


Nguyễn Phúc Khải

CHƯƠNG 9: *HÀM*

Các nội dung:

- Khái niệm hàm
- Khai báo hàm
- Đối số của hàm - đối số là tham trị
- Kết quả trả về của hàm - lệnh RETURN
- PROTOTYPE của một hàm
- Hàm đệ quy

KHÁI NIỆM HÀM

- Chương trình con là đoạn chương trình đảm nhận thực hiện một thao tác nhất định.
- Đối với C, chương trình con chỉ ở một dạng là hàm (function), không có khái niệm thủ tục (procedure).

KHÁI NIỆM HÀM

- Hàm **main**() là hàm đặc biệt của C, nó là một hàm mà trong đó các thao tác lệnh (bao gồm các biểu thức tính toán, gọi hàm, ...) được C thực hiện theo một trình tự hợp logic để giải quyết bài toán được đặt ra.
- Việc sử dụng hàm sẽ làm cho chương trình trở nên rất dễ quản lý, dễ sửa sai.
- Tất cả các hàm đều ngang cấp nhau. Các hàm đều có thể gọi lẫn nhau, dĩ nhiên hàm được gọi phải được khai báo trước hàm gọi.

KHÁI NIỆM HÀM

- Các hàm trong một chương trình có thể nằm trên các tập tin khác nhau và khác với tập tin chính (chứa hàm **main**()), mỗi tập tin được gọi là một **module** chương trình
- Các module chương trình sẽ được dịch riêng rẽ và sau đó được liên kết (link) lại với nhau để tạo ra được một tập tin thực thi duy nhất.
- Cách tạo chương trình theo kiểu nhiều module như vậy trong C là **project**

KHÁI NIỆM HÀM

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
main ()
{
    double a, b, c, delta, n1, n2;
    clrscr();
    printf ("Nhap 3 he so phuong trinh bac hai; ");
    scanf ("%lf %lf %lf", &a, &b, &c);
```

KHÁI NIỆM HÀM

```
if (a ==0)/* phuong trinh suy bien ve bac nhat */
    {     printf ("Phuong trinh suy bien ve bac nhat va ");
          if (b == 0)
                    if (c == 0)
                              printf ("vo so nghiem\n");
                    else /* c != 0 */
                              printf ("vo nghiem\n");
          else / * b != 0 */
{         n1 = -c/b;
          printf ("co 1 nghiem: = %5.2f \n", n1);
                    }
}
```

KHÁI NIỆM HÀM

```
    else /* a != 0 */
    {
        printf ("Phuong trinh bac hai va ");
        delta = b*b - 4*a*c;
if (delta < 0)
            printf ("vo nghiem thuc\n");
        else if (delta == 0)
        {
            n1 = n2 = -b/2/a;
            printf ("co nghiem kep x1=x2 = %5.2f \n" ,n1);
        }
```

KHÁI NIỆM HÀM

```
            else /* delta > 0 */
        {
            n1 = (-b + sqrt(delta))/2/a;
            n2 = (-b - sqrt(delta))/2/a;
            printf ("co hai nghiem phan biet; \n");
            printf ("x1 = %5.2f \n", n1);
            printf ( x2 = %5.2f \n", n2);
        }
    }
    getch();
}
```

KHÁI NIỆM HÀM

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void gptb1 (double a, double b);
void gptb2 (double a, double b, double c);
```

KHÁI NIỆM HÀM

```
void gptb1 (double a, double b)
{
    printf ("Phuong trinh suy bien ve bac nhat va ");
    if (a == 0)
        if (b == 0)
                printf ("vo so nghiem\n");
        else /* b != 0 */
                printf ("vo nghiem\n");
    else
                printf ("co 1 nghiem: x = %5.2f \n",-b/a);
}
```

