

HÀM (Function)

GV BIÊN SOẠN: PHẠM NGUYỄN TRƯỜNG AN

3. Nội dung



Đặt vấn đề

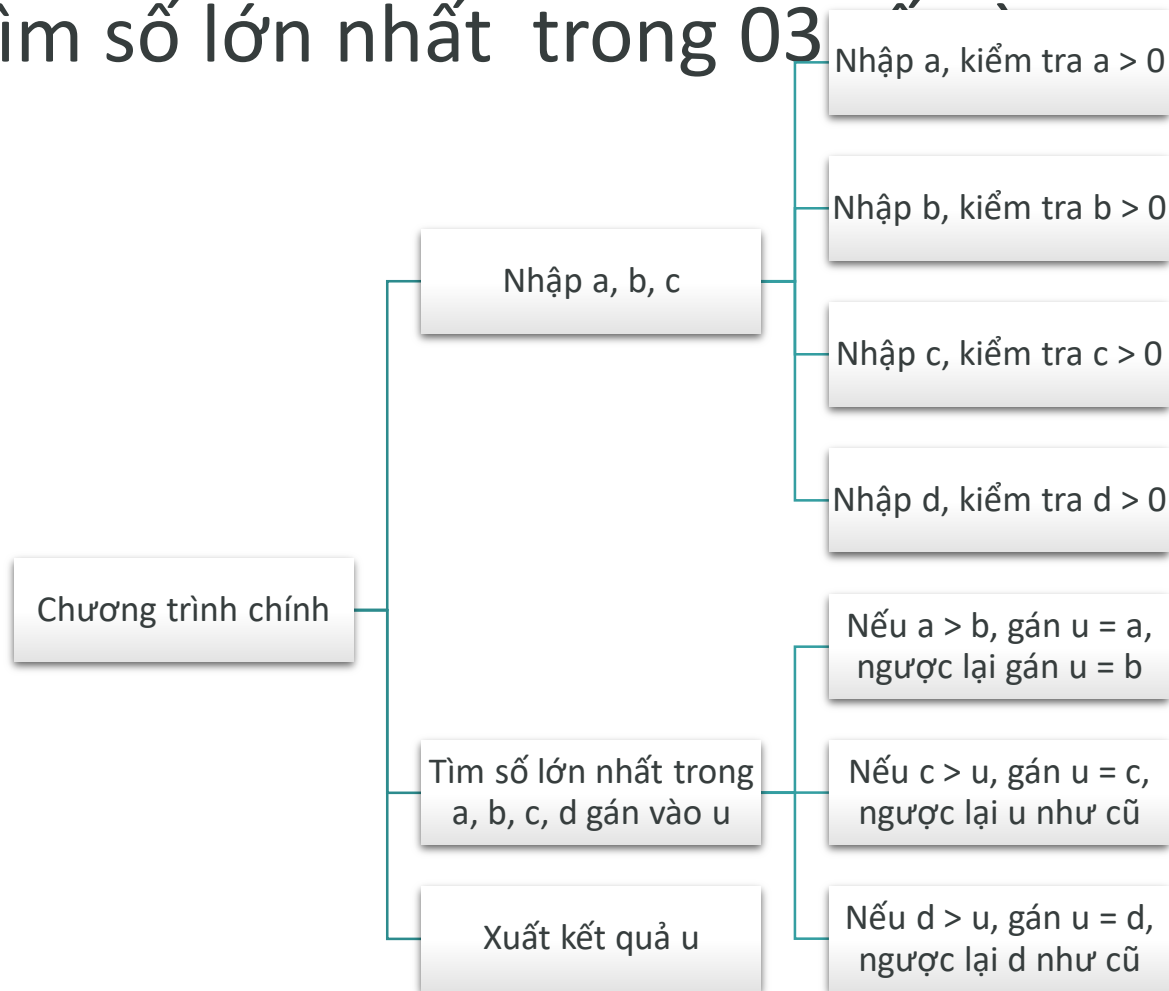
Khái niệm và cú pháp

Tham số và lời gọi hàm

Khai báo và tầm vực



- Nhập 04 số nguyên dương a, b, c, d .
Tìm số lớn nhất trong 03 số






- 4 đoạn lệnh nhập a, b, c, d

```
int a, b, c, d;

do {
    cout << "Nhap mot so nguyen duong";
    cin >> a;
} while (a <= 0);
do {
    cout << "Nhap mot so nguyen duong";
    cin >> b;
} while (b <= 0);
do {
    cout << "Nhap mot so nguyen duong";
    cin >> c;
} while (c <= 0);
do {
    cout << "Nhap mot so nguyen duong";
    cin >> d;
} while (d <= 0);
```



- Hai đoạn code tính u



```
int u;  
if (a > b) u = a;  
else u = b;  
  
if (c > u) u = c;  
  
if (d > u) u = d;
```

- Đoạn lệnh nhập và kiểm tra một số lớn hơn 0 **lặp lại 04 lần**.
- Đoạn lệnh tính u có **03 lệnh if tương tự** nhau lặp lại.
- Cần giải pháp **viết 01 lần** và nhưng có thể **dùng nhiều lần**



- Một đoạn chương trình có tên, đầu vào và đầu ra.
