


Nội dung

1. Hàm & Chương trình con
2. Xây dựng hàm
3. Gọi hàm
3. Bài tập

1. Hàm

- Là một đoạn chương trình độc lập, được đặt tên, thực hiện trọn vẹn một công việc cụ thể.
  - ✓ Có sẵn (built-in function)
  - ✓ Người lập trình tự xây dựng (user defined function)

- Hỗ trợ cho chương trình chính thực hiện ý đồ "chia để trị" trong việc giải quyết bài toán lớn.

- Một chương trình C bao gồm một hoặc nhiều hàm.
  Hàm *main*( ) là hàm chính, bắt buộc có của một chương trình C

- Đặc điểm của hàm:
  - Là một đơn vị độc lập của chương trình.
  - Không cho phép xây dựng một hàm bên trong một hàm khác.


2 Xây dựng hàm

1. Công việc cần để xây dựng hàm

- Xác định:
  - ✓ *Đầu vào*:
    - Giá trị
    - Các biến đại diện cho đầu vào
  - ✓ *Đầu ra*:
    - Các biến đại diện cho đầu ra
  - ✓ *Thuật toán*:
    - Các biến trung gian
    - Các bước tiến hành để thu được *đầu ra* từ *đầu vào*

2. Cấu trúc của hàm

1. Dòng **tiêu đề của hàm** mô tả các thông tin:

   Kiểu của hàm, *tên hàm*, *kiểu* và *tên của biến* đại diện cho đầu vào/ đầu ra

   - Danh sách biến vào/ra nằm trong dấu ( ) ngay sau tên hàm. Chúng phân tách nhau bởi dấu phẩy.
   - Kết thúc dòng tiêu đề KHÔNG có *dấu chấm phẩy*

   ➢ Một số hàm có thể không có biến vào/ra

2. **Thân hàm**

   Nằm ngay sau dòng tiêu đề

   - Thân hàm là các câu lệnh thực hiện công việc theo các bước. Tất cả được đặt trong cặp dấu { }
   - Trong thân hàm có thể sử dụng lệnh return để trả(gán) kết quả cho tên hàm.

   ➢ Một số trường hợp có thể bỏ qua câu lệnh return khi xây dựng hàm.

2.1. Mô hình của một hàm

```
Kiểu_của_hàm     tên_hàm( khai báo biến vào/ra )
{
    Khai báo các biến cục bộ;
    Các câu lệnh;
    return (biểu thức hoặc giá trị);
}
```

Dòng tiêu đề

Thân hàm

Có thể có hoặc không, tùy bài toán

2.2. Quy tắc xây dựng hàm

- Mỗi hàm có một tên, được đặt theo quy tắc như biến.
- Khai báo biến vào/ra:
  - *Biến đại diện đầu vào:* biến thường
  - *Biến đại diện cho đầu ra:* con trỏ
- Đưa giá trị ra bên ngoài hàm:
  - Đưa chỉ MỘT giá trị ra khỏi hàm, dùng lệnh *return* để gán giá trị đó cho tên hàm (hoặc gán vào biến đại diện cho đầu ra). Xem ví dụ 1a, 1b.
  - Đưa NHIỀU giá trị ra khỏi hàm, các giá trị đó được gán cho các biến đại diện đầu ra. Xem ví dụ 2.
- Kiểu của hàm:
  - Nếu gán giá trị cho tên hàm, kiểu của hàm có thể là *float, int, char, double*,…
  - Nếu không gán giá trị cho tên hàm, kiểu của hàm sẽ là *void*. Xem ví dụ 1b, 2

Ví dụ 1a

Lập hàm tính trung bình cộng của 3 số nguyên

(Lấy *kết quả ra* thông qua *tên của hàm*)

```
float  tbc(int x, int y, int z)
       {
       return( (x + y + z)/3.0);
       }
```

Cách này không dùng biến trung gian bên trong hàm

```
float   tbc(int x, int y, int z)
        {
        float  tg;
        tg = (x + y + z)/3.0;
        return(tg);
        }
```

Dòng tiêu đề

Thân hàm

Cách này dùng thêm biến trung gian *tg* bên trong hàm

Ví dụ 1b

Lập hàm tính trung bình cộng của 3 số nguyên

(Lấy *kết quả ra* thông qua *một con trỏ*)

```
void  tbc(int x, int y, int z, float  *avg)
      {
          *avg = (x + y + z)/3.0;
      }
```