KHÁI NIỆM HÀM

```
void gptb2 (double a,double b,double c)
{   double delta, x1, x2;
    printf ("Phuong trinh bac hai va ");
    delta = b*b - 4*a*c;
    if (delta < 0)
        printf ("vo nghiem thuc\n");
    else if (delta == 0)
        printf ("co nghiem kep x1 = x2 = %5.2f \n", -b/2/a);
else /* delta > 0 */
    {           x1 = (-b + sqrt(delta))/2/a;
        x2 = (-b - sqrt(delta))/2/a;
        printf ("co hai nghiem phan biet: \n");
        printf ("x1 = %5.2f \n ", x1);
        printf ("x2 = %5.2f \n" , x2);
    }
}
```

KHÁI NIỆM HÀM

```
main()
{
    double a, b, c;
    clrscr();
    printf ("Nhap 3 he so phuong trinh bac hai: ");
    scant ("%lf %lf %lf", &a, &b, &c);
    if (a == 0)  /* phuong trinh suy bien ve bac nhat */
          gptb1 (b, c);
    else /* a != 0 */
          gptb2 (a, b, c);
    getch();
}
```

KHAI BÁO HÀM

- Khai báo một hàm là chỉ ra rõ rằng trả về kiểu gì, đối số đưa vào cho hàm có bao nhiêu đối số, mỗi đối số có kiểu như thế nào và các lệnh bên trong thân hàm xác định thao tác của hàm.
- Có hai loại hàm: hàm trong thư viện và hàm do lập trình viên tự định nghĩa.

KHAI BÁO HÀM

- Nếu hàm sử dụng là hàm chuẩn trong thư viện thì việc khai báo hàm chỉ đơn giản là khai báo prototype của hàm, các prototype này đã được phân loại và ở trong các file .h, lập trình viên cần ra lệnh **#include** bao hàm các file này vào chương trình hoặc module chương trình sử dụng nó.

KHAI BÁO HÀM

- Nếu các hàm sử dụng là do lập trình viên tự định nghĩa thì việc khai báo hàm bao gồm hai việc: khai báo prototype của hàm đầu chương trình và định nghĩa các lệnh bên trong thân hàm (hay thường được gọi tắt là định nghĩa hàm).

KHAI BÁO HÀM

- Cú pháp khai báo hàm:

kiểu tên_hàm
(danh_sách_khai_báo_đối_số)
{
    khai_báo_biến_cục_bộ
        lệnh
}

KHAI BÁO HÀM

```
int so_sanh (int a, int b)
{
    int ket_qua;
    if (a >b)
        ket_qua = 1:
    else if (a == b)
        ket_qua = 0;
    else if (a < b)
        ket_qua = -1;
    return ket_qua;
}
```

KHAI BÁO HÀM

```
#include <stdio.h>
#include <conio.h>
int so_sanh (int a, int b); ← prototype của hàm so_sanh
main()
{
    int a, b, ket_qua;
    clrscr();
    printf ("Moi nhap hai so ");
    scanf ("%d %d" , &a, &b);
    ket_qua = so_sanh (a, b);          ← gọi hàm
```

KHAI BÁO HÀM

```
    switch (ket_qua)
        { case -1:
                printf ("So %d nho hon so %d \n" , a, b);
                break;
          case 0:
                printf ("So %d bang so %d \n", a, b);
                break;
          case 1:
                printf ("So %d lon hon so %d \n" , a, b);
                break;          }
    getch();
}
```