- Có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính.
- Có thể được gọi nhiều lần với các đối số khác nhau.
- Được sử dụng khi có nhu cầu:
 - Tái sử dụng.
 - Sửa lỗi và cải tiến.



- “chương trình con” - ***Subroutine*** - là thuật ngữ được đề xuất sớm (1951, 1952) và chuyên biệt cho khái niệm này¹
- Một số thuật ngữ khác: ***Subprogram, procedure, method, routine, function***
- Một số sách dùng thuật ngữ tổng quát: ***callable unit***
- C/C++ dùng thuật ngữ ***hàm - function***.

1 Wheeler, D. J. (1952). "The use of sub-routines in programmes". Proceedings of the 1952 ACM national meeting (Pittsburgh) on - ACM '52. p. 235.



```
kiểu_trả_về tên_hàm([danh sách tham số]){  
    <các câu lệnh>  
    return <giá_trị_trả_về>;  
}
```

kiểu_trả_về

- Return type
- Bất kỳ kiểu nào của C/C++. Nếu hàm không trả về thì kiểu là **void**

tên_hàm

- Function name
- Như quy tắc đặt tên biến

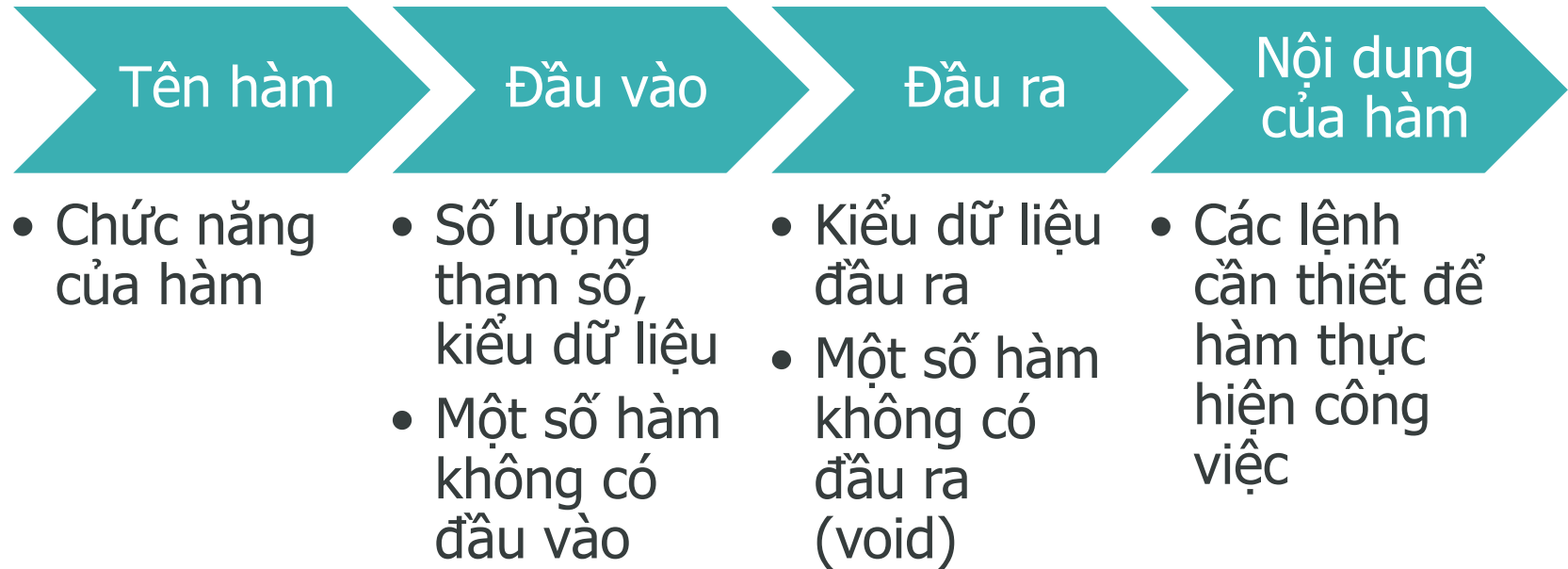
[danh sách tham số]

