] )
{
   <Các lệnh của hàm>
   [ return [<giá trị>]; ]
}
```

- Tên Hàm được đặt theo đúng qui tắc đặt tên *(không có khoảng trắng ở giữa, không bắt đầu bởi ký tự số, tránh trùng từ khóa,..).*

- <Kiểu dữ liệu trả về> có thể là bất kỳ kiểu dữ liệu gì của C, nếu hàm không có giá trị trả về thì dùng kiểu **void**.

- Phần <Danh sách tham số> có thể không có tham số, trong trường hợp có nhiều tham số thì phải dùng dấu ‘**,**’ để phân cách giữa các tham số. Mỗi tham số ở đây có thể khai báo tương tự như khai báo biến.

- Ngay sau khi thực hiện xong các lệnh của hàm phải có lệnh **return** để trả về giá trị kết quả (nếu hàm kiểu **void** thì không cần).

- Lệnh **return** cũng có thể đặt ở giữa thân hàm để hàm kết thúc sớm khi gặp một điều kiện gì đó (dùng kèm với cấu trúc **if** tương tự như lệnh **break**).

- Để tiện lợi cho việc chia sẻ và tái sử dụng, nên có chú thích thuyết minh về công dụng của hàm và ý nghĩa của các tham số đầu vào ngay tại dòng khai báo tên hàm.

- Nên xem xét đặt tham số cho hàm một cách hợp lý nhất có thể, để hàm có khả năng tái sử dụng cho nhiều bài toán khác nhau. *(trước mắt các thao tác nhập xuất trực tiếp ngay bên trong hàm cần xem xét thay thế bằng các tham số input và output)*

Ví dụ:

* Hàm tính giai thừa:

```
#include <stdio.h>
long GiaiThua (int n) {  // tính n!
  long S = 1;
  for ( int j = 2; j < n; j++ )
      S *= j;
   return S;
 }
 void main () {
    int m;
    printf ( " 7! = %ld\n", GiaiThua(7) );
    printf ( " Nhap so nguyen can tinh giai thua: " );
    scanf ( "%d", &m );
    printf ( " %d! = %ld\n", m, GiaiThua(m) );
 }
```

* Hàm xác định số thiếu, thừa hay đủ:

```
int SoHoanThien ( int n) {  // kết quả =0: n là số đủ;  =-1: thiếu;   =1:thừa
  int tong_uoc = 1;
  for ( int j = n/2; j > 1; j-- )
     if ( n % j == 0 )
        tong_uoc += j;
  if ( tong_uoc == n) return 0;
  else if ( tong_uoc < n) return -1;
         else return 1;
 }
```

* Hàm tìm ước số chung lớn nhất của 2 số:

```
int UCLN (int a, int b) {  // ước chung lớn nhất của 2 số nguyên dương a, b
  int uoc = a<b? a: b;
  while ( (a%uoc !=0) || (b%uoc !=0) )
      uoc --;
   return uoc;
 }
```

* Hàm + Chương trình kiểm 1 số có phải số nguyên tố:

```
#include "stdafx.h"
// Ham kiem tra so N co nguyen to hay khong.
// Ham tra ve 0 neu N khong phai nguyen to va tra ve 1 neu N nguyen to
int LaSoNguyenTo(int N) {
  for (int i = 2; i < N - 1; i++)
    if (N % i == 0)  // N chia het cho 1 so ko phai 1 hoac N nen KHONG nguyen to
      return 0; // lenh RETURN gan giong BREAK nhung ko chi thoat khoi cau truc ma thoat luon khoi ham
  return 1; //vong lap FOR chay het (ko bi thoat giua chung) => N ko chia het cho so nao => N nguyen to
}
//----------------------------------------
void main() {
        int a;
        scanf_s("%d", &a);
        if ( LaSoNguyenTo(a) ) // tuong duong voi ( LaSoNguyenTo(a) != 0 )
          printf("So %d la so nguyen to \n", a);
        else
          printf("So %d Khong phai so nguyen to \n", a);
}
```

Cũng có thể viết hàm kiểm tra số nguyên tố như sau:

```
int LaNguyenTo(int N) { // N co phai so nguyen to. Tri tra ve: 0 - ko phai / 1 - phai
  int NguyenTo = 1;
  for (int i = 2; i < N/2; i++)
    if (N % i == 0) {
      NguyenTo = 0;
      break;
    }
  return NguyenTo;
}
```

4.2 Truyền tham số cho Hàm

Khái niệm về tham số:

- Tham số hình thức là biến được liệt kê trong danh sách tham số của hàm (thường nằm tại dòng khai báo hàm). Còn tham số thực sự là giá trị cụ thể của biến đó tại thời gian chạy, chỉ đến các thông tin được truyền cho hàm trong các lời gọi hàm.

- Tham số hình thức thường được gọi tắt là tham số. Tham số thực sự còn được gọi là tham số thực hoặc đối số. *(Tuy nhiên vẫn có một ít tài liệu lại gọi chung tham số thực và tham số hình thức là tham số).*

- Mỗi tham số thực tương ứng với một tham số hình thức. Kiểu dữ liệu của tham số hình thức sẽ quyết định kiểu giá trị cho tham số thực tương ứng.

- Ví dụ, với hàm f(x) = x + 2, thì x là một tham số hình thức. Khi hàm được dùng, ví dụ: y = f(3) + 5, thì giá trị 3 là tham số thực.

Khái niệm về truyền tham số:

- Khi gọi thực hiện một hàm ta phải truyền các tham số cho nó *(dĩ nhiên ngoại trừ các hàm đơn giản không có tham số)*. Các tham số phải được truyền chính xác, đúng trật tự và đúng kiểu thì hàm mới thực hiện đúng.

Ví dụ hai lời gọi hàm printf("%d",a) và scanf("%d",&a) có hình thức truyền khác nhau ở tham số thứ hai.

- Cơ chế gán đối số cho một tham số được gọi là cơ chế truyền tham số.

Truyền tham số bằng giá trị (call by value)

- Trong cơ chế truyền tham số bằng giá trị *(gọi tắt là truyền tham trị, truyền giá trị hoặc truyền trị)*, khi hàm chạy thì một bản sao của tham số thực được gán cho tham số hình thức (sao chép giá trị).

-> Do đó các sửa đổi của hàm đối với tham số hình thức không gây ảnh hưởng tới biến được truyền vào.

- Các tham số được truyền bằng giá trị được gọi là tham trị. Do chỉ có giá trị được truyền vào chương trình con, tham số thực sự không nhất thiết phải là một biến thông thường mà có thể là hằng giá trị, hằng biến, biểu thức trả về giá trị...

Ví dụ:

```
void Hoanvi_0 ( int x, int y) {
  int tam;
  tam = x;
  x = y;
  y = tam;
}
void main () {
  int a, b;
  a = 2;
  b = 3;
  Hoanvi_0 (a, b);   // kết quả là a và b vẫn không bị thay đổi giá trị

  Hoanvi_0 (0, 1); // không bị báo lỗi
  Hoanvi_0 (a+3*b, b/a); // không bị báo lỗi
}
```

Truyền tham số bằng biến (call by reference)

- Trong cơ chế truyền tham số bằng biến *(gọi tắt là truyền tham biến hoặc truyền biến)*, khi hàm chạy thì tham số hình thức trở thành tham chiếu tham số thực.

-> Do đó các sửa đổi của hàm đối với tham số hình thức sẽ có tác dụng tới biến được truyền vào.

- Các tham số được truyền bằng biến được gọi là tham biến. Khác với hình thức truyền trị có thể truyền một giá trị, hình thức truyền tham biến bắt buộc phải truyền vào một biến.

- Đối số được truyền cho hàm ở dạng địa chỉ. Ở dòng khai báo hàm phải có ký hiệu **&** trước tên biến.

- Hình thức này được sử dụng khi có nhu cầu thay đổi giá trị của tham số sau khi thực hiện hàm.

Ví dụ 1:

```
void Hoanvi_1 ( int &x, int y) {
  int tam;
  tam = x;
  x = y;
  y = tam;
}
void main ( ) {
  int a, b;
  a = 2;
  b = 3;
  Hoanvi_1 (a, b);   // kết quả là a = 3 và b = 2
```

```
}
```

Ví dụ 2:

```
int TinhTong ( int x, int y) {
  return (x+y) ;
}
void TinhTong2 ( int x, int y, int &tong) {
  tong = x+y;
}
void TinhTongHieu ( int x, int y, int &tong, int &hieu) {
  tong = x+y;
  hieu = x–y ;
}
void main ( ) {
  int a, b, c, d, e, f;
  a = 2;
  b = 3;
  c =  TinhTong ( a, b ) ;
  TinhTong2 ( a, b, d ) ;
  TinhTongHieu ( a, b, e, f) ;
}
  // kết quả sau khi chạy chương trình là c = d = e = 5  và  f = -1

int TichThuong ( double a, double b, double &thuong, double &tich)
{  // tinh tich & thuong cua 2 so a, b. Ham tra ve 0 neu ko co thuong
  tich = a*b;
  if (b==0) return 0 ;
  thuong = a/b;
  return 1 ;
}
```

Truyền tham số bằng con trỏ (call by address)

- Trong cơ chế truyền tham số bằng con trỏ *(gọi tắt là truyền tham trỏ hoặc truyền địa chỉ)*, tham số hình thức phải để ở dạng con trỏ (dùng

ký hiệu * trước tên tham số), tham số thực khi truyền phải là địa chỉ (dùng ký hiệu & trước tên tham số). Các sửa đổi của hàm đối với tham trỏ sẽ có tác dụng tới biến được truyền vào.

- Hình thức này được sử dụng khi có nhu cầu thay đổi giá trị của tham số sau khi thực hiện hàm, hoặc khi tham số là một dãy phần tử (mảng,..)

Ví dụ:

```
void Hoanvi_2 ( int *x, int *y) {
  int tam;
  tam = *x;
  *x = *y;
  *y = tam;
}
void main ( ) {
  int a, b;
  a = 2;
  b = 3;
  Hoanvi_2 (&a, &b);   // kết quả là a = 3 và b = 2
}
```

4.3 Hàm đệ qui

- Hàm được gọi là đệ qui nếu trong quá trình thực hiện nó có gọi đến chính nó.

- Hàm đệ qui phải được thiết kế sao cho số lần tự gọi đến chính nó là có giới hạn *(không chỉ để hàm có thể kết thúc sau một khoảng thời gian hữu hạn mà còn để không bị tràn stack)*.

Ví dụ: Hàm đệ qui tính n!

Ta nhận thấy n! = n * (n-1)! ➔ cần tính (n-1)!
Mà (n-1)! = (n-1) * (n-2)! ➔ cần tính (n-2)!
Tương tự cần tính (n-3)!, …, 2!, 1!, 0!
Và 0!=1

⇨ Hàm đệ qui có thể viết như sau:

```
int GiaiThua (int n) {
  if (n == 0)  return 1;
  return n*GiaiThua(n – 1);
}
```

4.4 Biến toàn cục và biến cục bộ

- Các biến khai báo bên ngoài tất cả các hàm gọi là biến toàn cục và có hiệu lực trên toàn bộ chương trình.

- Các biến khai báo trong hàm hoặc khối lệnh gọi là biến cục bộ và chỉ có hiệu lực trong hàm hoặc khối lệnh đó. Biến cục bộ sẽ không còn tồn tại ở bên ngoài khối khai báo nó.

- Biến cục bộ được quyền đặt trùng tên với biến toàn cục, khi đó biến toàn cục sẽ không có hiệu lực trong khối lệnh chứa biến cục bộ đó.

Ví dụ 1:

```
int y=1;
void f1 (int x) {
  int y;
  y = 12+x;
}
void f2 ( int x) {
  y = 12+x;
}
```

```
void main ( ) {
  f1 (y); // y=1
  f2 (y); // y = 13
}
```

Ví dụ 2:

```
int y=1;
void f1 (int x) {
  y = 12+x;
}
int f2 ( int x) {
  int y;
  y = 12+x;
  return y+1;
}
void main ( ) {
  f1 ( f2 (y) ); // y = 13
}
```

Ví dụ 3:

```
int a, b;

void Hoanvi_3 ( ) {
  int tam;
  tam = a;
  a = b;
  b = tam;
}

void main ( ) {
  a = 2;
  b = 3;
  Hoanvi_3( );   // kết quả là a = 3 và b = 2
}
```

BÀI TẬP:

1. Viết hàm kiểm tra ba số nguyên a,b,c có thể là ba cạnh của tam giác hay không.
2. Viết hàm xác định:
    a. Tổng các chữ số của số N.
    b. Các chữ số của số N có tăng dần hay giảm dần không?
    c. Số N có phải là số đối xứng? (77, 232, 2002,..)
    d. Số N có phải là số nguyên tố?
    e. Số N có phải là số đủ? (bằng tổng các ước nhỏ hơn nó)
3. Viết hàm tính:
    a. $S = 1 + 1/2 + \ldots + 1/n$
    b. $S = 1! + 2! + \ldots + n!$
    c. $S = 1^1 + 2^2 + 3^3 + \ldots + n^n$
4. Viết hàm giải phương trình bậc 2.

```
int SoDu ( int n) {  // kết quả =0: n là số đủ;  =-1: thiếu;   =1:thừa

  int tong_uoc = 1;

  for ( int j = n/2; j > 1; j-- )
     if ( n % j == 0 )
        tong_uoc += j;
```

```
  if ( tong_uoc == n) return 0;
  else if ( tong_uoc < n) return -1;
               else return 1;
 }

int a, b;
void main() {
for (a=1; a<=1000; a++)
  for (b=1; b<=1000; b++)
    if ( SoDu(a)==-1 && SoDu(b)==1 && SoDu(a+b)==0 )
              printf (“%d : thieu + %d : thua = %d : du”, a, b, a+b);
}
```