



28TECH

Become A Better Developer



# ĐỆ QUY (RECURSION)





# 1. Cấu trúc dữ liệu ngăn xếp:



**Ngăn xếp (stack)** là một cấu trúc dữ liệu có **quan hệ mật thiết** với cơ chế hoạt động của đệ quy. Để hiểu được cách hàm đệ quy hoạt động, ta cần **nhắm được cách hoạt động** của cấu trúc dữ liệu ngăn xếp,




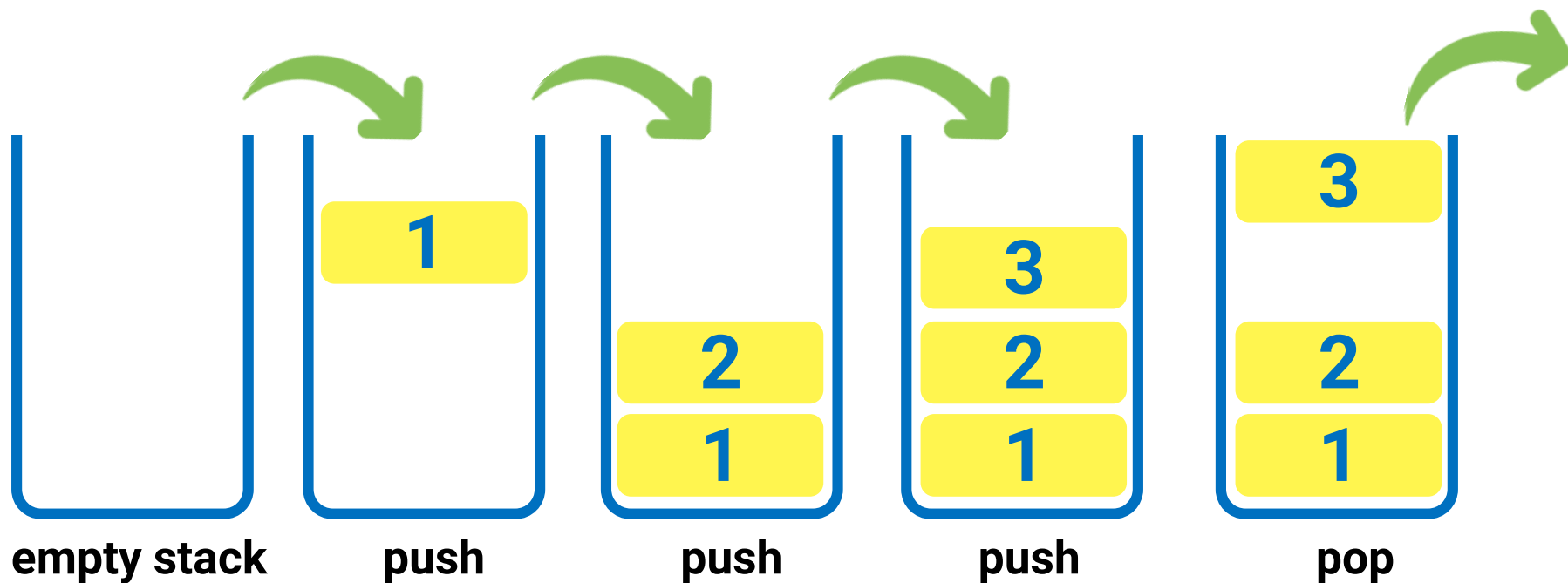
**Ngăn xếp** là một cấu trúc dữ liệu hỗ trợ **2 thao tác push và pop**. Trong đó push giúp thêm 1 phần tử vào đỉnh ngăn xếp, pop giúp xóa 1 phần tử khỏi đỉnh ngăn xếp. Cả 2 thao tác này đều **được thực hiện ở đỉnh ngăn xếp**.






# 1. Cấu trúc dữ liệu ngăn xếp:

 Ngăn xếp hoạt động theo nguyên tắc viết tắt là **LIFO (Last In First Out)** nghĩa là vào cuối thì ra đầu. Các phần tử vào cuối cùng sẽ được ra đầu tiên.