- Parameter list
- Giống như khi khai báo biến trên một dòng, cách nhau bằng dấu ,

<giá_trị_trả_về>

- Return value
- Là kết quả đầu ra của hàm, phải cùng kiểu với kiểu_trả_về. Dùng được bất cứ cú pháp nào có thể tính thành giá trị
- Từ khóa return sẽ kết thúc quá trình thực thi của hàm.

Các bước viết hàm





- Hàm có đầu ra, không có đầu vào:
 - *Tên hàm*: `nhap_so_duong` , Hàm yêu cầu người dùng nhập vào một số nguyên dương. Nếu không phải số dương yêu cầu nhập lại.
 - Đầu vào: Không có
 - Đầu ra: số nguyên dương.

```
int nhap_so_duong(){  
    int n;  
    do {  
        cout << "Nhap mot so nguyen duong";  
        cin >> n;  
    } while (n <= 0);  
    return n;  
}
```



- Hàm có đầu vào, không có đầu ra:
 - *Tên hàm*: `xuat_so_lon`, Xuất ra màn hình số lớn hơn trong 02 số.
 - Đầu vào: Hai số nguyên. Đặt tên là `a` và `b`
 - Đầu ra: Không có

```
void xuat_so_lon(int a, int b){  
    int m;  
    if (a > b) m = a;  
    else m = b;  
}  
cout << "so lon nhat giua "  
      << a << " va " << b << " la " << m;  
}
```



- Hàm không có đầu vào lẫn đầu ra
 - *Tên hàm*: `nhap_xuat_so_lon`, Yêu cầu nhập vào 02 số nguyên và xuất ra màn hình ước chung lớn nhất của 02 số đó.
 - Đầu vào: Không có
 - Đầu ra: Không có

```
void nhap_xuat_so_lon(){
    int m, n;
    cout << "Nhap so nguyen duong"; cin >> m;
    cout << "Nhap so nguyen duong"; cin >> n;
    cout << "So lon hon trong "
           << m << " va " << n << " la ";
    if (n > m) m = n;
    cout << m;
}
```



- Hàm có cả đầu vào và đầu ra
 - *Tên hàm*: so_lon, Nhận vào 02 số nguyên dương và trả về số lớn hơn trong 02 số đó.
 - Đầu vào: Hai số nguyên dương, đặt tên m và n
 - Đầu ra: Số nguyên dương có giá trị lớn hơn trong m và n

```
int so_lon(int m, int n){  
    if (n > m) m = n;  
    return m;  
}
```



- Lệnh return dùng để trả về giá trị đầu ra của hàm
- Hàm chỉ trả về được **duy nhất 01 giá trị**. Lệnh return sẽ kết thúc quá trình thực thi của hàm

```
int so_lon(int m, int n){  
    if (n > m) return n;  
    return m;  
}
```