Cách này không dùng biến trung gian bên trong hàm

```
void   tbc(int x, int y, int z, float *avg)
       {
          float  tg;
          tg = (x + y + z)/3.0;
          *avg = tg;
       }
```

Cách này dùng thêm biến trung gian *tg* bên trong hàm


2.3. Một số khái niệm liên quan tới hàm

- Tham số của hàm
- Phạm vi của biến trong chương trình


Tham số của hàm

- Có thể coi *tham số* chính là các đại lượng vào/ra của hàm.
  - Các biến đại diện (mang tính hình thức) cho các dữ liệu vào/ra của hàm, được khai báo ở dòng tiêu đề *khi xây dựng hàm*, có thể coi là các tham số hình thức.
  - Tham số thực tế là các dữ liệu thực tế được truyền cho tham số hình thức tương ứng *khi gọi hàm*
- Khi gọi hàm:
  - Tham số thực tế của hàm có thể là *giá trị* hoặc *địa chỉ*
- Một số hàm có thể không có tham số, vì vậy khi gọi hàm đó không cần truyền tham số thực tế.

Phạm vi của biến trong chương trình

- Biến cục bộ: Là các biến khai báo *bên trong các hàm*, kể cả hàm main(). Kết thúc hàm, bộ nhớ của các biến này được giải phóng.
  - *Nếu biến cục bộ chứa đựng giá trị rất quan trọng, khi hàm được gọi kết thúc công việc, biến cục bộ được giải phóng, ta mất quyền kiểm soát biến cục bộ. Vậy làm sao lấy giá trị quan trọng đó ra khỏi hàm ?*
- Biến toàn cục: Là các biến khai báo từ đầu chương trình, *bên ngoài các hàm* và không thuộc về hàm nào. Mục đích để sử dụng chung giữa các hàm.
  - *Biến toàn cục rất dễ bị các hàm thay đổi giá trị mà nó lưu trữ*
  - *Nên khai báo ít biến toàn cục để tránh lãng phí bộ nhớ.*
- Nếu tên của biến toàn cục và cục bộ trùng nhau thì sẽ ưu tiên cho biến cục bộ.


3 Gọi hàm


3. Gọi hàm

- Sau khi xây dựng xong một hàm, khi cần ta có thể gọi nó để sử dụng. Việc này người ta gọi là gọi hàm.

- Để gọi hàm cần gõ tên hàm và truyền các dữ liệu thật sự (*tham số thực*) vào cho các đối tương ứng (*tham số hình thức*) của hàm.

- Việc truyền dữ liệu cho hàm khi gọi nó thường được biết với tên gọi là *truyền tham số*, hay *truyền đối* cho hàm.

- Khi gọi hàm cần lưu ý một số điểm sau đây:
  - Số lượng *tham số thực* phải bằng *tham số hình thức*
  - Kiểu của tham số thực phải cùng kiểu với tham số hình thức.

3.1. Mô hình hoạt động của hàm

```
KiểuDL   Tênhàm (tham số hình thức)
      {
          Các biến cục bộ;
          các lệnh thực hiện công việc;
          return (expr);
      }

int  main()
      {
          …
          Tênhàm(tham số thực);
          …
          …
      }
```

Step 2

Step 1

- **Kiểu dữ liệu** (KiểuDL): Kiểu giá trị trả về cho *tên hàm*.
  VD: *int, float, double, void*
- **Tên hàm**: theo quy ước đặt tên biến
- **Tham số**: danh sách các giá trị đưa vào và đưa ra của hàm
- **Các biến cục bộ**: là các biến được khai báo và chỉ sử dụng trong hàm con
- Gọi hàm trong main() bằng tên hàm và tham số thực

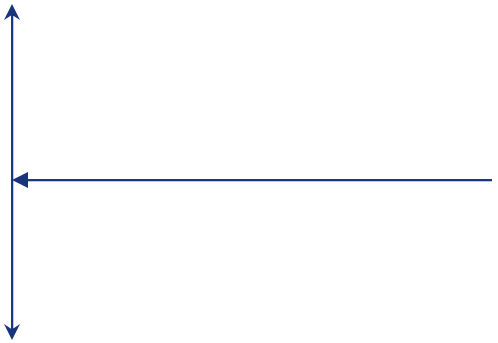
3.2. Hoạt động của hàm khi nó được gọi

- Khi hàm được gọi để thực hiện công việc:
  - Cấp phát bộ nhớ cho các biến cục bộ trong hàm
  - Gán dữ liệu (tham số thực) cho các tham số hình thức.
  - Thực hiện các lệnh trong thân hàm
  - Khi gặp lệnh return hoặc dấu } cuối cùng của thân hàm, bộ nhớ dành cho các biến cục bộ được giải phóng và thoát khỏi hàm được gọi.
  - Nếu câu lệnh return có chứa biểu thức thì giá trị của biểu thức đó sẽ được gán cho tên hàm

