



IT001 - NHẬP MÔN LẬP TRÌNH

CHƯƠNG 5: HÀM

Hàm số (hay hàm, function) đóng một vai trò rất quan trọng trong lập trình, nhất là lập trình theo hướng thủ tục, chúng vừa giúp chương trình chính tinh gọn và dễ hiểu hơn, vừa tiết kiệm chi phí cài đặt và bảo trì. Chương này sẽ trình bày về hàm số, cách xây dựng và sử dụng hàm trong lập trình.

Khoa Khoa học Máy tính



NỘI DUNG

5.1 Đặt vấn đề

5.2 Khái niệm hàm và lợi ích của việc sử dụng hàm

5.3 Định nghĩa hàm

5.4 Khai báo hàm, nguyên mẫu hàm

5.5 Khai báo hàm trùng tên

5.6 Lời gọi hàm

5.7 Phạm vi hoạt động của biến trong hàm

5.8 Tham số, đối số của hàm

5.9 Giá trị trả về của hàm

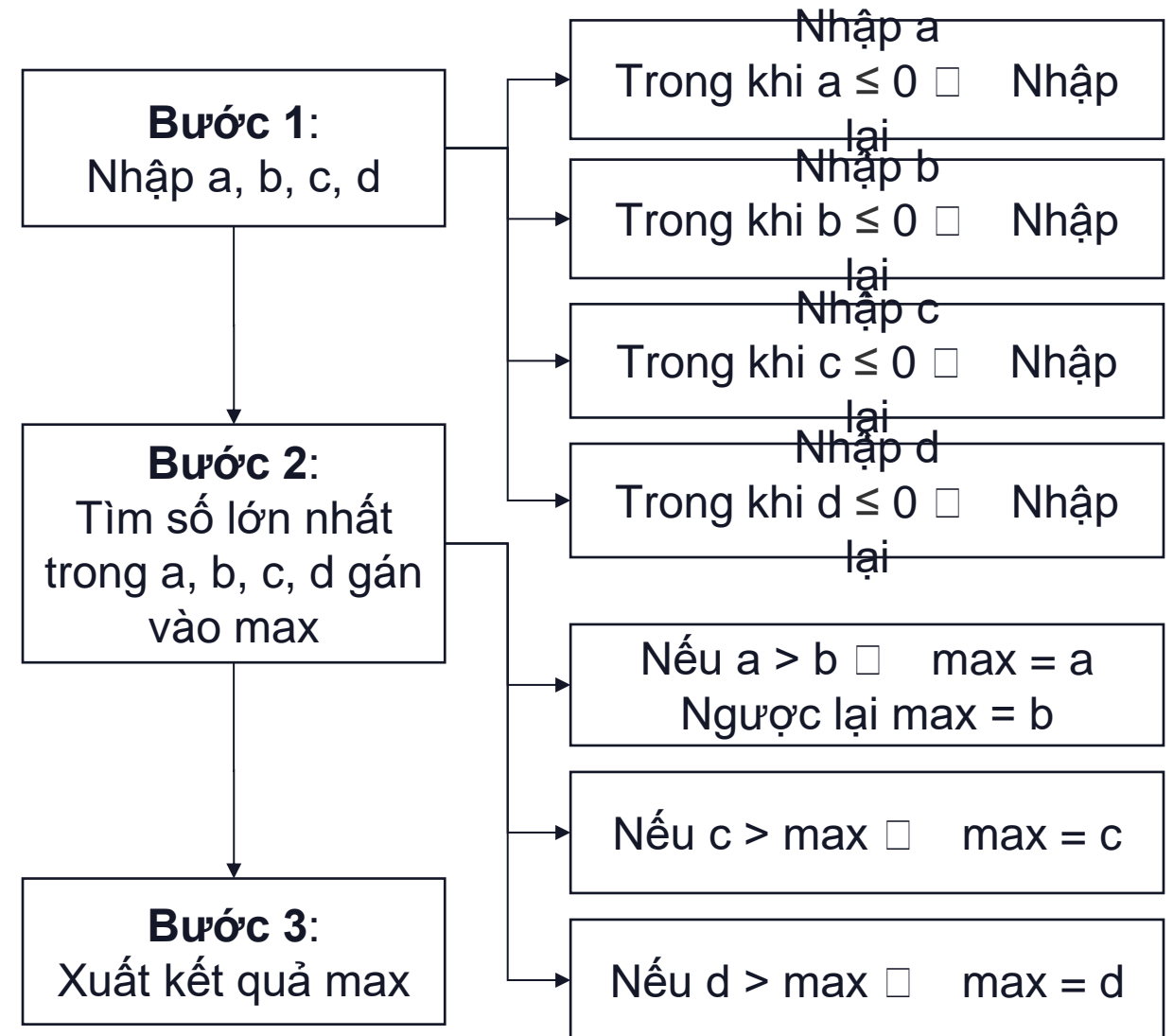
Bài tập



5.1 Đặt vấn đề

5.1 Đặt vấn đề

- Ví dụ: Viết chương trình nhập vào 04 số nguyên dương a, b, c, d . Tìm số lớn nhất trong 04 số này.
- **Input:** Nhập vào 4 số nguyên a, b, c, d (Có kiểm tra điều kiện nhập đảm bảo đúng)
- **Output:** Xuất số lớn nhất trong 4 số
- Sơ đồ các bước giải như sau:





5.1 Đặt vấn đề

- Ta có 4 đoạn lệnh **do-while** nhập a, b, c, d như sau:

```
int a, b, c, d;  
do {  
    cout << "Nhập một số nguyên dương: "; cin >> a;  
} while (a <= 0);  
do {  
    cout << "Nhập một số nguyên dương: "; cin >> b;  
} while (b <= 0);  
do {  
    cout << "Nhập một số nguyên dương: "; cin >> c;  
} while (c <= 0);  
do {  
    cout << "Nhập một số nguyên dương: "; cin >> d;  
} while (d <= 0);
```



5.1 Đặt vấn đề

- Đoạn mã tính giá trị lớn nhất max:

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

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



5.2 Khái niệm hàm và lợi ích của việc sử dụng hàm



5.2 Khái niệm hàm và lợi ích của việc sử dụng hàm

- **Khái niệm hàm:**

- Hàm trong C++ là một khối lệnh có thể tái sử dụng, được thiết kế để thực hiện một chức năng cụ thể trong chương trình. Hàm có thể nhận đầu vào (tham số) và trả về kết quả (giá trị trả về).

(Lưu ý: C++ không cho phép xây dựng một hàm bên trong một hàm khác)

- **Lợi ích của việc dùng hàm:**

- Tái sử dụng: Có thể được gọi nhiều lần với các đối số khác nhau.
- Tổ chức mã nguồn: quản lý và duy trì mã nguồn trở nên dễ dàng hơn.
- Tăng tính chuyên môn hóa: tập trung vào các chức năng cụ thể



5.3 Định nghĩa hàm (Function definition)



5.3 Định nghĩa hàm (Function definition)

- Cú pháp:

```
<kiểu_trả_về> <tên_hàm> (<[danh_sách_tham_số]>){  
    <Các_câu_lệnh>  
    return <giá_trị_trả_về>;  
}
```
- Trong đó:
 - <kiểu_trả_về>: Kiểu dữ liệu C/C++ (không trả về thì kiểu là **void**)
 - <tên_hàm>: Như quy tắc đặt tên định danh
 - <[danh_sách_tham_số]>: Như khi khai báo các biến trên một dòng, cách nhau bằng dấu `,`
 - <giá_trị_trả_về>: Kết quả trả về của hàm (phải cùng kiểu với <kiểu_trả_về>)



5.3 Định nghĩa hàm (Function definition)

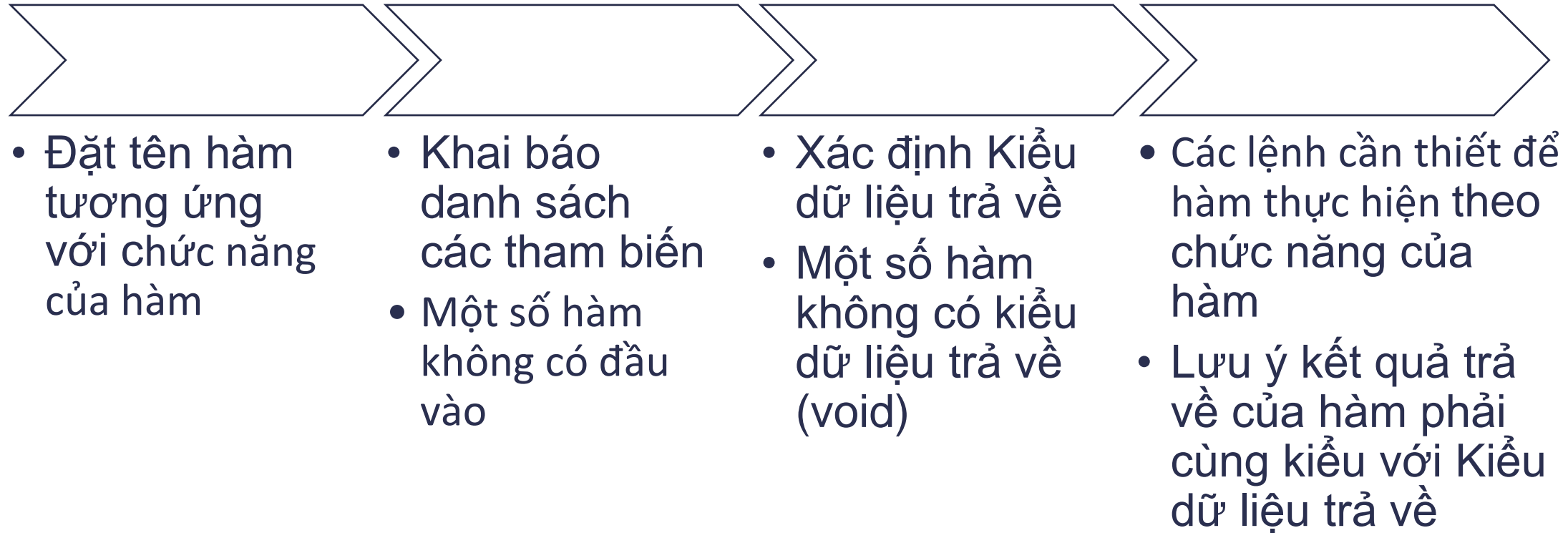
- Ví dụ hàm:

```
int so_lon(int a, int b) {  
    int max = (a > b ? a : b);  
    return max;  
}
```

```
bool kiemTraSNT(int n) {  
    if (n < 2) return 0;  
  
    for (int i = 2; i <= sqrt(double(n)); i++)  
        if (n%i == 0) return 0;  
  
    return 1;  
}
```



Các bước viết hàm





5.4 Khai báo hàm, nguyên mẫu hàm



5.4 Khai báo hàm, nguyên mẫu hàm

- Khai báo hàm: Một **khai báo** hàm thông báo cho trình biên dịch về tên hàm và cách gọi của hàm. Phần thân hàm có thể định nghĩa sau.
 - Cú pháp: `<kiểu_trả_về> <tên_hàm> (<[danh sách tham số]>);`
 - Ví dụ: `int so_lon(int a, int b);`
- Nguyên mẫu hàm (**Function prototype**): Là khai báo hàm nhưng không ghi tên tham số, chỉ ghi kiểu dữ liệu tham số.

Ví dụ: `int so_lon(int , int);`

Ví dụ

- 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.

```
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);
}
```



5.5 Khai báo hàm trùng tên (function overloading)



5.5 Khai báo hàm trùng tên (function overloading)

- Nạp chồng hàm (function overloading) trong C++ là một tính năng cho phép định nghĩa nhiều **hàm có cùng tên** nếu thỏa 2 điều kiện sau:
 - **Số lượng các tham số** trong hàm là khác nhau
 - Hoặc: **Kiểu của tham số** trong hàm là khác nhau
- Ví dụ: Các hàm sau đây hợp lệ khi cùng khai báo trong 1 chương trình.

```
int  maximun(int  a, int  b) { return (a > b) ? a: b ; }
float maximun(float a, float b) { return (a > b) ? a: b ; }
char  maximun(char  a, char  b) { return (a > b) ? a: b ; }
long  maximun(long  a, long  b) { return (a > b) ? a: b ; }
double maximun(double a, double b) { return (a > b) ? a: b ; }
void draw(Diem A, Diem B) ;
void draw(Diem A, Diem B, Diem C) ;
void draw(Diem A, Diem B, Diem C, Diem D) ;
```




5.6 Lời gọi hàm



5.6 Lời gọi hàm

- 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> (<Đối_số_1>, <Đối_số_2>, <Đối_số_3>, ...);



Ví dụ

- Ví dụ 1: Gọi hàm có đối số

```
1. #include <iostream>
2. using namespace std;
3. int so_nho(int, int);
4. int so_lon(int m, int n) {
5.     return m > n ? m : n;
6. }
7. int main() {
8.     int a = 5, b = 10;
9.     cout << so_lon(a, b) << endl;
10.    cout << so_nho(a, b);
11.    return 0;
12. }
13. int so_nho(int m, int n) {
14.     return n > m ? m : n;
15. }
```

- Ví dụ 2: Gọi hàm không đối số

```
1. #include <iostream>
2. void sayHello();
3. int main() {
4.     sayHello();
5.     return 0;
6. }
7. void sayHello() {
8.     std::cout << "Hello world!";
9. }
```




5.7 Phạm vi hoạt động của biến trong hàm



5.7 Phạm vi hoạt động của biến trong hàm

- Trong C++, các biến có phạm vi hoạt động khác nhau **tùy thuộc vào vị trí và cách biến được khai báo**. Dưới đây là các loại phạm vi biến trong hàm:
 - Biến cục bộ (Local variable)
 - Biến tĩnh cục bộ (Static local variable)
 - Biến toàn cục (Global variable)
 - Tham số (Sẽ trình bày ở các mục sau)



Biến cục bộ (Local variable)

- Biến cục bộ (**Local variable**) 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.
- Ví dụ:

```
#include <iostream>
void func() {
    int a=10; // Biến cục bộ
    a++;
    std::cout << a << std::endl;
}
int main() {
    int a=5;
    func();
    func();
    return 0;
}
```