- Các hàm không có đầu ra sẽ có kiểu trả về là void
- Không có biến kiểu void
- Lệnh return với các hàm không có đầu ra sẽ không kèm theo giá trị (nhưng vẫn sẽ kết thúc việc thực thi hàm)

```
void xuat_so_lon(int a, int b){  
    cout << "so lon nhat giua "  
        << a << " va " << b << " la " ;  
    if (a > b) {  
        cout << a;  
        return;  
    }  
    cout << b;  
}
```



- Gọi hàm – ***to call (a) function*** - là hành động yêu cầu hệ thống thực hiện các công việc của hàm
- Lời gọi hàm – ***function call*** – phải có tên hàm và danh sách các thông số sẽ được đưa vào cho hàm trong cặp *ngoặc đơn*
- Lời gọi hàm có thể tính ra giá trị, chính là giá trị trả về của hàm.
- Cú pháp:

↓ tên_hàm ([danh sách đối số])



```
1. int nhap_so_duong(){
2.     int n;
3.     do {
4.         cout << "Nhap mot so nguyen duong";
5.         cin >> n;
6.     } while (n <= 0);
7.     return n;
8. }
9. int main()
10. {
11.     int a = nhap_so_duong();
12.     cout << "So vua nhap la " << a << endl;
13.     cout << "Tong hai so la " << a + nhap_so_duong() << endl;
14.     return 0;
15. }
```

Output:

```
Nhap mot so nguyen duong 5
So vua nhap la 5
Nhap mot so nguyen duong 8
Tong hai so la 13
```

• Chú ý kỹ dòng 13



- **Parameter**

- tạm dịch: Tham số hoặc tham số hình thức
- Là các thông số mà **hàm nhận vào**
- Xác định khi khai báo hàm

- **Argument**

- Tạm dịch: Đối số hoặc Tham số thực sự
- Là các thông số được **đưa vào hàm** khi tiến hành gọi hàm
- Hai thuật ngữ này đôi khi dùng lẫn lộn và gọi chung là Tham số



- Truyền đối số - **to pass argument** – là công việc đưa các thông số cho hàm hoạt động khi gọi hàm.
- Đối số phải được truyền tương ứng với cách tham số đã được khai báo.
- Có 02 cách truyền đối số chính
 - **Pass by value** – Truyền giá trị (truyền tham trị)
 - **Pass by reference** – Truyền tham chiếu

Truyền giá trị



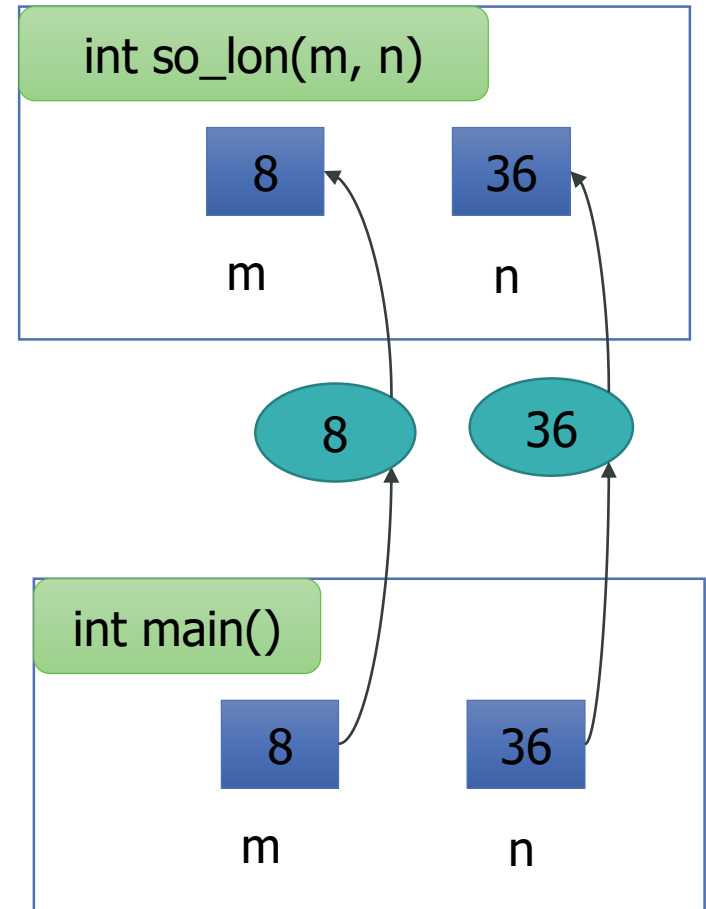
- Là cách mặc định của C/C++
- Tham số chứa bản sao giá trị của đối số. Thay đổi tham số không ảnh hưởng đến đối số.

```
int so_lon(int m, int n){  
    if (n > m) m = n;  
    return m;  
}  
int main()  
{  
    int m = 8, n = 36;  
    int o = so_lon(m, n);  
    cout << "UCLN của " << m << " va " << n << " la " << o;  
}
```



```
int so_lon(int m, int n){
    if (n > m) m = n;
    return m;
}
int main()
{
    int m = 8, n = 36;
    int o = so_lon(m, n);
    cout << "UCLN của " << m
          << " va " << n << " la " << o;
}
```

