**Hàm đệ quy là gì? Hoạt động ra sao?**

**Trong bài này mình sẽ giới thiệu đến các bạn hàm đệ quy, đây là một hàm rất căn bản và được sử dụng rất nhiều trong lập trình.**



Chúng ta sẽ cùng nhau tìm hiểu về hàm đệ quy là gì? Cơ chế hoạt động của nó như thế nào? Và làm một vài ví dụ biểu diễn trong ngôn ngữ lập trình C++.

**Mục lục**

* [1. Hàm đệ quy là gì?](https://freetuts.net/ham-de-quy-2945.html#goto-h2-0)
* [2. Cơ chế hoạt động của đệ quy](https://freetuts.net/ham-de-quy-2945.html#goto-h2-1)
* [Lời kết](https://freetuts.net/ham-de-quy-2945.html#goto-h2-2)

**1. Hàm đệ quy là gì?**

Một hàm được gọi là hàm đệ quy nếu trong thân hàm có một hoặc nhiều lệnh gọi đến chính hàm đó.

Đệ quy giúp giải quyết bài toán theo cách nghĩ thông thường một cách tự nhiên.

*Bài viết này được đăng tại [free tuts .net]*

Đệ quy cũng tương tự như các vòng lặp, nó phải xác định được điểm dừng. Nếu không xác định chính xác điểm dừng, bài toán có thể lặp vĩnh cửu (**Stack Overhead**).

**Ví dụ:** Chúng ta có định nghĩa giai thừa của một số nguyên dương n như sau:

5! = 5 \* 4!

4! = 4 \* 3!

…

1! = 1 \* 0!

Theo quy luật ở trên, nếu ta biết được (n-1) giai thừa thì ta sẽ tính được n giai thừa: n! = n \* (n-1)!

Ta thấy n = 0 hoặc n = 1 thì giai thừa luôn bằng 1, vì vậy đây chính là điểm dừng.

Công thức tổng quát của n! như sau:

n! = 1 \* 2 \* 3 \*…\* (n-1) \* n = (n-1)! \* n với (0! = 1).

Áp dụng công thức trên, ta có thể viết một hàm tính giai thừa của một số nguyên dương n trong C++.

|  |  |
| --- | --- |
| 1  2  3  4 | Int giaiThua(**int** n){      If(n<=1) return 1;      return n \* giaiThua(n-1);  } |

**Điểm dừng** của hàm đệ quy trên chính là n <= 1.

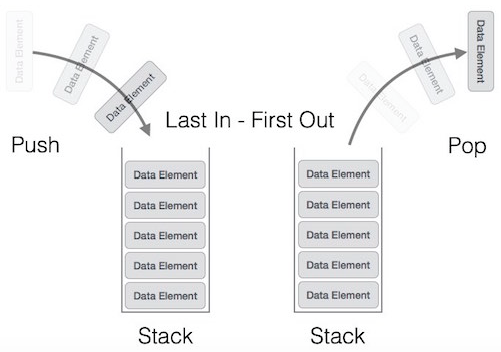
**\* Lưu ý**: Bản chất của hàm đệ quy là lặp vô hạn, vì vậy nếu không có điểm dừng thì chương trình sẽ chạy liên tục và dẫn đến tràn tài nguyên.

Có rất nhiều hàm đệ quy như:

* Đệ quy tuyến tính.
* Đệ quy đuôi.
* Đệ quy nhị phân.
* Đệ quy đa tuyến.
* Đệ quy lồng.
* Đệ quy tương hỗ.

**2. Cơ chế hoạt động của đệ quy**

Cơ chế hoạt động của đệ quy tuân thủ theo LIFO (**Last In First Out**), hay còn được gọi là cơ chế Stack.



Cơ chế Stack được hiểu đơn giản là vào sau – ra trước.

Ví dụ chúng ta muốn lắp một chiếc bánh xe, thì việc đầu tiên sẽ lắp bánh xe, rồi đến lắp lồng đền và cuối cùng mới lắp ốc. Nhưng khi tháo ra, ta lại tháo ốc trước rồi tới lồng đền và cuối cùng mới tháo bánh xe. Đây là một ví dụ thực tế cho các bạn dễ hình dung cơ chế Stack hoạt động như thế nào.

Ở các bài sau mình sẽ giải thích chi tiết cơ chế hoạt động của từng hàm đệ quy,vì chúng ta có rất nhiều hàm đệ quy khác nhau. Các bạn chỉ cần hiểu đơn giản rằng hàm đệ quy thực hiện theo cơ chế Stack.

**Lời kết**

Như vậy trong bài này chúng ta sẽ học tổng cộng 6 hàm đệ quy. Đây là một trong các chương cơ bản trong học phần cấu trúc dữ liệu và giải thuật. Khi các bạn nắm được bản chất và cố lõi của nó, thì việc chuyển sang các ngôn ngữ lập trình khác là điều rất dễ dàng. Đây cũng chỉ là bài cơ bản về đệ quy, các bạn hãy xem hết các bài sau để có thể nắm rõ hết tất cả các hàm đệ quy nhé.

**Đệ quy tuyến tính (Linear Recursion)**

Trong bài này mình sẽ giới thiệu đến các bạn một trong các hàm đệ quy đó chính là đệ quy tuyến tính. Đây là một hàm đệ quy cơ bản nhất và được sử dụng rất nhiều trong lập trình.



Chúng ta sẽ cùng nhau tìm hiểu về khái niệm đệ quy tuyến tính (Linear Recursion) cũng như cơ chế hoạt động của nó.

**Mục lục**

* [1. Đệ quy tuyến tính là gì?](https://freetuts.net/de-quy-tuyen-tinh-2946.html#goto-h2-0)
* [2. Cơ chế hoạt động của đệ quy tuyến tính (Linear Recursion)](https://freetuts.net/de-quy-tuyen-tinh-2946.html#goto-h2-1)

**1. Đệ quy tuyến tính là gì?**

Đệ quy tuyến tính là hàm đệ quy chỉ gọi chính nó một lần trong thân hàm. Hiểu đơn giản là trong một hàm, nếu có duy nhất một câu lệnh gọi chính hàm đó thì được gọi là hàm đệ quy tuyến tính.

Ví dụ hàm tính giai thừa chính là một hàm đệ quy tuyến tính, vì nó chỉ gọi lại chính nó một lần duy nhất.