KHAI BÁO HÀM

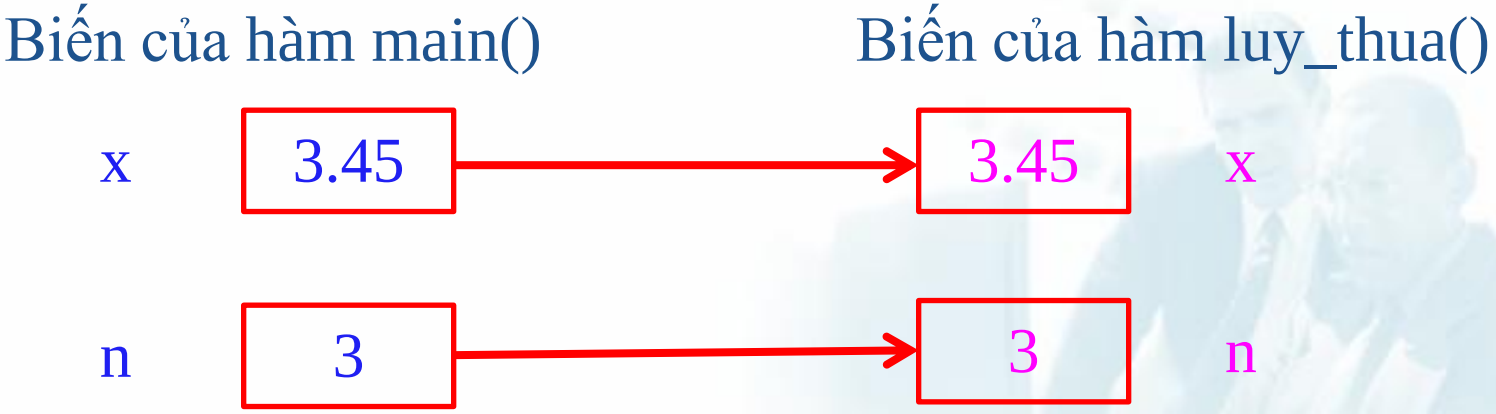
```
int so_sanh (int a, int b)
{
    int ket_qua:
    if (a >b)
        ket_qua = 1;
    else if (a == b)
        ket_qua = 0;
    else if (a < b)
        ket_qua = -1;
    return ket_qua;
}
```

ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

- Khi gọi hàm thì đối số thật cần gởi cho hàm chỉ được gởi dưới dạng tham số trị, có nghĩa là các biến, trị hoặc biểu thức được gởi đến cho một hàm, qua đối số của nó, sẽ được lấy trị để tính toán trong thân hàm.
- Có thể nói trị của biến thật bên ngoài khi gọi hàm đã được chép sang đối số giả, ta có thể xem như là biến cục bộ của hàm, và mọi việc tính toán chỉ được thực hiện trên biến cục bộ này mà thôi.

ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

- Khi gọi hàm lũy thừa, trị của biến x và n sẽ được chép vào cho hai đối số giả x và n, do đó ta có đồng thời các hộp biến như sau khi vào trong hàm luy_thua():



ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

```
#include <stdio.h>
#include <conio.h>
double luy_thua(double x, int n);
main()
{   int n;
    double x, xn;
    clrscr();
    printf ("Moi nhap so tinh luy thua: ");
    scant ("%lf", &x);
    printf ("Moi nhap so luy thua: ");
    scanf ("%d", &n);
    xn =luy_thua (x, n);
```

ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

```
    printf("Ket qua: %5.2f luy thua %d bang: %7.2f\n", x,n,xn);
    printf ("Tri cua so mu la %d", n);
    getch();
}
double luy_thua(double x, int n)
{   double t = 1;
    for ( ; n > 0; n--)
       t *= x;
    return t;     }
```

ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

- Ta có thể gọi hàm *luy_thua()* và truyền cho hàm này một biểu thức:

```
xn = luy_thua( 3*a + x , 5);
```

- Tuy nhiên, cách truyền tham số như trên không thể thay đổi trị của biến, mà điều này đôi khi lại cần thiết.

ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

```
#include <stdio.h>
#include <conio.h>
void nhap_tri (int a, int b);
main()
{   int a = 0, b = 0;
    clrscr();
    printf ("Truoc khi goi ham nhap_tri: a = %d, b = %d\n", a,
    b);
    nhap_tri (a, b);
    printf("Sau khi goi ham nhap_tri a = %d, b = %d\n", a, b);
    getch();
}
```

ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

```
void nhap_tri (int a, int b)
{
    printf ("Moi nhap hai so: ");
    scanf ("%d %d", &a, &b);
}
```

