

Hàm

Lập trình nâng cao

Outline

- Mục đích sử dụng
- Cách sử dụng
- Cơ chế truyền tham số
 - Truyền giá trị - Pass-by-value
 - Truyền tham chiếu - Pass-by-reference
- Biến địa phương và tổ chức bộ nhớ
-

Hàm

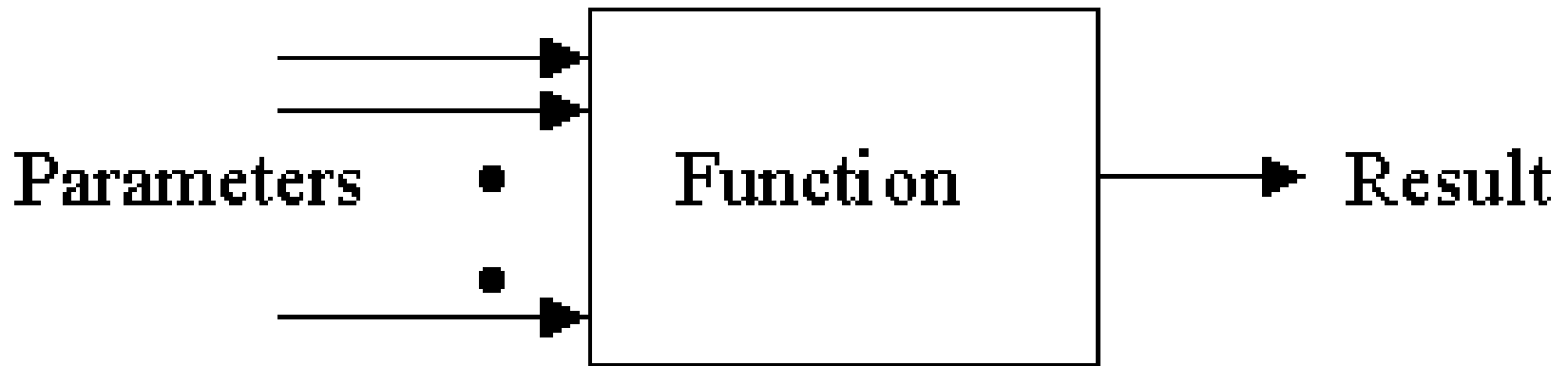
- Để làm gì?
 - Chia bài toán lớn thành các bài toán nhỏ hơn
 - Tách khái niệm ra khỏi cài đặt
 - Bạn có phải biết code của hàm `sqrt()`?
 - Chương trình dễ hiểu hơn
 - Tránh code lặp đi lặp lại
 - Tái sử dụng
- Lập trình có cấu trúc – **structured** programming

Internal vs. External function

- Internal : bạn tự định nghĩa
- External : ví dụ `abs`, `sqrt`, `exp`... được nhóm thành các thư viện `math`, `iostream`, `stdlib`...

Input/output

Các tham số $\rightarrow$ hàm $\rightarrow$ giá trị trả về



Hàm đặt sau main cần có function prototype đặt trước

```
int absolute(int x); // function prototype
```

```
int main() { ...  
    a = absolute(b); // function use  
}
```

```
int absolute(int x) { // function definition  
    if (x >= 0) return x;  
    else return -x;  
}
```

Hàm đặt trước không cần prototype

```
int absolute(int x) { // function definition
    if (x >= 0) return x;
    else return -x;
}

int main() { ...
    a = absolute(b); // function use
}
```

Cú pháp định nghĩa hàm

```
<return type> <function name>(<parameter list>) {  
    <local declarations>  
    <sequence of statements>  
}
```

```
int absolute(int x) {  
    if (x >= 0) return x;  
    else return -x;  
}
```

Cú pháp khai báo prototype hàm

<return type> <function name>(<parameter list>);

```
int absolute(int x) ;
```

Truyền tham số - pass-by-value

```
int argument1;  
double argument2;  
// function call (in another function, such as main)  
result = thefunctionname(argument1, argument2);
```

copy giá trị



```
// function definition  
int thefunctionname(int parameter1, double parameter2){  
// Now the function can use the two parameters  
// parameter1 = argument 1, parameter2 = argument2
```

pass-by-value

```
void swap(int x, int y) {  
    int t = x; x = y; y = t;  
}
```

```
int main() {  
    int a = 2;  
    int b = 3;
```

2,3

```
    swap(a,b);
```

```
    cout << a << ", " << b;
```

```
}
```

Sai! Vì x, y là
bản sao của a, b

pass-by-reference

```
void swap(int& x, int& y) {  
    int t = x; x = y; y = t;  
}
```

```
int main() {  
    int a = 2;  
    int b = 3;
```

```
    swap(a,b);
```

```
    cout << a << ", " << b;
```

```
}
```

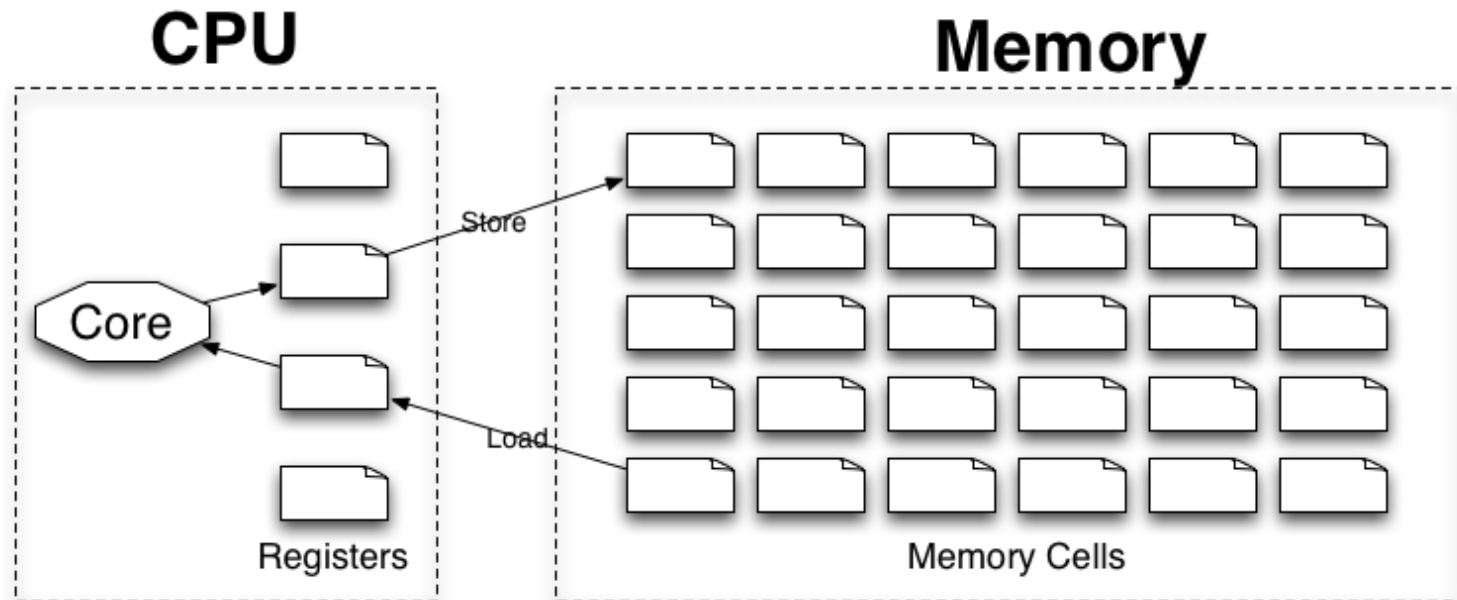
3, 2

Đúng. Vì x, y là
tham chiếu tới a, b

Cấu trúc bộ nhớ

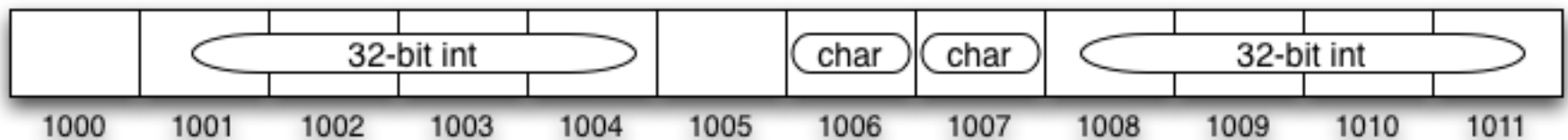
CPU và Bộ nhớ

- CPU tính toán với dữ liệu ghi tại các thanh ghi
- Dữ liệu được chuyển qua lại giữa bộ nhớ và các thanh ghi



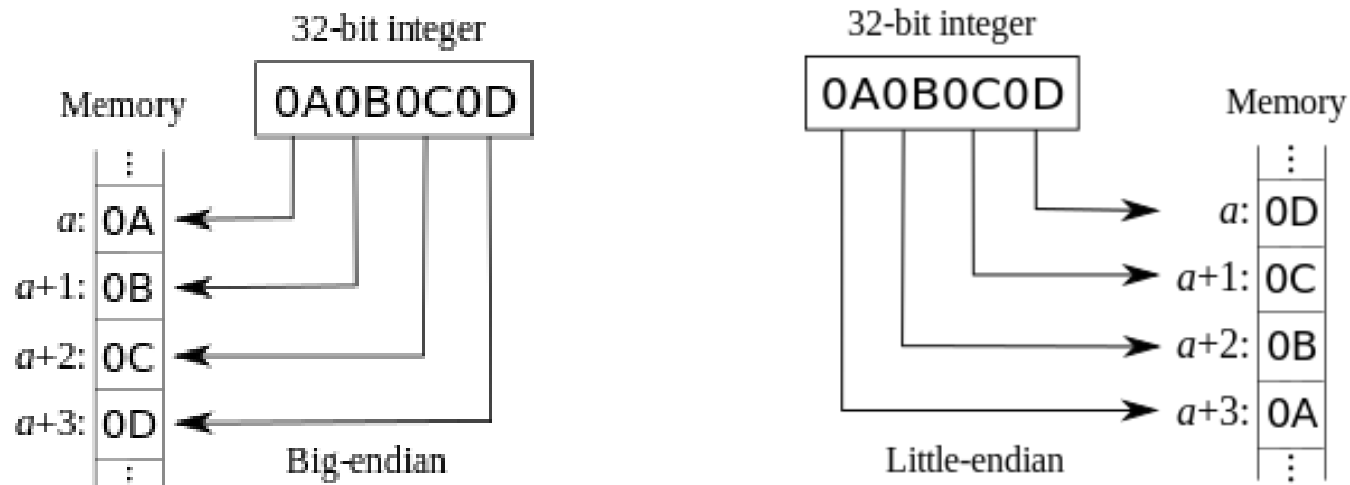
Lưu dữ liệu trong bộ nhớ

- Kích thước mỗi ô là 8 bit – 1 byte
- Các kiểu dữ liệu lớn cần một chuỗi byte liên tiếp, xác định bởi
 1. địa chỉ byte đầu tiên, và
 2. kích thước



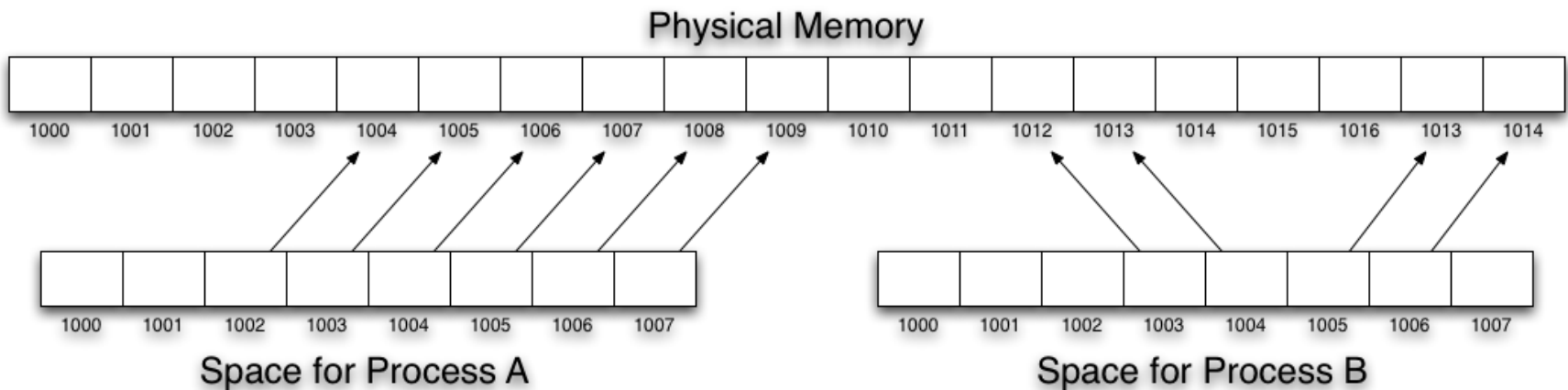
Bit $\leftrightarrow$ giá trị dữ liệu

- Thứ tự byte mã hóa và giải mã cần nhất quán
- Big-endian: từ trái sang phải, địa chỉ các byte tăng dần (mainframe, IPv4...)
- Little-endian: từ trái sang phải, địa chỉ các byte giảm dần (Intel x86, x86-64)



