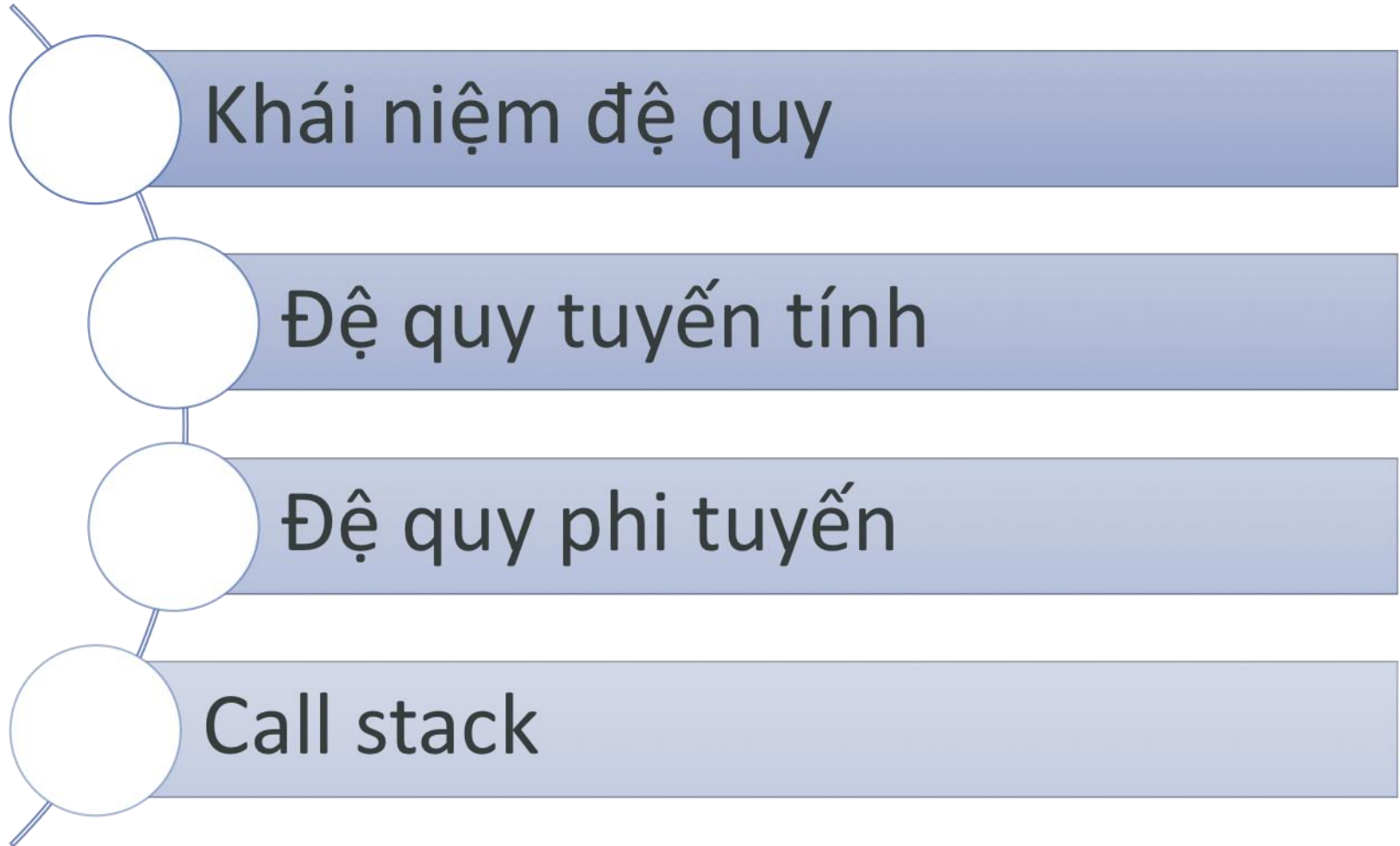


ĐỆ QUY (RECURSION)

3. Nội dung





- Một vấn đề mang tính đệ quy nếu như nó có thể được giải quyết thông qua kết quả của chính vấn đề đó nhưng với đầu vào đơn giản hơn.
- VD: Giai thừa:

$$n! = 1 \times 2 \times \cdots \times (n - 1) \times n$$

$$(n - 1)! = 1 \times 2 \times \cdots \times (n - 1)$$

$$\longrightarrow n! = (n - 1)! \times n$$



- **Recursion** – Đệ quy
- **Recursive** – Tính đệ quy.
- **Recursive problem** – Vấn đề đệ quy
- VD Tổng $S(n)$ của các số tự nhiên từ 1 đến n

$$S(n) = 1 + 2 + \cdots + n - 1 + n$$

$$S(n - 1) = 1 + 2 + \cdots + n - 1$$

$$\rightarrow S(n) = S(n - 1) + n$$



- Trường hợp cơ bản – **base case** – là một input đủ nhỏ để ta có thể giải quyết vấn đề mà không cần lời gọi đệ quy.