KẾT QUẢ TRẢ VỀ CỦA HÀM

- Đối với C không có sự phân biệt giữa thủ tục (procedure) và hàm (function), mà thủ tục cũng được xem là một hàm mà không trả về giá trị nào cả. Để khai báo kiểu trả về từ hàm như vậy C đưa ra kiểu **void**, tạm gọi là **kiểu không hiểu.**
- Ví dụ: so sánh 2 trường hợp sử dụng hàm

c = getch(); và getch();

- hoặc

c = getche(); và getche();

KẾT QUẢ TRẢ VỀ CỦA HÀM

- Trong chương trình, ta cũng biết lệnh **return** dùng để thực hiện việc trả trị của hàm về cho nơi gọi nó, dù trị này có được sử dụng hay không tùy nơi gọi.

KẾT QUẢ TRẢ VỀ CỦA HÀM

```
int so_sanh (int a, int b)
{   if (a >b)
    {     printf ("So %d lon hon so %d", a, b);
          return 1;       }
    else if (a == b)
    {     printf ("So %d bang so %d", a, b);
          return 0;       }
    else /* a <b */
    {     printf ("So %d nho hon so %d", a, b);
          return -1;      }
}
```

KẾT QUẢ TRẢ VỀ CỦA HÀM

```
main()
        {
                int so1, so2;
                clrscr();
                printf("Moi nhap hai so: ");
                scanf ("%d %d", &so1, &so2);
                so_sanh (so1, so2);
                getch();
        }
```

KẾT QUẢ TRẢ VỀ CỦA HÀM

```
main()
{
    int so1, so2;
    int kq;
    clrscr();
    printf ("Moi nhap hai so: ");
    scanf ("%d %d", &so1, &so2);
    kq = so_sanh (so1, so2);
```

KẾT QUẢ TRẢ VỀ CỦA HÀM

```
    switch (kq)
    {
     case -1:
      printf (", nen tri tuyet doi hieu hai so la: %d\n,so2-so1");
      break;
     case 0:
     case 1:
      printf (", tri tuyet doi hieu hai so la %d\n , so1-so2");
      break;
    }
    getch();
}
```

KẾT QUẢ TRẢ VỀ CỦA HÀM

```
void so_sanh (int a, int b)
{   if (a > b)
        printf ("So %d lon hon so %d", a, b);
    else if (a == b)
        printf ("So %d bang so %d", a, b);
    else /* a < b */
        printf ("So %d nho hon s %d", a, b);
}
```

KẾT QUẢ TRẢ VỀ CỦA HÀM

- Khi khai báo hàm mà ta không nêu cụ thể kiểu trả về của hàm, C mặc nhiên xem như hàm trả về kết quả là **int**.

```
so_sanh (int a, int b)
        {       if (a > b)
                        return 1;
                else if (a == b)
                        return 0;
                else /* a < b */
                        return -1;          }
```

KẾT QUẢ TRẢ VỀ CỦA HÀM

- Đối với các hàm có kiểu trả về trị khác **int**, khi khai báo cần phải trình bày đầy đủ các thành phần của hàm.
- Khi gọi sử dụng hàm thì trong hàm gọi cần phải có nêu kết quả trả về của các hàm được gọi trong đó.
- Kiểu khai báo kết quả này có thể được đặt bên ngoài tất cả các hàm để thông báo cho tất cả các hàm về trị trả về của nó, hoặc có thể được đặt trong hàm mà hàm sử dụng được gọi: **kiểu tên_hàm();**

KẾT QUẢ TRẢ VỀ CỦA HÀM

```
main()
        {       int so_sanh ();
                int so1, so2;
                clrscr();
                printf ("Moi nhap hai so: ");
                scanf ("%d %d", &so1, &so2);
                so_sanh (so1, so2);
                getch();                }
```

PROTOTYPE CỦA MỘT HÀM

- Như vậy để một hàm có thể sử dụng trong một hàm khác thì trong hàm sử dụng phải có khai báo hàm cần sử dụng.
- Tuy nhiên khai báo này rất hạn chế ở chỗ không cho phép kiểm tra số đối số thật đưa vào hàm cũng như kiểu của đối số có phù hợp không