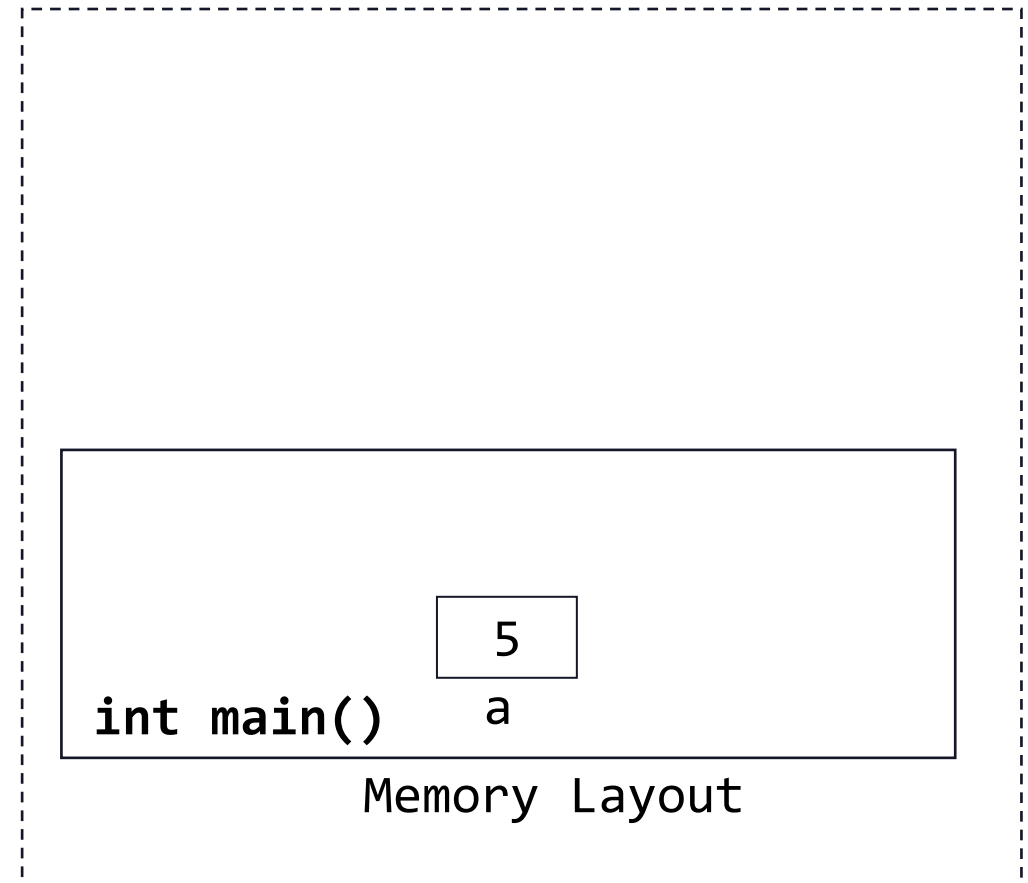


Biến cục bộ (Local variable)

- Biến cục bộ (**Local variable**) 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.

- Ví dụ:

```
#include <iostream>
void func() {
    int a=10; // Biến cục bộ
    a++;
    std::cout << a << std::endl;
}
int main() {
    int a=5;
    func();
    func();
    return 0;
}
```



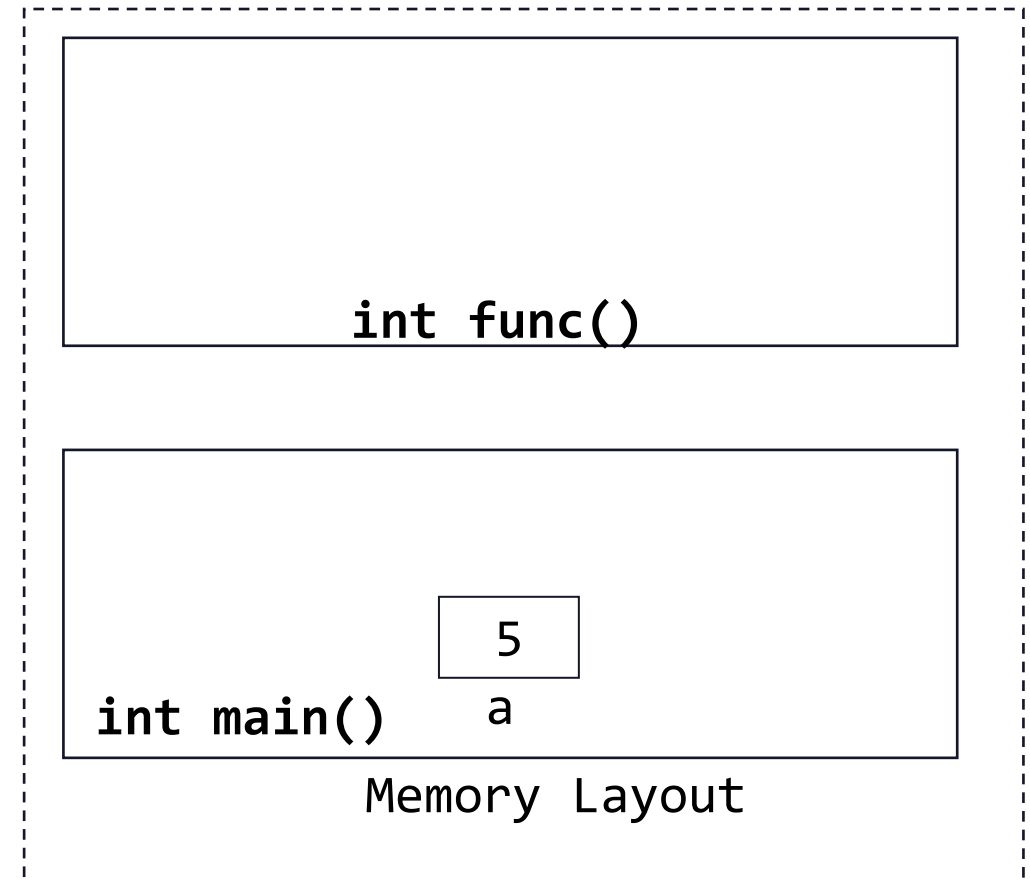


Biến cục bộ (Local variable)

- Biến cục bộ (**Local variable**) 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.

- Ví dụ:

```
#include <iostream>
void func() {
    int a=10; // Biến cục bộ
    a++;
    std::cout << a << std::endl;
}
int main() {
    int a=5;
    func();
    func();
    return 0;
}
```



Biến cục bộ (Local variable)

- Biến cục bộ (**Local variable**) 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.

- Ví dụ:

```
#include <iostream>
```

```
void func() {
```

```
    int a=10; // Biến cục bộ
```

```
    a++;
```

```
    std::cout << a << std::endl;
```

```
}
```

```
int main() {
```

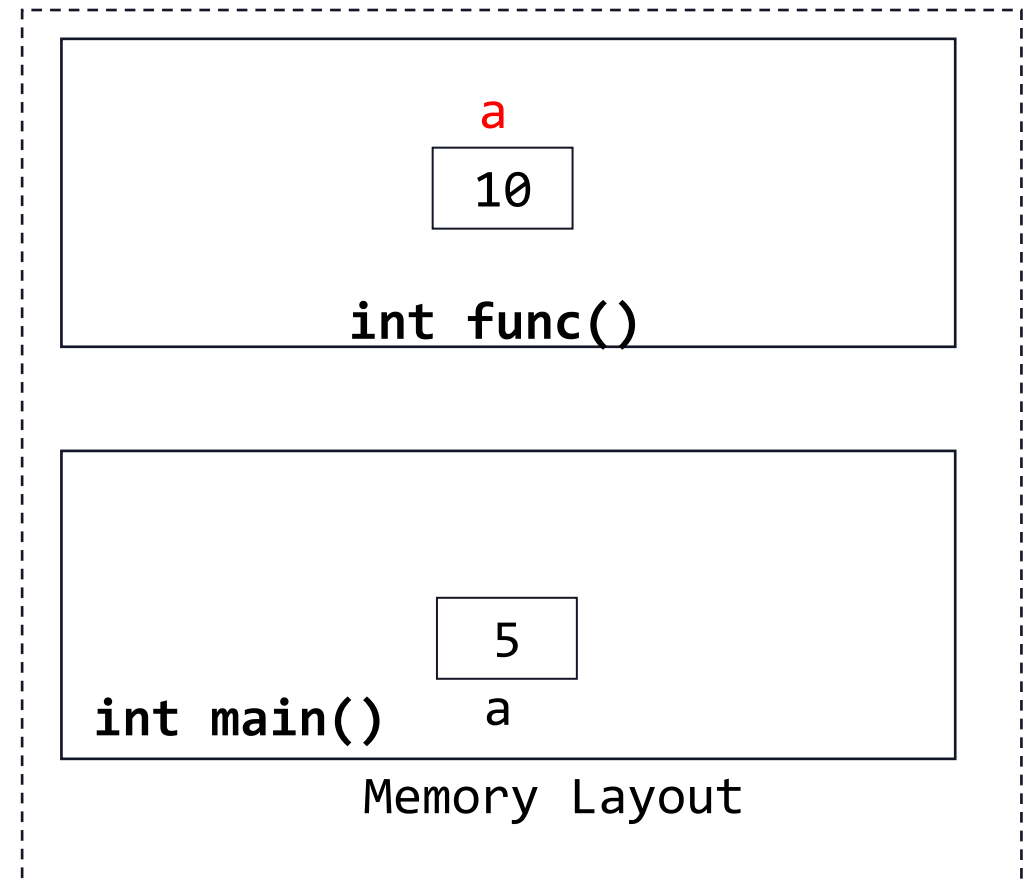
```
    int a=5;
```

```
    func();
```

```
    func();
```

```
    return 0;
```

```
}
```

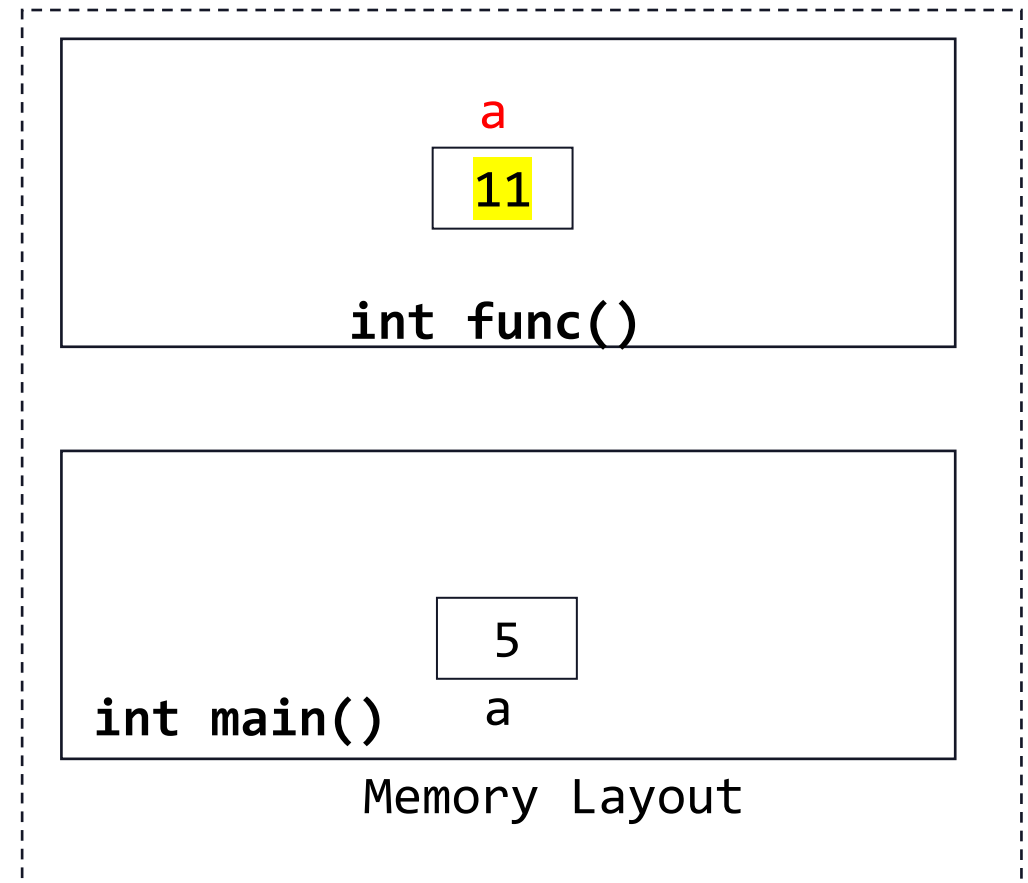


Biến cục bộ (Local variable)

- Biến cục bộ (**Local variable**) 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.

- Ví dụ:

```
#include <iostream>
void func() {
    int a=10; // Biến cục bộ
    a++;
    std::cout << a << std::endl;
}
int main() {
    int a=5;
    func();
    func();
    return 0;
}
```



Biến cục bộ (Local variable)

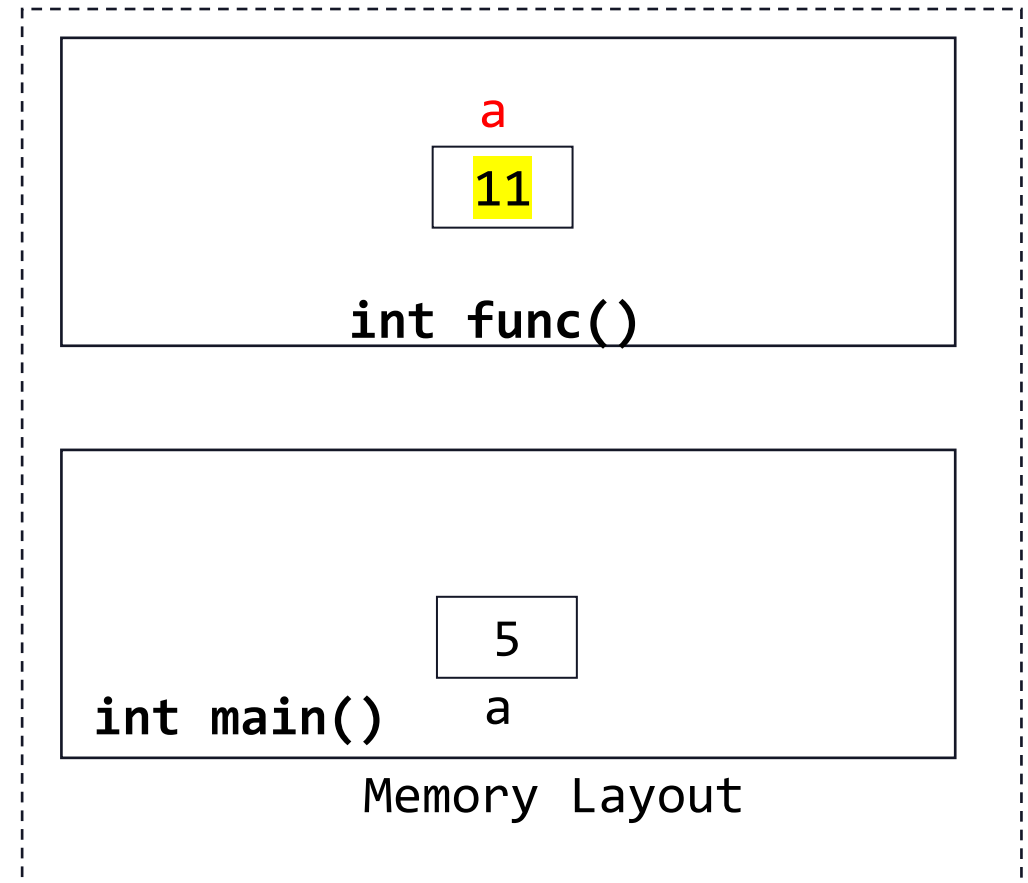
- Biến cục bộ (**Local variable**) được khai báo trong một khối ngoặc nhọn **{ }**. Biến chỉ có tác dụng trong khối ngoặc nhọn đó và sẽ bị xóa khỏi bộ nhớ khi chương trình chạy ra khỏi khối.

