

## Đệ quy

*Một hàm đệ quy* là một hàm gọi lại chính nó trực tiếp hoặc gián tiếp thông qua các hàm khác.

Đây là một chủ đề phức tạp sẽ được thảo luận và ứng dụng cụ thể hơn trong loạt bài về cấu trúc dữ liệu và giải thuật.

Trong nội dung này ta sẽ:

- tiếp cận với khái niệm và
- những ví dụ cơ bản nhất về đệ quy.

Hàm đệ quy thường dùng để giải quyết các bài toán có cùng đặc điểm chung. Trong lời gọi đệ quy, hàm đệ quy thường tự biết xử lý trường hợp cơ bản nhất gọi là trường hợp cơ sở/ điểm dừng. Nếu hàm đệ quy được gọi với vấn đề ở trường hợp cơ sở, nó trả về kết quả. Nếu hàm đệ quy được gọi với vấn đề ở trường hợp phức tạp hơn, nó chia vấn đề thành 2 phần: một phần nó biết cách xử lý, và phần còn lại nó không biết cách xử lý. Để có thể đệ quy, vấn đề mà nó chưa biết giải quyết sẽ trông giống với vấn đề gốc ban đầu nhưng nhỏ hơn, tức đỡ phức tạp hơn. Do vấn đề nhỏ hơn này giống y như vấn đề mà hàm đang xử lý nên hàm đệ quy tiếp tục gọi đệ quy với bản sao của hàm hiện tại để giải quyết vấn đề đang được nói tới. Hàm đệ quy thường đi kèm với keyword return vì kết quả của lời gọi hàm là sự kết hợp của phần vấn đề mà hàm biết cách xử lý và phần vấn đề hàm chưa biết cách xử lý để tạo thành kết quả trả về cho nơi gọi hàm. Vấn đề cứ được chia nhỏ ra tới khi hàm gặp trường hợp cơ sở, lúc này hàm sẽ lần lượt trả về kết quả theo chiều ngược lại của lời gọi hàm.

Ví dụ tính  $5!$  bằng đệ quy (hình ảnh lấy trong tài liệu deitel-c trang 189).

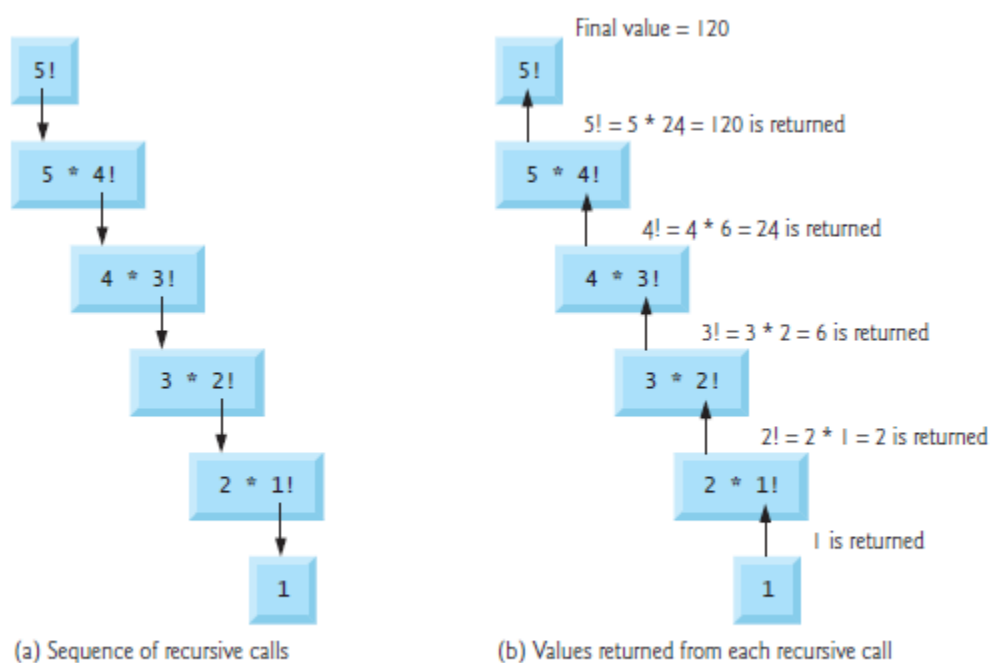


Fig. 5.17 | Recursive evaluation of 5!.

