

IT003 - CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Trường ĐH Công Nghệ Thông Tin, ĐHQG-HCM



CHỦ ĐỀ 1: CÁC THUẬT TOÁN ĐỆ QUY

NHÓM 07 - THÀNH VIÊN

- 19520839 - Nguyễn Chí Phú
- 19521095 - Phạm Mai Trúc Uyên
- 19522423 - Lê Đức Trung

NỘI DUNG



TOÁN 3

$$142 \times 3 = 426$$
$$426 : 3 = ?$$



Nhìn bức
ảnh này bạn
có thấy...



...điểm gì
đặc biệt
không?



```
2. #include <stdio.h>
```

```
3.
```

```
4. int sum(int n){
```

```
5.     if(n == 0) // điều kiện dừng (phần cơ sở)
```

```
6.         return 0;
```

```
7.     return n + sum(n-1);
```

```
8. }
```

```
9.
```

```
10. int main(){
```

```
11.     int sum = sum(5);
```

```
12.     printf("Sum = %d", sum);
```

```
13. }
```

OUTPUT:

Sum = 15

Với $n = 5$

$5 + \text{sum}(4)$

$5 + 4 + \text{sum}(3)$

$5 + 4 + 3 + \text{sum}(2)$

$5 + 4 + 3 + 2 + \text{sum}(1)$

$5 + 4 + 3 + 2 + 1 + 0$

Định nghĩa đệ quy

- Về mặt thuật toán (Algorithmically): Đệ quy là **chia vấn đề phức tạp thành các vấn đề tương tự nhau và đơn giản hơn để giải quyết**

```
int GiaiThua(int n)
{
    if (n == 1)
        return 1;
    return n * GiaiThua(n - 1);
}
```

- Về mặt ngữ nghĩa (Semantically): đệ quy là một kỹ thuật lập trình khi **một hàm tự gọi lại chính nó**



$$\begin{aligned} n! &= n * (n - 1)! \\ &= n * (n - 1) * (n - 2) * \dots * 1 \end{aligned}$$

Cấu trúc chương trình đệ quy

Base Case

- ĐK để thoát khỏi đệ quy, trả về giá trị có thể tính toán trực tiếp

Recursive Case

- Thân hàm có chứa phần gọi đệ quy

```
//HÀM TÍNH N!  
int Fact (int n)  
{  
    if (n == 1)  
        return 1;  
    //BASE CASE: 1! = 1  
  
    else  
        return n * Fact (n - 1);  
    //RECURSIVE CASE  
}
```

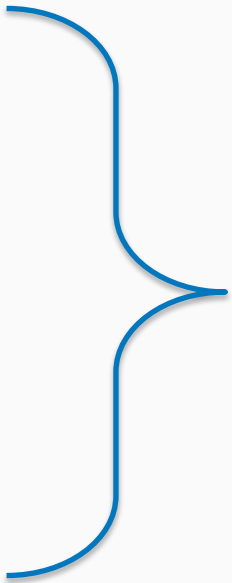


```
4 void hello(int count)
5 {
6     count++;
7
8
9
10     cout<<"hello "<< count<<endl;
11     hello(count);
12 }
13 }
14 int main()
15 {
16     int count=0;
17     hello(count);
18 }
```

```
hello 4451
hello 4452
hello 4453
hello 4454
hello 4455
hello 4456
hello 4457
hello 4458
hello 4459
hello 4460
hello 4461
hello 4462
hello 4463
hello 4464
hello 4465
hello 4466
hello 4467
hello 4468
hello 4469
hello 4470
hello 4471
hello 4472
hello 4473
hello 4474
```