*Bài viết này được đăng tại [free tuts .net]*

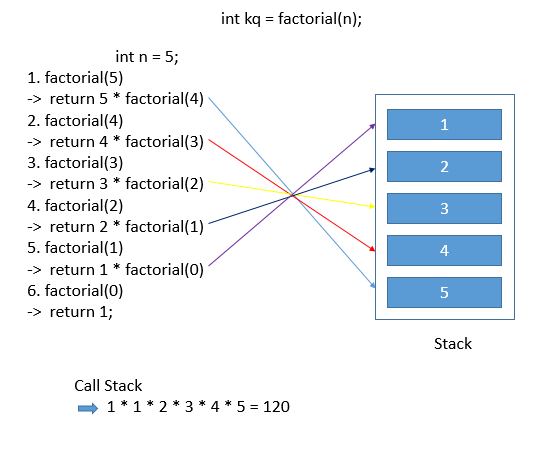
|  |  |
| --- | --- |
| 1  2  3  4 | Int factorial(**int** n){      If(n == 0) return 1;      return n \* factorial(n-1);  } |

**2. Cơ chế hoạt động của đệ quy tuyến tính (Linear Recursion)**

Chúng ta sẽ lấy ví dụ hàm factorial() tính giai thừa, để xem cơ chế hoạt động của nó như thế nào nhé.

|  |  |
| --- | --- |
| 1  2  3  4 | Int factorial(**int** n){      If(n == 0) return 1;      return n \* factorial(n-1);  } |

Giả sử chúng ta muốn tính 5! thì khi đó cơ chế hoạt động như hình bên dưới đây:

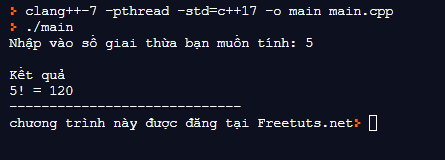


Cứ sau mỗi lần gọi hàm factorial(), kết quả sẽ được đẩy vào Stack cho đến khi gặp điều kiện dừng hàm sẽ kết thúc và trả về kết quả 1. Sau đó gọi Stack để tính kết quả, cơ chế Stack sẽ lấy từ trên xuống vì vậy kết quả sẽ là : 1 \* 1 \* 2 \* 3 \* 4 \* 5 = 120.

**Code mẫu:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include <iostream>  using namespace std;    //hàm đệ quy tuyến tính  **int** factorial(**int** n){    if(n == 0) return 1; // điểm dừng của hàm, nếu n == 0 thì kết thúc hàm và trả về 1    return n \* factorial(n-1);  }  //hàm main  **int** main() {  **int** n;    cout<<"Nhập vào số giai thừa bạn muốn tính: ";    cin>>n;  **int** kq = factorial(n);//gọi hàm factorial() để tính giai thừa cho n và gán kết quả vào biến kq    cout<<"\nKết quả \n"<<n<<"! = "<<kq;      cout<<"\n-----------------------------\n";    cout<<"chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình tính giai thừa của số nguyên n áp dụng hàm đệ quy. Cũng như kết thúc hướng dẫn đệ quy tuyến tính (Linear Recursion), chúc các bạn thực hiện thành công!!!

# Đệ quy đuôi (Tail Recursion)

**Trong bài này mình sẽ giới thiệu đến các bạn một trong các hàm đệ quy tiếp theo, đó là hàm đệ quy đuôi (Tail Recursion). Đây là một hàm đệ quy cơ bản và được sử dụng khá nhiều trong lập trình.**



Chúng ta sẽ cùng nhau tìm hiểu về đệ quy đuôi là gì? và cơ chế hoạt động của đệ quy đuôi như thế nào thông qua 2 ví dụ.

**Mục lục**

* [1. Đệ quy đuôi là gì?](https://freetuts.net/de-quy-duoi-tail-recursion-2951.html#goto-h2-0)
* [2. Cơ chế hoạt động đệ quy đuôi](https://freetuts.net/de-quy-duoi-tail-recursion-2951.html#goto-h2-1)
  + [Ví dụ 1](https://freetuts.net/de-quy-duoi-tail-recursion-2951.html#goto-h3-0)
  + [Ví dụ 2](https://freetuts.net/de-quy-duoi-tail-recursion-2951.html#goto-h3-1)
* [3. Kết luận](https://freetuts.net/de-quy-duoi-tail-recursion-2951.html#goto-h2-2)

## 1. Đệ quy đuôi là gì?

Đệ quy đuôi là một trường hợp đặc biệt của [đệ quy tuyến tính](https://freetuts.net/de-quy-tuyen-tinh-2946.html). Giống như tên của nó, đệ quy đuôi là hàm thực hiện gọi đệ quy ở sau cùng.

Trong hàm đệ quy đuôi có thể gọi nhiều lần chính nó, nó có dạng tương tự như dưới đây.

*Bài viết này được đăng tại [free tuts .net]*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **int** UCLN(**int** m, **int** n){  **int** r;    if(m<n) return UCLN(n,m);    r = m % n;    if(r == 0) return n;    else return UCLN(n,r);  } |

Đây là hàm tìm ước số chung lớn nhất sử dụng đệ quy đuôi để thực hiện. Như các bạn thấy trong hàm này gọi lại chính nó 2 lần và kết thúc hàm gọi lại chính đệ quy đó.

## 2. Cơ chế hoạt động đệ quy đuôi

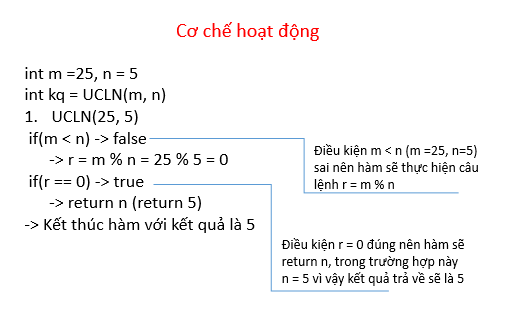
Ở phần này mình sẽ lấy hàm đệ quy UCLN() để chạy hai ví dụ với hai tham số khác nhau, để các bạn hiểu rõ hơn về cơ chế hoạt động của đệ quy đuôi.

**Hàm UCLN()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **int** UCLN(**int** m, **int** n){  **int** r;    if(m<n) return UCLN(n,m);    r = m % n;    if(r == 0) return n;    else return UCLN(n,r);  } |

### Ví dụ 1

Trong ví dụ này mình sẽ lấy tham số truyền vào cho hàm UCLN() là m = 25 và n = 5, cụ thể như hình dưới đây:

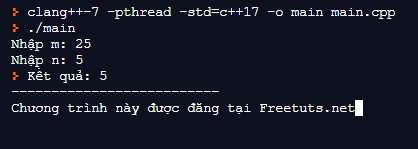


Trong trường hợp này hàm chỉ thực hiện một lần và không gọi chính nó lần nào nên không có kết quả nào được đẩy vào Stack.