- Ví dụ:

```
#include <iostream>
void func() {
    int a=10; // Biến cục bộ
    a++;
    std::cout << a << std::endl;
}
int main() {
    int a=5;
    func();
    func();
    return 0;
}
```

Kết quả thực thi:

11





Biến cục bộ (Local variable)

- Biến cục bộ (**Local variable**) 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.

- Ví dụ:

```
#include <iostream>
```

```
void func() {
```

```
    int a=10; // Biến cục bộ
```

```
    a++;
```

```
    std::cout << a << std::endl;
```

```
}
```

```
int main() {
```

```
    int a=5;
```

```
    func();
```

```
    func();
```

```
    return 0;
```

```
}
```

Kết quả thực thi:

11

int func()

int main()

5

a

Memory Layout



Biến cục bộ (Local variable)

- Biến cục bộ (**Local variable**) 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.

- Ví dụ:

```
#include <iostream>
```

```
void func() {
```

```
    int a=10; // Biến cục bộ
```

```
    a++;
```

```
    std::cout << a << std::endl;
```

```
}
```

```
int main() {
```

```
    int a=5;
```

```
    func();
```

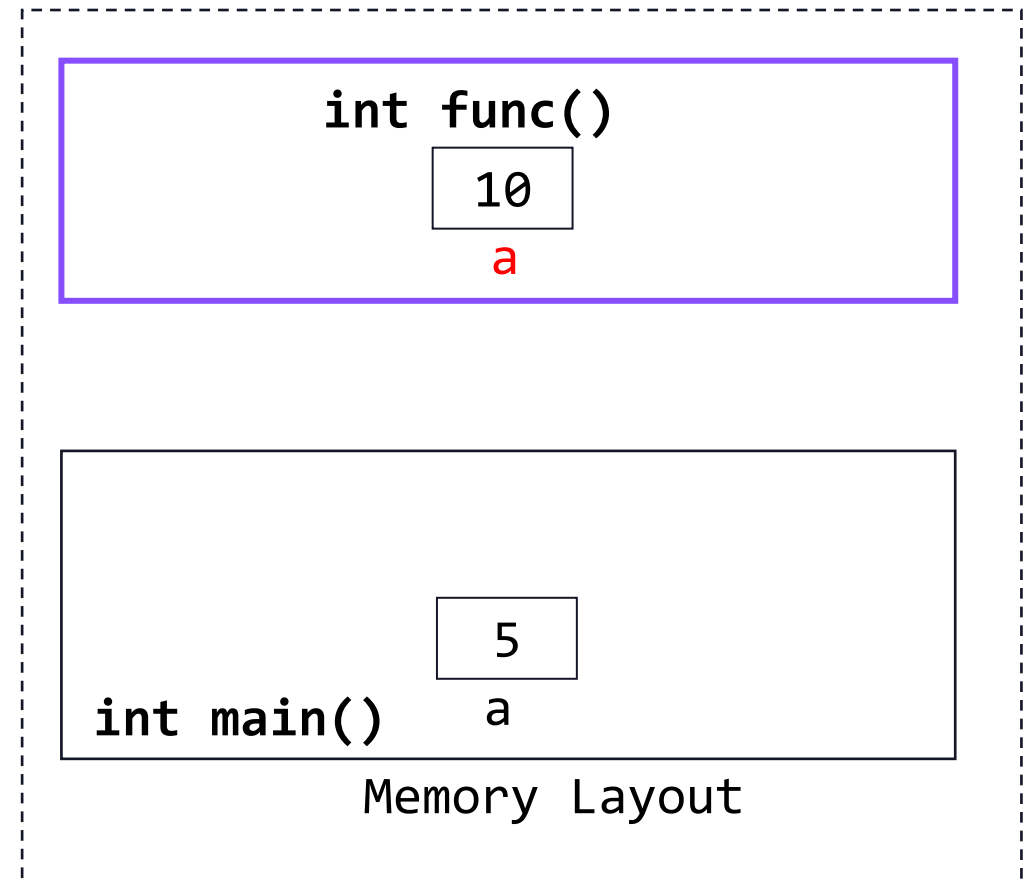
```
    func();
```

```
    return 0;
```

```
}
```

Kết quả thực thi:

11





Biến cục bộ (Local variable)

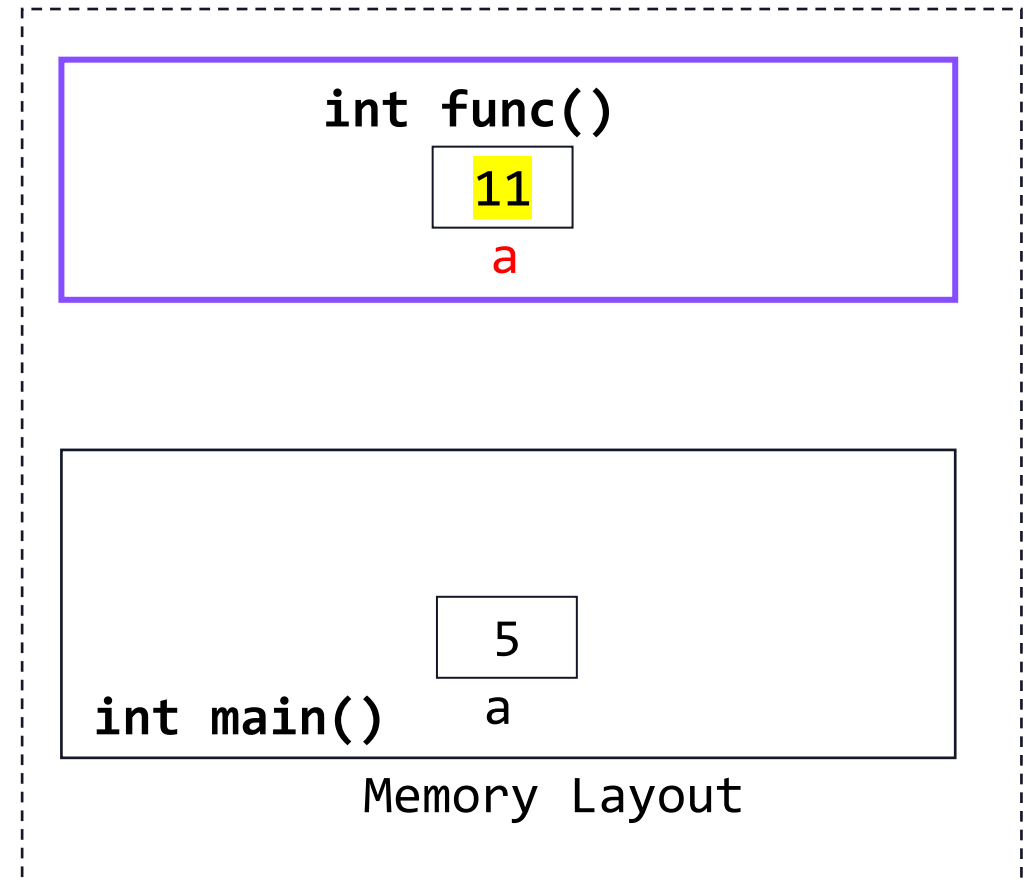
- Biến cục bộ (**Local variable**) được khai báo trong một khối ngoặc nhọn **{ }**. Biến chỉ có tác dụng trong khối ngoặc nhọn đó và sẽ bị xóa khỏi bộ nhớ khi chương trình chạy ra khỏi khối.

- Ví dụ:

```
#include <iostream>
void func() {
    int a=10; // Biến cục bộ
    a++;
    std::cout << a << std::endl;
}
int main() {
    int a=5;
    func();
    func();
    return 0;
}
```

Kết quả thực thi:

11





Biến cục bộ (Local variable)

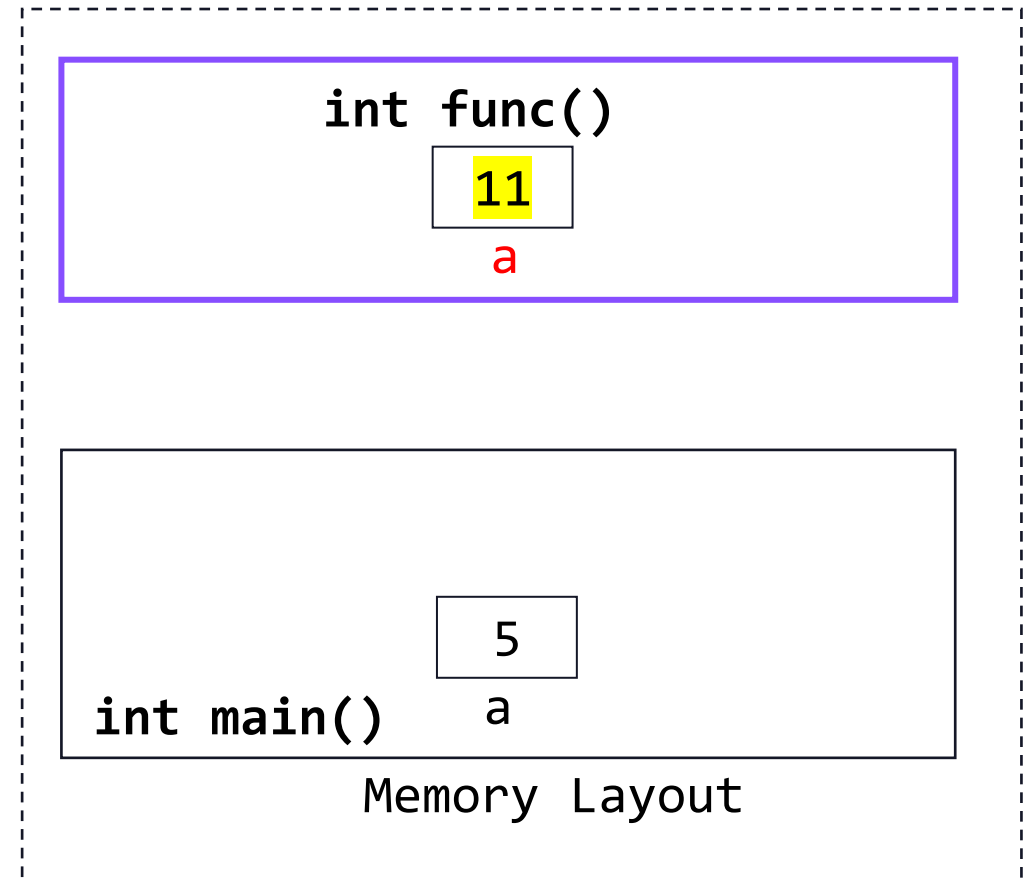
- Biến cục bộ (**Local variable**) được khai báo trong một khối ngoặc nhọn **{ }**. Biến chỉ có tác dụng trong khối ngoặc nhọn đó và sẽ bị xóa khỏi bộ nhớ khi chương trình chạy ra khỏi khối.

