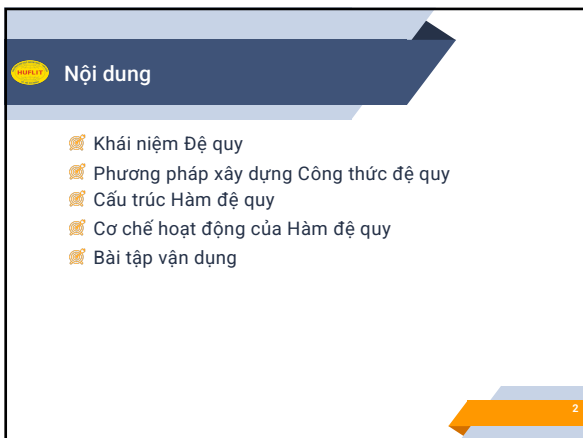
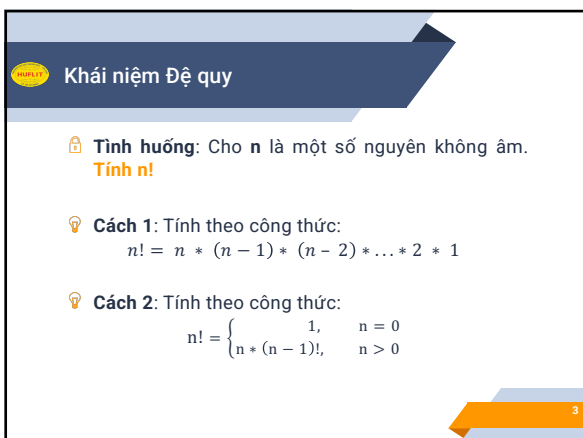




1



2



3


Khái niệm Đệ quy

- Trong Toán học, một **bài toán có tính chất đệ quy** khi nó có thể được định nghĩa thông qua một bài toán nhỏ hơn có dạng giống với nó.
- Trong lập trình, **đệ quy (recursion)** là kỹ thuật cài đặt chương trình, trong đó **một hàm tự gọi lại chính nó bên trong phần định nghĩa** của hàm đó.

4

4


Xây dựng Công thức Đệ quy

- Mỗi công thức đệ quy gồm 2 thành phần:
 - Phần không đệ quy:** gọi là **trường hợp cơ sở (base case)**
 - Là trường hợp **bài toán con nhỏ nhất có thể tìm được lời giải mà không cần gọi đệ quy**
 - Phải **có ít nhất một trường hợp cơ sở** để dừng quá trình đệ quy

5

5


Xây dựng Công thức Đệ quy

- Mỗi công thức đệ quy gồm 2 thành phần:
 - Phần đệ quy:** gọi là **trường hợp tổng quát (general case)**
 - Công thức mô tả quan hệ giữa lời giải bài toán với lời giải của bài toán nhỏ hơn (bài toán con) cùng dạng
 - Công thức phải **dẫn đưa bài toán về trường hợp cơ sở**

6

6

Xây dựng Công thức Đệ quy

■ Ví dụ:

$$n! = \begin{cases} 1, & n = 0 \\ n * (n - 1)!, & n > 0 \end{cases}$$

Trường hợp cơ sở

Trường hợp tổng quát

7

7

Xây dựng Công thức Đệ quy

Ba câu hỏi gợi ý để xây dựng Công thức đệ quy:

- Trường hợp cơ sở là gì?**
 - Trường hợp đơn giản nhất mà có thể biết ngay lời giải hoặc tính toán được lời giải mà không thông qua việc gọi đệ quy
- Chia nhỏ bài toán như thế nào?**
 - Xác định bài toán con cùng dạng; đảm bảo quá trình chia nhỏ phải dẫn đưa được bài toán về trường hợp cơ sở
- Mối liên hệ giữa lời giải của bài toán với bài toán con là gì?**
 - Xác định công thức liên hệ hoặc cách kết hợp lời giải của bài toán con thành lời giải của bài toán lớn hơn

8

8

Xây dựng Công thức Đệ quy

✍ **Bài tập:** Hãy xác định công thức đệ quy để tính a^n , với a là một số thực và n là số nguyên không âm

9

9

Xây dựng Công thức Đệ quy

Bài tập: Hãy xác định công thức đệ quy để tìm số Fibonacci thứ n , với n là một số nguyên dương và dãy số Fibonacci có dạng:

F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	...	F(n)
1	1	2	3	5	8		?