**Code mẫu:**

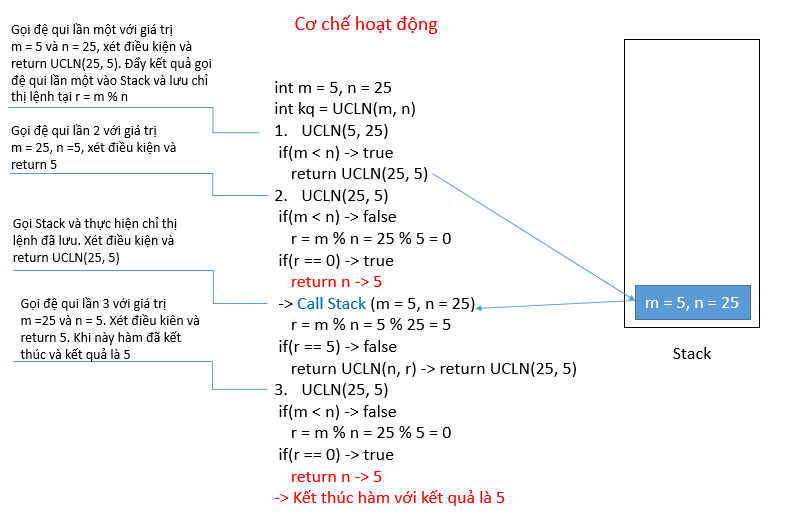
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <iostream>  using namespace std;    **int** UCLN(**int** m, **int** n){  **int** r;    if(m<n) return UCLN(n,m);    r = m % n;    if(r == 0) return n;    else return UCLN(n,r);  }    **int** main() {  **int** kq,m,n;    cout<<"Nhập m: ";    cin>>m;    cout<<"Nhập n: ";    cin>>n;    kq = UCLN(m,n);    cout<<"Kết quả: "<<kq;    cout<<"\n--------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



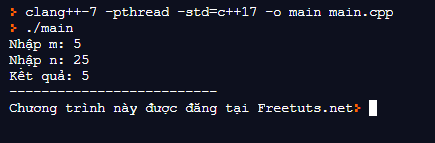
### Ví dụ 2

Cũng là hàm UCLN(), nhưng lần này mình sẽ truyền vào tham số m = 5 và n = 25. Hãy cùng xem cơ chế hoạt động của hàm lúc này sẽ như thế nào nhé:



**\*Lưu ý:** Các bạn sẽ dễ hiểu nhầm rằng hàm đã kết thúc ở bước gọi đệ quy lần hai sau khi return n -> 5. Nhưng trong Stack vẫn còn giá trị, vì vậy phải gọi Stack và thực hiện cho đến khi trong Stack rỗng.

**Kết quả:**



## 3. Kết luận

Thông qua hai ví dụ trên, đã biểu diễn cơ chế hoạt động của đệ quy đuôi rất rõ ràng. Các bạn có thể thấy cả hai ví dụ đều cho về cùng một kết quả, nhưng cách thức nó hoạt động lại khác nhau hoàn toàn. Vì vậy các bạn phải nắm thật kỹ các bước gọi hàm, chạy hàm, xét điều kiện và return kết quả. Nếu thiếu một bước nào đó có thể dẫn đến bài toán sai. Chúc các bạn thực hiện thành công!!!

**Đệ quy nhị phân (Binary Recursion)**

**Trong bài này mình sẽ giới thiệu đến các bạn một trong các hàm đệ quy tiếp theo đó là đệ quy nhị phân (Binary Recursion).**



Đây là một hàm đệ quy khá phổ biến trong quá trình xử lý về tìm kiếm, sắp xếp, các phép toán trên cây.

Chúng ta sẽ tìm hiểu về đệ quy nhị phân là gì? Cơ chế hoạt động của nó ra sao? Các bạn cùng mình tìm hiểu nhé.

**Mục lục**

* [1. Đệ quy nhị phân là gì?](https://freetuts.net/de-quy-nhi-phan-binary-recursion-2953.html#goto-h2-0)
* [2. Cơ chế của đệ quy nhị phân](https://freetuts.net/de-quy-nhi-phan-binary-recursion-2953.html#goto-h2-1)
* [3. Kết luận](https://freetuts.net/de-quy-nhi-phan-binary-recursion-2953.html#goto-h2-2)

**1. Đệ quy nhị phân là gì?**

Đệ quy nhị phân là dạng đệ quy gọi hai lần chính nó. Hiểu đơn giản là trong một hàm đệ quy, mà có dòng lệnh gọi chính hàm đó hai lần.

*Bài viết này được đăng tại [free tuts .net]*

Ta có một hàm tìm dãy số Fibonacci sử dụng đệ quy như sau:

|  |  |
| --- | --- |
| 1  2  3  4 | **int** fib(**int** n){  if(n <= 2) return 1; // điểm dừng  return fib(n - 1) + fib(n - 2);  } |

Một hàm tương tự như trên được gọi là hàm đệ quy nhị phân, trong hàm fib() gọi chính nó hai lần để tính.

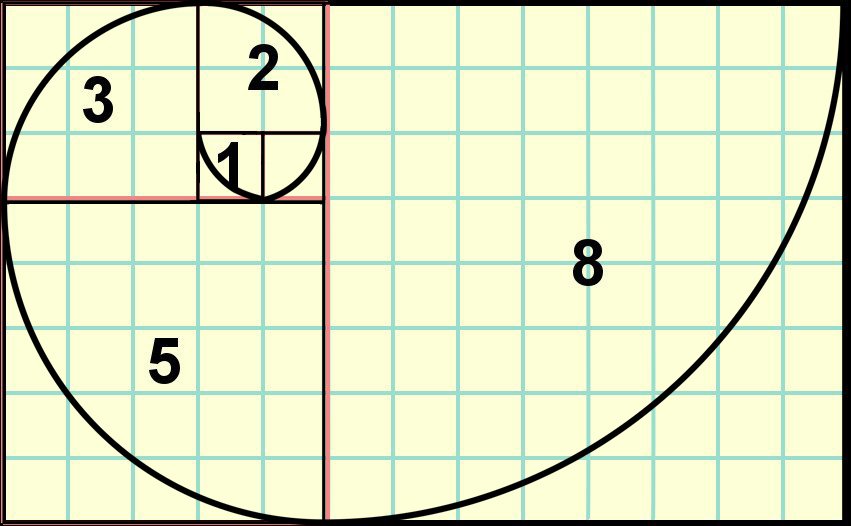
**2. Cơ chế của đệ quy nhị phân**

Để biểu diễn cơ chế của hàm đệ quy nhị phân ta có hai cách biểu diễn. Cách thứ nhất là biểu diễn bằng Stack và cách thứ hai là biểu diễn bằng cây (đây là cách thường được sử dụng nhất).

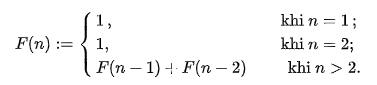
Trong phần này mình sẽ biểu diễn cơ chế hoạt động của đệ quy nhị phân bằng cây. Chúng ta sẽ lấy bài toán tìm số fibonacci thứ n (với n được nhập từ bàn phím).