Bộ nhớ ảo – virtual memory

- Mỗi tiến trình (chương trình đang chạy) được phân một không gian bộ nhớ riêng
 - Hệ điều hành ánh xạ một phần bộ nhớ logic với bộ nhớ vật lý
 - Địa chỉ trong các không gian khác nhau là độc lập

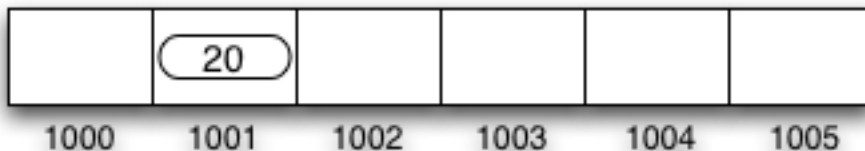


Biến và các lời gọi hàm

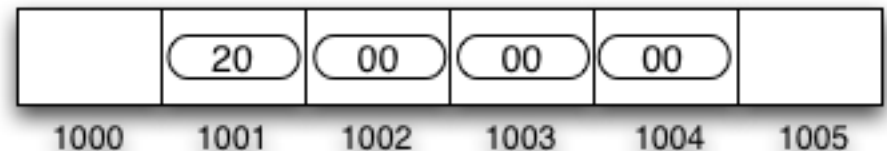
Biến

- Biến là tên gọi của một vùng bộ nhớ cụ thể
 - Có thể đọc và ghi nội dung
- Kiểu dữ liệu (data type): dùng để đọc lấy giá trị của biến
 - Biến gồm bao nhiêu ô nhớ
 - Tính giá trị biến từ giá trị các ô nhớ bằng cách nào

`char a = 0x20`



`int a = 0x20`



Cuộc đời của biến địa phương

- Được khai báo trong một khối lệnh
- Cuộc đời và phạm vi hiệu lực tương ứng với khối lệnh đó

```
int main (int argc, const char * argv[])
{
    int a = 20, b = 25;

    {
        int g;
        g = gcd(a,b);
        printf("GCD(%d,%d)=%d\n", a, b, g);
    }

    printf("GCD(%d,%d)=%d\n", a, b, g);

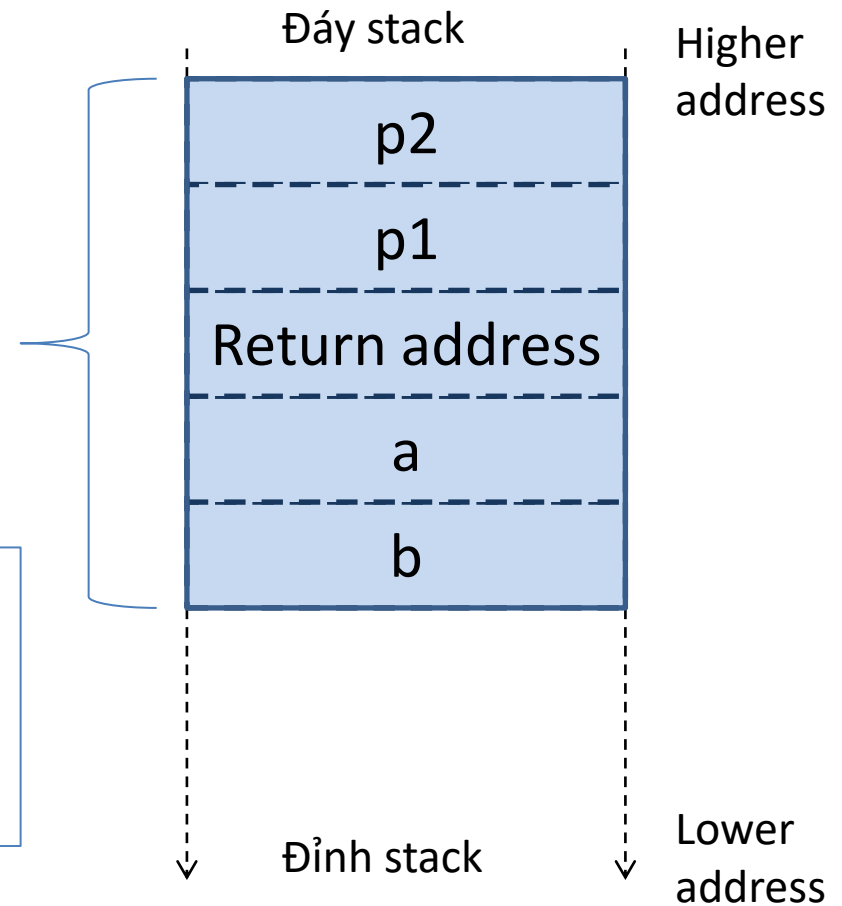
    return 0;
}
```

✖ error: 'g' undeclared (first use in this function)

Biến trong vùng bộ nhớ của lời gọi hàm – stack frame

```
int f(int p1, int p2) {  
    int a, b;  
    return a+b;  
}
```

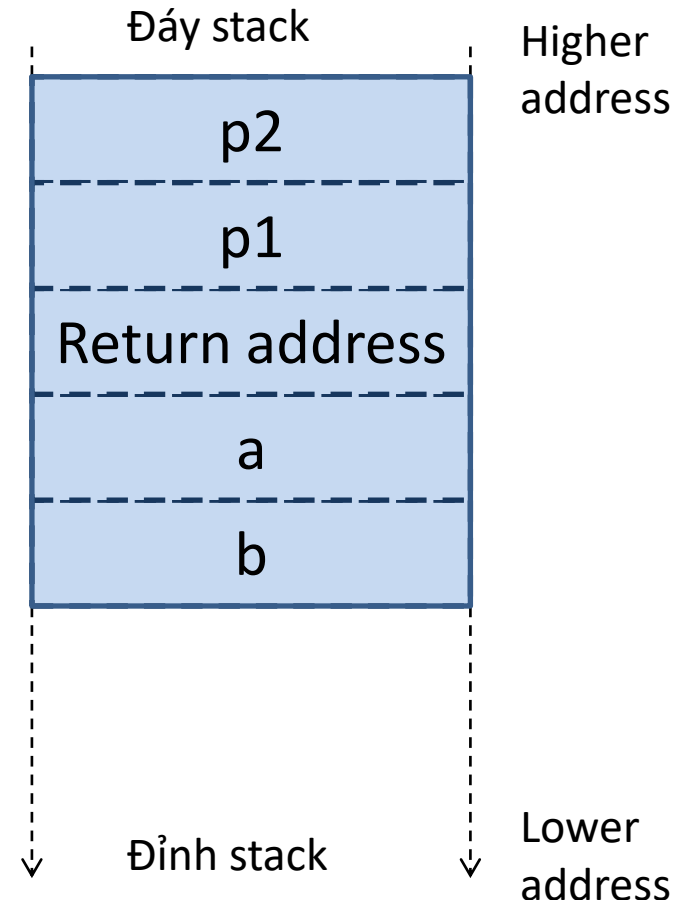
Stack frame của f()



Khi một lời gọi hàm được chạy

- Bản sao của các đối số được đẩy vào stack. Đó là các tham số.
- Địa chỉ lưu giá trị trả về được đẩy vào stack
- Các biến địa phương được cấp phát trong stack

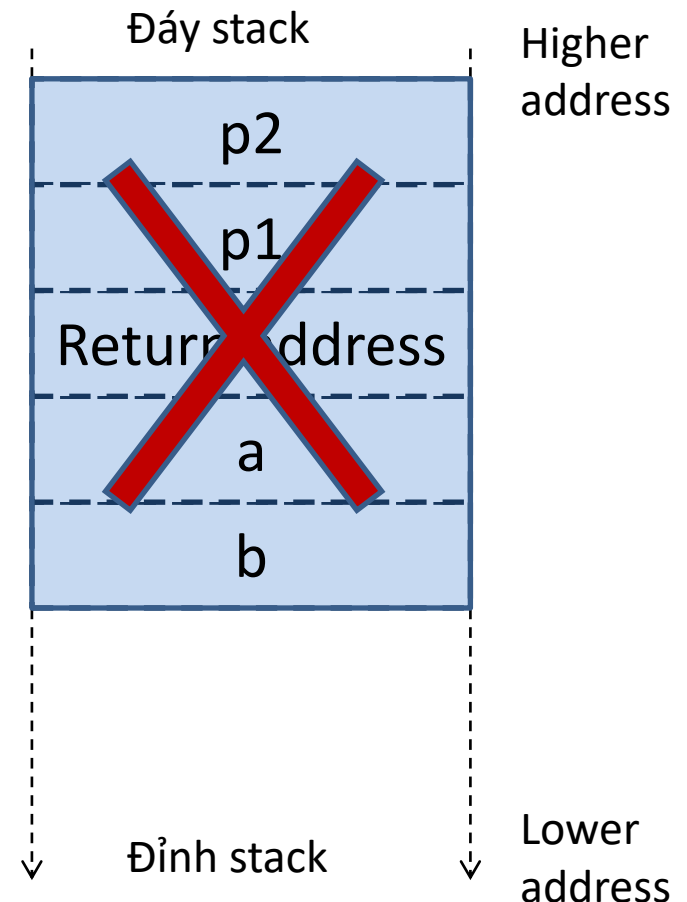
```
int f(int p1, int p2) {  
    int a, b;  
    return a+b;  
}
```



Khi hàm trả về (return)

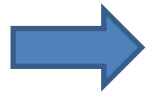
- Lưu giá trị trả về vào thanh ghi hoặc stack
- Đẩy (pop) toàn bộ frame của hàm ra khỏi stack, gồm:
 - Biến địa phương
 - Địa chỉ trả về
 - Tham số

```
int f(int p1, int p2) {  
    int a, b;  
    return a+b;  
}
```

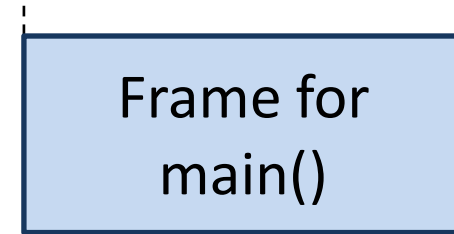


Function call stack

(Stack frame cho lời gọi hàm)



```
int main() {  
    a();  
    return 0;  
}  
  
int a() {  
    b();  
    c();  
    return 0;  
}  
  
int b() {return 0;}  
int c() {return 0;}
```

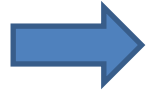


Higher
address

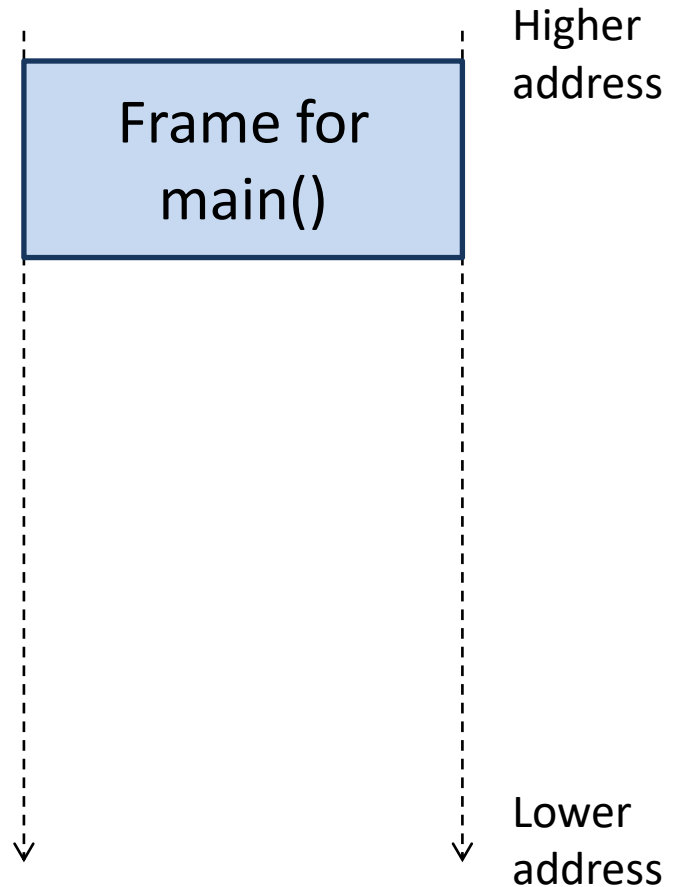
Lower
address

Stack memory

Function call stack



```
int main() {  
    a();  
    return 0;  
}  
  
int a() {  
    b();  
    c();  
    return 0;  
}  
  
int b() {return 0;}  
int c() {return 0;}
```