Ví dụ minh họa việc *lập hàm* và *gọi hàm*

```
// Lập hàm tính trung bình cộng của ba số nguyên
#include<stdio.h>
float   tbc(int x, int y, int z)
        {
            float  gt;
            gt = (x + y + z)/3.0;
            return(gt);
         }

void    main()
        {
            int   a, b,c;
            printf("Nhap ba so nguyen a, b, c \n");
            scanf("%d%d%d",&a,&b,&c);
            printf("Trung binh cong  3 so la: %f ", tbc(c,b,a));
        }
```



Lập hàm

Gọi hàm

Ví dụ về tham số

```
/*-----------------begin-----------------*/

#include<stdio.h>
float   tbc(int x, int y, int z)
    {
       float  gt;
       gt = (x + y + z)/3.0;
       return(gt);
    }

void main()
{
    int   a, b,c;
    float   tb3so;
    printf("Nhap ba so nguyen a, b, c \n");
    scanf("%d%d%d",&a,&b,&c);
    tb3so  = tbc(a,b,c);
    printf("Tb 3 so la: %f ", tb3so);
    //printf("Tb 3 so la: %f ", tbc(a,b,c));
}
/*-----------------end-----------------*/
```

Tham số hình thức

Tham số thực tế

Truyền tham số khi gọi hàm

- Truyền *giá trị (pass by value)*
  - Khi gọi hàm, giá trị của tham số thực tế sẽ được *gán* cho tham số hình thức (biến đại diện cho đại lượng cần đưa vào) tương ứng.
  - Lúc xây dựng hàm, biến đại diện cho đầu vào được khai báo ở dạng *biến thường*.
- Truyền *địa chỉ (pass by address)*
  - Khi gọi hàm, địa chỉ của tham số thực tế sẽ được *gán* cho tham số hình thức (biến đại diện cho đại lượng cần đưa ra khỏi hàm) tương ứng.
  - Lúc xây dựng hàm, biến đại diện cho đầu ra được khai báo ở *dạng con trỏ*.

Bản chất của truyền tham số khi gọi hàm

- Truyền giá trị: Khi gọi hàm, giá trị của *biến bên ngoài* hàm được sao chép (gán) cho *biến đại diện đầu vào* tương ứng của hàm. Trường hợp này, biến bên ngoài hàm và bên trong hàm *khác nhau* về địa chỉ, mọi sự thay đổi của giá trị copy được tiến hành cục bộ bên trong hàm sẽ không làm thay đổi dữ liệu gốc của biến bên ngoài hàm.

- Truyền địa chỉ: Khi gọi hàm, địa chỉ của *biến bên ngoài hàm* được truyền (gán) vào cho *biến đại diện đầu ra* (con trỏ) tương ứng của hàm [Con trỏ đó trỏ vào vùng nhớ của biến bên ngoài hàm]. Trường hợp này, biến bên trong hàm và biến bên ngoài hàm có *cùng* địa chỉ. Mọi sự thay đổi bên trong hàm đối với dữ liệu trên vùng địa chỉ được đưa vào sẽ làm thay đổi dữ liệu gốc của biến bên ngoài hàm [Vì cùng làm việc trên cùng một vùng nhớ]. Khi biến bên trong hàm được giải phóng, dữ liệu không bị mất đi, do vẫn còn được kiểm soát thông qua biến bên ngoài hàm.

Quy tắc truyền tham số cho hàm

- Khi xây dựng hàm:
  - Các biến đại diện cho *đầu vào*: biến thường
  - Các biến đại diện cho *đầu ra*: con trỏ
- Khi gọi hàm:
  - Biến *đầu vào* của hàm được truyền (gán): giá trị
  - Biến *đầu ra* của hàm được truyền (gán): địa chỉ
- Đối với trường hợp *đầu vào* cũng là *đầu ra*, biến đại diện sẽ là con trỏ. Xem ví dụ 4, 5

Ví dụ 2: truyền giá trị, địa chỉ

Lập hàm tính *tổng, tích* của hai số thực. Gọi hàm này trong chương trình chính để tính cho 2 số thực bất kì được nhập vào từ bàn phím.