Để làm được bài toán chúng ta cần hiểu được số fibonacci là số như thế nào?

Theo [wikipedia](https://freetuts.net/de-quy-nhi-phan-binary-recursion-2953.html) thì dãy số fibonacci là dãy vô hạn các số tự nhiên bắt đầu bằng hai phần tử 0 1 hoặc 1 1. Các phần tử sau đó được thiết lập theo quy tắt mỗi phần tử luôn bằng tổng hai phần tử trước nó.



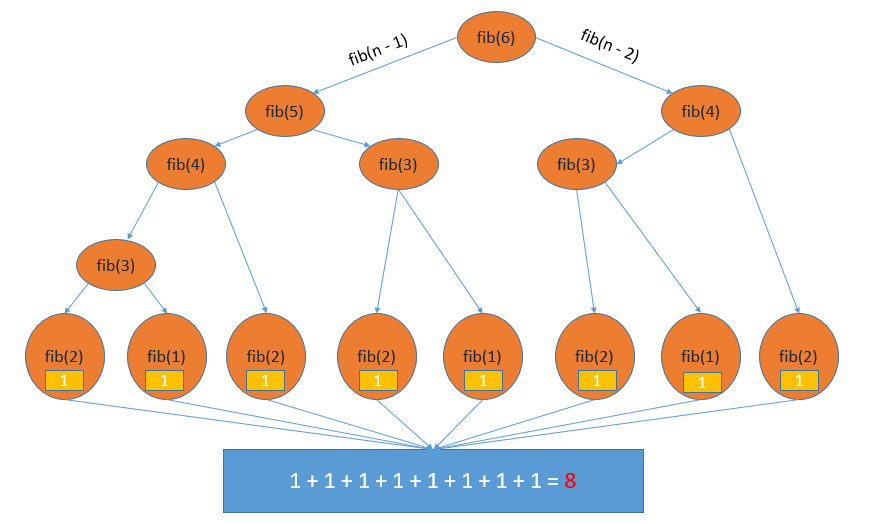
**Công thức của nó là:**



Ta có hàm fib() sử dụng đệ quy nhị phân để tìm số fibonacci thứ n như sau:

|  |  |
| --- | --- |
| 1  2  3  4 | **int** fib(**int** n){    if(n <= 2) return 1;    return fib(n - 1) + fib(n - 2);  } |

Giả sử chúng ta cần tìm số fibonacci ở vị trí 6 (fib(6)), khi đó cơ chế hoạt động được biểu diễn dưới dạng cây của nó như sau:



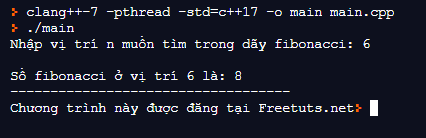
Điểm dừng trong hàm đệ quy trên chính là n <= 2, vì vậy fib(2) và fib(1) sẽ return 1. Sau khi hàm kết thúc, kết quả bằng tổng các fib(1) và fib(2) là 8, đây chính là giá trị của vị trí số 6 trong dãy fibonacci.

Các bạn có thể vẽ cây biểu diễn cho fib(7), fib(8), … để có thể hiểu được bài. Nếu các bạn thực hành bằng tay nhiều điều đó giúp các bạn nhớ rất lâu, hãy cố gắng luyện tập nhé.

**Code mẫu:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <iostream>  using namespace std;    **int** fib(**int** n){    if(n <= 2) return 1;    return fib(n - 1) + fib(n - 2);  }    **int** main() {  **int** kq,n;    cout<<"Nhập vị trí n muốn tìm trong dãy fibonacci: ";    cin>>n;    kq = fib(n);    cout<<"\nSố fibonacci ở vị trí "<<n<<" là: "<<kq;    cout<<"\n-----------------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



**3. Kết luận**

Như vậy là chúng ta đã tìm hiểu về khái niệm đệ quy nhị phân, cơ chế hoạt động của nó trong C++ thông qua ví dụ tìm vị trí của số n trong dãy fibonacci. Qua bài này mình muốn các bạn nắm rõ được cơ chế hoạt động của đệ quy nhị phân. Bởi vì đây là một hàm rất quan trọng để áp dụng cho sắp xếp, tìm kiếm, … Các bạn có thể tìm hiểu thêm các [bài tập C++](https://freetuts.net/hoc-c/bai-tap-c++-co-loi-giai).

**Đệ quy đa tuyến (Exponential Recursion)**

**Trong bài này mình sẽ giới thiệu đến các bạn một trong các hàm đệ quy tiếp theo đó chính là đệ quy đa tuyến (Expenential Recursion).**



Đây là một thuật toán được sử dụng khá nhiều trong các bài toán về sắp xếp.

Chúng ta sẽ cùng nhau tìm hiểu về đệ quy đa tuyến là gì? một vài ví dụ sử dụng đệ quy đa tuyến.

**Mục lục**

* [1. Đệ quy đa tuyến là gì?](https://freetuts.net/de-quy-da-tuyen-exponential-recursion-2958.html#goto-h2-0)
* [2. Ví dụ 1 đệ quy đa tuyến](https://freetuts.net/de-quy-da-tuyen-exponential-recursion-2958.html#goto-h2-1)
* [3. Ví dụ 2 đệ quy đa tuyến](https://freetuts.net/de-quy-da-tuyen-exponential-recursion-2958.html#goto-h2-2)
* [Kết luận](https://freetuts.net/de-quy-da-tuyen-exponential-recursion-2958.html#goto-h2-3)

**1. Đệ quy đa tuyến là gì?**

Một hàm được gọi là đệ quy đa tuyến nếu mỗi lần gọi đệ quy nó phát sinh ra khoảng n lần gọi đệ quy khác. Thông thường câu lệnh gọi đệ quy được đặt trong các vòng lặp.

*Bài viết này được đăng tại [free tuts .net]*

Ví dụ chúng ta có một hàm đệ quy đa tuyến như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | void daTuyen(**int** i, **int** n, **int** \*X)  {  **int** val;      for (val = 0; val < 2; val++)      {          X[i] = val;          if (i == (n-1))          {  **int** j;              for(j = 0; j < n; j ++)              {                  cout<< X[j];              }              cout<<"\n";          }          else          {              daTuyen(i+1, n, X);          }      }  } |

Đây là một hàm tìm các dãy nhị phân có chiều dài n. Như các bạn thấy trong hàm có sử dụng vòng lặp và thực hiện gọi đệ quy ngay trong vòng lặp.