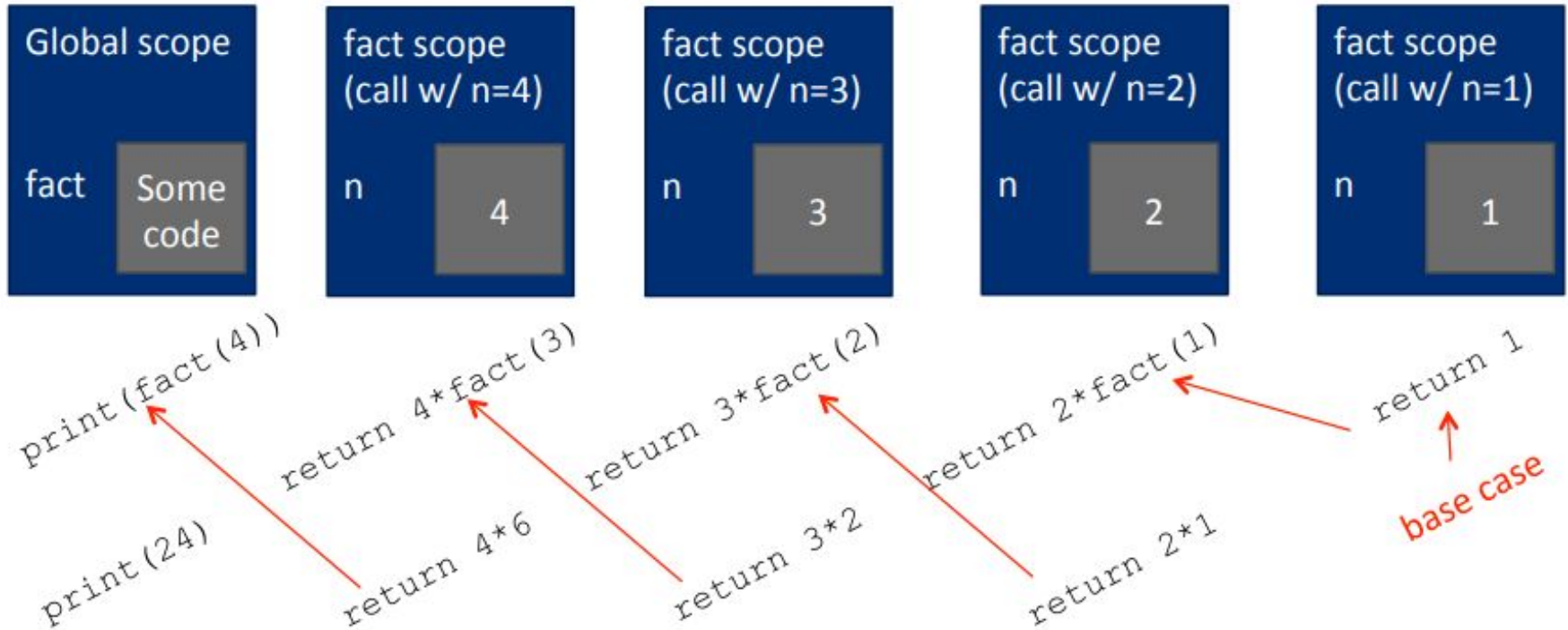
Cấu trúc chương trình đệ quy

- Mỗi lời gọi đệ quy trong hàm sẽ tạo ra một phạm vi/môi trường (scope) riêng
- Các ràng buộc về biến trong 1 phạm vi (scope) sẽ không bị thay đổi bởi lời gọi đệ quy



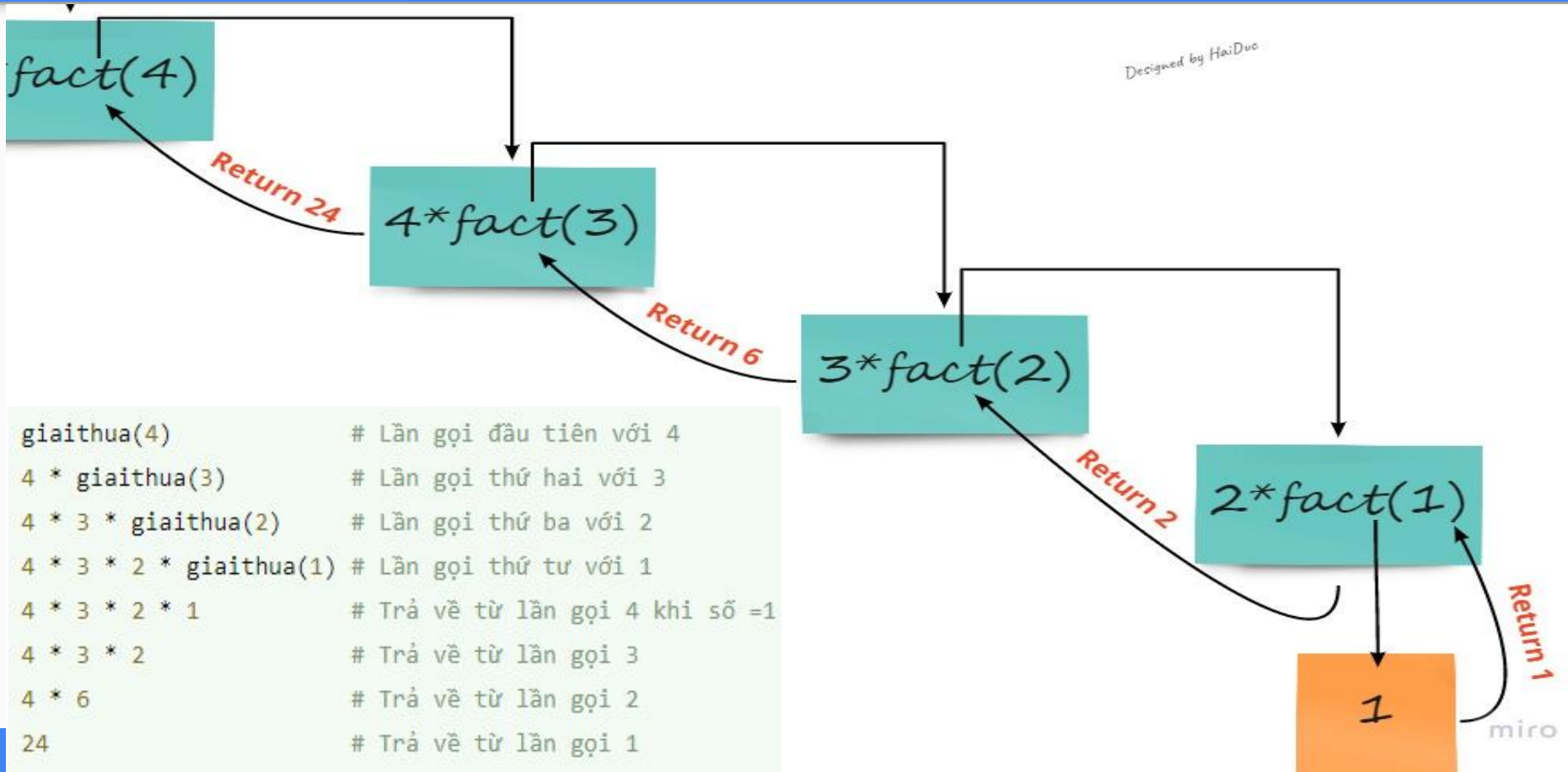
Mỗi lần đệ quy sử dụng tên biến (n) như nhau nhưng chúng là những objects khác nhau trong những scope riêng biệt