- Truyền giá trị tạo ra bản sao của đối số và lưu vào trong vùng nhớ của tham số





- Có thể truyền đối số là bất cứ cú pháp nào tính được thành giá trị (hằng, biến, biểu thức, lời gọi, v.v...)

```
int so_lon(int m, int n){
    if (m > n) n = m;
    return n;
}
int nhap_so_nguyen(){
    int n; cout << "Nhap mot so nguyen "; cin >> n;
    return n;
}
int main()
{
    cout << "So lon hon la " << so_lon(9*4, nhap_so_duong())
}
```



```
int so_lon(int m, int n){
    if (n > m) m = n;
    return m;
}
int nhap_so_duong(){
    int n;
    do {
        cout << "Nhap mot so nguyen duong";
        cin >> n;
    } while (n <= 0);
    return n;
}
int main()
{
    cout << "so lon nhat trong 04 so la "
        << so_lon(
            so_lon(nhap_so_duong(), nhap_so_duong())
            , so_lon(nhap_so_duong(), nhap_so_duong())
        );
}
```

- Giải quyết vấn đề đặt ra ở đầu bài.



- Áp dụng cho các tham số khi khai báo có dấu **&** phía sau kiểu dữ liệu.
- Chỉ có thể truyền các đối số là biến (hoặc hằng nếu tham số khai báo là *const*)
- Các tham số là tham chiếu không được cấp phát vùng nhớ
 - Tham số được truyền tham chiếu sẽ trở đến cùng địa chỉ vùng nhớ của đối số truyền cho nó
 - Tham số sẽ trở thành một ánh xạ đến đối số. Mọi thay đổi lên tham số sẽ thay đổi luôn đối số.

Truyền tham chiếu



```
void hoan_vi(int& a, int& b){  
    int c = a;  
    a = b;  
    b = c;  
}  
int main()  
{  
    int a, b;  
    cin >> a >> b;  
    hoan_vi(a, b);  
    cout << a << " " << b;  
}
```