Cơ chế hoạt động của nó tương tự các hàm đệ quy khác, thay vào đó nó chỉ lặp đi lặp lại nhiều lần hơn mà thôi. Các bạn có thể thao khảo cơ chế hoạt động của các hàm đệ quy [tại đây](https://freetuts.net/de-quy-duoi-tail-recursion-2951.html).

Bây giờ các bạn cùng mình tìm hiểu một vài ví dụ áp dụng hàm đệ quy đa tuyến để có thế hiểu rõ hơn về nó nhé.

**2. Ví dụ 1 đệ quy đa tuyến**

Ở ví dụ đầu tiên mình sẽ thực hiện bài toán tìm tất cả các dãy nhị phân có chiều dài n nhập từ bàn phím.

Hàm dayNhiPhan()

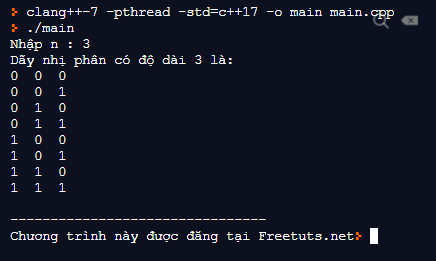
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | void dayNhiPhan(**int** i, **int** n, **int** \*X)  {  **int** val;    // val là các giá trị có thể gán cho các vị trí trong x      for (val = 0; val < 2; val++)      // val có thể nhận hai giá trị là 0 và 1      {          X[i] = val;            if (i == (n-1))         // nếu i là phần tử cuối của dãy          {  **int** j;              for(j = 0; j < n; j ++)         // thì tin ra nhị phân mới tìm được              {                  cout<<X[j];              }              cout<<"\n";          }            else              // nếu i chưa phải là phần tử cuối thì gán cho i sau là i+1.          {              dayNhiPhan(i+1, n, X); // gọi đệ quy tiếp tục thực hiện hàm          }      }  } |

Mình đã chú thích rất rõ ràng trong hàm, các bạn hãy đọc kỹ để hiểu rõ hơn nhé.

**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | #include<stdio.h>  #include<iostream>  using namespace std;  void dayNhiPhan(**int** i, **int** n, **int** \*X)  {  **int** val;    // val là các giá trị có thể gán cho các vị trí trong x      for (val = 0; val < 2; val++)      // val có thể nhận hai giá trị là 0 và 1      {          X[i] = val;            if (i == (n-1))         // nếu i là phần tử cuối của dãy          {  **int** j;              for(j = 0; j < n; j ++)         // thì tin ra nhị phân mới tìm được              {                  cout<<X[j];              }              cout<<"\n";          }            else              // nếu i chưa phải là phần tử cuối thì gán cho i sau là i+1.          {              dayNhiPhan(i+1, n, X); // gọi đệ quy tiếp tục thực hiện hàm          }      }  }    **int** main()  {  **int** n;      cout<<"Nhập n : ";      cin>>n;    **int** X[n];      cout<<"Dãy nhị phân có độ dài "<<n<<" là:\n";      dayNhiPhan(0, n, X);        cout<<"\n--------------------------------\n";      cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



**3. Ví dụ 2 đệ quy đa tuyến**

Trong ví dụ này mình sẽ thực hiện sắp xếp các số trong mảng M, sử dụng đệ quy đa tuyến để sắp xếp. Với các phần tử trong mảng M được nhập từ người dùng.

Hàm sort()

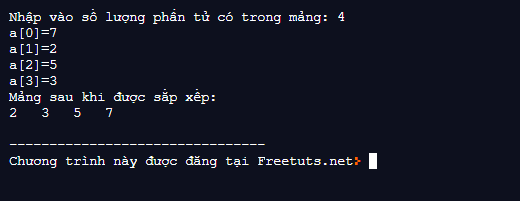
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void sort(**int** arr[], **int** n, **int** i){  **int** j, swap;    //thực hiện vòng lặp để sắp xếp các phần tử    for(j = i + 1; j < n; j++){      if(arr[i] > arr[j]){ // Nếu phần tử trước lớn hơn phần tử sau thì thực hiện tráo đổi vị trí giữa hai phần tử        swap = arr[i];        arr[i] = arr[j];        arr[j] = swap;      }      sort(arr, n, i + 1);//Tiếp tục gọi đệ quy và thực hiện đến khi hàm kết thúc    }  } |

Mình đã giải thích từng dòng lệnh trong hàm, các bạn có thể chạy tay hàm này để có thể hiểu được từng bước hoạt động của nó. Ở các bài trước mình đã biểu diễn cơ chế hoạt động một cách cụ thể thông qua Stack và cây.

**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | #include<stdio.h>  #include<iostream>  using namespace std;  /\* hàm xuất các phần tử trong mảng \*/  void printArr(**int** arr[], **int** n){    for(**int** i = 0; i < n; i++) //chạy vòng lặp từng phần tử trong mảng      cout<<arr[i]<<"\t"; // xuất các phần tử trong mảng    cout<<endl;  }  /\* hàm sắp xếp các phần tử trong mảng \*/  void sort(**int** arr[], **int** n, **int** i){  **int** j, swap;    //thực hiện vòng lặp để sắp xếp các phần tử    for(j = i + 1; j < n; j++){      if(arr[i] > arr[j]){ // Nếu phần tử trước lớn hơn phần tử sau thì thực hiện tráo đổi vị trí giữa hai phần tử        swap = arr[i];        arr[i] = arr[j];        arr[j] = swap;      }      sort(arr, n, i + 1);//Tiếp tục gọi đệ quy và thực hiện đến khi hàm kết thúc    }  }    **int** main()  {  **int** n;  **int** a[n];      do{          cout << "\nNhập vào số lượng phần tử có trong mảng: ";          cin >> n;      }while(n <= 0);        for(**int** i = 0;i < n;i++){          cout<<"a["<<i<<"]=";         cin >> a[i];      };       sort(a, n, 0);      cout<<"Mảng sau khi được sắp xếp: \n";      printArr(a, n);        cout<<"\n--------------------------------\n";      cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



**Kết luận**

Trên đây mình đã thực hiện hai ví dụ về việc sử dụng đệ quy đa tuyến. Các bạn hãy luyện tập bằng cách chạy tay các bài tập trên để nắm rõ các bước thực hiện của nó. Nếu các bạn thực hiện nhiều lần và thành thạo, điều đó giúp các bạn nhớ rất lâu và có một tư duy logic tốt.

**Đệ quy lồng (Nested Recursion)**

**Trong bài này mình sẽ giới thiệu các bạn một trong các hàm đệ quy tiếp theo đó chính là đệ quy lồng (Nested Recursion).**



Đây là một hàm đệ quy được sử dụng khá nhiều trong lập trình nói chung và C++ nói riêng.

Chúng ta sẽ cùng nhau tìm hiểu đệ quy lồng (Nested Recursion) là gì? cách nó hoạt động ra sao?