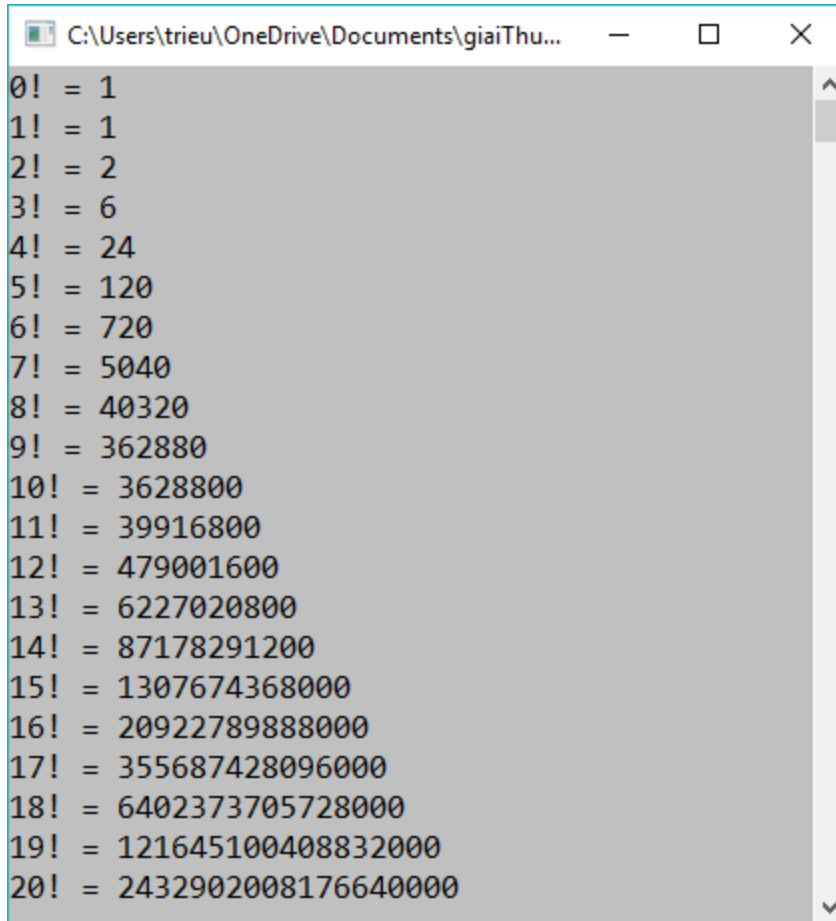
Thể hiện bằng chương trình:

```

1  #include<stdio.h>
2
3  unsigned long long giaiThua( int n ){
4      if( n == 0 || n == 1 ) // trường hợp cơ sở
5          return 1;
6      else // trường hợp tiếp tục gọi đệ quy
7          return n * giaiThua( n - 1 );
8  }
9
10 int main(){
11
12     int i;
13     for( i = 0; i < 21; i++ ){ // hiển thị từ 0! tới 20!
14         printf("%d! = %lld\n", i, giaiThua(i));
15     }
16
17     return 0;
18 }

```

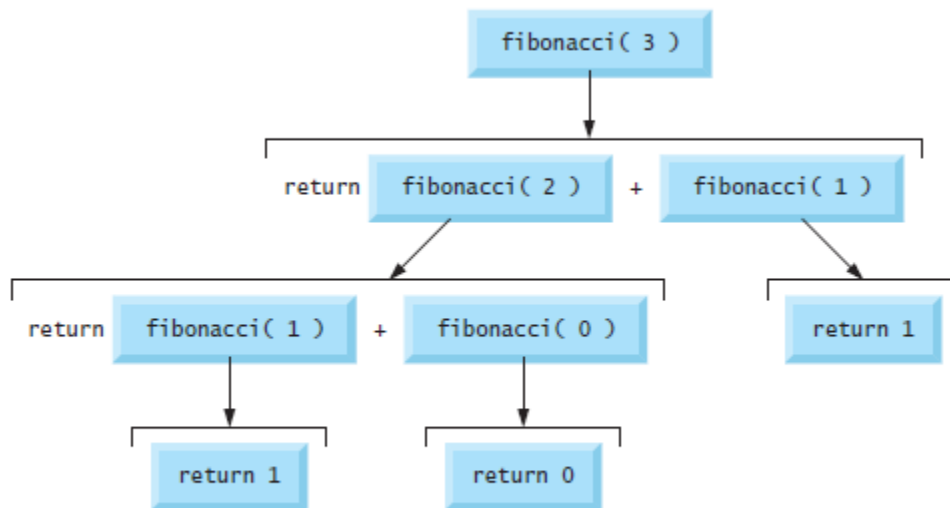
Kết quả thực hiện:



A screenshot of a Windows Notepad window. The title bar shows the file path: C:\Users\trieu\OneDrive\Documents\giaiThu... The window contains a list of factorial calculations from 0! to 20!. The text is as follows:

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
```

Hình sau minh họa việc giải quyết vấn đề tìm số fibonacci thứ n. (cùng tài liệu trang 193).



**Fig. 5.20** | Set of recursive calls for fibonacchi(3).

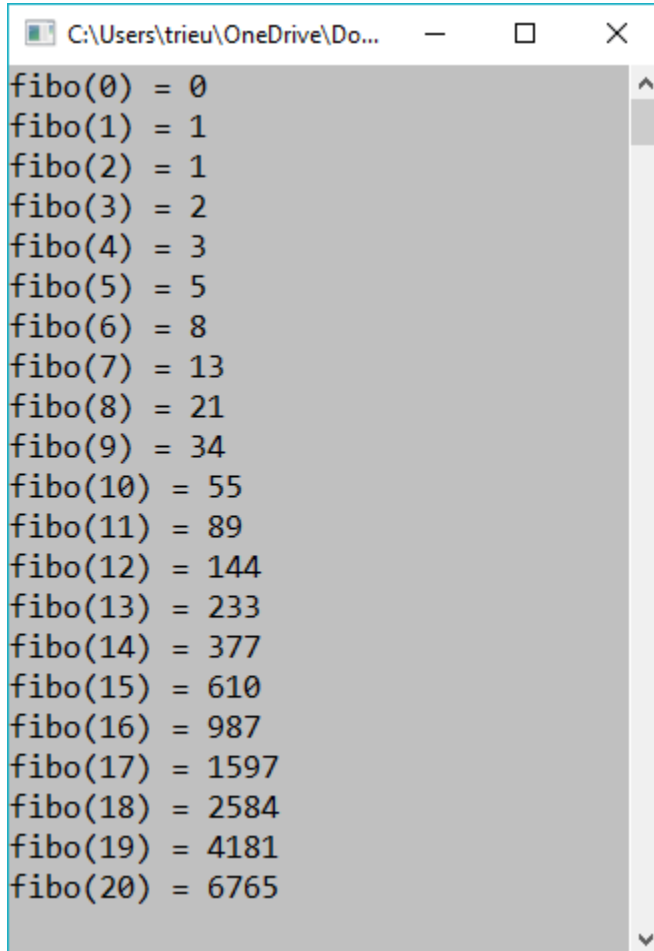
Chương trình:

```

1  #include<stdio.h>
2
3  unsigned long long fibo( int n ){
4      if( n == 0 || n == 1 ) // điều kiện dừng
5          return n;
6      else // trường hợp cần thực hiện đệ quy
7          return ( fibo( n - 1 ) + fibo( n - 2 ) );
8  }
9
10 int main(){
11
12     int i;
13     for( i= 0; i< 21; i++ ){ // hiển thị 21 kết quả
14         printf("fibo(%d) = %lld\n", i, fibo(i));
15     }
16
17     return 0;
18 }

```

Kết quả thực hiện:



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Users\trieu\OneDrive\Do...". The window contains a list of Fibonacci numbers calculated by a function named "fibo". The results are as follows:

n	fibo(n)
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55
11	89
12	144
13	233
14	377
15	610
16	987
17	1597
18	2584
19	4181
20	6765

## So sánh đệ quy và vòng lặp:

**Đệ quy** và **vòng lặp** có một số điểm chung và riêng như sau:

- Cùng dựa trên cấu trúc điều khiển: vòng lặp dựa trên cấu trúc điều khiển lặp, còn đệ quy dựa trên cấu trúc điều khiển lựa chọn.
- Cả vòng lặp và đệ quy dựa trên việc thực hiện lặp lại cùng một công việc giống nhau, điều này ta đã thấy rõ ràng trong vòng lặp, với đệ quy thì nó lặp lại việc gọi hàm để tiến hành thực hiện tiếp vấn đề con của vấn đề hiện tại.
- Cả vòng lặp và đệ quy đều có điều kiện kết thúc việc thực hiện lặp đi lặp lại: vòng lặp kết thúc lặp khi điều kiện lặp không còn thỏa mãn. Còn hàm đệ quy kết thúc lặp việc gọi đệ quy khi gặp trường hợp cơ sở hay còn gọi là điểm dừng.
- Cả hai đều có thể lặp vô hạn: vòng lặp lặp vô hạn nếu điều kiện lặp luôn đúng, đệ quy lặp vô hạn nếu vấn đề không nhỏ hơn sau mỗi lần đệ quy, kết quả là điều kiện dừng không bao giờ được tìm thấy.
- Với đệ quy, sau mỗi lần gọi đệ quy, vấn đề sẽ nhỏ dần. Với vòng lặp, vấn đề không thay đổi sau mỗi lần lặp.
- Đệ quy thường tiêu tốn tài nguyên máy: tốn CPU và bộ nhớ vì mỗi lần gọi đệ quy, chương trình lại yêu cầu cấp phát thêm không gian nhớ để lưu trữ kết quả và thực hiện chương trình.
- Vấn đề nào giải quyết được bằng đệ quy đều có thể giải quyết được bằng vòng lặp. Chọn đệ quy vì đệ quy thể hiện cách giải quyết vấn đề rõ ràng, tự nhiên và dễ hiểu, dễ sửa lỗi.

*Thank you for reading this tutorial!*