Stack memory

Function call stack

```
int main() {  
    a();  
    return 0;  
}
```

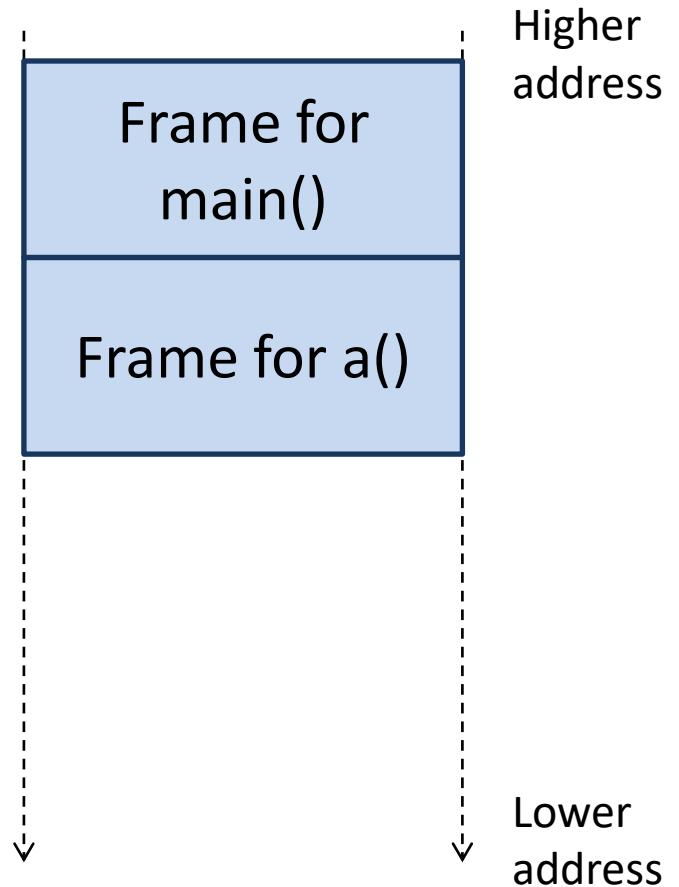
main()
gọi a()



```
int a() {  
    b();  
    c();  
    return 0;  
}
```

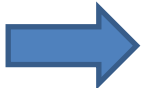
```
int b() {return 0;}
```

```
int c() {return 0;}
```

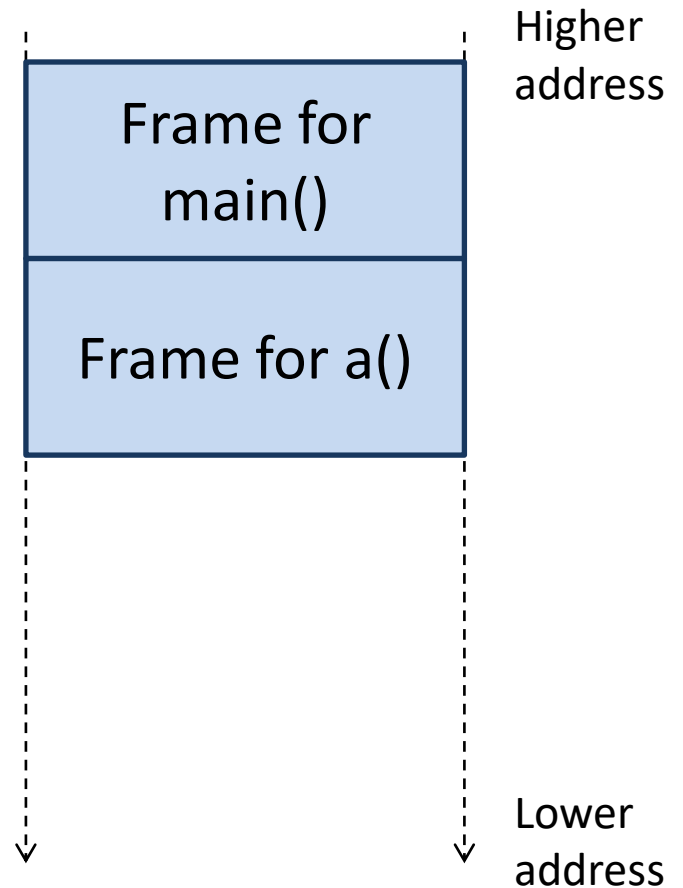


Stack memory

Function call stack



```
int main() {  
    a();  
    return 0;  
}  
  
int a() {  
    b();  
    c();  
    return 0;  
}  
  
int b() {return 0;}  
int c() {return 0;}
```

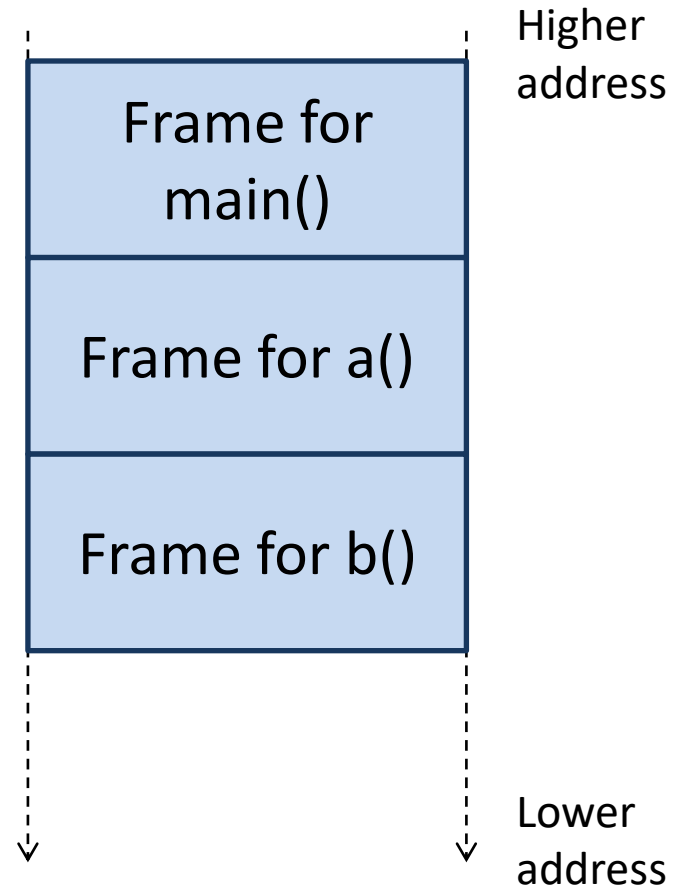


Stack memory

Function call stack

```
int main() {  
    a();  
    return 0;  
}  
  
int a() {  
    b();  
    c();  
    return 0;  
}  
  
int b() {return 0;}  
int c() {return 0;}
```

→
a() gọi b()



Stack memory

Function call stack

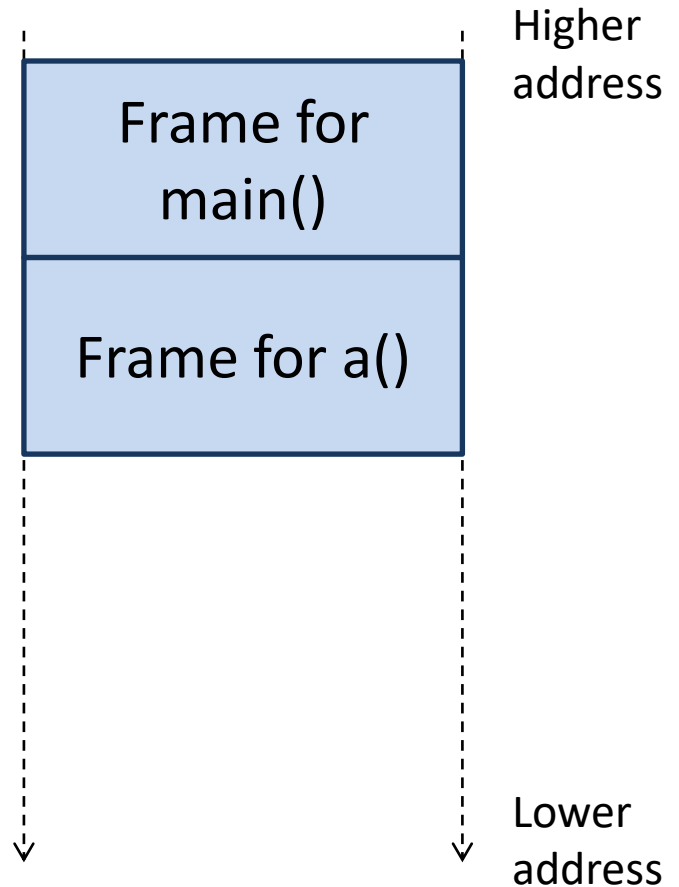
```
int main() {  
    a();  
    return 0;  
}
```

```
int a() {  
    b();  
    c();  
    return 0;  
}
```

```
int b() {return 0;}
```

```
int c() {return 0;}
```

b() trả về

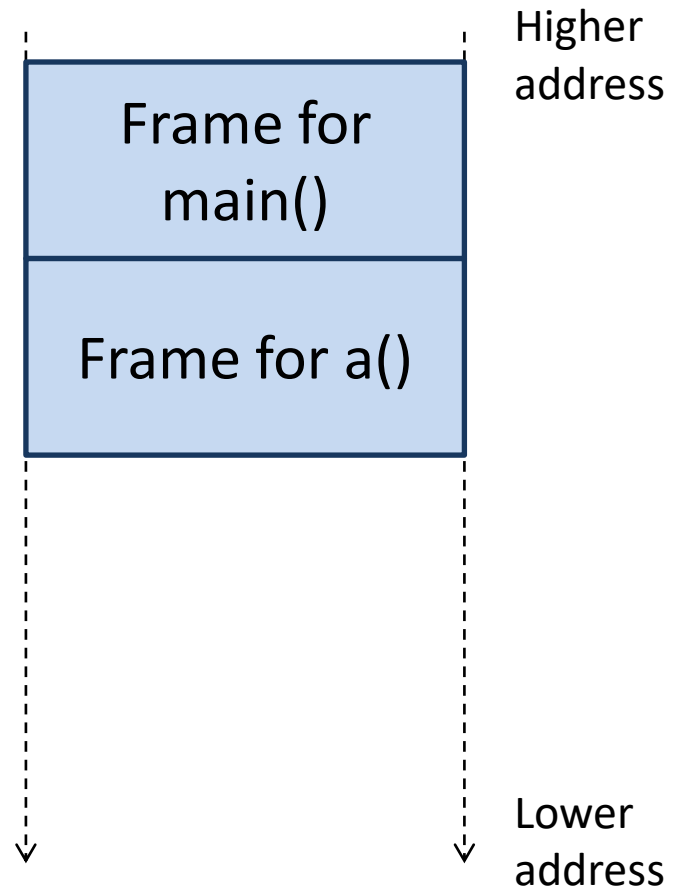


Stack memory

Function call stack



```
int main() {  
    a();  
    return 0;  
}  
  
int a() {  
    b();  
    c();  
    return 0;  
}  
  
int b() {return 0;}  
int c() {return 0;}
```