**Mục lục**

* [1. Đệ quy lồng là gì?](https://freetuts.net/de-quy-long-nested-recursion-2959.html#goto-h2-0)
* [2. Cơ chế đệ quy lồng (Nested Recursion)](https://freetuts.net/de-quy-long-nested-recursion-2959.html#goto-h2-1)
* [3. Kết luận](https://freetuts.net/de-quy-long-nested-recursion-2959.html#goto-h2-2)

**1. Đệ quy lồng là gì?**

Đệ quy lồng là loại đệ quy gọi đối số của nó là một đệ quy. Hiểu đơn giản là tham số truyền vào của hàm đệ quy là một đệ quy.

*Bài viết này được đăng tại [free tuts .net]*

Ví dụ chúng ta có một hàm đệ quy lồng như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **int** ackerman(**int** m, **int** n){    if(m == 0)       return n + 1;    else if(n == 0)       return ackerman(m - 1, 1);    else       return ackerman(m-1, ackerman(m, n-1));  } |

Trong hàm trên, tham số truyền vào khi return ackerman(m-1, ackerman(m, n-1)) là một đệ quy. Một hàm tương tự như vậy được gọi là đệ quy lồng.

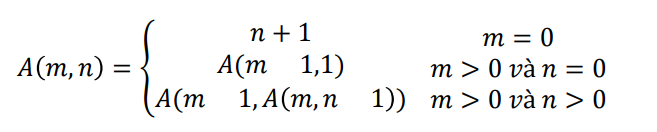
Các bạn cùng mình xem cơ chế hoạt động của nó ra sao nhé.

**2. Cơ chế đệ quy lồng (Nested Recursion)**

Trong phần này mình sẽ lấy hàm ackerman() để làm ví dụ thực hiện xem cơ chế hoạt động của nó như thế nào.

Để làm được nó, việc đầu tiền các bạn phải hiểu được nó là hàm như thế nào. Các bạn có thể tìm hiểu thêm về [hàm ackerman()](https://freetuts.net/de-quy-long-nested-recursion-2959.html).

Ở đây mình sẽ để phương trình truy hồi của nó để các bạn dễ hình dung nhé.



Trong phương trình trên có hai phần, phần thứ nhất là kết quả trả về của phương trình và phần thứ hai là điểm dừng.

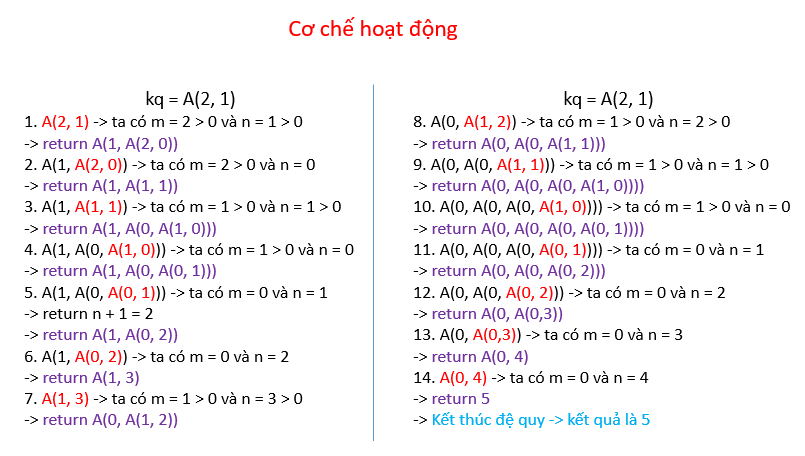
Trong trường hợp m > 0 và n > 0, thì tham số của hàm ackerman() chính là một đệ quy.

Để ngắn gọn hơn, mình sẽ ký hiệu hàm ackerman() = A().

**Hàm A()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **int** A(**int** m, **int** n){    if(m == 0)      return n + 1;    else if(n == 0)      return A(m - 1, 1);    else      return A(m-1, A(m, n-1));  } |

Giả sử mình truyền vào hàm A() với hai tham số là m = 2 và n = 1. Khi đó cơ chế hoạt động của hàm trên được biểu diễn như hình dưới đây.



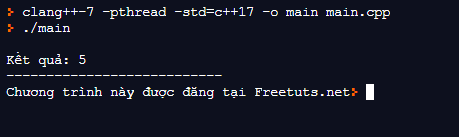
Các bạn có thể thấy nó được thực hiện rất nhiều lần gọi đệ quy và các đệ quy này được lồng vào nhau. Chúng ta đơn giản chỉ cần thực hiện từng đệ quy một, theo đúng tuần tự thì nó không có gì khó cả.

Các bạn có thể luyện tập bằng cách chạy code bằng tay, truyền tham số với m và n tùy chọn. Điều này giúp các bạn rèn luyện tư duy logic rất tốt.

**Code mẫu:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include <iostream>  using namespace std;    **int** A(**int** m, **int** n){    if(m == 0) // nếu m == 0 thì return n + 1      return n + 1;    else if(n == 0) // nếu n == 0 thì return đệ quy A(m - 1, 1)      return A(m - 1, 1);    else // nếu m > 0 và n > 0 thì return đệ quy lồng A(m-1, A(m, n-1))      return A(m-1, A(m, n-1));  }  **int** main() {  **int** kq = A(2,1); // truyển tham số đầu vào cho A() là m = 2 và n = 1    cout<<"\nKết quả: "<<kq;      cout<<"\n---------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



**3. Kết luận**

Như vậy là chúng ta đã tìm hiểu xong về hàm đệ quy lồng. Đây là một hàm được sử dụng rất nhiều trong lập trình, vì vậy các bạn hãy luyện tập thật nhiều để thành thạo nó nhé. Các bạn có thể tham khảo thêm các bài tập C++ và bài tập C tại đây.

**Đệ quy tương hỗ (Mutual Recursion)**

**Trong bài này mình sẽ giới thiệu các bạn một hàm đệ quy cuối cùng trong các hàm đệ quy cơ bản đó chính là đệ quy tương hỗ (Mutual Recursion).**



Chúng ta sẽ cùng nhau tìm hiểu về đệ quy tương hỗ là gì? hoạt động như thế nào?

**Mục lục**

* [1. Đệ quy tương hỗ là gì?](https://freetuts.net/de-quy-tuong-ho-mutual-recursion-2961.html#goto-h2-0)
* [2. Cơ chế đệ quy tương hỗ (Mutual Recursion)](https://freetuts.net/de-quy-tuong-ho-mutual-recursion-2961.html#goto-h2-1)
* [3. Kết luận](https://freetuts.net/de-quy-tuong-ho-mutual-recursion-2961.html#goto-h2-2)

**1. Đệ quy tương hỗ là gì?**

Đệ quy tương hỗ là loại đệ quy không gọi đệ quy trực tiếp chính nó, mà gọi một hàm khác. Trong hàm khác lại gọi lại nó. Ví dụ chúng ta có hàm A() gọi đệ quy hàm B() và trong hàm B() gọi lại đệ quy hàm A().