$$n! = n \times (n - 1)!$$

$$\longrightarrow 1! = 1 \times 0! \quad \longrightarrow 0! = ? ? ? ?$$


$$S(n) = n + S(n - 1)$$

$$\longrightarrow S(2) = 1 + S(1) \longrightarrow S(1) = ? ? ? ?$$



- Hàm đệ quy là hàm có lời gọi lại chính nó trong thân hàm

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






- Đệ quy tuyến tính - **Single recursion** – là trường hợp đệ quy chỉ đề cập lại đến nó một lần
- Đệ quy phi tuyến - **Multiple recursion** – là trường hợp đệ quy đề cập lại chính nó nhiều lần.
- Đệ quy hỗ tương – **Mutual recursion** – là trường hợp hiếm gặp khi hàm không trực tiếp gọi lại chính nó mà thông qua hàm khác




- Hàm đệ quy tuyến tính chỉ có duy nhất một lần gọi lại chính nó

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



- Ngay cả khi lời gọi đệ quy xuất hiện nhiều lần nhưng chỉ một lần được chạy

```
int uscln(int a, int b)
{
    if (a == b) return a;
    else if (a > b) return uscln(a - b, b);
    else uscln(a, b - a);
}
```




Đệ quy tuyến tính




- Đệ quy tuyến tính rất dễ chuyển sang vòng lặp có chức năng tương đương → Khử đệ quy

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



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



Đệ quy tuyến tính



- Dạng vòng lặp thường chạy nhanh hơn đệ quy, dùng ít bộ nhớ hơn → chạy được input lớn hơn

```
int tong(int n)
{
    if (n > 0) return n + tong(n - 1);
    else return 0;
}
```

```
int tong_2(int n)
{
    int kq = 0;
    for (int i = 1; i <= n; i++){
        kq = kq + i;
    }
    return kq;
}
```

```
int main()
{
    for(int i = 0; i < 1000; i++)
    {
        ///cout << tong(100000) << endl;
        cout << tong_2(100000) << endl;
    }
}
```

0.5s

0.25s

Đệ quy phi tuyến



- Cho dãy số fibonacci: số sau bằng tổng hai số liền trước nó:

$$\bullet F(n) = \begin{cases} 1 & \text{khi } n < 2 \\ F(n-1) + F(n-2) & \text{khi } n \geq 2 \end{cases}$$

0	1	1	2	3	5	8	13	21
F(0)	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)	F(8)

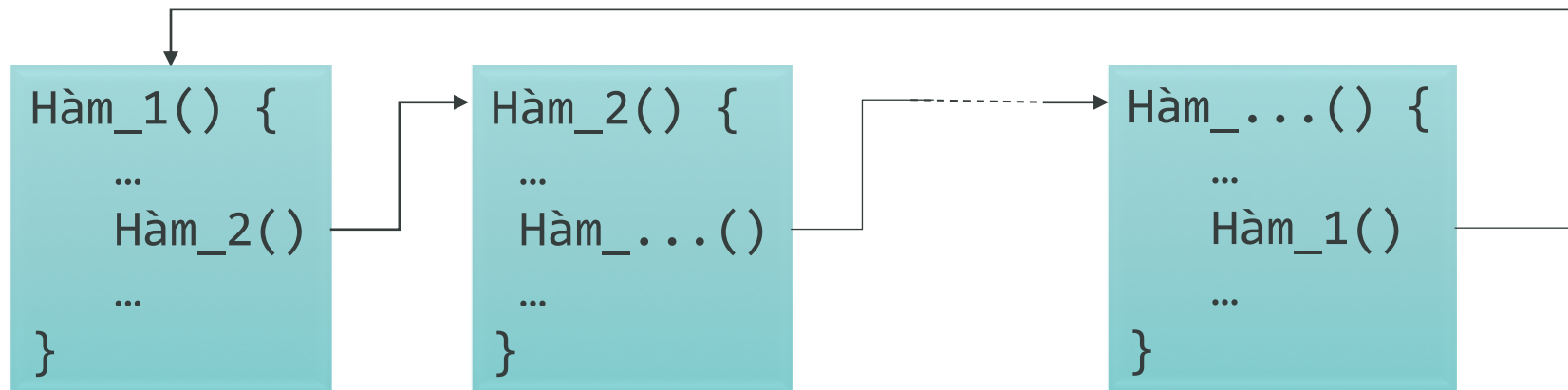


```
int fibonacci(int n)
{
    if (n < 2) return 1;
    else return fibonacci(n - 1) + fibonacci(n - 2);
}
```

- Hàm Fibonacci gọi lại chính nó 02 lần
 - → trường hợp đặc biệt của đệ quy phi tuyến: **đệ quy nhị phân**
- Đoạn code trên khó chuyển sang cấu trúc lặp