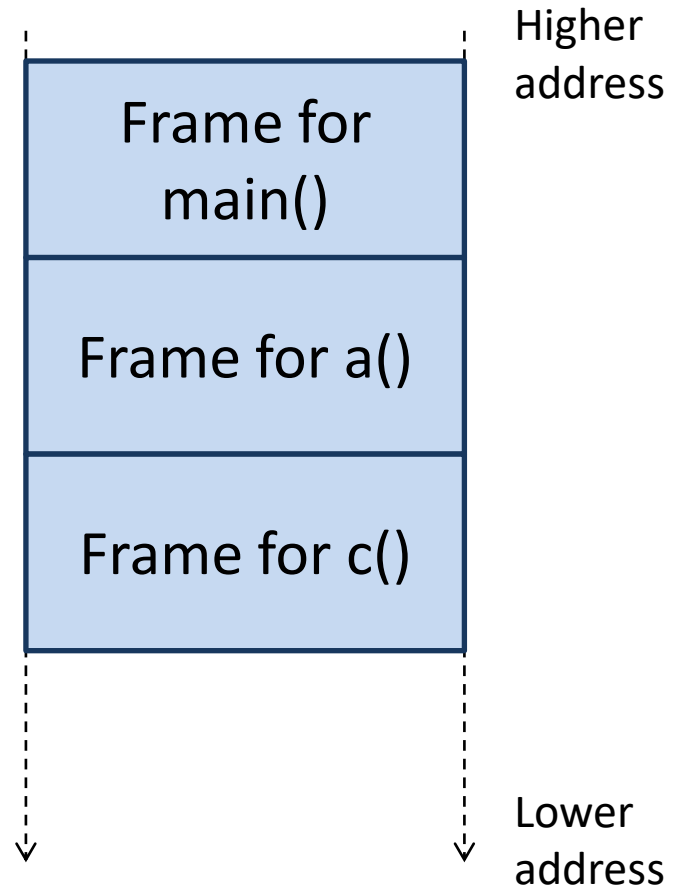


Stack memory

Function call stack

```
int main() {  
    a();  
    return 0;  
}  
  
int a() {  
    b();  
    c();  
    return 0;  
}  
  
int b() {return 0;}  
int c() {return 0;}
```

a() gọi c()



Stack memory

Function call stack

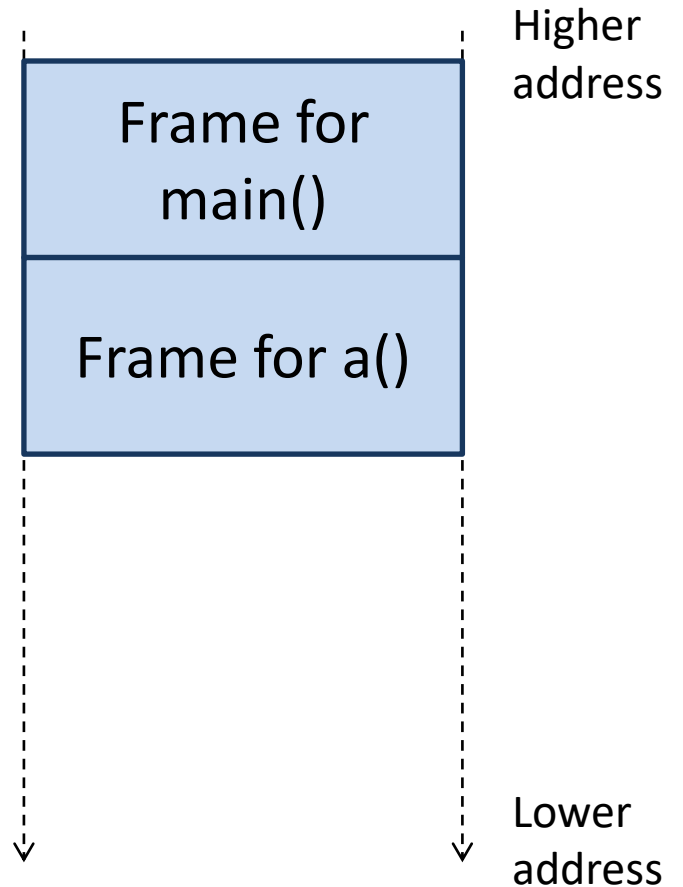
```
int main() {  
    a();  
    return 0;  
}
```

```
int a() {  
    b();  
    c();  
    return 0;  
}
```

```
int b() {return 0;}
```

```
int c() {return 0;}
```

c() trả về

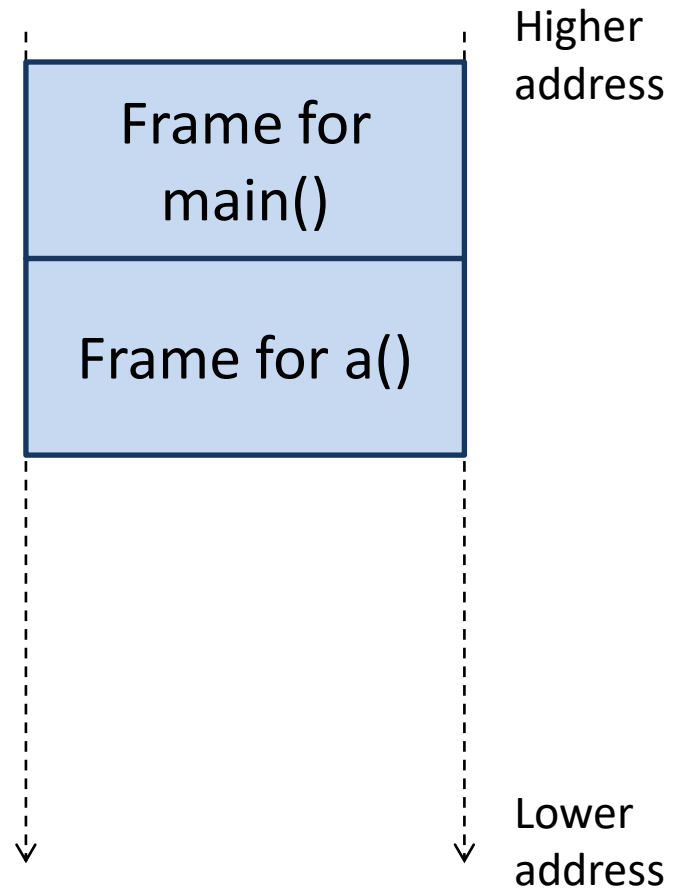


Stack memory

Function call stack

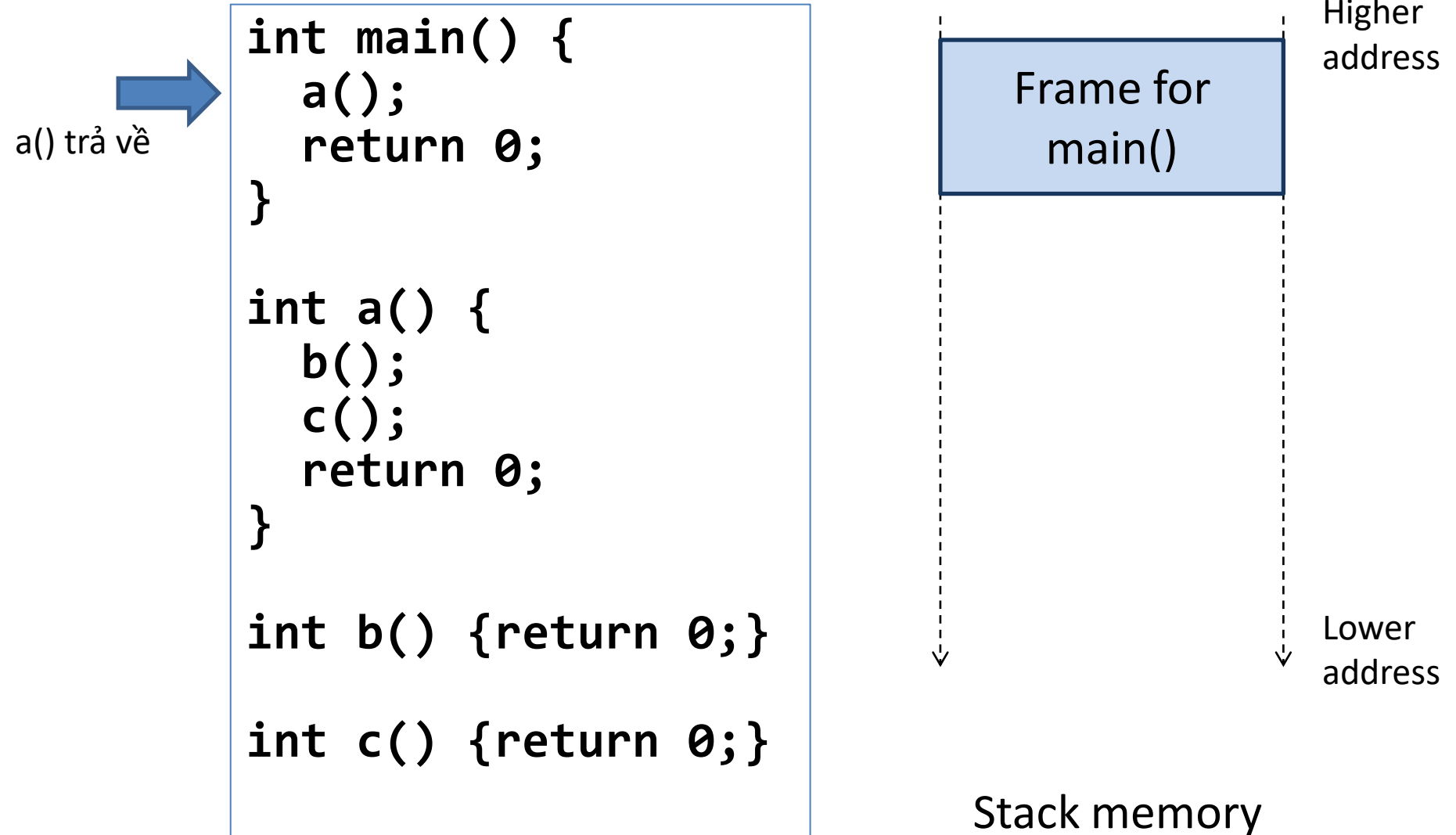


```
int main() {  
    a();  
    return 0;  
}  
  
int a() {  
    b();  
    c();  
    return 0;  
}  
  
int b() {return 0;}  
int c() {return 0;}
```