10

10

Xây dựng Công thức Đệ quy

Bài tập: Hãy xác định công thức đệ quy để tìm ước số chung lớn nhất của 2 số nguyên a, b theo giải thuật Euclid:

- Bước 1. [Tìm số dư].** Tính $r = a \bmod b$ (ta có $0 \leq r < b$)
- Bước 2. [Có phải số dư bằng 0].** Nếu $r = 0$ thì dừng thuật toán, b là kết quả
- Bước 3. [Giản lược]**
 - ☐ Đặt $a = b$
 - ☐ Đặt $b = r$
 - ☐ Quay lại Bước 1

11

11

Cấu trúc Hàm đệ quy

Ngôn ngữ C# cho phép định nghĩa Hàm mà bên trong thân hàm có thể gọi lại chính nó để diễn đạt các thuật toán đệ quy → Hàm đệ

```
[return-type] FunctionName([parameters])
{
    //Trường hợp cơ sở
    if (điều kiện dừng)
        [return] Kết quả của bài toán con nhỏ nhất
    //Trường hợp tổng quát
    else
        [return] Công thức đệ quy
}
```

12

12

Cấu trúc Hàm đệ quy

■ Ví dụ:

$$n! = \begin{cases} 1, & n = 0 \\ n * (n - 1)!, & n > 0 \end{cases}$$

Trường hợp cơ sở

Trường hợp tổng quát

```
int Factorial(int n)
{
    //Trường hợp cơ sở
    if (n == 0)
        return 1;
    //Trường hợp tổng quát
    else
        return n * Factorial(n - 1);
}
```

Gọi Đệ quy

13

13

Xây dựng Công thức Đệ quy

✍ **Bài tập:** Hãy viết hàm đệ quy để giải các bài toán ở phần trước sau khi đã biết công thức đệ quy

14

14

Cơ chế hoạt động của Hàm đệ quy

```
[return-type] FunctionName([parameters])
{
    . . .
    FunctionName(. . .);
    . . .
}
```

■ Xem xét 2 cơ chế:

- ☐ Cơ chế tính toán
- ☐ Cơ chế bộ nhớ

15

15

Cơ chế hoạt động của Hàm đệ quy

- Cơ chế tính toán:
 - Khi gặp lời gọi đệ quy, bài toán con sẽ được giải hoàn tất trước khi thực hiện phần còn lại của hàm
 - Quá trình gọi đệ quy sẽ được lặp lại cho đến khi gặp trường hợp cơ sở

16

16

Cơ chế hoạt động của Hàm đệ quy

Ví dụ:

```
int Factorial(int n)
{
    if (n == 0)
        return 1;
    else
    {
        int ans = n * Factorial(n - 1);
        Console.WriteLine($"Tinh {n}! Hoàn tất");
        return ans;
    }
}
```

```
Factorial(2)
{
    ans = 2 * Factorial(1);
    ...
}
```

```
ans = 1 * Factorial(0);
return 1;
Tinh 1! Hoàn tất
return ans;
Tinh 2! Hoàn tất
return ans;
}
```

Tính Factorial(2)
Tinh Factorial(1)
Tinh Factorial(0)

17

17

Cơ chế hoạt động của Hàm đệ quy

- Cơ chế bộ nhớ:
 - Nhắc lại:** Khi thực thi Hệ điều hành cấp cho chương trình 2 vùng nhớ:

```
static void Main(string args)
{
    int a = 2;
    int[] b = new int[3];
}
```

Stack: lưu trữ các biến cục bộ (bao gồm các tham số của hàm)

Computer memory

Heap: lưu trữ giá trị của các biến kiểu tham chiếu

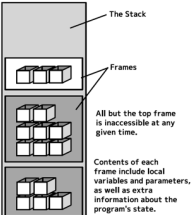
18

18

Cơ chế hoạt động của Hàm đệ quy

■ Cơ chế bộ nhớ:

- Stack còn giúp quản lý trạng thái hiện tại của chương trình (hàm nào đang thực thi):



- Khi một hàm được gọi, vùng chứa dữ liệu của hàm đó được tạo trên đỉnh Stack → gọi là **frame**
- Tại mỗi thời điểm chỉ có thể truy cập frame nằm trên đỉnh Stack
- Sau khi hàm thực hiện xong, frame của hàm đó được xóa khỏi Stack

19

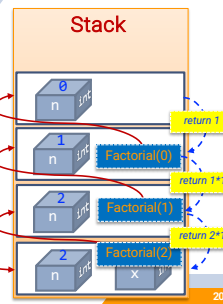
19

Cơ chế hoạt động của Hàm đệ quy

■ Cơ chế bộ nhớ:

```
static int Factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * Factorial(n - 1);
}

static void Main(string args)
{
    int n = 2;
    int x = Factorial(n);
    Console.WriteLine($"{n}! = {x}");
}
```