- Có 2 kết quả đơn lẻ (*tổng, tích*) cần lấy ra khỏi hàm sau khi tính toán. Trường hợp này, *tên hàm* không thể chứa cùng lúc 2 giá trị đó. Do vậy, ta sẽ dùng 2 con trỏ để lấy *tổng*, *tích* ra khỏi hàm.
- Hàm này sẽ có 4 tham số.
  - Hai *biến thường* đại diện cho *hai số thực* cần đưa vào hàm
  - Hai *con trỏ* đại diện cho *tổng, tích* cần lấy ra khỏi hàm

Chương trình minh họa cho ví dụ 2

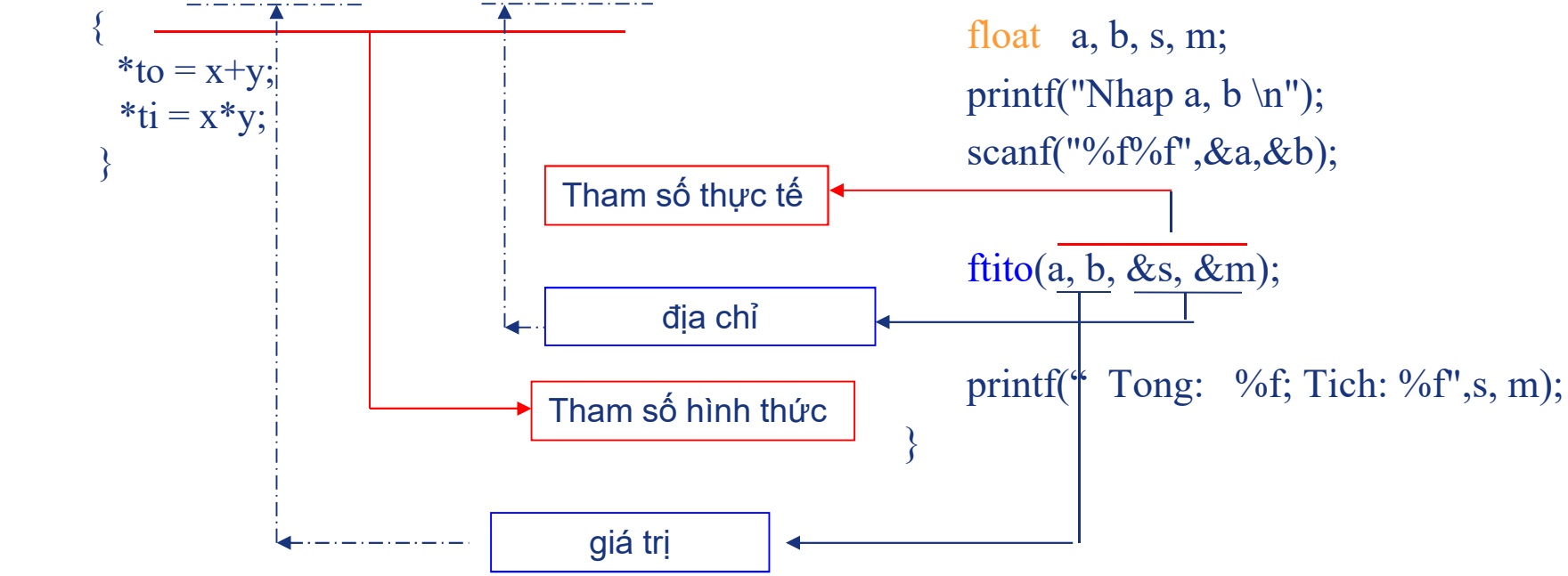
```
#include<stdio.h>

void  ftito(float x, float y, float *to, float *ti)
      {
        *to = x+y;
        *ti = x*y;
      }
```

```
void  main()
{
     float   a, b, s, m;
     printf("Nhap a, b \n");
     scanf("%f%f",&a,&b);

     ftito(a, b, &s, &m);

     printf(" Tong:   %f; Tich: %f",s, m);
}
```



Ví dụ: Lập hàm kiểm tra một số có phải số nguyên tố

```
# include <stdio.h>
int n;        //biến toàn cục

int snt(int so)
{
    int i, kt = 1;     //biến cục bộ

    for(i = 2; i <= sqrt(so); i++)
       if(so % i == 0)
       {
            kt = 0;
            break;
       }
    return (kt);
}
```

```
main()
{
     printf("Nhap vao so can kiem tra:\n ");
     scanf("%d", &n);

     if (snt(n) = =1)
          printf("Day la so nguyen to\n");
     else
          printf("Khong la so nguyen to\n");

}
```

Ví dụ 4: Đầu vào cũng là đầu ra

Lập hàm con để hoán vị 2 số nguyên bất kì. Từ chương trình chính, nhập vào 2 số nguyên từ bàn phím, gọi hàm con đã lập vào để hoán đổi giá trị cho 2 số này.