Output:

```
5 3  
3 5
```

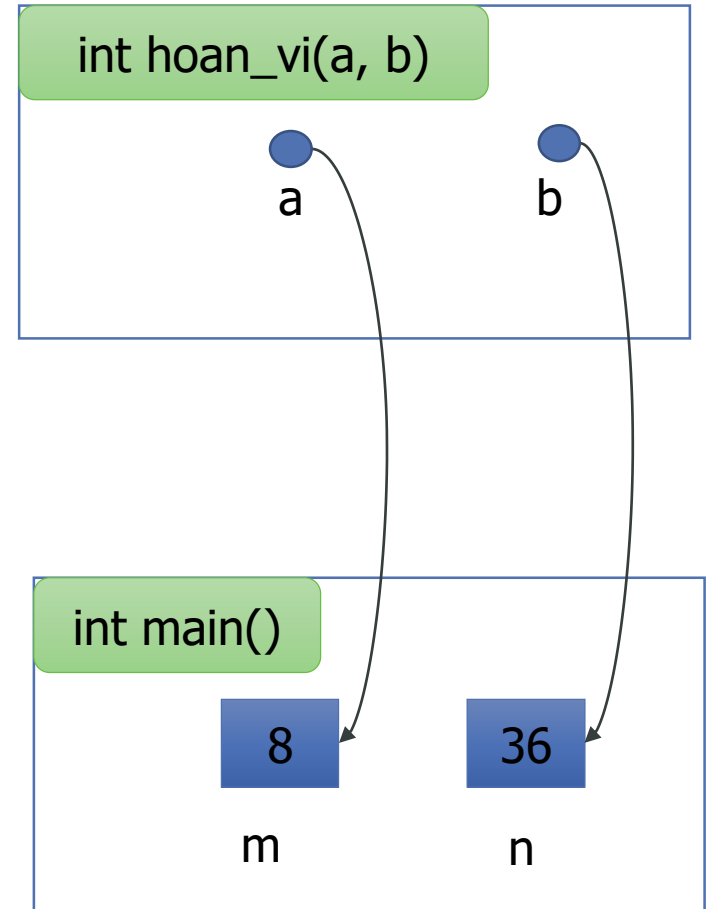
- Chương trình xuất ra hai số ngược với thứ tự chúng được nhập vào
- Đối số truyền vào bắt buộc phải là biến, không thể dùng hàm nhập_so_duong trong trường hợp này

Truyền tham chiếu



```
void hoan_vi(int& a, int& b){
    int c = a;
    a = b;
    b = c;
}
int main()
{
    int m, n;
    cin >> m >> n;
    hoan_vi(m, n);
    cout << m << " " << n;
}
```

- Truyền tham chiếu liên kết tham số đến vùng nhớ của đối số. Tham số không có vùng nhớ



Truyền tham chiếu



```
bool phep_chia(int x, int y, double& thuong){
    if (y != 0) {
        thuong = double(x)/y;
        return true;
    } else {
        return false;
    }
}

int main(){
    double thuong;
    if (phep_chia(5, 3, thuong)){
        cout << "Thuong so la " << thuong;
    } else {
        cout << "Khong the chia duoc ";
    }
}
```

- Dùng truyền tham chiếu như một cách trả về kết quả

Scope - Phạm vi của biến



- Bên cạnh tham số, hàm có thể tự khai báo thêm các biến.
- Mỗi biến có một phạm vi tác dụng nhất định gọi là **Scope** – tạm dịch: Tâm vực
- **Local variable** - biến cục bộ - được khai báo trong một khối ngoặc nhọn **{ }**. Biến chỉ có tác dụng trong khối ngoặc nhọn đó và sẽ bị xóa khỏi bộ nhớ khi chương trình chạy ra khỏi khối.
- **Global variable** – biến toàn cục – khai báo bên ngoài các cặp ngoặc nhọn. Có phạm vi toàn chương trình.
 - Biến toàn cục chỉ bị xóa khi chương trình kết thúc => tốn bộ nhớ nếu lạm dụng
 - Biến toàn cục có thể được/bị thay đổi bởi bất kỳ hàm nào => Dễ gây lỗi logic nếu dùng bất cẩn

Ví dụ




```
1. int a
2. int ham1(){
3.     int a1;
4.     ///Các biến có tác dụng: a, a1
5. }
6. int ham2(){
7.     int a2;
8.     ///Các biến có tác dụng: a, a2
9.     {
10.        int a21;
11.        ///Các biến có tác dụng: a, a2, a21
12.        int a;
13.        ///Biến a này sẽ thay thế cho biến a toàn cục
14.    }
15.    ///Các biến có tác dụng: a, a2
16.}
17.int main(){
18.    int a3;
19.    ///Các biến có tác dụng: a, a3
20.}
```

- Biến a khai báo tại dòng 13 sẽ có tác dụng trong phạm vi cặp ngoặc { } chứa nó thay cho biến toàn cục ở dòng 1
- Khi dùng hàm nên **hạn chế dùng biến toàn cục**