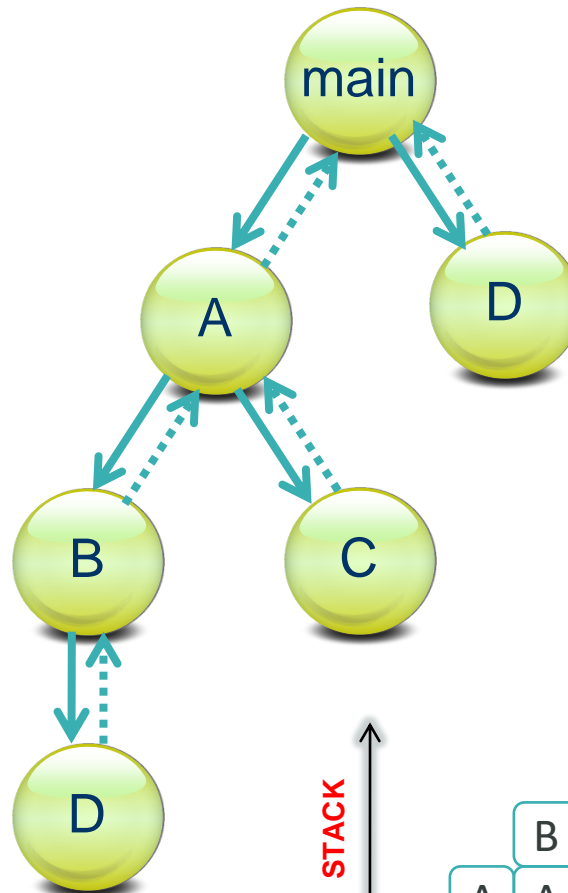
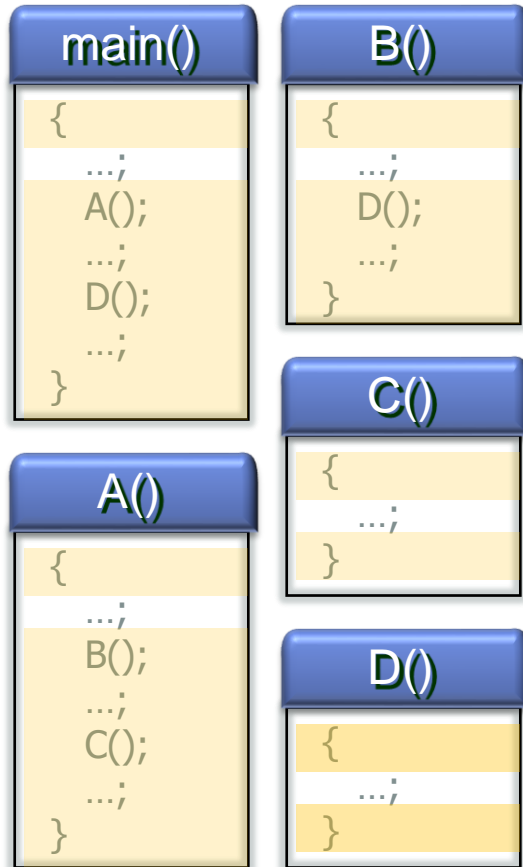


- Đệ quy hỗ tương – mutual recursion. Còn gọi đệ quy gián tiếp – indirect recursion.
- Hàm không trực tiếp gọi lại chính nó mà gọi thông qua một (hoặc nhiều) hàm khác

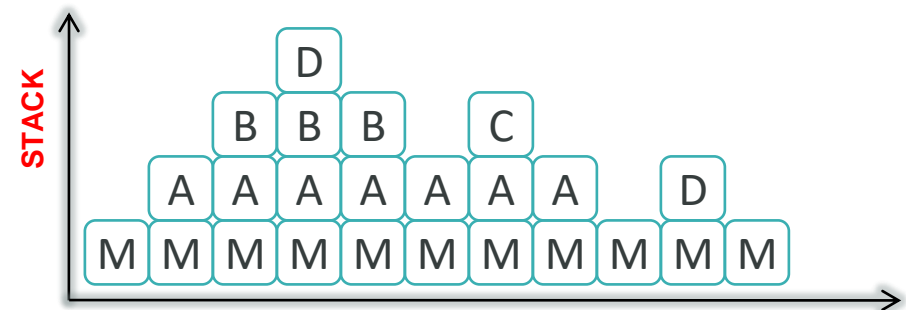




Call stack



- Mỗi lời gọi hàm tạo ra một phần tử mới trong stack
- C++ luôn chạy phần tử ở đỉnh của stack trước





- 

[illegible]



- Viết hàm đệ quy giải các bài toán:
 1. Đếm số lượng chữ số của số nguyên dương n
 2. Tính giá trị của x lũy thừa n
 3. Tính giá trị của $n!$
 4. Tính ước số chung nhỏ nhất của 2 số nguyên.
 5. Tìm số thứ n của dãy Fibonacci
 6. Tìm số thứ n của dãy Padovan
 7. Bài toán Tháp Hà Nội

Bài tập



```
void ThapHaNoi (int n, char a, char b, char c)
{
    if (n==1)
    {
        cout<<"\t"<<a<<" ----->"<<c<<endl;
        return;
    }
    ThapHaNoi (n-1, a, c, b);
    ThapHaNoi (1, a, b, c);
    ThapHaNoi (n-1, b, a, c);
}
```