PROTOTYPE CỦA MỘT HÀM

- Để khắc phục những lỗi trên, trong những phát triển sau này của C theo ANSI, người ta đưa ra khái niệm prototype của một hàm, đây thật sự là một dạng khai báo hàm mở rộng hơn, có dạng tổng quát như sau:

kiểu tên_hàm (danh_sách_khai_báo_đối_số);

- **Ví dụ :** int so_sanh (int a, int b);
- void gptb1 (double a, double b, doubbe c);
- char kiem_tra (double n);

PROTOTYPE CỦA MỘT HÀM

- C cho phép khai báo prototype của hàm trong phần khai báo đối số chỉ cần có kiểu mà không cần có tên của đối số giả.
- **Ví dụ :**

int so_sanh (int, int);

PROTOTYPE CỦA MỘT HÀM

- Công dụng của prototype của hàm: prototype của một hàm ngoài việc dùng để khai báo kiểu của kết quả trả về từ một hàm, nó còn được dùng để kiểm tra số đối số.
- **Ví dụ:** Nếu đã khai báo prototype

```
int so_sanh (int a, int b);
```

- mà khi gọi hàm ta chỉ gửi một đối số như sau:

```
so_sanh (so2);
```

- thì sẽ bị C phát hiện và báo lỗi

PROTOTYPE CỦA MỘT HÀM

- **Chuyển kiểu của đối số:** khi một hàm được gọi, mà hàm đó có prototype, các đối số được gởi cho hàm sẽ được chuyển kiểu bắt buộc theo kiểu của các đối số được khai báo trong prototype, sự chuyển kiểu này làm cho các đối số được sử dụng phù hợp với các phép toán trong thân hàm.
- **Trường hợp mà sự chuyển kiểu không cho phép thực hiện** thì C sẽ đưa ra các thông báo lỗi, hoặc một lời cảnh báo (warning).

PROTOTYPE CỦA MỘT HÀM

- Đối với các hàm chuẩn trong thư viện C, prototype của chúng đã được viết sẵn và để trong các file có phần mở rộng là .h, muốn lấy các prototype này vào chương trình ta cần ra chỉ thị bao hàm file .h chứa prototype của các hàm cần sử dụng vào đầu chương trình bằng lệnh tiền xử lý **#include** theo cú pháp sau:

 # include <file. h>

HÀM ĐỆ QUY

- C cho phép một hàm có thể gọi đến chính nó một cách trực tiếp, hoặc gián tiếp (tức là gọi qua trung gian một hàm khác), khi đó ta nói hàm đó có tính đệ quy (recursive).
- Một giải thuật đệ quy sẽ dẫn đến một sự lặp đi lặp lại không kết thúc các thao tác, nhưng trong thực tế, chúng cần phải được kết thúc, sử dụng các điều kiện kết thúc đệ quy.

HÀM ĐỆ QUY

```
#include <stdio.h>
#include <conio.h>
long factorial (long so);
int main()
{   long so, kq = 0; clrscr();
    printf ("Moi nhap mot so khac 0: ");
    scanf ("%ld", &so);
    kq = factorial (so);
    printf ("Ket qua %ld! la %ld \n", so, kq);
    getch();
    return 0;
}
```

```
long factorial (long so)
{
    if (so > 1)
        return (factorial(so - 1) * so);
    else
        return 1;
}
```

Bài tập

- Thiết kế hàm tính các biểu thức sau đây:

S =(1)! + (1+2)! +…+ (1+…+n)!

- Thiết kế hàm in ra màn hình ***n*** chuỗi số Fibonaci, với ***n*** thông số nhập từ bàn phím trong số từ 1 đến 100.
- Viết một hàm nhận một số thực dương có phần lẻ và in ra màn hình phần nguyên và phần lẻ riêng biệt.

Kết thúc chương 9