- Ví dụ:

```
#include <iostream>
void func() {
    int a=10; // Biến cục bộ
    a++;
    std::cout << a << std::endl;
}
int main() {
    int a=5;
    func();
    func();
    return 0;
}
```

Kết quả thực thi:

11
11





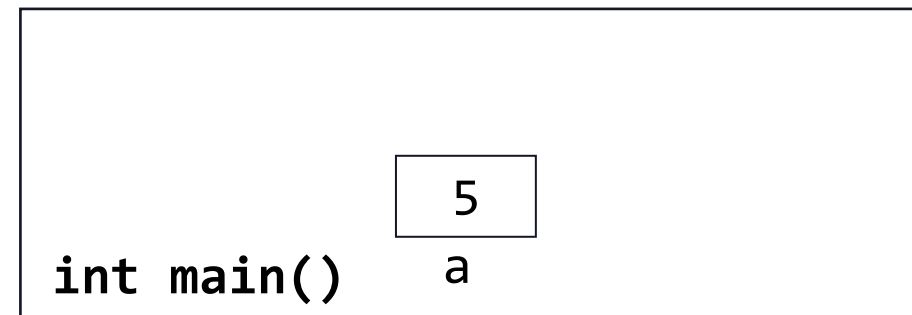
Biến cục bộ (Local variable)

- Biến cục bộ (**Local variable**) 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.
- Ví dụ:

```
#include <iostream>
void func() {
    int a=10; // Biến cục bộ
    a++;
    std::cout << a << std::endl;
}
int main() {
    int a=5;
    func();
    func();
    return 0;
}
```

Kết quả thực thi:

```
11
11
```



Memory Layout



Biến tĩnh cục bộ (Static local variable)

- **Biến tĩnh cục bộ** (static local variable): Khác với biến cục bộ thông thường, biến tĩnh cục bộ không bị hủy khi hàm kết thúc mà tồn tại suốt thời gian chạy của chương trình. Giá trị của biến tĩnh cục bộ được bảo lưu giữa các lần gọi hàm.
- Ví dụ:

```
#include <iostream>

void func() {
    static int a = 5; // Biến tĩnh cục bộ
    a++;
    std::cout << a << std::endl;
}

int main() {
    func(); func(); // In ra 7
    return 0;
}
```

Kết quả thực thi:

6
7



Biến toàn cục (Global variable)

- Biến toàn cục (**Global variable**): Biến toàn cục chỉ bị xóa khi chương trình kết thúc ☐ tồn 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 ☐ Khi dùng hàm nên **hạn chế dùng biến toàn cục**
- Ví dụ:

```
#include <iostream>
int a = 5; // Biến toàn cục
void func() {
    a++;
    std::cout << a << std::endl;
}
int main() {
    func();
    func();
    return 0;
}
```

Kết quả thực thi:

6
7



5.8 Tham số và đối số



5.8 Tham số và đối số

5.8.1 Khái niệm Tham số, Đối số

5.8.2 Tham số mặc định (Default Parameter)

5.8.3 Truyền đối số theo giá trị (Pass by Value)

5.8.4 Truyền đối số theo tham chiếu (Pass by Reference)

5.8.5 Truyền đối số theo con trỏ (Pass by Pointer)

5.8.6 Truyền đối số hằng theo tham chiếu (Pass by Const Reference)



5.8 Tham số và đối số

5.8.1 Khái niệm Tham số và đối số



5.8.1 Khái niệm Tham số và đối số

- **Parameter:**

- Tạm dịch: *Tham số* hoặc *tham số hình thức*
- Là các thông số mà **hàm nhận vào**
- Xác định khi khai báo hàm
- Ví dụ : `void func(int x, int y);` // x và y là các tham số hình thức

- **Argument:**

- Tạm dịch: *Đối số* hoặc *Tham số thực sự*
- Là các thông số được **đưa vào hàm** khi tiến hành gọi hàm
- Ví dụ: `func(10, 20);` // 10 và 20 là các đối số thực tế

- Hai thuật ngữ này đôi khi dùng lẫn lộn và gọi chung là *Tham số*



5.8 Tham số và đối số

5.8.2 Tham số mặc định (Default Parameter)



5.8.2 Tham số mặc định (Default Parameter)

- Giá trị mặc định của tham số trong hàm là giá trị được gán sẵn cho tham số đó khi **hàm được khai báo**.
- Giá trị mặc định sẽ **được sử dụng khi gọi hàm nếu không cung cấp đối số cho tham số** có giá trị mặc định.
- Vị trí của tham số mặc định: Các tham số mặc định phải được đặt sau các tham số không có giá trị mặc định. Ví dụ:

// Đúng: Tham số mặc định được **đặt sau tham số không mặc định**

```
void func(int x, int y = 10, int z = 10, int t = 10); // ĐÚNG
```

// Sai: Tham số mặc định **không được đặt trước tham số không mặc định**

```
void func(int x = 10, int y = 10, int z, int t); // Lỗi biên dịch
```

```
void func(int x, int y = 10, int z, int t = 10); // Lỗi biên dịch
```



5.8.2 Tham số mặc định (Default Parameter)

- Ví dụ:

```
#include <iostream>
using namespace std;
float divide (int a, int b=2) {
    return float(a)/b;
}
int main () {
    cout << divide (12) << '\n';
    cout << divide (21, 4) << '\n';
    return 0;
}
```

- Ví dụ: (Trong định nghĩa hàm không khai báo lại tham số mặc định)

```
#include <iostream>
using namespace std;
float divide (int a, int b=2);
int main () {
    cout << divide (12) << '\n';
    cout << divide (21, 4) << '\n';
}
float divide (int a, int b) {
    return float(a)/b;
}
```



5.8 Tham số và đối số

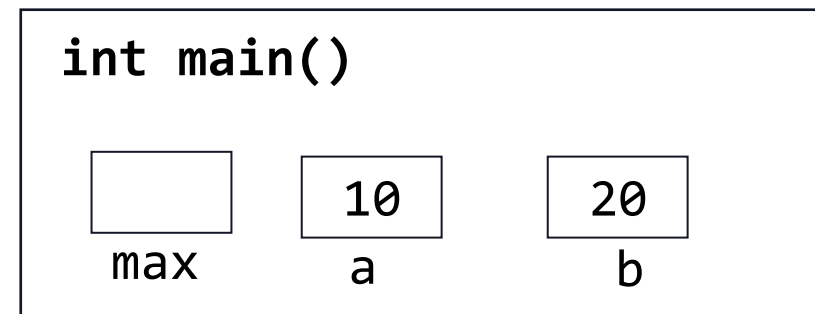
5.8.3 Truyền đối số theo giá trị (Pass by Value)



5.8.3 Truyền đối số theo giá trị (Pass by Value)

- 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.
- Ví dụ:

```
int so_lon(int m, int n){  
    return m > n ? m : n;  
}  
int main() {  
    int a = 10, b = 20;  
    int max = so_lon(a, b);  
    cout << max;  
}
```



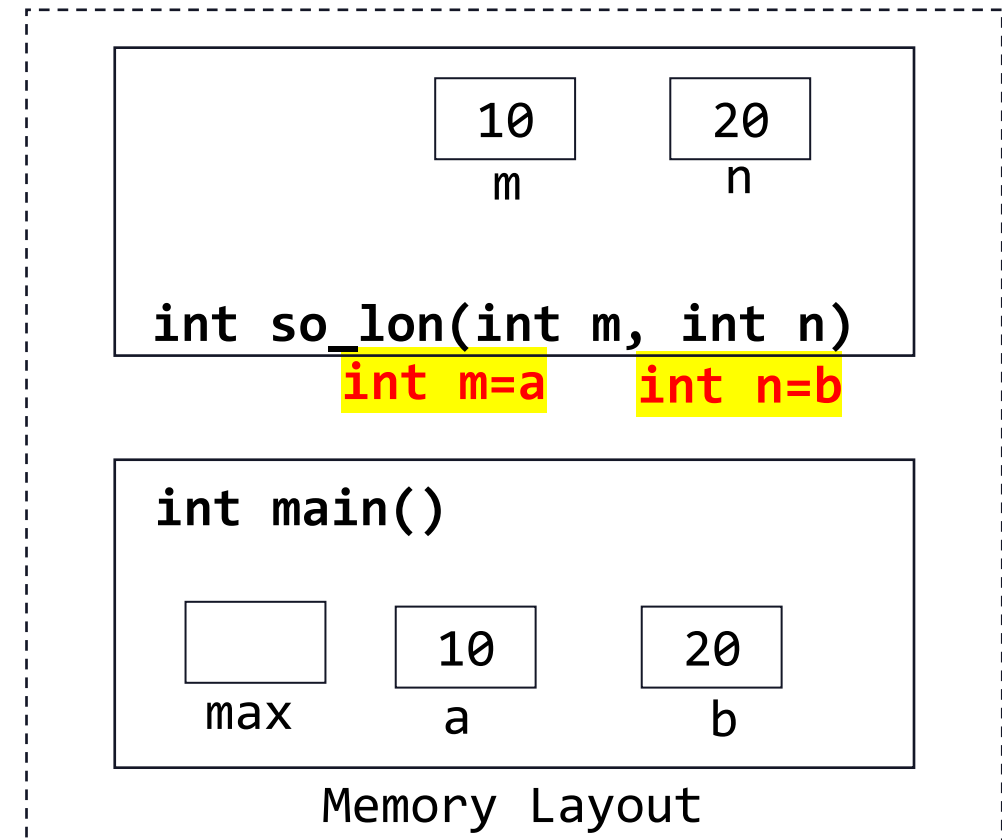
Memory Layout



5.8.3 Truyền đối số theo giá trị (Pass by Value)

- 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.
- Ví dụ:

```
int so_lon(int m, int n){  
    return m > n ? m : n;  
}  
int main() {  
    int a = 10, b = 20;  
    int max = so_lon(a, b);  
    cout << max;  
}
```

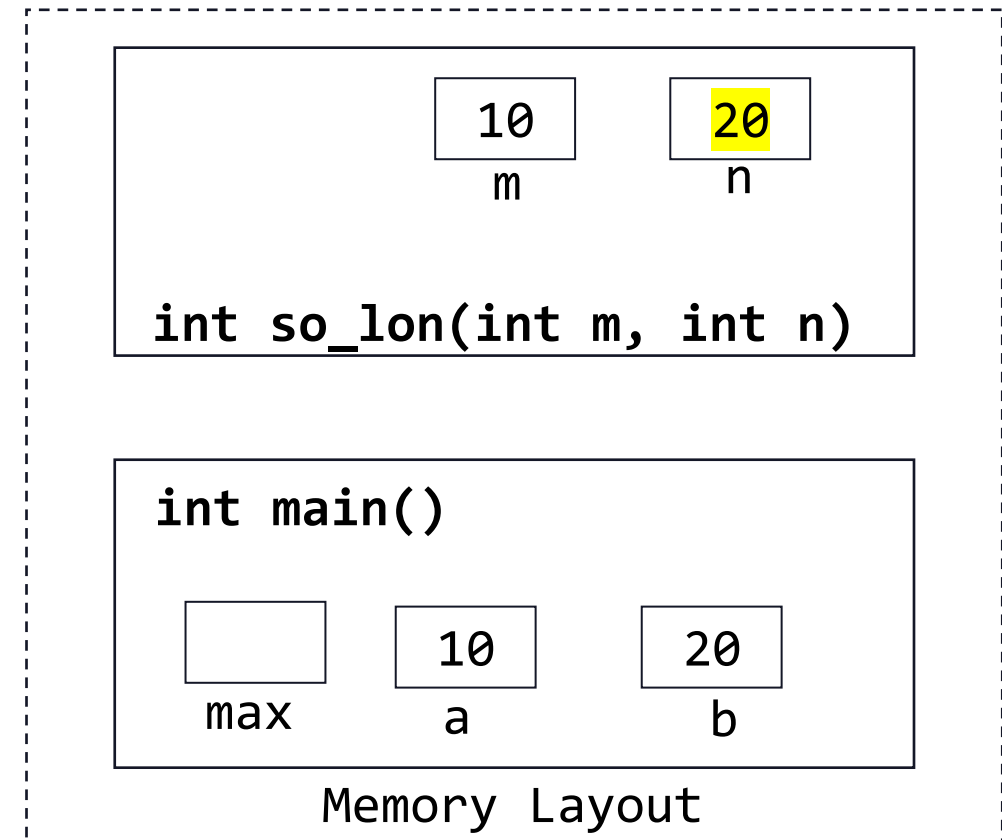




5.8.3 Truyền đối số theo giá trị (Pass by Value)

- 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.
- Ví dụ:

```
int so_lon(int m, int n){  
    return m > n ? m : n;  
}  
int main() {  
    int a = 10, b = 20;  
    int max = so_lon(a, b);  
    cout << max;  
}
```



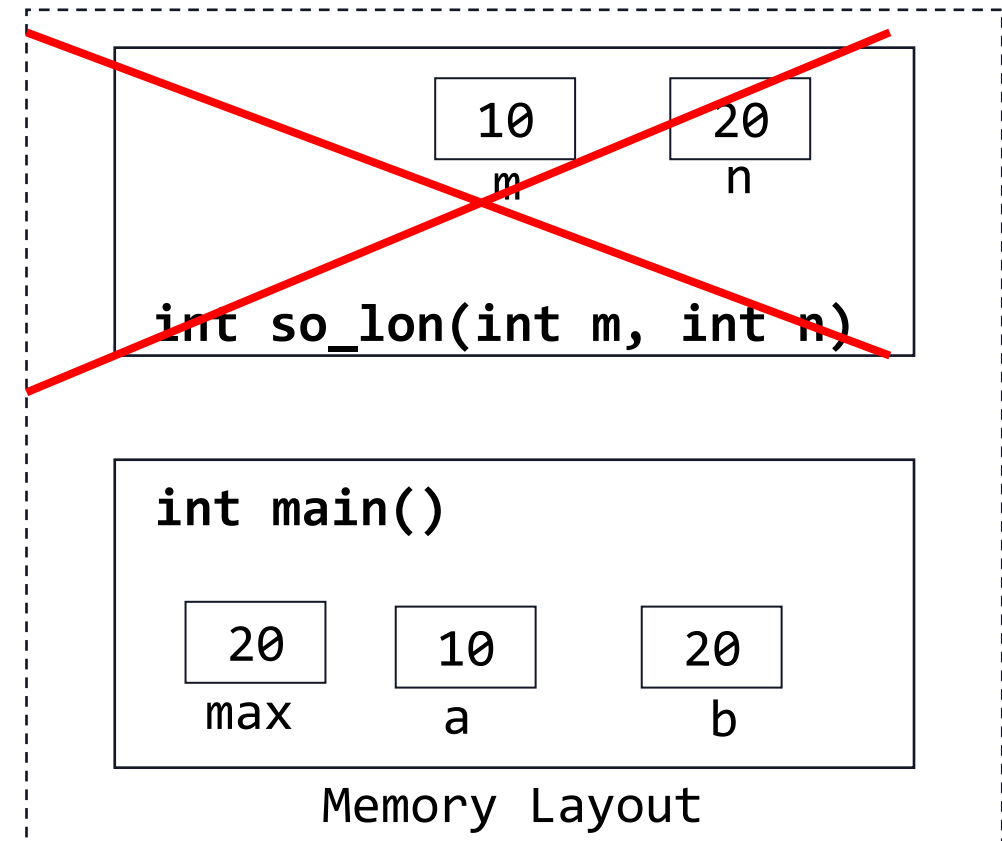


5.8.3 Truyền đối số theo giá trị (Pass by Value)

- 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.
- Ví dụ:

Kết quả thực thi:
20

```
int so_lon(int m, int n){  
    return m > n ? m : n;  
}  
int main() {  
    int a = 10, b = 20;  
    int max = so_lon(a, b);  
    cout << max;  
}
```





5.8.3 Truyền đối số theo giá trị (Pass by Value)

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

- Ví dụ:

```
int so_lon(int m, int n){ return m > n ? m : n; }
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 hon la " << so_lon(9*4, nhap_so_duong());
    return 0;
}
```



5.8.3 Truyền đối số theo giá trị (Pass by Value)

- Vậy bài toán ở mục “Đặt vấn đề” có thể viết lại như sau:

```
1. int so_lon(int m, int n){ return m > n ? m : n; }
2. int nhap_so_duong(){
3.     int n;
4.     do {
5.         cout << "Nhap mot so nguyen duong: "; cin >> n;
6.     } while (n <= 0);
7.     return n;
8. }
9. int main() {
10.    cout << "So lon nhat trong 04 so la "
11.        << so_lon(
12.            so_lon(nhap_so_duong(), nhap_so_duong())
13.            , so_lon(nhap_so_duong(), nhap_so_duong())
14.        );
15.    return 0;
16. }
```