Stack memory

Function call stack

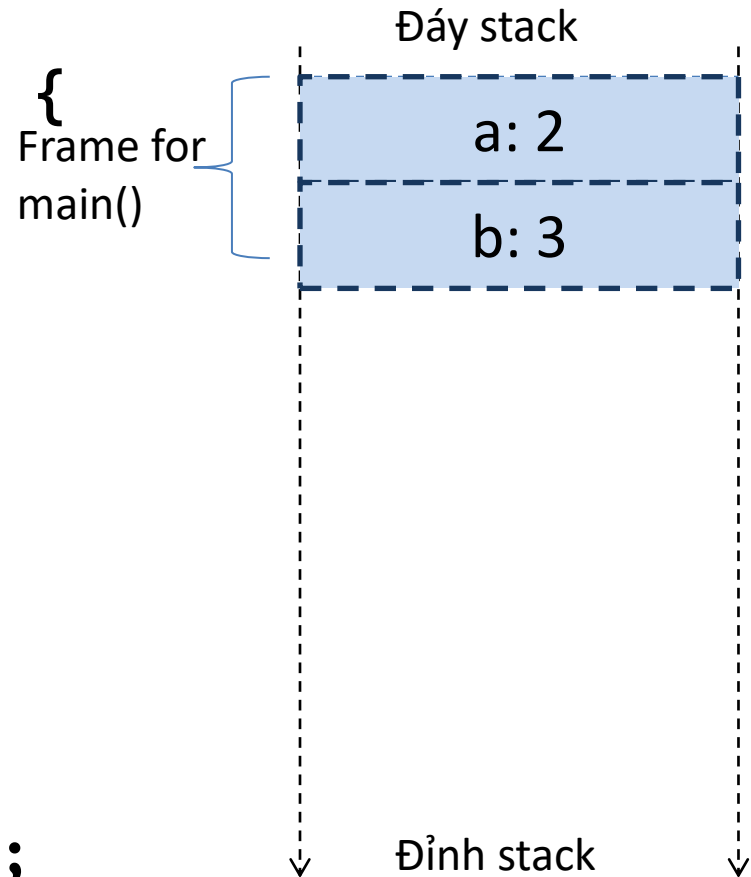


Cơ chế truyền tham số

Tại sao swap(int x, int y) không có tác dụng

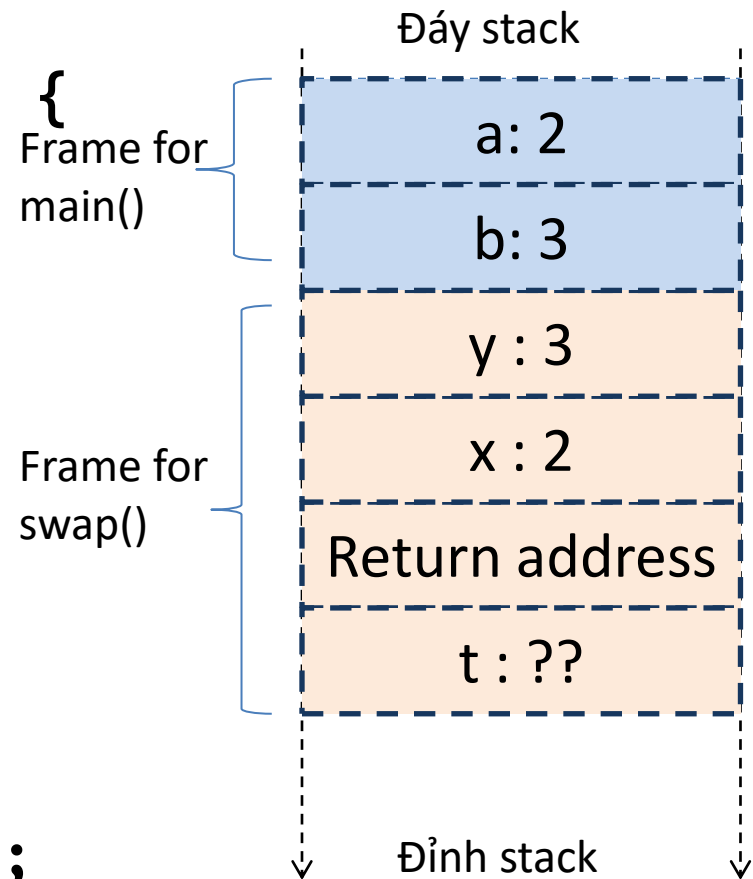
```
void swap(int x, int y) {  
    int t;  
    t = x;  
    x = y;  
    y = t;  
}
```

```
int main() {  
    int a = 2;  
    int b = 3;  
    swap(a,b);  
    cout << a << ", " << b;  
}
```



Tại sao swap(int x, int y) không có tác dụng

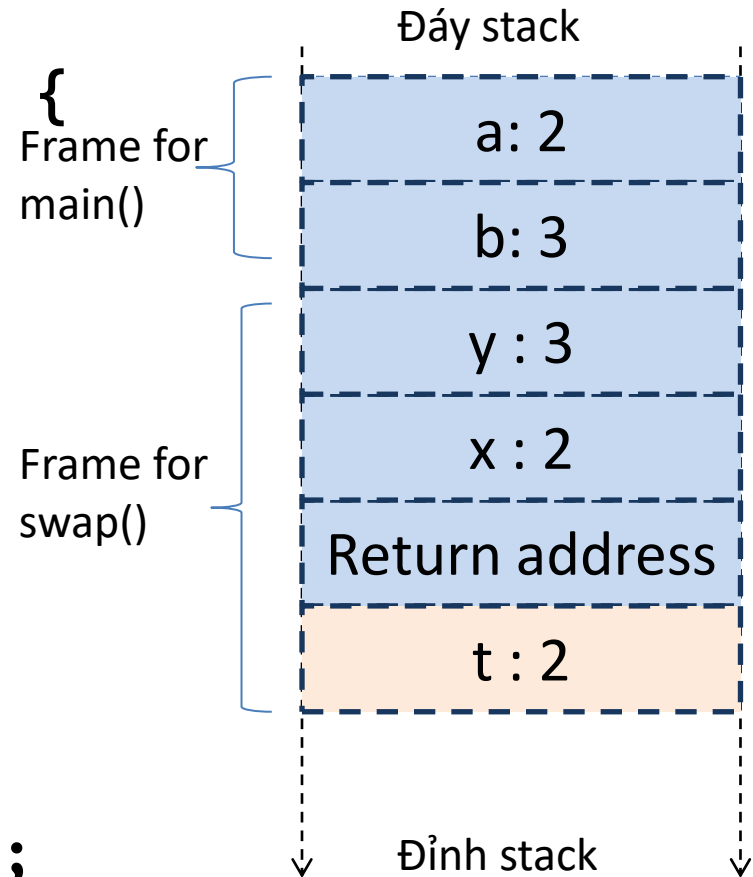
```
void swap(int x, int y) {  
    int t;  
    t = x;  
    x = y;  
    y = t;  
}  
  
int main() {  
    int a = 2;  
    int b = 3;  
    swap(a,b);  
    cout << a << ", " << b;  
}
```



Tại sao swap(int x, int y) không có tác dụng

```
void swap(int x, int y) {  
    int t;  
    t = x;  
    x = y;  
    y = t;  
}
```

```
int main() {  
    int a = 2;  
    int b = 3;  
    swap(a,b);  
    cout << a << ", " << b;  
}
```

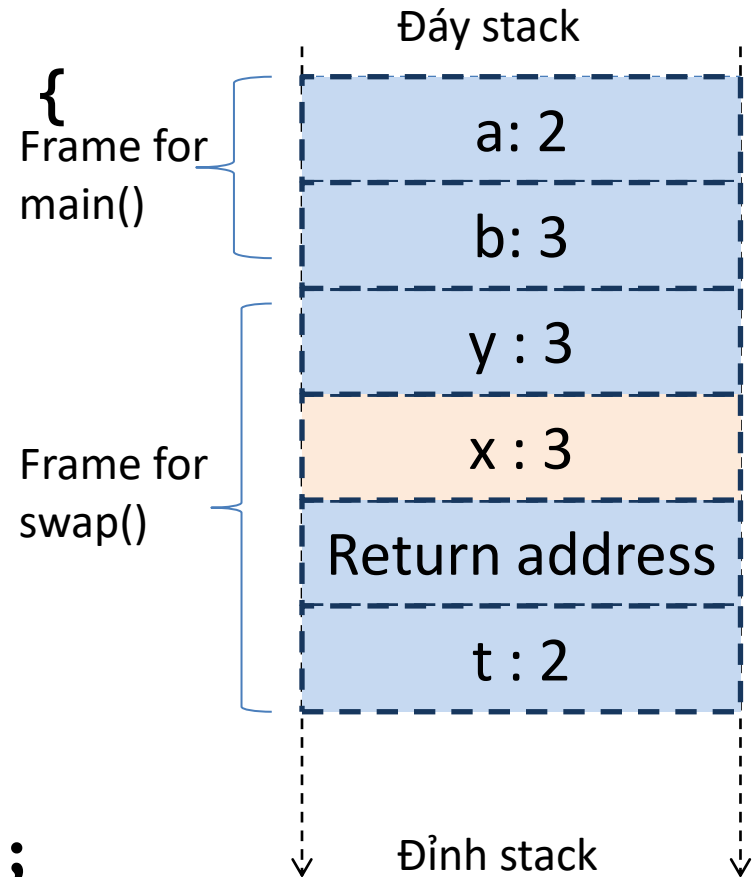


Tại sao swap(int x, int y) không có tác dụng

```
void swap(int x, int y) {  
    int t;  
    t = x;  
    x = y;  
    y = t;  
}
```



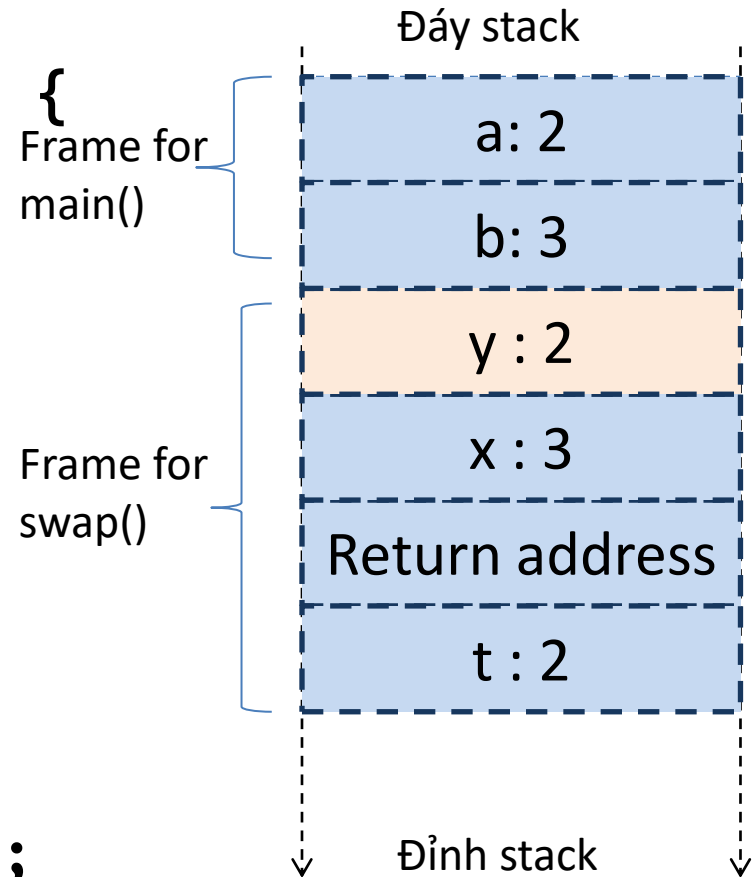
```
int main() {  
    int a = 2;  
    int b = 3;  
    swap(a,b);  
    cout << a << ", " << b;  
}
```



Tại sao swap(int x, int y) không có tác dụng

```
void swap(int x, int y) {  
    int t;  
    t = x;  
    x = y;  
    y = t;  
}
```

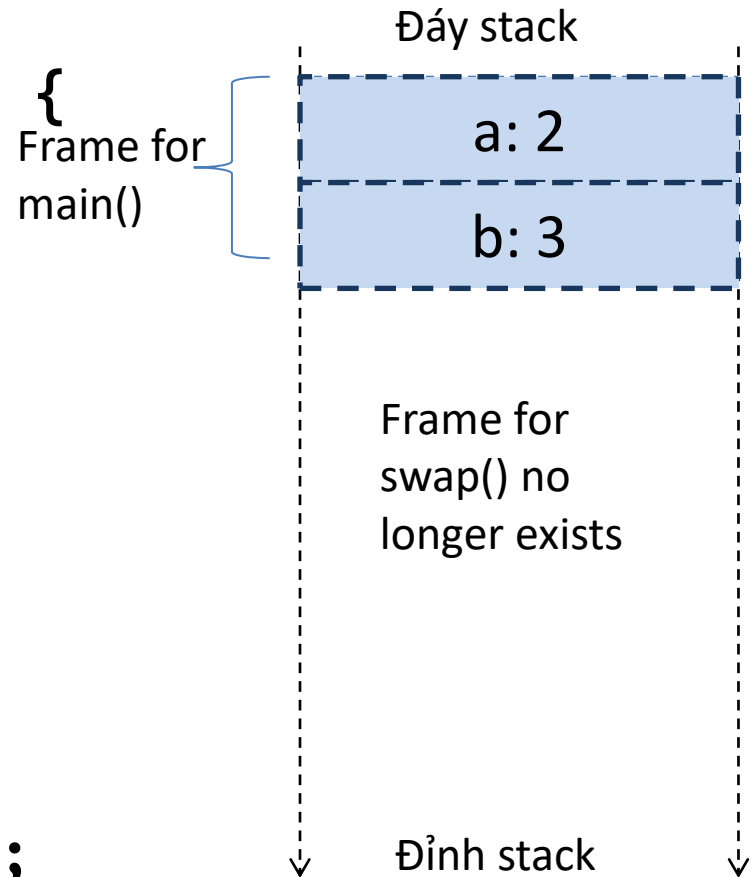
```
int main() {  
    int a = 2;  
    int b = 3;  
    swap(a,b);  
    cout << a << ", " << b;  
}
```



Tại sao swap(int x, int y) không có tác dụng

```
void swap(int x, int y) {  
    int t;  
    t = x;  
    x = y;  
    y = t;  
}
```