- Bài này cần 2 biến đại diện cho 2 giá trị cần đưa vào hàm. Sau khi đưa vào hàm, 2 biến này đổi giá trị cho nhau.
- Làm thế nào để khi hàm con thực hiện đổi giá trị bên trong nó, dữ liệu gốc bên ngoài cũng thay đổi theo. Để làm được việc này, biến mô hình đại diện cho đầu vào của hàm sẽ quản lý chung vùng nhớ với biến ngoài hàm chứa dữ liệu gốc. Như vậy ta cần con trỏ đại diện cho đầu vào.
- Bài này tham số của hàm là 2 con trỏ.

Ví dụ 4: Đầu vào cũng là đầu ra (tiếp)

```
#include<stdio.h>
void   hoanvi(int  *x, int  *y)
{
      int tg;
      tg = *x;
      *x = *y;
      *y = tg;
}
```

Bài này tên hàm không được gán giá trị. Do đó, hàm có kiểu void

```
main()
{
      int   a, b;
      printf("Nhap a:  ");
      scanf("%d", &a);
      printf("Nhap b:  ");
      scanf("%d", &b);
      hoanvi(&a, &b);
      printf("a la: %d \n", a);
      printf("b la: %d \n", b);
}
```

Ví dụ 5: Lập hàm *sắp xếp* dãy tăng dần (interchange sort)

```
# include <stdio.h>
# include <stdlib.h>
void    sapxep(float *x, int  n)
{
    float   tg;
    for(i = 0; i < n -1; i++)
      for(j = i + 1; j < n ; j++)
       if ( *(x + i) > *(x + j) )
       {
          tg = *(x + i);
          *(x+i) = *(x+j);
          *(x+j) = tg;
       }
}
```

Hàm sapxep có kiểu void do nó không được gán giá trị.

```
int    main()
{
    float *a;
    int i, n;
    printf("Nhap n: ");
    scanf("%d", &n);
    a = (float *)malloc(n*sizeof(float));
    if(a = = NULL) exit(0);
    printf("Nhap mang:\n");
    for(i = 0; i < n; i++)
            scanf("%f", a + i );
    sapxep(a, n);
    puts("Day sau khi da sap xep la:");
    for(i = 0; i < n; i++)
            printf("%f\t", *(a + i) );
    free(a);
    return 0;
}
```

Ví dụ 6: Lập hàm *sắp xếp* dãy tăng dần (interchange sort)

```
# include <stdio.h>
# include <stdlib.h>
int n, i, j; //cac bien toan cuc

float   *nhapday(int *n)
{
    float *b;
    printf("\n Nhap vao so phan tu: ");
    scanf("%d", n);
    b = (float*) calloc(*n, sizeof(float));
    for(i = 0; i < *n; i++) {
        printf("\n Nhap b[%d] = ",i);
        scanf("%f", b + i);
     }
    return b;
}
```

```
void    sapxep(float *a, int n)
{
    float tg;
    for(i = 0; i < n -1; i++)
      for(j = i + 1; j < n ; j++)
        if (a[i] > a[j])
        {
          tg = a[i];
          a[i] = a[j];
          a[j] = tg;
        }
}
```

Ví dụ 6: Sắp xếp dãy tăng dần (tiếp)

```
int    main()
       {
              float *a;
              a = nhapday(&n);
              sapxep(a, n);
              puts("Day sau khi da sap xep la:");
              for(i = 0; i < n; i++)
                     printf("%f\t",a[i]);
              free(a);
              return 0;
       }
```


Nội dung bài học

- Hàm và chương trình con
- Xây dựng hàm
  - Tham số của hàm
  - Phạm vi của biến trong chương trình
- Gọi hàm
- Các ví dụ minh họa


Bài tập

1. Viết hàm tính n!
2. Viết hàm tính tổng $S = 1+2^n+\ldots.+m^n$
3. Nhập hai dãy kích thước N từ bàn phím. Trộn hai dãy này thành một dãy tăng. Chia các công đoạn thành các hàm con gồm: nhập, trộn, sắp xếp thành dãy tăng.
4. Viết các hàm nhập, in và tính tổng hai ma trận cỡ n x n.