5.8 Tham số và đối số

5.8.4 Truyền đối số theo tham chiếu (Pass by Reference)



5.8.4 Truyền đối số theo tham chiếu (Pass by Reference)

- 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.**

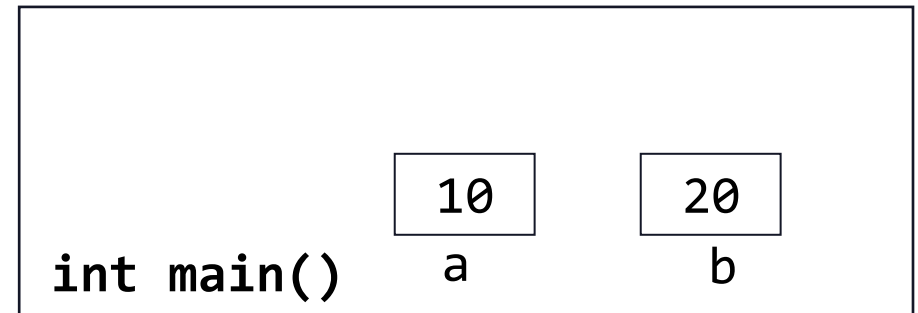
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ 1: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int& x, int& y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20



Memory Layout



5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int& x, int& y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20

```
int hoan_vi(int &x, int &y)
```

```
    int &x=a    int &y=b
```

```
          x      y
        [ 10 ]  [ 20 ]
    int main()  a      b
```

Memory Layout

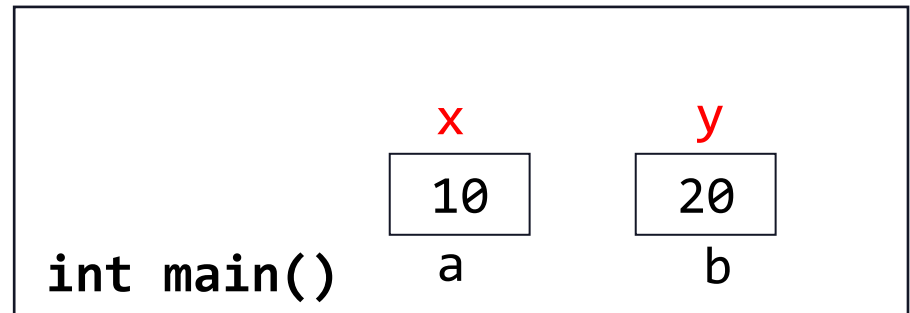
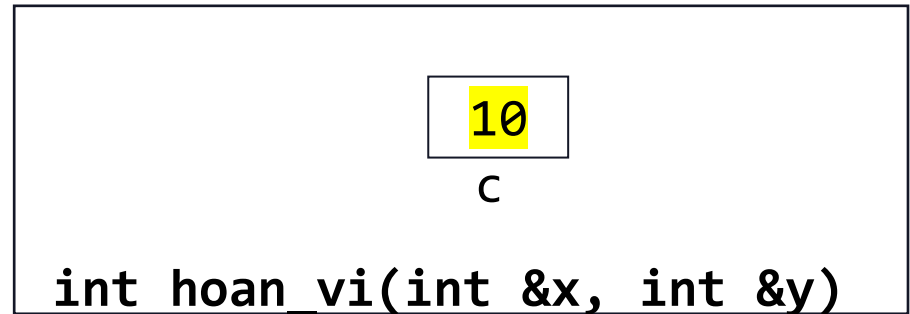
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int& x, int& y){  
2.     int c = x;  
3.     x = y;  
4.     y = c;  
5. }  
6. int main() {  
7.     int a=10, b=20;  
8.     cout << a << " " << b << endl;  
9.     hoan_vi(a, b);  
10.    cout << a << " " << b;  
11.    return 0;  
12. }
```

Kết quả thực thi:

10 20



Memory Layout



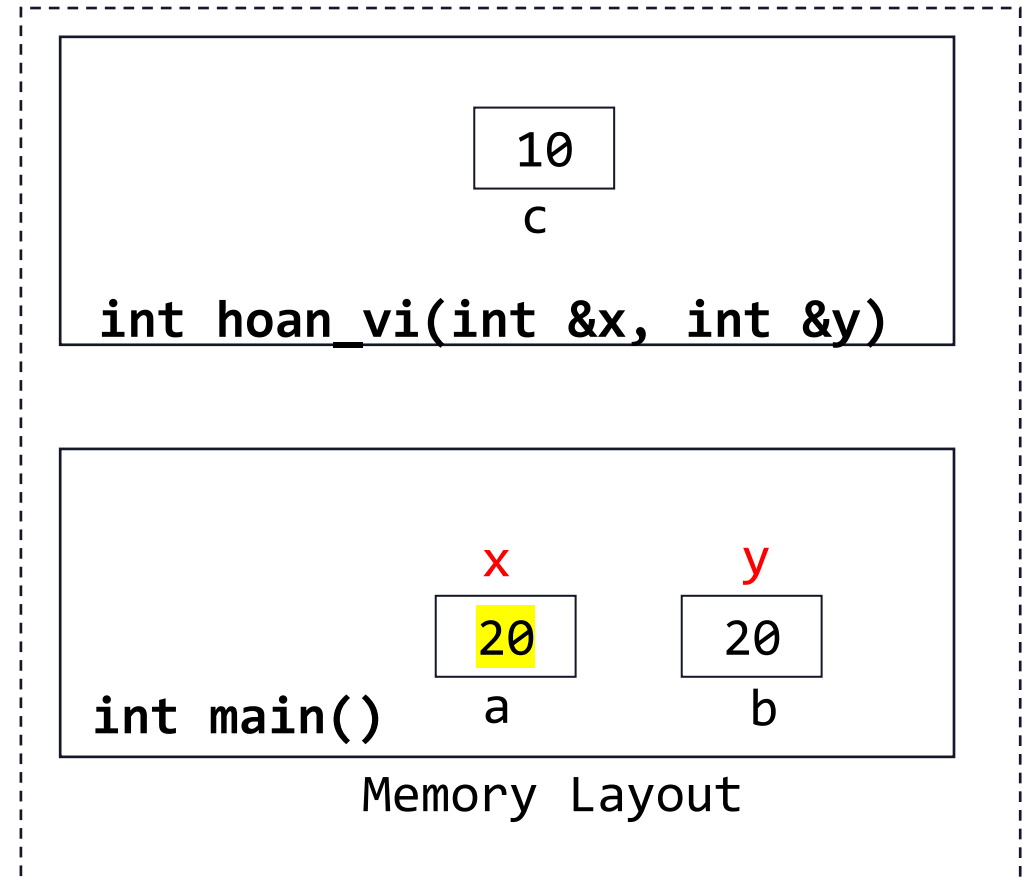
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int& x, int& y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20



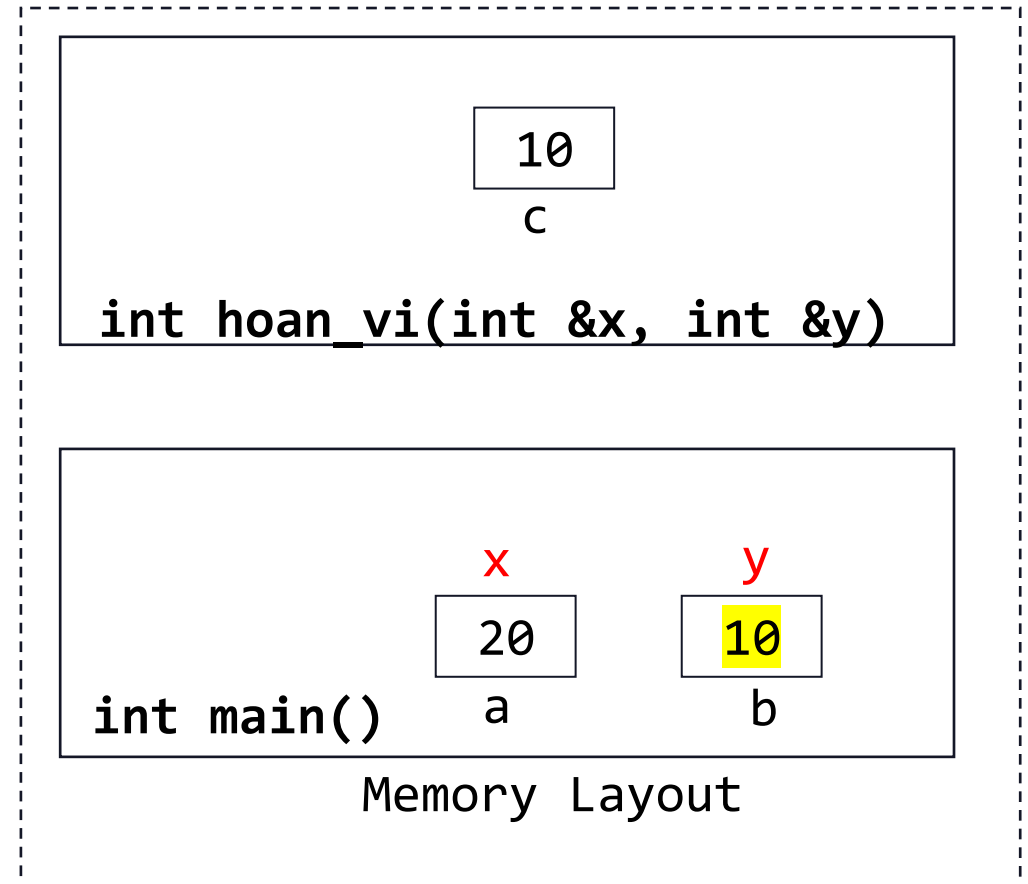
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int& x, int& y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20



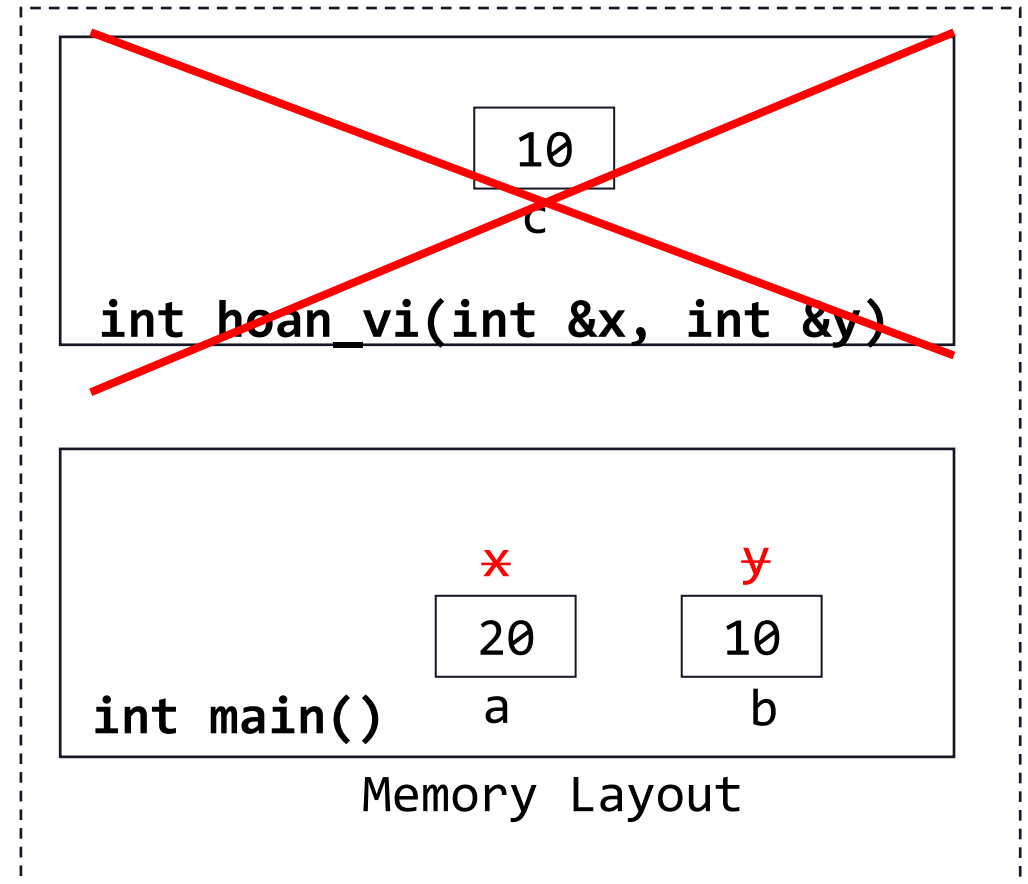
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int& x, int& y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20
20 10





5.8.4 Truyền đối số theo tham chiếu

- Ví dụ 2: Hỏi hàm hoán vị sau có hoạt động đúng theo yêu cầu không?

```
1. void hoan_vi(int x, int y){  
2.     int c = x;  
3.     x = y;  
4.     y = c;  
5. }  
6. int main() {  
7.     int a=10, b=20;  
8.     cout << a << " " << b << endl;  
9.     hoan_vi(a, b);  
10.    cout << a << " " << b;  
11.    return 0;  
12. }
```



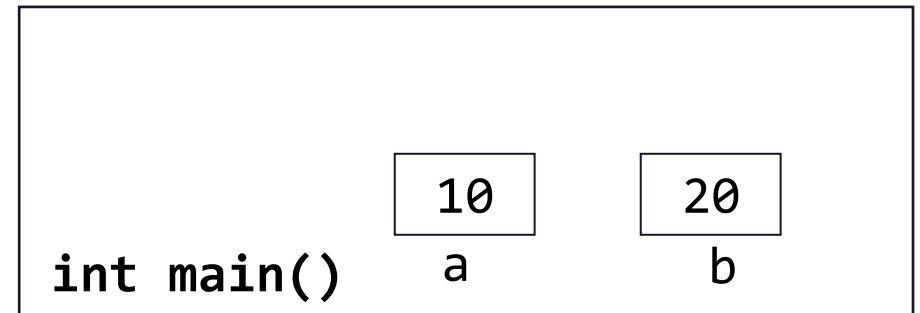
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ 2: Hỏi hàm hoán vị sau có hoạt động đúng theo yêu cầu không?