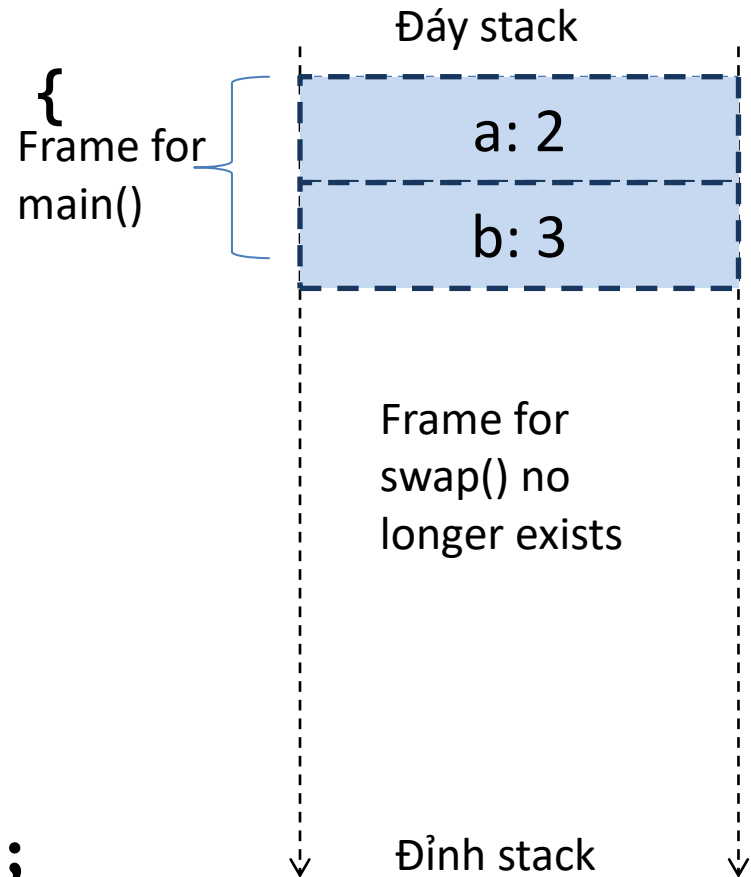
```
int main() {  
    int a = 2;  
    int b = 3;  
    swap(a,b);  
    cout << a << ", " << b;  
}
```



Tại sao swap(int x, int y) không có tác dụng

```
void swap(int x, int y) {  
    int t;  
    t = x;  
    x = y;  
    y = t;  
}
```

```
int main() {  
    int a = 2;  
    int b = 3;  
    swap(a,b);  
    cout << a << ", " << b;  
}
```



swap có tác dụng

```
void swap(int& x, int& y) {  
    int t;  
    t = x;  
    x = y;  
    y = t;  
}
```

```
int main() {  
    int a = 2;  
    int b = 3;  
    swap(a,b);  
    cout << a << ", " << b;  
}
```

Cơ chế truyền tham số

- **Pass-by-value :**

int f (**int** x, **int** y)

tham số là bản sao của đối số

sửa tham số không ảnh hưởng đến đối số

x, y còn gọi là tham trị

- **Pass-by-reference :**

int f (**int&** x, **int&** y)

tham số là nickname của đối số

sửa tham số chính là sửa đối số

x, y còn gọi là tham biến

Đối số / Tham số

Tham trị / Tham biến

- Argument - Đối số: a, b
- Parameter - Tham số: x, y
 - Tham trị: x
 - Tham biến: y

```
void f(int x) { ... }  
void g(int& y) { ... }
```

```
int main() {  
    int a = 2;  
    int b = 3;  
    f(a);  
    g(b);  
}
```

Hằng tham số

```
void f (const string x)
```

```
void g (const string& y)
```

f không được sửa x, g không được sửa y

Hằng tham số - best practice

```
void f (const string x)
```

```
void g (const string& y)
```

Nên dùng const cho tất cả các tham số
không nên bị sửa giá trị.

Lý do: đề phòng lỗi của lập trình viên

Hằng tham biến – performance

```
void f (string x)
```

```
void f (const string& y)
```

- Hằng tham biến và tham trị tương đương về hiệu ứng phụ:
đảm bảo hàm f không sửa giá trị đối số.
- Với kiểu dữ liệu lớn, hằng tham biến cho hiệu năng tốt hơn do không phải sao chép giá trị

Biến tham chiếu - reference

```
int a = 1;  
int& b = a;  
b++;  
cout << a << " " << b; // 2 2
```

- b được khai báo là tham chiếu tới a
- b thực chất chỉ là một nick khác của a

Giá trị mặc định của tham số

```
int divide (int a, int b = 2) {  
    int r;  
    r = a/b;  
    return r;  
}
```

Tham số b có giá trị mặc định bằng 2.
divide(12) thực chất là divide(12,2)

Chỉ áp dụng được cho các tham số cuối

```
int main () {  
    cout << divide (12) << '\n';  
    cout << divide (20,4) << '\n';  
    return 0;  
}
```

6

5

Hàm đệ quy

Hàm đệ quy

- Hàm gọi chính nó

```
long factorial(long x) {  
    if (x > 1)  
        return (x * factorial(x-1));  
    else  
        return 1;  
}  
  
int main() {  
    long number = 9;  
    cout << number << "! = " << factorial(number);  
    return 0;  
}
```

Hàm đệ quy

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

- Trường hợp cơ bản: $a \leq 1$: $f(x) = 1$
- Trường hợp thường: $a > 1$
Công thức đệ quy: $f(x) = x * f(x-1)$

Hàm đệ quy

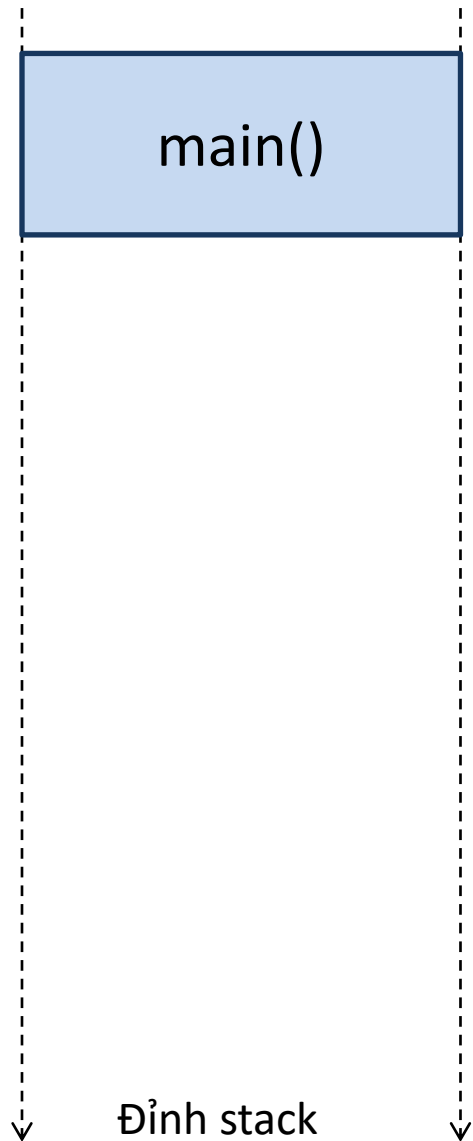
```
long factorial(long x) {  
    if (x > 1)  
        return (x * factorial(x-1));  
    else  
        return 1;  
}
```

- Đưa $f(x)$ về $f(x-1)$
- Đưa $f(x-1)$ về $f(x-2)$
- ...
- Hội tụ về một trong các trường hợp cơ bản

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

```
int main() {  
    long res = factorial(4);  
    return 0;  
}
```

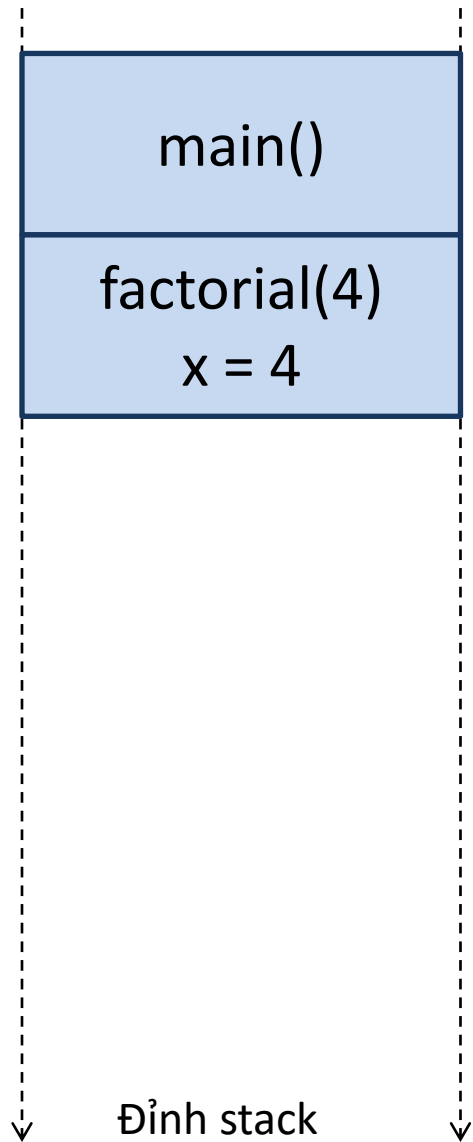
Frame for
swap()



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

```
int main() {  
    long res = factorial(4);  
    return 0;  
}
```

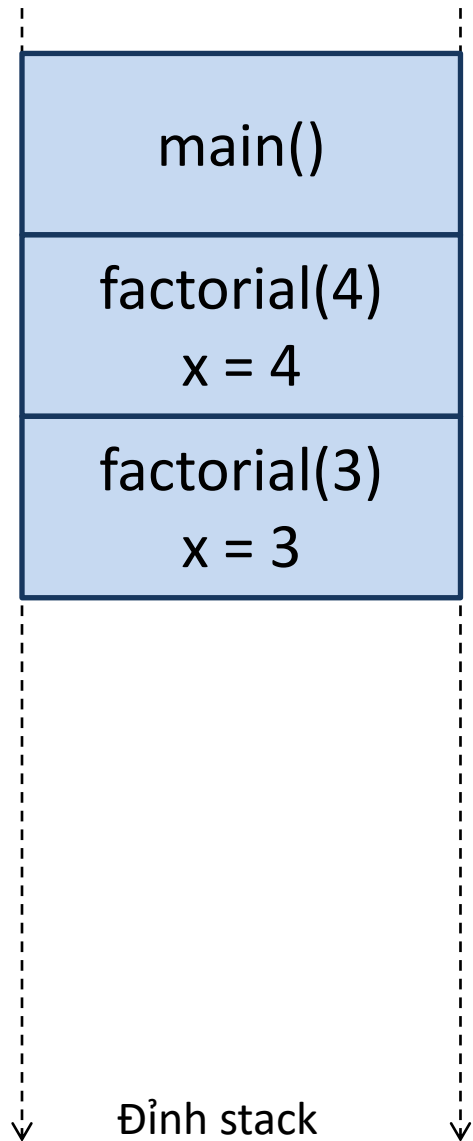
Frame for
swap()



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

```
int main() {  
    long res = factorial(4);  
    return 0;  
}
```

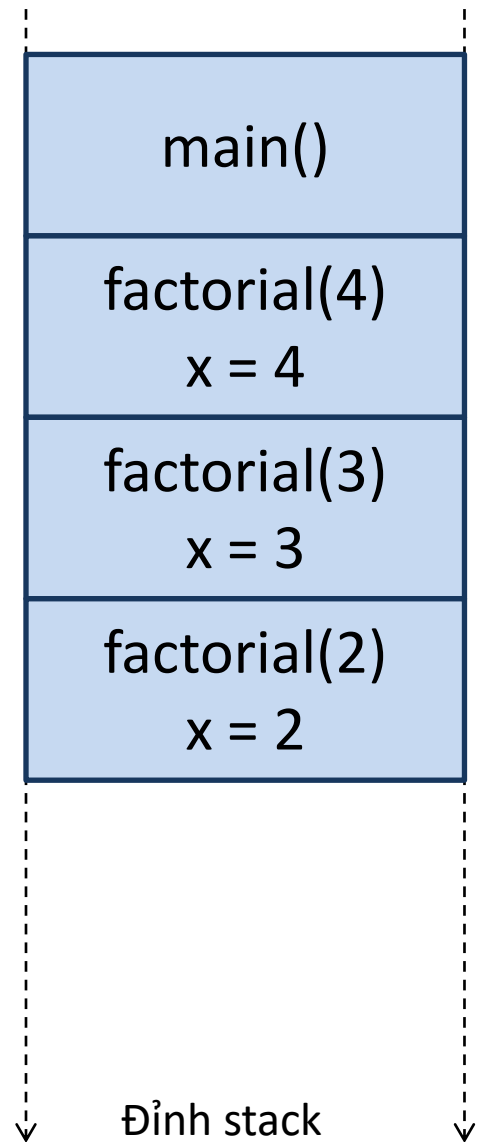
Frame for
swap()



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

```
int main() {  
    long res = factorial(4);  
    return 0;  
}
```