Ta có đệ quy tương hỗ như sau:

*Bài viết này được đăng tại [free tuts .net]*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **bool** isEven(**int** n);  **bool** isOdd(**int** n);    **bool** isEven(**int** n){    if(n == 0)      return true;    else      return isOdd(n - 1);  }    **bool** isOdd(**int** n){    return !isEven(n);  } |

Ở trên chúng ta có hai hàm là hàm isEven() và hàm isOdd(). Hai hàm này gọi đệ quy qua lại lẫn nhau, như vậy đây là hai hàm tương hỗ.

Để hiểu hơn chúng ta cùng xem cơ chế hoạt động của nó như thế nào nhé.

**2. Cơ chế đệ quy tương hỗ (Mutual Recursion)**

Trong phần này mình sẽ sử dụng hai hàm trên để kiểm tra một số n nhập và là số chẵn hay số lẻ. Thực ra bài toán này không cần sử dụng hàm đệ quy, nhưng vì đây là một bài toán đơn giản nên mình sử dụng nó để các bạn dễ nắm bắt hơn.

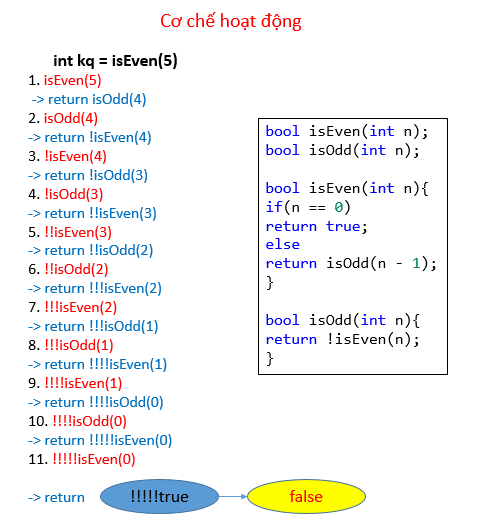
Như các bạn đã biết thì số chẵn là số chia hết cho 2 (2n) và số lẻ là số chia cho 2 dư 1 (2n - 1).

Để kiểm tra số n là số chẵn hay số lẻ sử dụng hàm đệ quy, ta có hàm đệ quy như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **bool** isEven(**int** n);  **bool** isOdd(**int** n);    **bool** isEven(**int** n){    if(n == 0)      return true;    else      return isOdd(n - 1);  }    **bool** isOdd(**int** n){    return !isEven(n);  } |

Nếu hàm IsEven() trả về true tức là n là số chẵn và ngược lại trả về false thì n là số lẻ.

Giả sử chúng ta truyền vào n = 5 thì cơ chế hoạt động của nó như hình dưới đây:



**\* Lưu ý:** Nếu các bạn đã học qua khóa học [C++ căn bản](https://freetuts.net/hoc-c/hoc-c++-can-ban-va-nang-cao) thì chắc chắn các bạn đã biết được toán tử NOT (!). Đây là một toán tử đảo ngược trạng thái logic. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false.

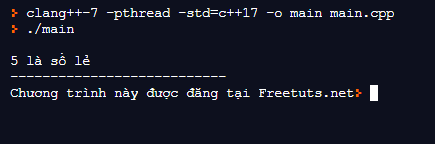
Dựa vào đó ta có !!!!!true = false. Như vậy kết quả trả về là false.

Các bạn có thể luyện tập bằng cách chạy code bằng tay với các tham số n khác. Đây là một trong những cách luyên tập giúp các bạn rèn luyện tư duy logic rất tốt.

**Code mẫu:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <iostream>  using namespace std;    **bool** isEven(**int** n);  **bool** isOdd(**int** n);    **bool** isEven(**int** n){    if(n == 0)      return true;    else      return isOdd(n - 1);  }    **bool** isOdd(**int** n){    return !isEven(n);  }  **int** main() {  **int** n = 5;  **bool** kq = isEven(n);    if(kq == true)      cout<<n<<" là số chẵn";    else      cout<<n<<" là số lẻ";      cout<<"\n---------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



**3. Kết luận**

Như vậy chúng ta đã tìm hiểu xong về đệ quy tương hỗ, đây là một hàm đệ quy được sử dụng nhiều trong các bài toàn phức tạp. Các bạn hãy luyện tập thật nhiều để thành thạo nó nhé. Đây cũng là bài cuối cùng trong series thuật toán đệ quy. Chúc các bạn học thật tốt nhé !!!

# Bài toán tháp Hà Nội: Sử dụng đệ quy để giải

**Trong bài này mình sẽ thực hiện giải một bài toán rất kinh điển đó chính là bài toán tháp Hà Nội.**



Đây là một bài toán áp dụng đệ quy để giải, có lẽ các bạn cũng đã từng nghe qua bài toán này rồi. Và chúng ta sẽ lần lượt tìm hiểu về bài toán tháp Hà Nội là gì? Sau đó mình sẽ đưa ra giải thuật để giải.

**Mục lục**

* [Bài toán tháp Hà Nội là gì?](https://freetuts.net/bai-toan-thap-ha-noi-2964.html#goto-h2-0)
* [Ý tưởng đệ quy](https://freetuts.net/bai-toan-thap-ha-noi-2964.html#goto-h2-1)
* [Giải bài toán tháp Hà Nội bằng C++](https://freetuts.net/bai-toan-thap-ha-noi-2964.html#goto-h2-2)
  + [Giải thuật](https://freetuts.net/bai-toan-thap-ha-noi-2964.html#goto-h3-0)
  + [Hàm main()](https://freetuts.net/bai-toan-thap-ha-noi-2964.html#goto-h3-1)

## Bài toán tháp Hà Nội là gì?

Bài toán tháp Hà Nội là một trò chơi toán học rất phổ biến. Nó đơn giản chỉ là việc dịch chuyển các đĩa từ cột này sang cột khác. Nhưng để thành thạo luật chơi của nó thì rất khó.

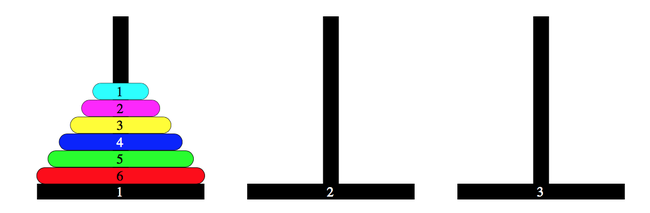
**Luật chơi được mô tả như sau:**

*Bài viết này được đăng tại [free tuts .net]*

Trò chơi này gồm một bộ các đĩa kích thước khác nhau, có lỗ ở giữa, nằm xuyên trên ba cái cột. Bài toán đố bắt đầu bằng cách sắp xếp các đĩa theo trật tự kích thước vào một cột, sao cho đĩa nhỏ nhất nằm ở trên cùng, tức là tạo thành một hình nón.