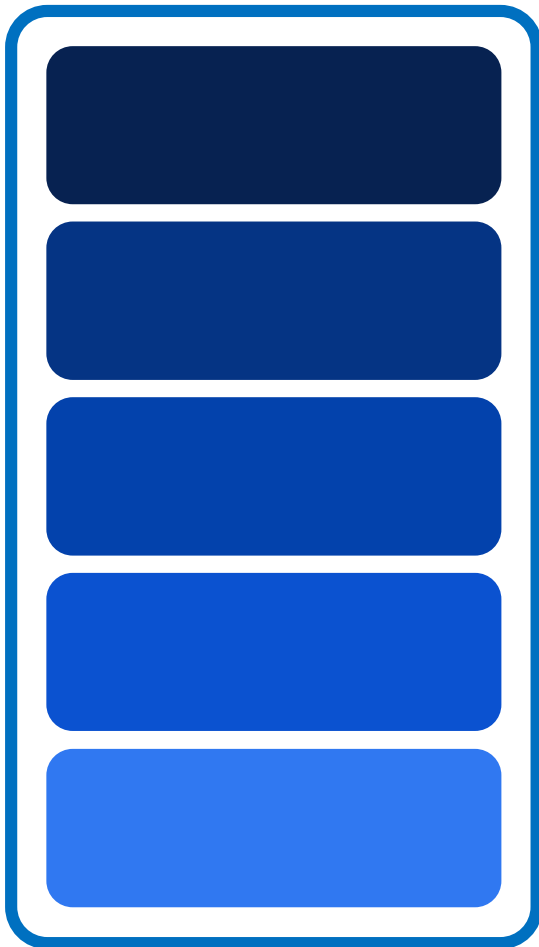


 Trong chương trình tồn tại một bộ nhớ là **bộ nhớ ngăn xếp**, cách hoạt động của bộ nhớ này **tương tự** như cách hoạt động của cấu trúc dữ liệu ngăn xếp.





## 2. Stackframe là gì ?



**Stack frame** là một kỹ thuật quản lý bộ nhớ xuất hiện trong một số ngôn ngữ lập trình, nó có nhiệm vụ tạo ra và loại bỏ các biến tạm thời.



Có thể hiểu **Stack frame** là một tập hợp tất cả các thông tin liên quan đến một chương trình con (được tạo ra khi xuất hiện lời gọi hàm)

**Stack frame** chỉ tồn tại trong quá trình chương trình thực thi, **stack frame** giúp các ngôn ngữ lập trình hỗ trợ được chức năng đệ quy cho chương trình con.





## 2. Stackframe là gì ?

-  Những thành phần của **stack frame** có thể kể đến như biến cục bộ, đối số, địa chỉ trả về của một chương trình con.
-  Mỗi khi một lời gọi hàm được thực hiện, **stack frame** chứa thông tin của hàm đó **được đẩy vào** bộ nhớ stack và khi hàm đó kết thúc thì **stack frame** này **được loại bỏ** khỏi bộ nhớ stack.

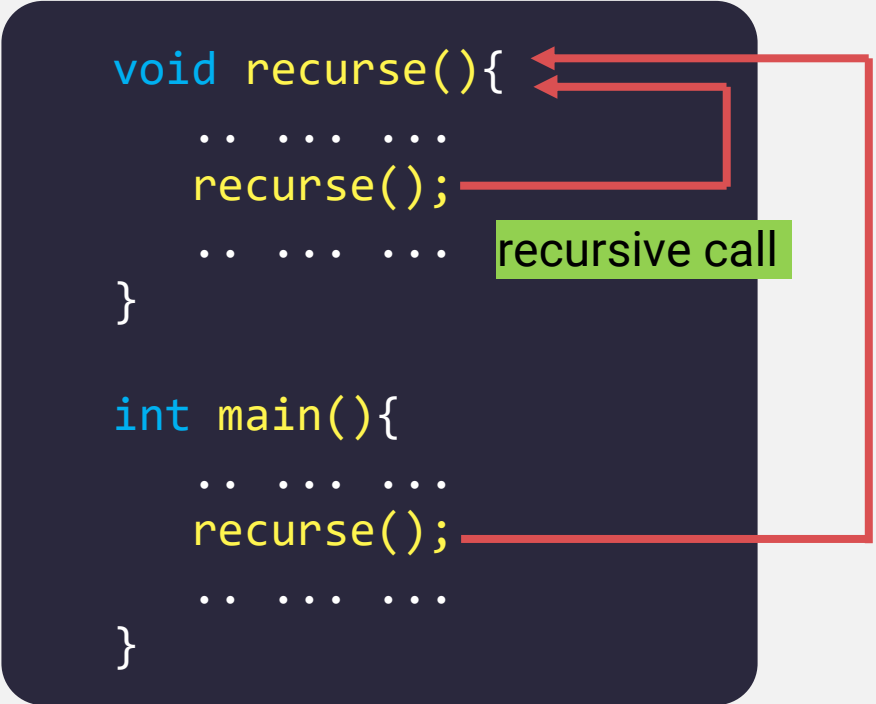


### 3. Hàm đệ quy:

#### Hoạt động của hàm đệ quy

Hàm đệ quy là một hàm gọi lại chính nó.

```
void recurse(){  
    .. .. .  
    recurse();  
    .. .. .  
}  
  
int main(){  
    .. .. .  
    recurse();  
    .. .. .  
}
```