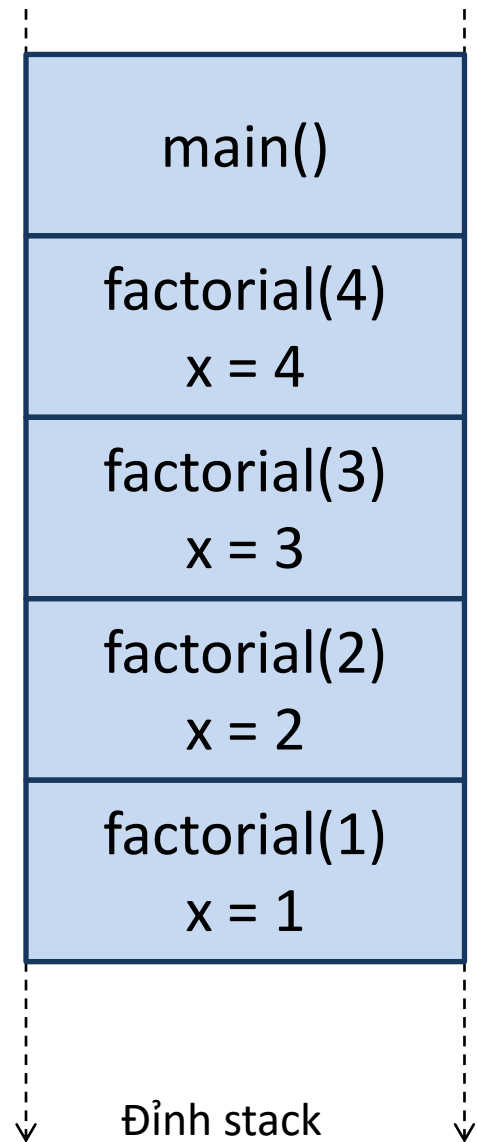
Frame for
swap()



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

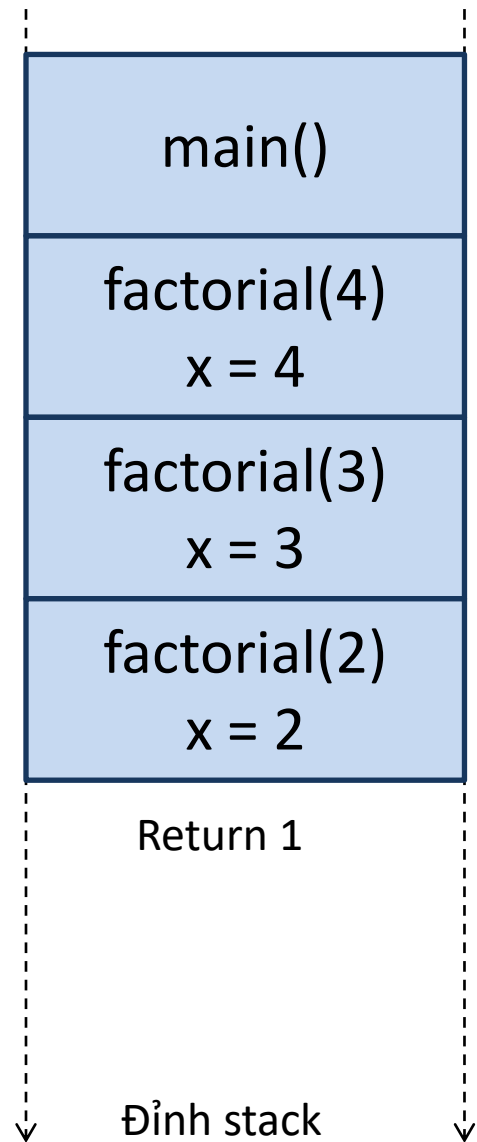
```
int main() {  
    long res = factorial(4);  
    return 0;  
}
```

Frame for
swap()



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

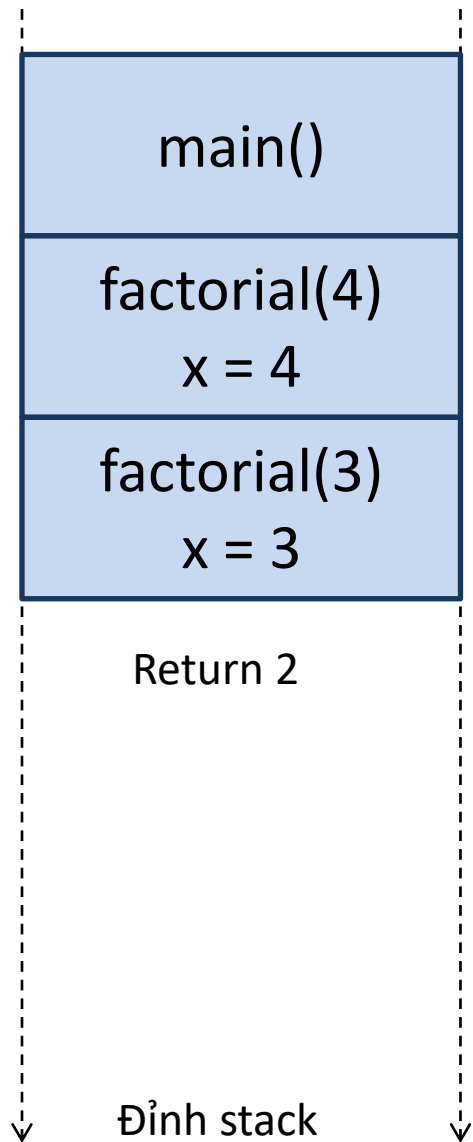
```
int main() {  
    long res = factorial(4);  
    return 0;  
}
```



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

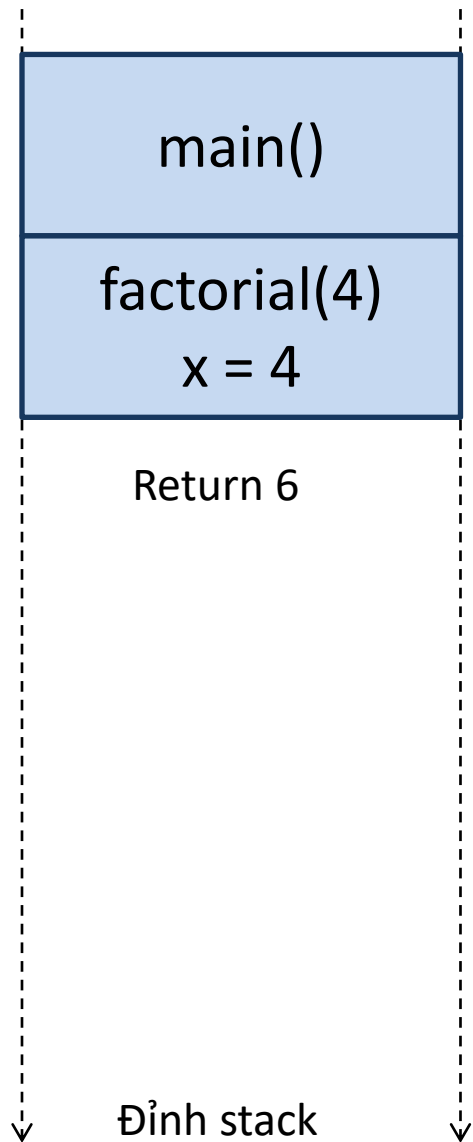
```
int main() {  
    long res = factorial(4);  
    return 0;  
}
```

Frame for
swap()



```
long factorial(long x) {  
    if (x > 1)  
        return (x * factorial(x-1));  
    else  
        return 1;  
}  
  
int main() {  
    long res = factorial(4);  
    return 0;  
}
```

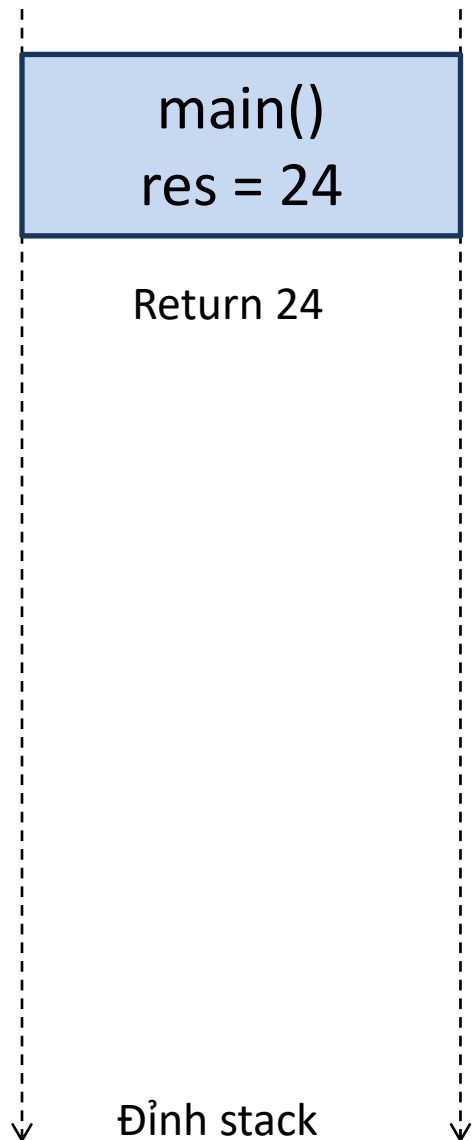
Frame for
swap()



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

```
int main() {  
    long res = factorial(4);  
    return 0;  
}
```

Frame for
swap()

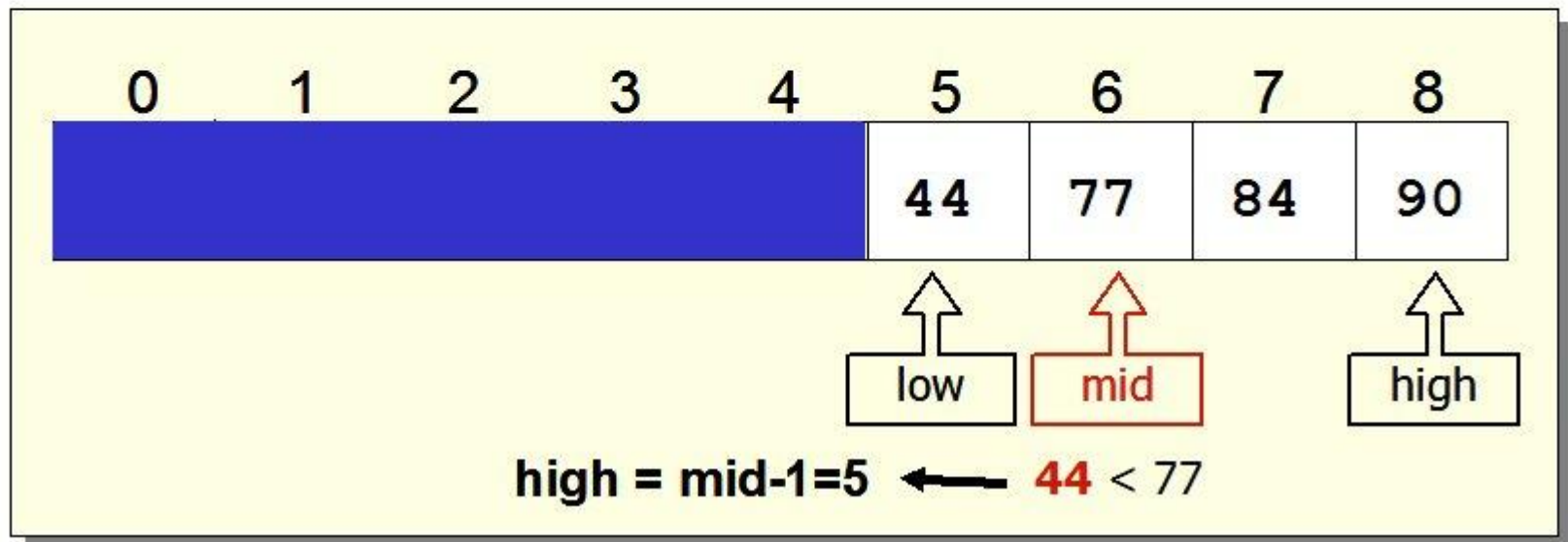


Tìm kiếm nhị phân đệ quy

	low	high	mid
#1	0	8	4
#2	5	8	6

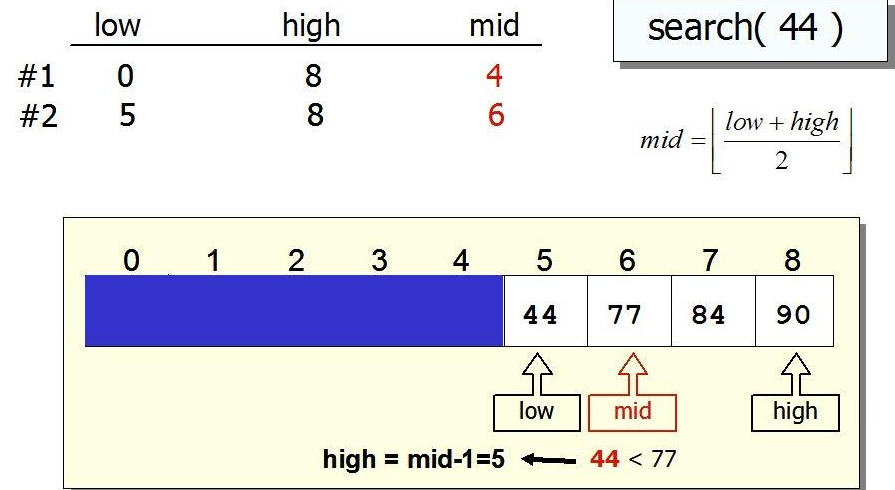
search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