Yêu cầu của trò chơi là di chuyển toàn bộ số đĩa sang một cột khác, tuân theo các quy tắc sau:

* Một lần chỉ có 3 cột để di chuyển
* Chỉ di chuyển một đĩa trên cùng (không được di chuyển đĩa nằm giữa hay nằm cuối).
* Một đĩa chỉ có thể đặt lên một đĩa lớn hơn (không nhất thiết hai đĩa này phải có kích thước liền kề, tức là đĩa nhỏ nhất vẫn có thể đặt trên đĩa lớn nhất).



## Ý tưởng đệ quy

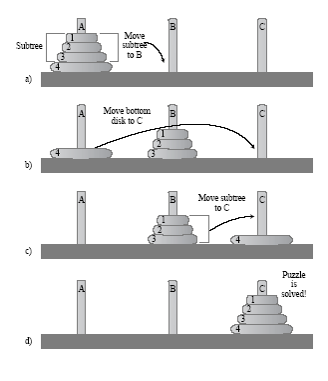
Dựa vào luật chơi của trò chơi, chúng ta sẽ áp dụng nó vào đệ quy để giải bài toán này bằng ngôn ngữ C++ nhé.

Trong bài toán này chúng ta cần quan tâm 4 vấn đề: số đĩa N, cột A, cột B, cột C.

Nhiệm vụ của chúng ta là chuyển N số đĩa từ cột A sang cột C, lấy cột B làm cột tạm.

**Ý tưởng:**

* Nếu đã biết cách chuyển N - 1 đĩa từ cột A sang cột B, ta chỉ cần chuyển đĩa thứ N (đĩa cuối cùng) từ cột A sang cột C, rồi chuyển N - 1 đĩa từ cột B sang cột C.
* Giải thuật không còn đệ quy khi chỉ có 1 đĩa, vì ta chuyển trực tiếp từ cột A sang cột C luôn mà không cần thông qua cột B.
* Độ phức tạp của nó là: 2n - 1 (lần dịch chuyển).



## Giải bài toán tháp Hà Nội bằng C++

Chúng ta đã có ý tưởng giải bài toán, chỉ cần dựa vào đó và áp dụng thêm kiến thức về đệ quy để bắt tay vào việc giải thôi nào.

### Giải thuật

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void move(**int** n,**char** A,**char** B,**char** C)   {        if(n==1){          cout<<A<<" ==> "<<C<<"\n";// nếu n = 1 thì dịch chuyển từ A -> C        }        else {          // Nếu n > 1 thì thực hiện lần lượt các bước          move(n - 1, A, C, B); // 1. Dịch chuyển n-1 đĩa từ A -> B          cout<<A<<" ==> "<<C<<"\n"; // 2. Dịch chuyển đĩa thứ n từ A -> C          move(n - 1, B, A, C); // 3. Dịch chuyển n-1 đĩa từ B -> C        }   } |

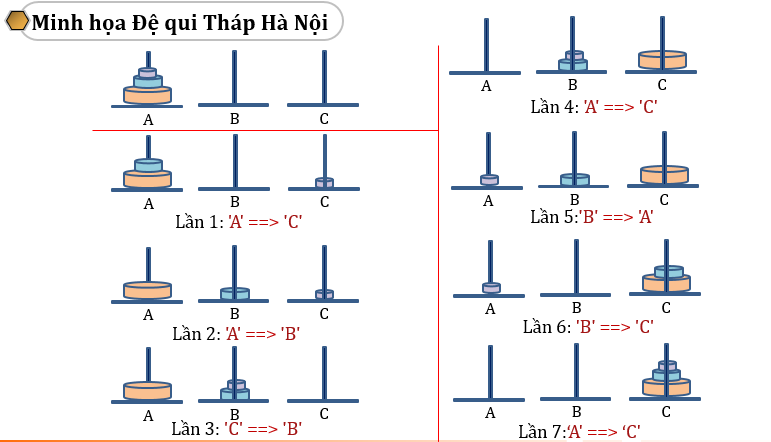
Như các bạn thấy chúng ta cần truyền 4 tham số cho hàm move() là: số đĩa n, cột A, cột B, cột C.

Nếu như n == 1 (chỉ có một đĩa) thì chúng ta chỉ cần chuyển đĩa đó từ cột A sang cột C là xong.

Trường hợp số đĩa n lớn hơn 1 thì chúng ta cần thực hiện dịch chuyển ba lần:

1. Chuyển n - 1 đĩa từ cột A sang cột B -> move(n - 1, A, C, B);
2. Chuyển đĩa còn lại (đĩa thứ n) từ cột A sang cột C -> cout<<A<<" ==> "<<C<<"\n";
3. Chuyển n - 1 đĩa từ cột B sang cột C -> move(n - 1, B, A, C);

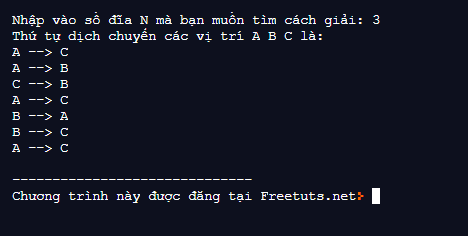
Giả sử chúng ta có n = 3 thì số lần thực hiện dịch chuyển bằng 2n - 1 = 23 - 1 = 7 (lần).



### Hàm main()

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #include <iostream>  using namespace std;    void move(**int** n,**char** A,**char** B,**char** C)   {        if(n==1){          cout<<A<<" ==> "<<C<<"\n";// nếu n = 1 thì dịch chuyển từ A -> C        }        else {          //// Nếu n > 1 thì thực hiện lần lượt các bước          move(n - 1, A, C, B); // 1. Dịch chuyển n-1 đĩa từ A -> B          cout<<A<<" ==> "<<C<<"\n"; //2. Dịch chuyển đĩa thứ n từ A -> C          move(n - 1, B, A, C); // 3. Dịch chuyển n-1 đĩa từ B -> C        }   }    **int** main() {  **int** n;    cout<<endl<<"Nhập vào số đĩa N mà bạn muốn tìm cách giải: ";    cin>>n;    cout<<"Thứ tự dịch chuyển các vị trí A B C là: \n";    move(n, 'A', 'B', 'C');      cout<<"\n------------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình tìm cách giải của trò chơi tháp Hà Nội. Chúc các bạn thực hiện thành công!!!