```
1. void hoan_vi(int x, int y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20



Memory Layout



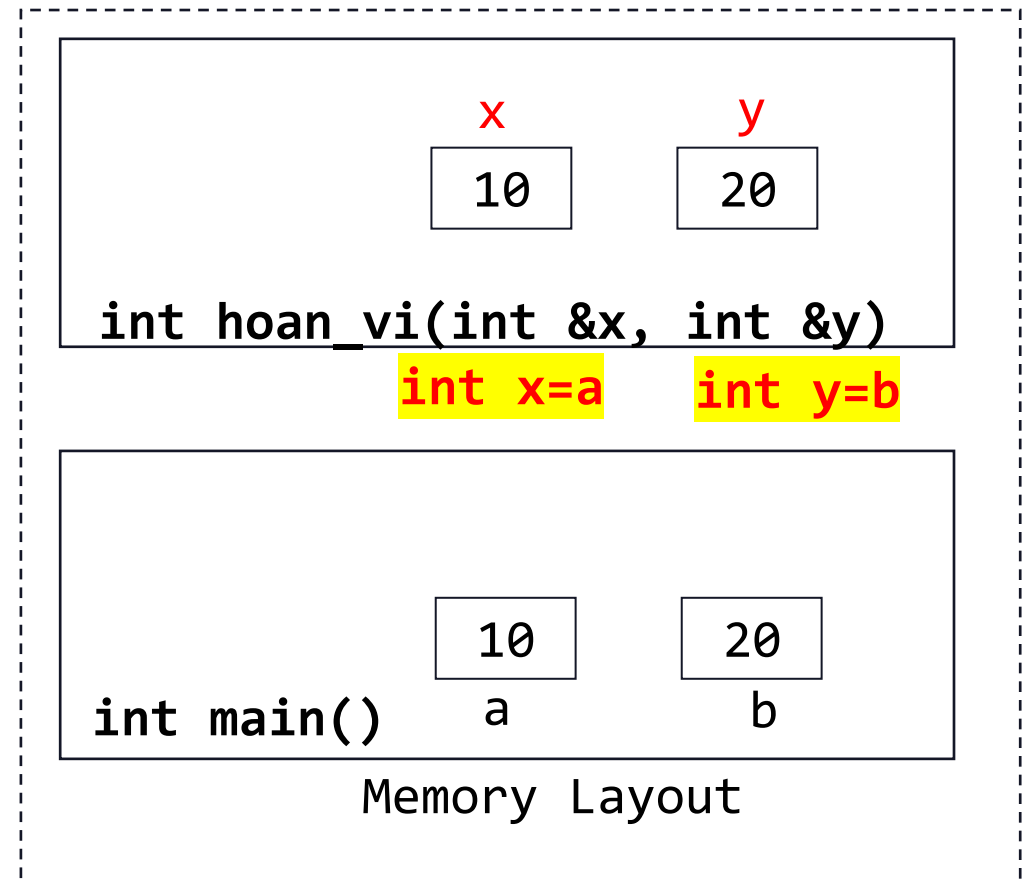
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int x, int y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20



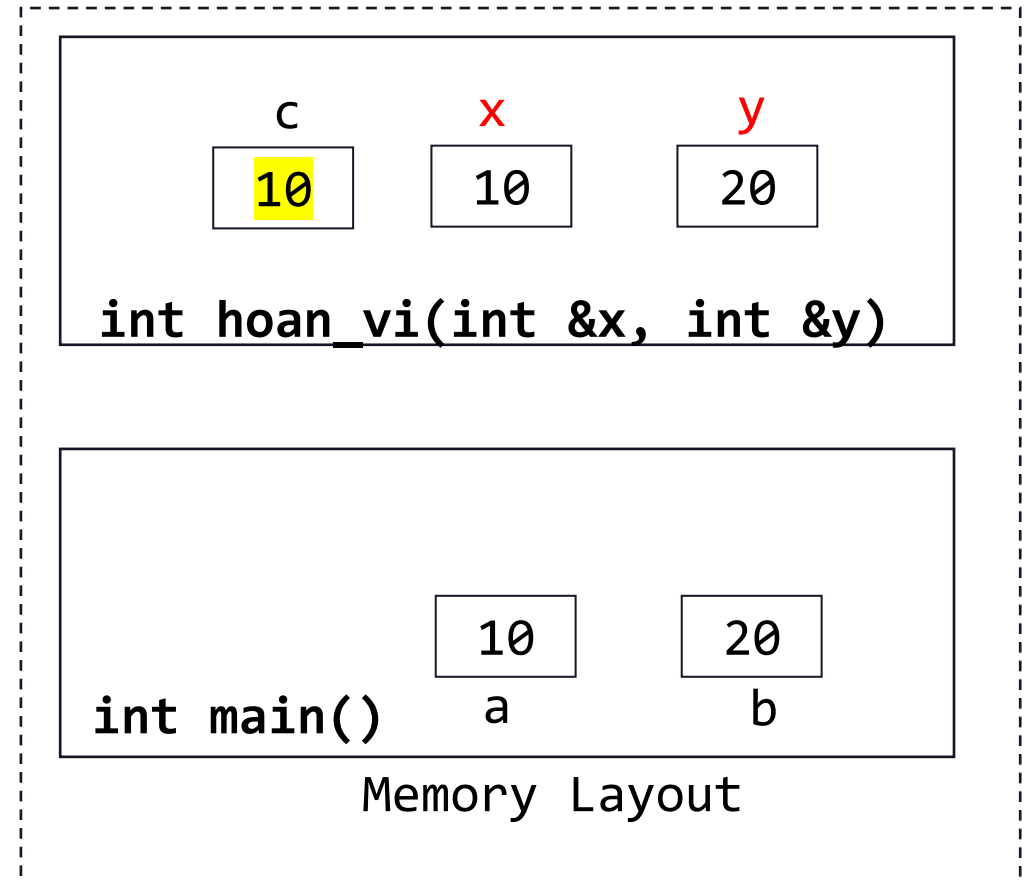
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int x, int y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20





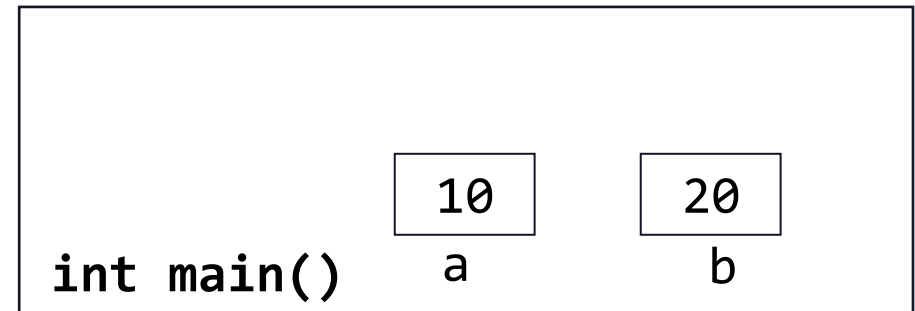
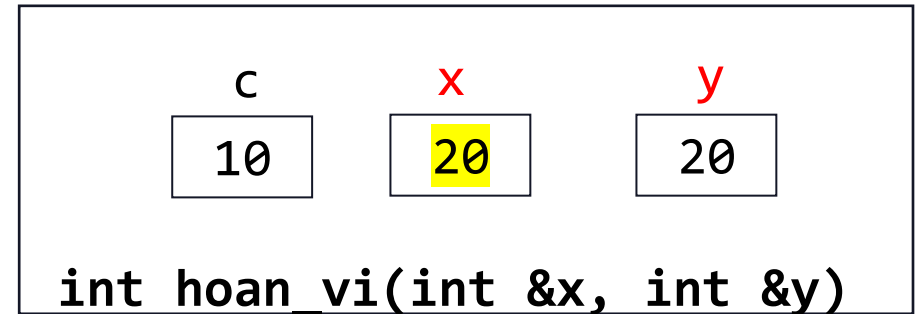
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int x, int y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20



Memory Layout



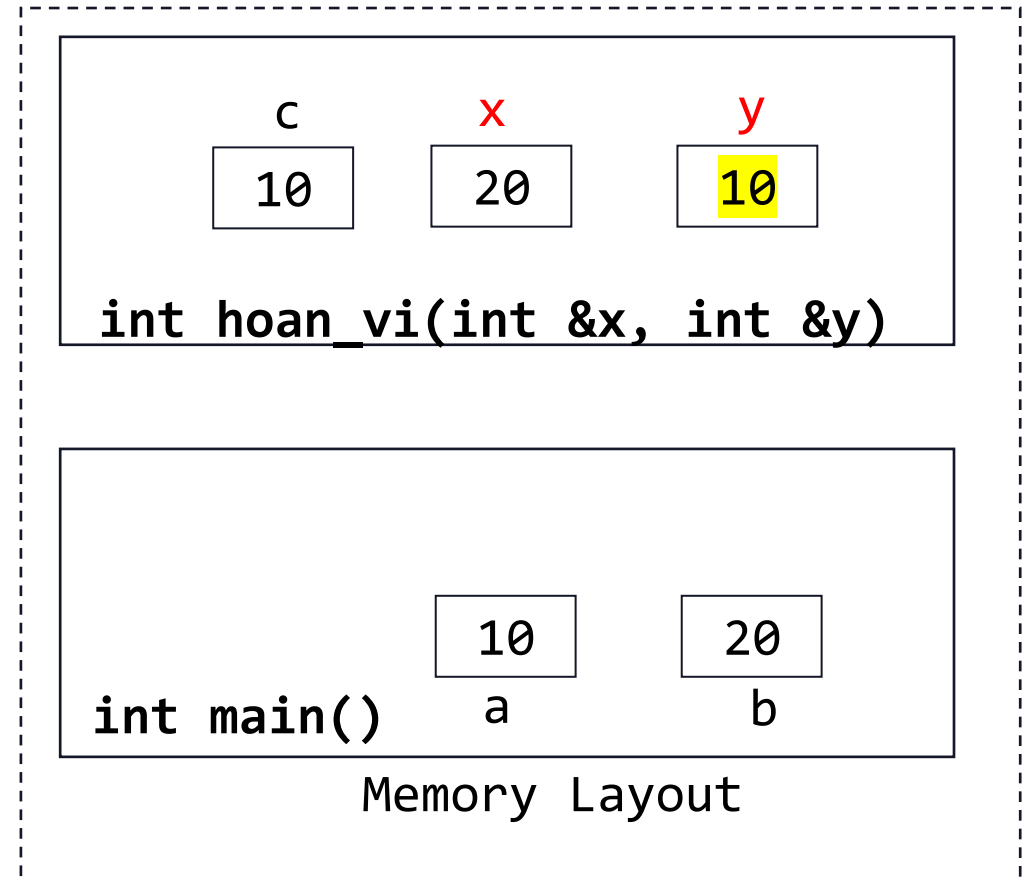
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int x, int y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20



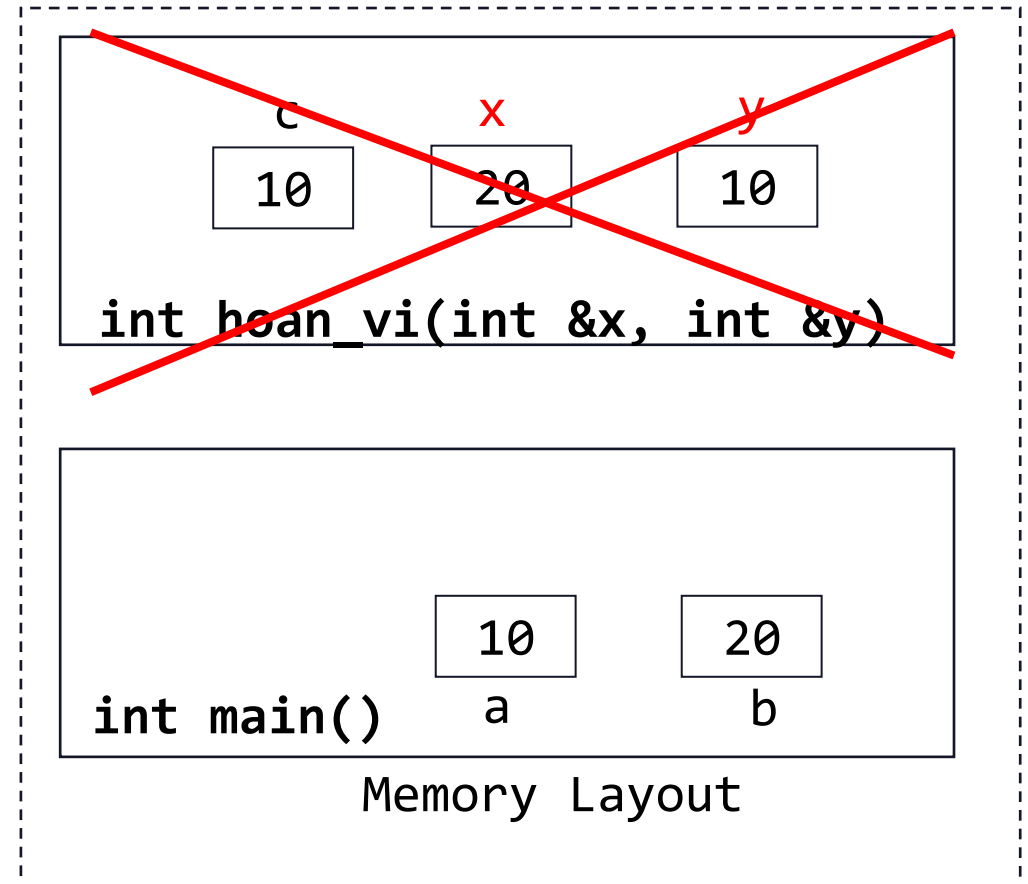
5.8.4 Truyền đối số theo tham chiếu

- Ví dụ: Viết chương trình hoán vị 2 số nguyên.

```
1. void hoan_vi(int x, int y){
2.     int c = x;
3.     x = y;
4.     y = c;
5. }
6. int main() {
7.     int a=10, b=20;
8.     cout << a << " " << b << endl;
9.     hoan_vi(a, b);
10.    cout << a << " " << b;
11.    return 0;
12. }
```

Kết quả thực thi:

10 20
10 20



Memory Layout



5.8.4 Truyền đối số theo tham chiếu

- Dùng truyền **tham chiếu** như một cách trả về kết quả.
- Ví dụ 3:

```
bool phep_chia(int x, int y, double& thuong){  
    if (y != 0) {  
        thuong = double(x)/y;  
        return true;  
    }  
    return false;  
}  
  
int main(){  
    int a, b;  
    cin >> a >> b;  
    double thuong;  
    if (phep_chia(a, b, thuong))  
        cout << "Thuong so la " << thuong;  
    else cout << "Khong the chia duoc. ";  
    return 0;  
}
```



5.8 Tham số và đối số

5.8.5 Truyền đối số theo con trỏ (Pass by Pointer)

5.8.5 Truyền đối số theo con trỏ (Pass by Pointer)



- Tham số là một con trỏ, cho phép hàm **thao tác trực tiếp với địa chỉ của đối số thực tế.**
- Ví dụ:

```
#include <iostream>
using namespace std;
void increment(int* num) {
    (*num)++;
    cout << "Trong ham: " << *num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    increment(&a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

5.8.5 Truyền đối số theo con trỏ (Pass by Pointer)

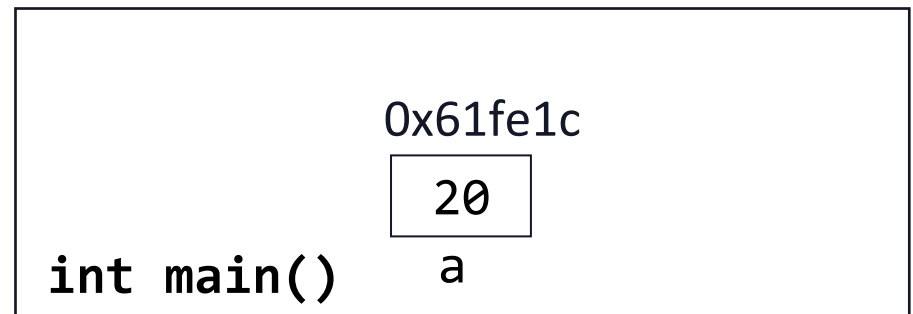


- Tham số là một con trỏ, cho phép hàm **thao tác trực tiếp với địa chỉ của đối số thực tế.**

- Ví dụ:

```
#include <iostream>
using namespace std;
void increment(int* num) {
    (*num)++;
    cout << "Trong ham: " << *num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    increment(&a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

Kết quả thực thi:
Truoc khi gọi ham: 20



Memory Layout

5.8.5 Truyền đối số theo con trỏ (Pass by Pointer)

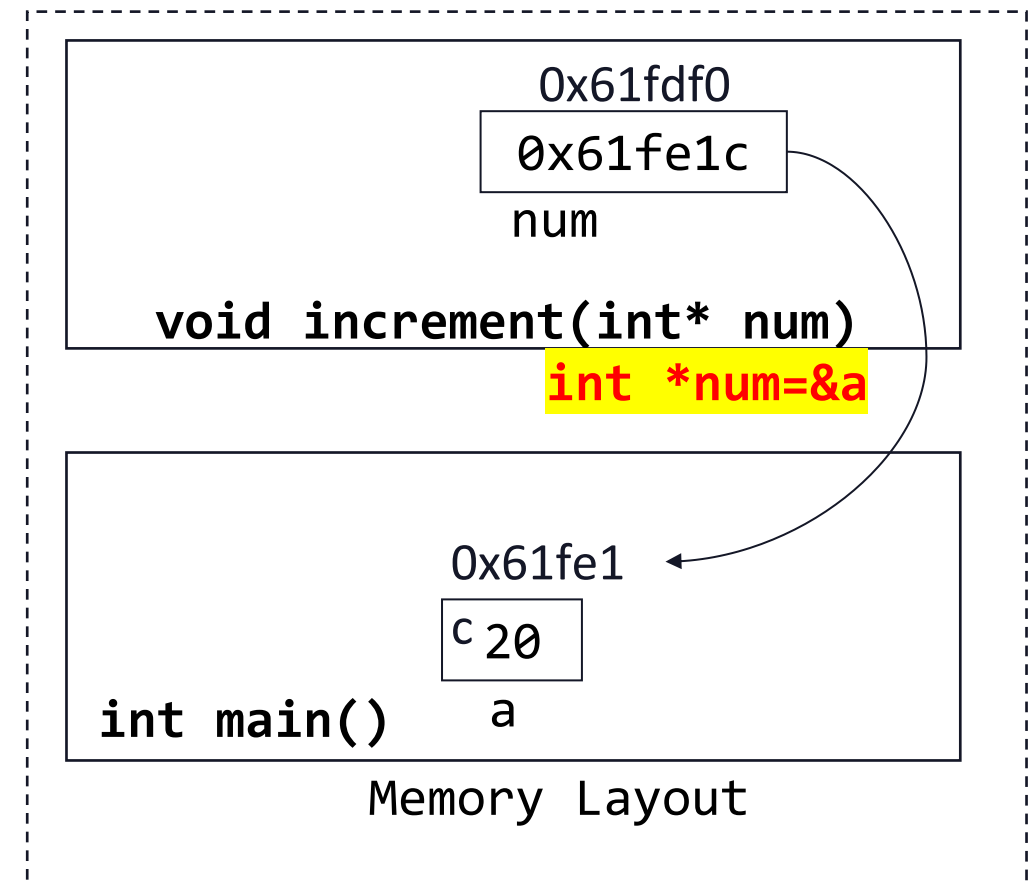


- Tham số là một con trỏ, cho phép hàm thao tác trực tiếp với địa chỉ của đối số thực tế.

- Ví dụ:

```
#include <iostream>
using namespace std;
void increment(int* num) {
    (*num)++;
    cout << "Trong ham: " << *num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    increment(&a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

Kết quả thực thi:
Truoc khi gọi ham: 20



5.8.5 Truyền đối số theo con trỏ (Pass by Pointer)

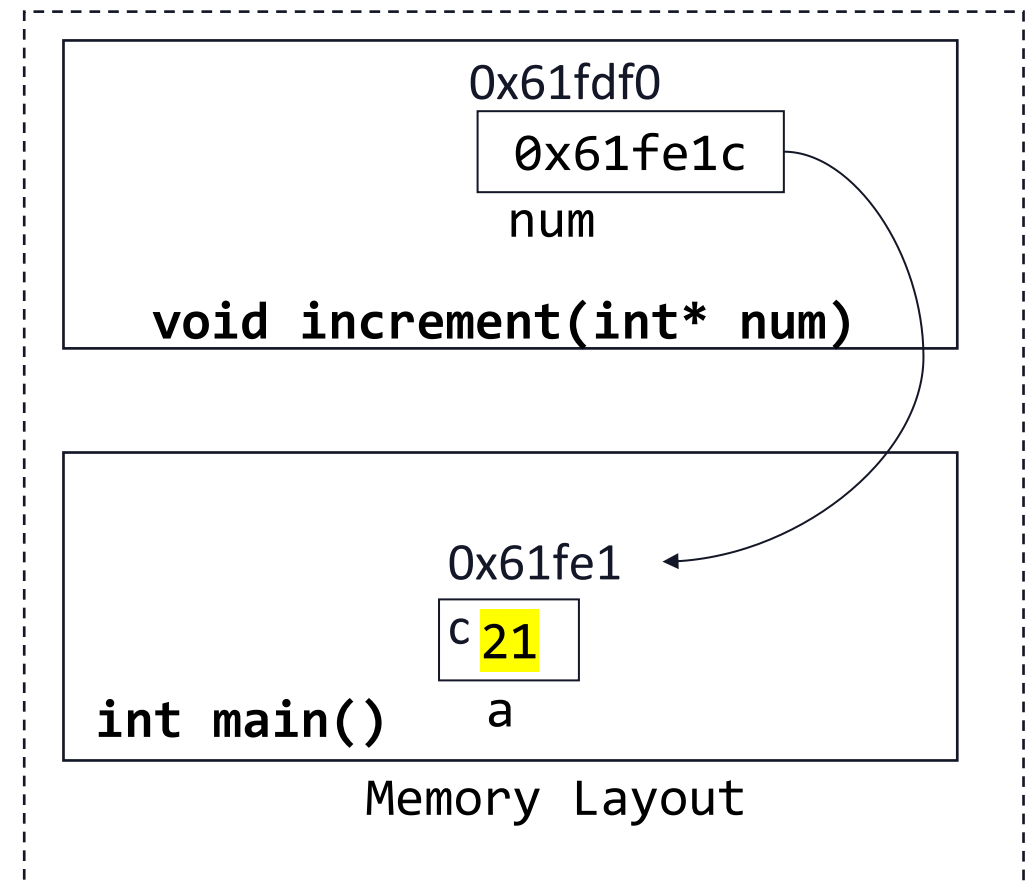


- Tham số là một con trỏ, cho phép hàm **thao tác trực tiếp với địa chỉ của đối số thực tế**.

- Ví dụ:

```
#include <iostream>
using namespace std;
void increment(int* num) {
    (*num)++;
    cout << "Trong ham: " << *num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    increment(&a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

Kết quả thực thi:
Truoc khi gọi ham: 20



5.8.5 Truyền đối số theo con trỏ (Pass by Pointer)

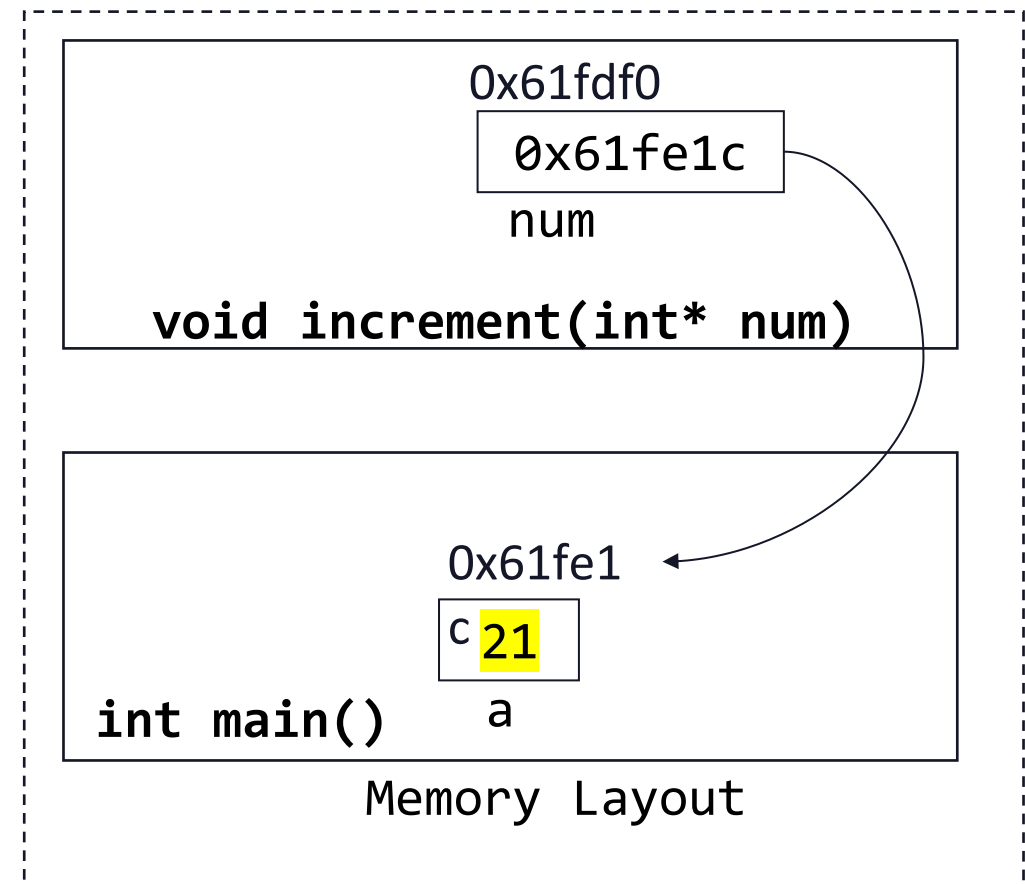


- Tham số là một con trỏ, cho phép hàm **thao tác trực tiếp với địa chỉ của đối số thực tế**.

- Ví dụ:

```
#include <iostream>
using namespace std;
void increment(int* num) {
    (*num)++;
    cout << "Trong ham: " << *num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    increment(&a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

Kết quả thực thi:
Truoc khi gọi ham: 20
Trong ham: 21



5.8.5 Truyền đối số theo con trỏ (Pass by Pointer)

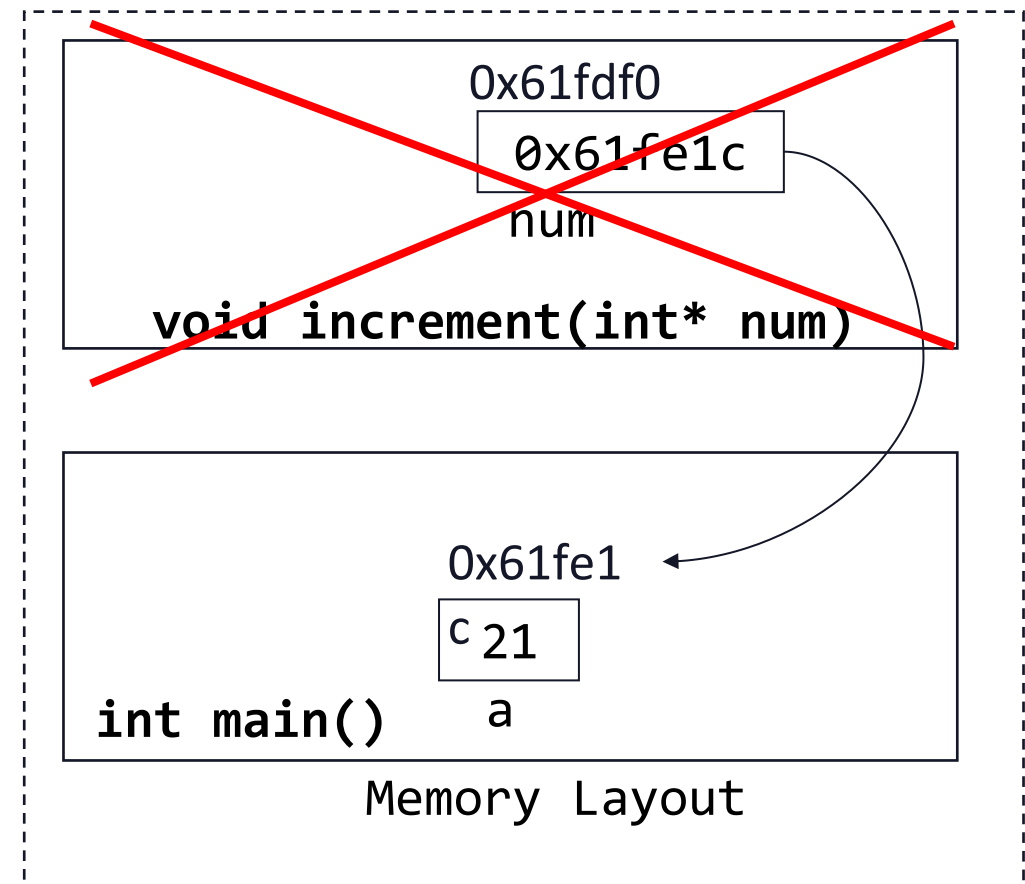


- Tham số là một con trỏ, cho phép hàm **thao tác trực tiếp với địa chỉ của đối số thực tế**.

- Ví dụ:

```
#include <iostream>
using namespace std;
void increment(int* num) {
    (*num)++;
    cout << "Trong ham: " << *num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    increment(&a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

Kết quả thực thi:
Truoc khi gọi ham: 20
Trong ham: 21
Sau khi gọi ham: 21





5.8 Tham số và đối số

5.8.6 Truyền đối số hằng theo tham chiếu (Pass by Const Reference)

5.8.6 Truyền đối số hằng theo tham chiếu (Pass by Const Reference)



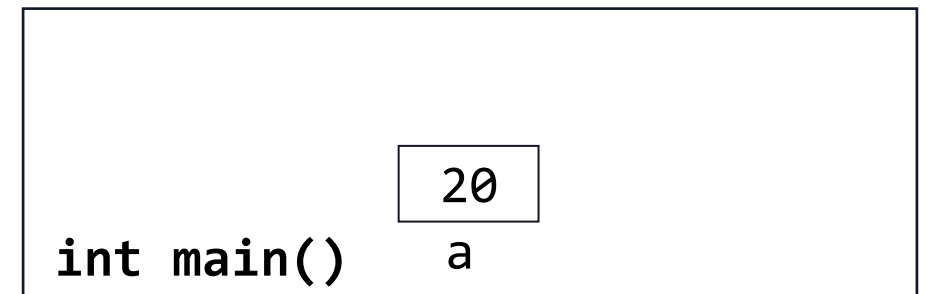
- Tham số tham chiếu được khai báo với từ khóa **const**, không cho phép hàm thay đổi giá trị của tham số tham chiếu đó.

- Ví dụ:

```
#include <iostream>
using namespace std;
void print(const int& num) {
    cout << "Trong ham: " << num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    print(a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

Đặt lệnh `num++` vào trong hàm được không?

Kết quả thực thi:
Truoc khi gọi ham: 20



Memory Layout

5.8.6 Truyền đối số hằng theo tham chiếu (Pass by Const Reference)



- Tham số tham chiếu được khai báo với từ khóa **const**, không cho phép hàm thay đổi giá trị của tham số tham chiếu đó.

- Ví dụ:

```
#include <iostream>
using namespace std;
void print(const int& num) {
    cout << "Trong ham: " << num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    print(a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

Đặt lệnh num++ vào trong hàm được không?

Kết quả thực thi:
Truoc khi gọi ham: 20

```
int print(const int &num)
    const int &num=a
```

```
num
20
int main()    a
```

Memory Layout

5.8.6 Truyền đối số hằng theo tham chiếu (Pass by Const Reference)



- Tham số tham chiếu được khai báo với từ khóa **const**, không cho phép hàm thay đổi giá trị của tham số tham chiếu đó.

- Ví dụ:

```
#include <iostream>
using namespace std;
void print(const int& num) {
    cout << "Trong hàm: " << num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi hàm: " << a << endl;
    print(a);
    cout << "Sau khi gọi hàm: " << a << endl;
    return 0;
}
```

Đặt lệnh `num++` vào trong hàm được không?
=> **KHÔNG ĐƯỢC** => Do `num` không thể thay đổi giá trị!

Kết quả thực thi:
Truoc khi gọi hàm: 20
Trong hàm: 20

`int print(const int &num)`

`int main()` **num**
 20
 a

Memory Layout



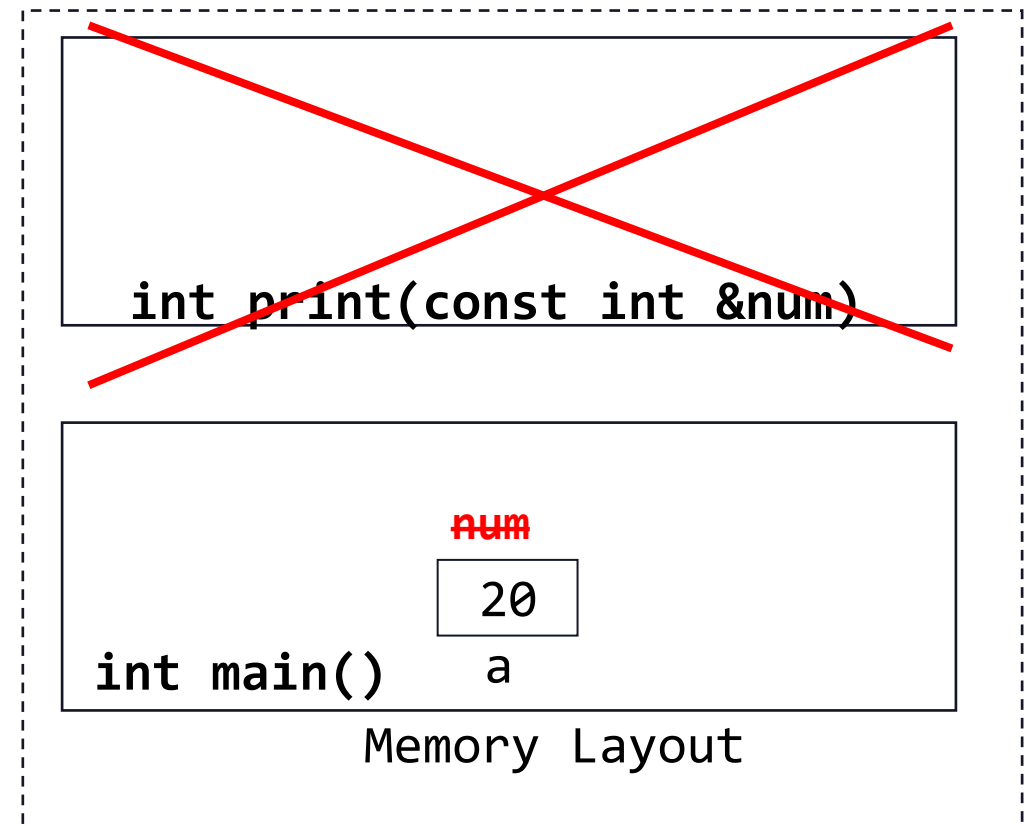
5.8.6 Truyền đối số hằng theo tham chiếu (Pass by Const Reference)

- Tham số tham chiếu được khai báo với từ khóa **const**, không cho phép hàm thay đổi giá trị của tham số tham chiếu đó.

- Ví dụ:

```
#include <iostream>
using namespace std;
void print(const int& num) {
    cout << "Trong ham: " << num << endl;
}
int main() {
    int a = 20;
    cout << "Truoc khi gọi ham: " << a << endl;
    print(a);
    cout << "Sau khi gọi ham: " << a << endl;
    return 0;
}
```

Kết quả thực thi:
Truoc khi gọi ham: 20
Trong ham: 20
Sau khi gọi ham: 20





5.9 Giá trị trả về của hàm



5.9 Giá trị trả về của hàm

5.9.1 Hàm trả về giá trị

5.9.2 Hàm trả về void

5.9.3 Hàm trả về tham chiếu

5.9.4 Hàm trả về địa chỉ

5.9.5 Giá trị trả về trong hàm main



5.9 Giá trị trả về của hàm

5.9.1 Hàm trả về giá trị

5.9.1 Hàm trả về giá trị

- Hàm có thể trả về một giá trị có kiểu dữ liệu nhất định (**bool**, **int**, **float**, ...) sau khi thực hiện xong. Kết quả trả về của hàm có thể sử dụng trong các phép toán hoặc gán cho biến khác.
- **Giá trị trả về** của hàm **là biến**, hằng hoặc biểu thức với kiểu dữ liệu tương ứng với giá trị trả về..
- Khi sử dụng câu lệnh **return**, hàm chỉ có thể trả về một giá trị duy nhất.
- Lệnh **return** sẽ kết thúc quá trình thực thi của hàm.
- Ví dụ:

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

```
bool equal(int m, int n) {  
    if (m == n) return 1;  
    return 0;  
}
```



5.9.1 Hàm trả về giá trị

- Ví dụ: So sánh 2 hàm `nhap_so_duong` bên dưới:

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

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

```
int main() {
    int a, b;
    a = nhap_so_duong();
    nhap_so_duong(b);
    cout << "So vua nhap la "
        << a << endl;
    cout << "Tong hai so la "
        << a + b << endl;
    return 0;
}
```



5.9 Giá trị trả về của hàm

5.9.2 Hàm trả về void



5.9.2 Hàm trả về void

- Hàm có thể trả về **void**, nghĩa là không trả về gì hết.
- Lệnh **return** với các hàm không có đầu ra sẽ không kèm theo giá trị (nhưng vẫn sẽ kết thúc việc thực thi hàm)
- Ví dụ:

```
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;  
}
```

```
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;  
}
```




5.9 Giá trị trả về của hàm

5.9.3 Hàm trả về tham chiếu



5.9.3 Hàm trả về tham chiếu

- **Hàm trả về tham chiếu** là cho phép hàm trả về một tham chiếu đến một biến thay vì trả về một giá trị. **Giá trị trả về** của hàm là **biến**, không phải hằng hoặc biểu thức.
- Ví dụ:

```
int& Nhap(int& n) {  
    std::cout << "Nhap gia tri n = ";  
    std::cin >> n;  
    return n;  
}
```



5.9.3 Hàm trả về tham chiếu

- Hàm có thể trả về **tham chiếu** đến **tham chiếu đối số**.
- Ví dụ:

```
#include <iostream>
int& Nhap(int& n) {
    std::cout << "Nhap gia tri n = ";
    std::cin >> n;
    return n;
}
int main() {
    int n;
    Nhap(n) -= 2;
    std::cout << "Gia tri n la " << n;
    return 0;
}
```

Kết quả thực thi:

Nhap gia tri n = 5
Gia tri n la 3



5.9.3 Hàm trả về tham chiếu

- Hàm có thể **trả về tham chiếu** đến **biến tĩnh** vì biến tĩnh tồn tại trong suốt chương trình.
- Ví dụ: **Hàm sau đúng**

```
int& Nhap() {  
    static int y;  
    cin >> y;  
    return y;  
}
```



5.9.3 Hàm trả về tham chiếu

- Hàm không trả về tham chiếu đến biến cục bộ bên trong hàm hoặc tham số (truyền giá trị), vì các biến đó sẽ bị hủy sau khi hàm kết thúc.
- Ví dụ: 2 hàm sau đây đều sai

```
int& Nhap() {  
    int y;  
    cin >> y;  
    return y;  
}
```

```
int& Nhap(int y) {  
    cin >> y;  
    return y;  
}
```



5.9.3 Hàm trả về tham chiếu

- Ví dụ:

```
#include <iostream>
```

```
int& Decrement(int& num) { return --num; }
```

```
int main() {
```

```
    int a = 5;    int& b = Decrement(a);
```

```
    std::cout << "a: " << a << std::endl;
```

```
    std::cout << "b: " << b << std::endl;
```

```
    b += 6;
```

```
    std::cout << "a: " << a << std::endl;
```

```
    std::cout << "b: " << b << std::endl;
```

```
    return 0;
```

```
}
```

Kết quả thực thi:

a: 4

b: 4

a: 10

b: 10



5.9.3 Hàm trả về tham chiếu

- Ví dụ: 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;  
}
```



5.9 Giá trị trả về của hàm

5.9.4 Hàm trả về con trỏ



5.9.4 Hàm trả về con trỏ

- **Giá trị trả về của hàm** chỉ có thể là **địa chỉ của biến**, không phải hằng hoặc biểu thức (không có địa chỉ).

- Ví dụ:

```
#include <iostream>
```

```
int* Nhap(int& n) {
```

```
    std::cout << "Nhap gia tri n = "; std::cin >> n;
```

```
    return &n;
```

```
}
```

```
int main() {
```

```
    int n; int m=*Nhap(n);
```

```
    std::cout << "Gia tri nhap la " << m << std::endl;
```

```
    return 0;
```

```
}
```



5.9.4 Hàm trả về con trỏ

- Lưu ý: Hàm không thể trả về địa chỉ của biến cục bộ bên trong hàm.
- Ví dụ: **Hàm sau sẽ lỗi**

```
#include <iostream>
using namespace std;
int* func( ) {
    int n=10;
    return &n;
}
int main() {
    cout << *func();
    return 0;
}
```




5.9.4 Hàm trả về con trỏ

- Hàm trả về địa chỉ thường được sử dụng để trả về địa chỉ vùng nhớ được cấp phát động. **Sẽ được học ở bài con trỏ.**

```
#include <iostream>
using namespace std;
int* Nhap() {
    int* n=new int; // Cấp phát động
    cout << "Nhập giá trị: "; cin >> *n;
    return n;
}
int main() {
    cout << "Giá trị nhập là " << * Nhap();
    return 0;
}
```



5.9 Giá trị trả về của hàm

5.9.5 Giá trị trả về trong hàm main



5.9.5 Giá trị trả về trong hàm main

- Giá trị trả về của hàm `main` trong C++ thường là một số nguyên, được dùng để báo kết quả của chương trình khi kết thúc. Giá trị trả về này có ý nghĩa đặc biệt đối với hệ điều hành và các chương trình gọi (caller).
- Các giá trị trả về thường dùng
 - `0`: Hàm `main` đã kết thúc thành công
 - Một số khác `0`: Chương trình đã gặp lỗi

• Ví dụ:

```
#include <iostream>
int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```



Bài tập

- Câu 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.



Bài tập

- Câu 2: Làm lại các bài tập chương câu lệnh lặp:
- Câu 3: 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



Chúc các em học tốt!