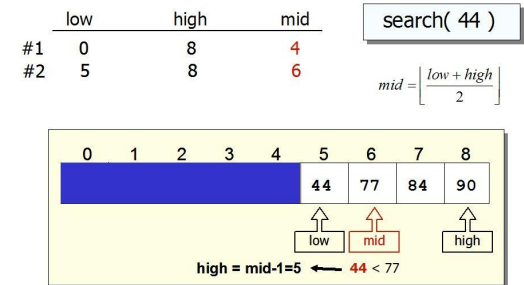
Tìm kiếm nhị phân đệ quy

- Tìm từ 0 đến 8
 - Tìm từ 5 đến 8
 - Tìm từ 5 đến 6
 - Tìm từ 5 đến 5



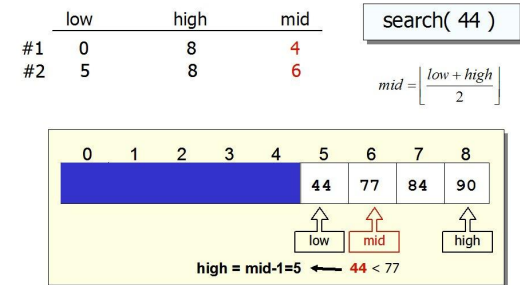
Tìm kiếm nhị phân đệ quy

- Tìm từ 0 đến 8
 - Do $a[4] < 44$ nên tìm từ 5 đến 8
 - Do $a[7] > 44$ nên tìm từ 5 đến 6
 - Do $a[6] > 44$ nên tìm từ 5 đến 5
 - Trường hợp cơ bản: $a[5] = 44$



Tìm kiếm nhị phân đệ quy

- Công thức đệ quy:
 - if $a[mid] < key$: $search(mid+1, high)$
 - if $a[mid] > key$: $search(low, mid-1)$
- Trường hợp cơ bản #1:
 $a[mid] == key$: tìm thấy
- Trường hợp cơ bản #2:
 $low > high$: không tìm thấy



Tìm kiếm nhị phân đệ quy

- Công thức đệ quy:
 - if $a[mid] < key$: $search(mid+1, high)$
 - if $a[mid] > key$: $search(low, mid-1)$
- Trường hợp cơ bản #1: $a[mid] == key$: tìm thấy
- Trường hợp cơ bản #2: $low > high$: không tìm thấy

```
int search(int key, int a[], int low, int high) {  
    if (low > high) return -1;  
    int mid = (low + high) / 2;  
    if (a[mid] == key) return mid;  
    if (a[mid] > key)  
        return search(key, a, low, mid-1);  
    return search(key, a, mid+1, high);  
}
```

Đọc thêm

- <http://www.drdobbs.com/security/anatomy-of-a-stack-smashing-attack-and-h/240001832>

Công thức đệ quy

- $\text{factorial}(n) = \text{factorial}(n-1) * n$
- $f(n)$: số bước chuyển n đĩa từ cọc này -> cọc khác



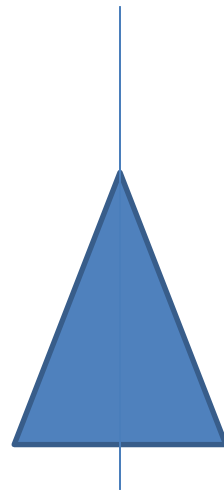
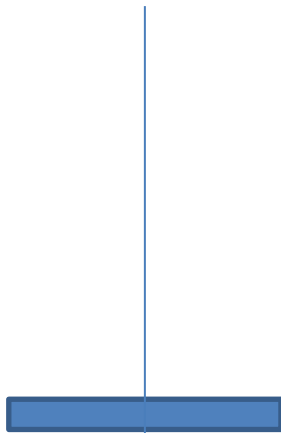
- $\text{factorial}(n) = \text{factorial}(n-1) * n$
- Tháp Hà Nội



- $\text{factorial}(n) = \text{factorial}(n-1) * n$
- Tháp Hà Nội $\text{move}(n, 1, 3)$

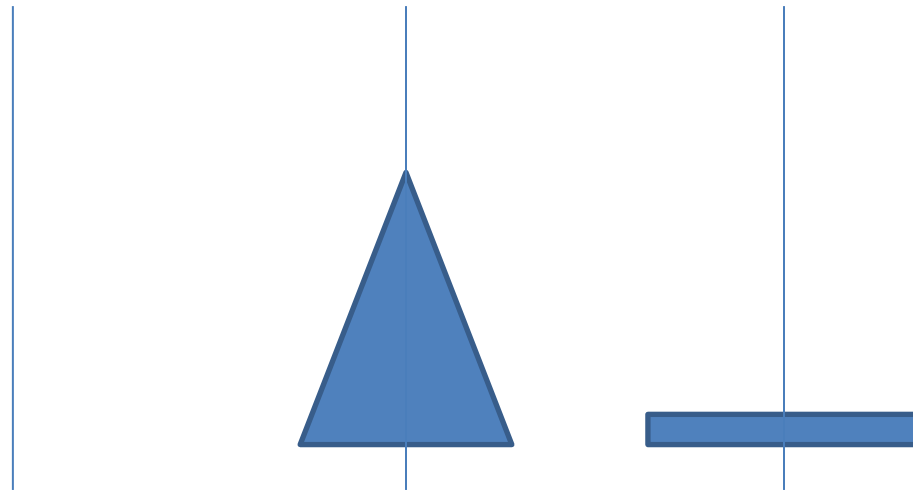


- $\text{factorial}(n) = \text{factorial}(n-1) * n$
- Tháp Hà Nội $f(n) = f(n-1) \dots$



- $\text{move}(n-1, 1, 2)$

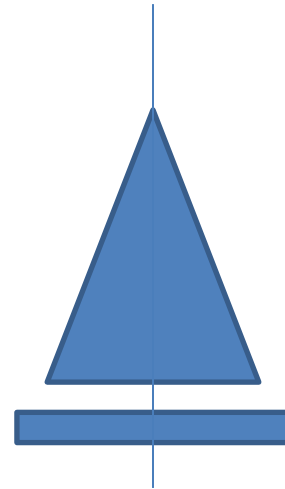
- $\text{factorial}(n) = \text{factorial}(n-1) * n$
- Tháp Hà Nội $f(n) = f(n-1) + 1 \dots$



- Output(move n from 1 to 3)

- $\text{factorial}(n) = \text{factorial}(n-1) * n$
- Tháp Hà Nội $f(n) = f(n-1) + 1 + f(n-1)$

- $\text{move}(n-1, 2, 3)$



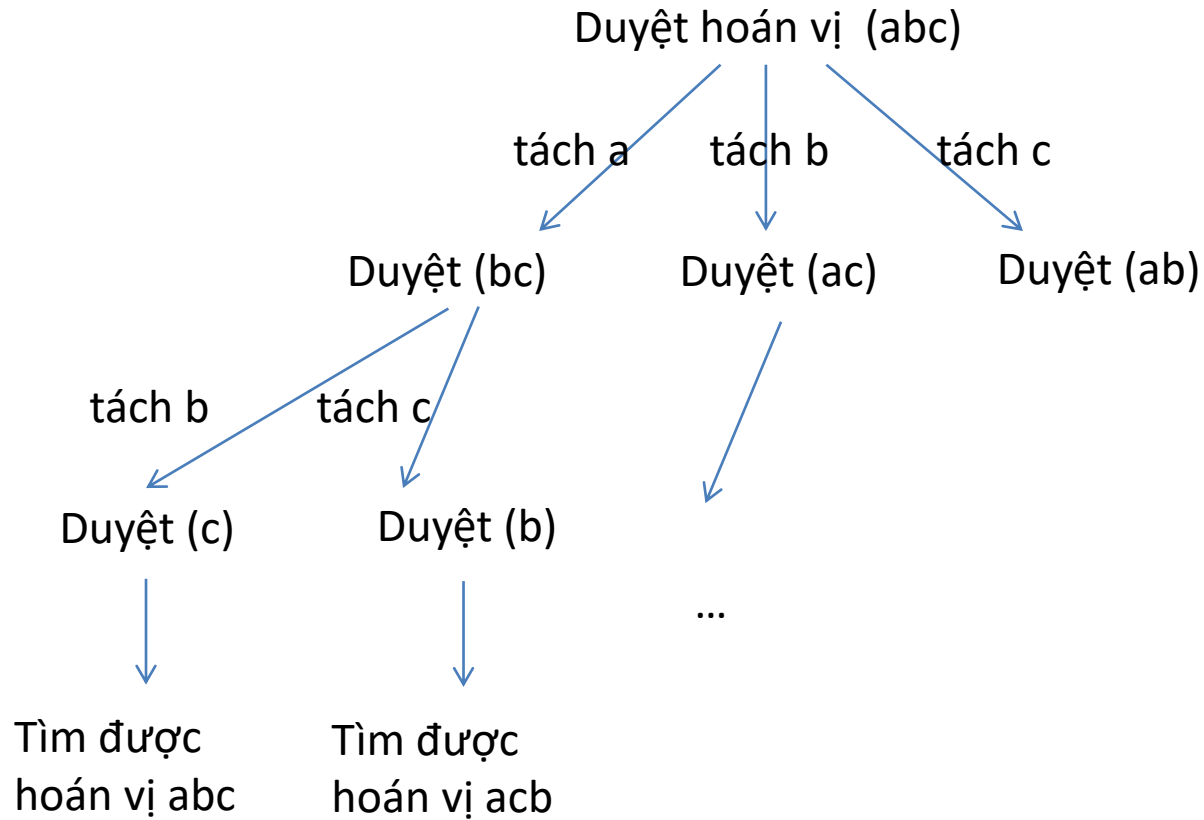
Move(n-1, 1, 2)

Output("move disc n from 1 to 3");

Move(n-1, 2, 3)

```
void move(int n, int from, int to) {  
    move(n-1, from, other(from, to));  
    output("move disc n from 'from' to 'to'");  
    move(n-1, other(from, to), to);  
}
```

Duyệt hoán vị



Duyệt hoán vị

$P(abcdef)$ danh sách tất cả các hoán vị của $abcdef$

$P(abcdef) = a + P(bcdef)$

$b + P(acdef)$

.....

permutation(s[], lo, hi): liệt kê tất cả hoán vị

```
if (lo == hi) { output(s); return; }
```

```
for i: lo-> hi {
```

```
    swap(s, lo, i); // tách lấy phần tử đứng đầu
```

```
    permutation(s, lo+1, hi) // đệ quy phần còn lại
```

```
    swap(s, lo, i); // quay lui về như cũ để thử cách khác
```

```
}
```

Duyệt tổ hợp

- Liệt kê các tập con của $[a,b,c]$
- Có a: abc, ab, ac, a
- Không có a: $bc, b, c, []$

$C(N)$ là tập của các tập con của $[1...N]$

Thì

$C(N) = C(N-1) + ([N] + s)$, với mọi s thuộc $C(N-1)$

Duyệt tổ hợp

$C(N) = C(N-1) + ([N] + s)$, với mọi s thuộc $C(N-1)$

Combination("abc", 3) -> in ra 8 kết quả

```
void combination(k) {  
    if (k<1) {  
        output();    return;  
    }  
    member[k] = true;  
    combination(k-1);  
    member[k] = false;  
    combination(k-1);  
}
```