20

20

Cơ chế hoạt động của Hàm đệ quy

✍ Bài tập: Minh họa quá trình tính a^n theo 2 công thức đệ quy dưới đây:

(1)
$$a^n = \begin{cases} 1 & \text{nếu } n = 0 \\ a^{n-1} \times a & \text{nếu } n > 0 \end{cases}$$

(2)
$$a^n = \begin{cases} 1 & \text{nếu } n = 0 \\ a^{n/2} \times a^{n/2} & \text{nếu } n > 0 \text{ và } n \text{ chẵn} \\ a^{(n-1)/2} \times a^{(n-1)/2} \times a & \text{nếu } n > 0 \text{ và } n \text{ lẻ} \end{cases}$$

21

21

Cơ chế hoạt động của Hàm đệ quy

Nhận xét:

- Nếu **hàm đệ quy không hội tụ** (không đến được trường hợp cơ sở) hoặc **số lần gọi đệ quy quá nhiều** thì sẽ dẫn đến **lỗi tràn bộ nhớ Stack** (thuật ngữ **Stack Overflow**)

22

22

Cơ chế hoạt động của Hàm đệ quy

Cải tiến: Trong công thức (2) phải gọi đệ quy 2 lần để tính giá trị $a^{n/2}$ hoặc $a^{(n-1)/2}$ → **cải tiến bằng cách sử dụng biến phụ**

$$a^n = \begin{cases} 1 & \text{nếu } n = 0 \\ t \times t & \text{(với } t = a^{n/2}) \\ t \times t \times a & \text{(với } t = a^{(n-1)/2}) \end{cases}$$

Bài tập: Viết hàm đệ quy **tính a^n** theo công thức trên có cải tiến bằng cách sử dụng biến phụ

23

23

Cơ chế hoạt động của Hàm đệ quy

Nhận xét: Đệ quy và vòng lặp đều thực hiện một quá trình nào đó lặp lại nhiều lần nhưng khác nhau về cơ chế

Đệ quy	Vòng lặp
Lặp lại việc gọi hàm	Lặp lại việc thực hiện một đoạn lệnh
Mô tả chương trình ngắn gọn, tự nhiên, dễ hiểu	Chương trình có thể dài và phức tạp hơn
Tốn nhiều bộ nhớ	Tốn ít bộ nhớ hơn
Tốc độ thực thi chậm	Tốc độ thực thi nhanh hơn

24

24

Cơ chế hoạt động của Hàm đệ quy

Khi nào có thể sử dụng đệ quy:

- Ngôn ngữ lập trình hỗ trợ đệ quy
- Vấn đề cần giải quyết có thể biểu diễn đệ quy và khó mô tả dưới dạng vòng lặp
- Kích thước dữ liệu không quá lớn để gây ra lỗi Stack Overflow

25

25

Bài tập vận dụng

Bài tập 1: Viết hàm đệ quy chuyển đổi một số nguyên dương n từ hệ thập phân (DEC) sang hệ nhị phân (BIN)

- **Input:** một dòng duy nhất chứa số nguyên n
- **Output:** một dòng duy nhất chứa giá trị ở hệ nhị phân của số nguyên n .

2	20	0
2	10	0
2	5	1
2	2	0
1		

INPUT	OUTPUT
20	10100

26

26

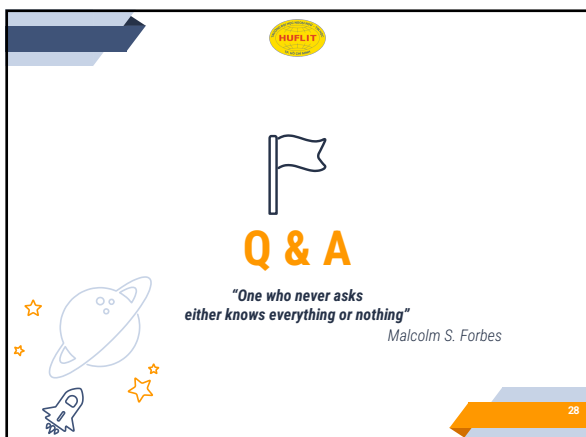
Bài tập vận dụng

Bài tập 2: Cài đặt lại thuật toán Tìm kiếm nhị phân bằng cách đệ quy

```
static int BinarySearch(int[] a, int x)
{
    int left = 0;
    int right = a.Length - 1;
    while (left <= right)
    {
        int mid = (left + right) / 2;
        if (a[mid] == x) return mid;
        if (a[mid] > x) right = mid - 1;
        else left = mid + 1;
    }
    return -1;
}
```

27

27



28