Ví dụ về phạm vi (scope) của hàm đệ quy



Cấu trúc chương trình đệ quy

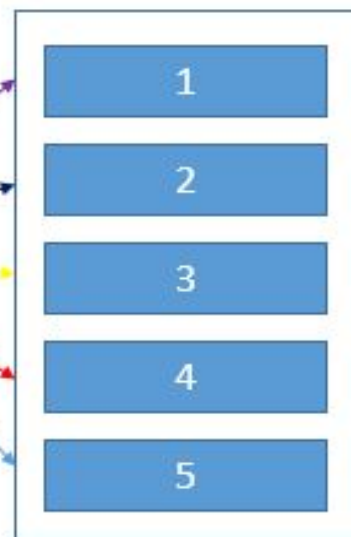
Designed by HaiDuc



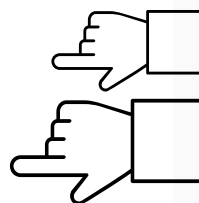
```
int kq = factorial(n);
```

```
int n = 5;
```

1. factorial(5)
-> return 5 * factorial(4)
2. factorial(4)
-> return 4 * factorial(3)
3. factorial(3)
-> return 3 * factorial(2)
4. factorial(2)
-> return 2 * factorial(1)
5. factorial(1)
-> return 1 * factorial(0)
6. factorial(0)
-> return 1;



Stack



Call Stack

➡ $1 * 1 * 2 * 3 * 4 * 5 = 120$

	Đệ Quy	Vòng lặp
<u>Định nghĩa</u>	Gọi một hàm trong cùng một hàm	Thực thi một khối lệnh nhiều lần trong chương trình
<u>Tốc độ</u>	Thực thi chậm hơn	Thực thi nhanh hơn
<u>Stack</u>	Được dùng để lưu trữ các biến cục bộ	Không sử dụng stack
<u>Điều kiện chấm dứt</u>	Nếu không có điều kiện chấm dứt -> đệ quy vô hạn	Nếu không có điều kiện hoặc điều kiện không bao giờ trở thành sai -> vòng lặp vô hạn
<u>Dễ đọc</u>	Dễ đọc, dễ hiểu hơn	
<u>Space Complexity</u> <u>Độ phức tạp bộ nhớ</u>	Độ phức tạp cao hơn	Thấp hơn

Đệ Quy vs Vòng lặp

Recursion

```
int GiaiThua(int n)
{
    if (n == 1)
        return 1;
    return n * GiaiThua(n - 1);
}
```

```
int giaiThua()
{
    int giai_thua = 1;
    for ( int i = 1; i <= 5; i ++ )
        giai_thua *= i;
    return giai_thua;
}
```

Loop