# Ví dụ về hàm đệ quy

```
public static int main(){  
    ... ..  
    result = sum(number);  
    ... ..  
}
```

Trả về  $3 + 3 = 6$

```
public static int sum(int n){  
    if (n == 0)  
        return n;  
    else return n + sum(n - 1);  
}
```

Trả về  $2 + 1 = 3$

```
public static int sum(int n){  
    if (n == 0)  
        return n;  
    else return n + sum(n - 1);  
}
```

Trả về  $1 + 0 = 1$

```
public static int sum(int n){  
    if (n == 0)  
        return n;  
    else return n + sum(n - 1);  
}
```

Trả về 0

```
public static int sum(int n){  
    if (n == 0)  
        return n;  
    else return n + sum(n - 1);  
}
```





### 3. Hàm đệ quy:



Đệ quy thường dựa trên công thức toán học gọi là **công thức truy hồi** và **một bài toán con nhỏ nhất**.

Khi viết hàm đệ quy ta **cần xác định** được bài toán con nhỏ nhất để **làm điểm dừng** cho hàm đệ quy và công thức truy hồi để **tìm ra lời giải** của bài toán lớn hơn thông qua đáp án của bài toán nhỏ hơn.



Nếu đệ quy không có điểm dừng, khi **số lượng hàm đệ quy gọi đủ lớn** sẽ làm **bộ nhớ stack bị tràn**.





# Một số ví dụ minh họa

## Tính tổng $S(n) = 1 + 2 + 3 + \dots + n$ :

- Bài toán con nhỏ nhất:  $S(0) = 0$  khi  $n = 0$
- Công thức truy hồi:  $S(n) = n + S(n - 1)$  khi  $n \geq 1$

```
public static int sum(int n){  
    if (n == 0)  
        return 0;  
    else  
        return n + sum(n - 1);  
}
```

## Tính giai thừa: $F(n) = 1.2.3....n$ :

- Bài toán con nhỏ nhất:  $F(0) = 1$  khi  $n = 0$
- Công thức truy hồi:  $F(n) = n * F(n - 1)$  khi  $n \geq 1$

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

# Một số ví dụ minh họa

## Tính số Fibonacci:

- Bài toán con nhỏ nhất :  $F(0) = 0, F(1) = 1$
- Công thức truy hồi:  $F(n) = F(n - 1) + F(n - 2)$  khi  $n \geq 2$
- Code này chạy rất chậm, độ phức tạp là  $O(1.618^n)$

```
public static int fibo(int n):  
    if (n == 0 || n == 1)  
        return n;  
    return fibo(n - 1) + fibo(n - 2);
```

## Tính tổng chữ số của số nguyên dương n: SumDigit(n)

- Bài toán con nhỏ nhất :  $\text{SumDigit}(n) = n$  nếu  $n < 10$
- Công thức truy hồi :  
 $\text{SumDigit}(n) = n \% 10 + \text{SumDigit}(n / 10)$  nếu  $n \geq 10$

```
public static int SumDigit(int n):  
    if (n < 10)  
        return n;  
    return n % 10 + SumDigit(n / 10);
```

# Một số ví dụ minh họa

## Tính tổ hợp chập k của n:

- Bài toán con nhỏ nhất:  $C(n, k) = 0$  nếu  $k = 0$  hoặc  $n = k$
- Công thức truy hồi (Công thức pascal) :  
$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$$

```
public static int nCk(int n, int k):  
    if (k == 0 || n == k)  
        return 1;  
    return nCk(n - 1, k - 1) + nCk(n - 1, k);
```

## Tìm ước chung lớn nhất bằng đệ quy:

- Bài toán con nhỏ nhất  $\text{gcd}(a, b) = a$  khi  $b = 0$
- Công thức truy hồi:  $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$

```
public static int gcd(int a, int b):  
    if (b == 0)  
        return a;  
    return gcd(b, a % b)
```



## Chuyển đổi số thập phân thành nhị phân bằng đệ quy

```
public static void convert(int n){  
    if (n < 2){  
        System.out.print(n % 2);  
    }  
    else{  
        convert(n / 2);  
        System.out.print(n % 2);  
    }  
}
```