Prototype – nguyên mẫu hàm




• Xét ví dụ:



```
int ham(int tham_so1, double tham_so2){  
    Cau_lenh;  
    return 0;  
}
```

- Function declaration – khai báo hàm – phần màu vàng
- Function's body – Thân hàm – Phần màu xám
- Khai báo hàm đi kèm với thân hàm tạo thành định nghĩa hàm – Function definition
- Khai báo hàm nhưng không ghi tên tham số, chỉ ghi kiểu dữ liệu tham số được gọi là nguyên mẫu hàm – function prototype:



```
int ham(int, double)
```



```
void le (int x);
void chan (int);
int main(){
    int i;
    do
    {
        cout << "Nhap 1 so (Nhap 0 de thoat): ";
        cin >> i;
        le (i);
    }
    while (i!=0);
    return 0;
}
void le (int x){
    if ((x%2)!=0) cout << "So le.\n";
    else chan (x);
}
void chan (int x){
    if ((x%2)==0) cout << "So chan.\n";
    else le (x);
}
```

- Hàm cần phải được khai báo trước khi gọi
- Có thể khai báo hàm trước và định nghĩa hàm sau
- Khi khai báo hàm có thể dùng prototype thay cho lời khai báo (declaration), nếu dùng lời khai báo thì tên tham số phải khớp với khi định nghĩa

Hàm trả về tham chiếu

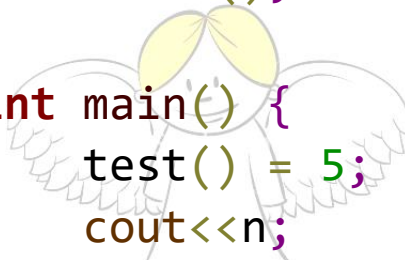



• Đoạn code nào đúng?

```
#include <iostream>
using namespace std;
int n;
int& test();

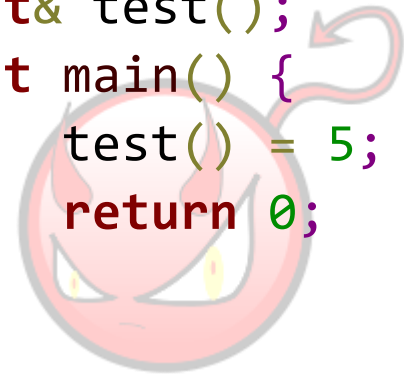

int main() {
    test() = 5;
    cout<<n;
    return 0;
}

int& test() {
    return n;
}
```



```
#include <iostream>
using namespace std;
int& test();
int main() {
    test() = 5;
    return 0;
}

int& test() {
    int n = 5;
    return n;
}
```





- Tìm ước số chung lớn nhất

```
int &so_lon(int &a, int &b){  
    if(a > b) return a;  
    return b;  
}  
int &so_be(int &a, int &b){  
    if(a < b) return a;  
    return b;  
}  
int main(){  
    int a, b;  
    cin >> a >> b;  
    while (a != b) so_lon(a,b) -= so_be(a, b);  
    cout << a  
}
```



1. Làm lại các bài tập chương câu lệnh điều kiện và rẽ nhánh dưới dạng hàm:
 - a) Viết hàm đổi một ký tự hoa sang ký tự thường.
 - b) Viết hàm giải phương trình bậc nhất và xuất kết quả ra màn hình
 - c) Viết hàm giải phương trình bậc hai và xuất kết quả ra màn hình
 - d) Viết hàm trả về giá trị nhỏ nhất của 4 số nguyên.
 - e) Viết hàm hoán vị hai số nguyên.
 - f) Viết hàm sắp xếp 4 số nguyên tăng dần.



1. Làm lại các bài tập chương câu lệnh lặp:
2. Viết hàm nhận vào số nguyên dương n và thực hiện:
 - a) Đếm số lượng chữ số của số đó
 - b) Tính tổng các chữ số của số đó
 - c) Tính tổng các chữ số lẻ.
 - d) Tính tổng các chữ số chẵn của số đó.
 - e) Tìm số đảo của số n