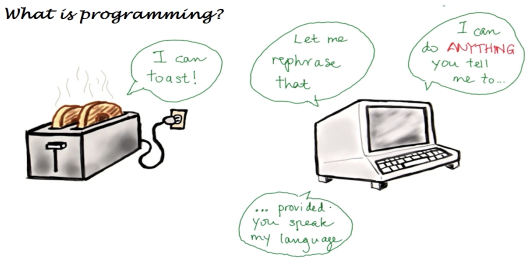


Tuần 05

Đệ quy

CT101 – Lập trình căn bản

Khoa CNTT&TT – ĐHCT

Mục đích

- Đến thời điểm này, để lập trình tính ra n! ta phải chạy qua vòng lặp để tính: n! = n*(n-1)*(n-2)*…*1
- Câu hỏi đặt ra là: Có cách nào để ta biểu diễn cách tính n! một cách toán học và gọn gàng hơn không?
- Nghĩa là ta chỉ cần tính n! = n*(n-1)! với trường hợp đơn giản nhất là 0! = 1. Trong trường hợp này, ta sử dụng hàm đệ quy!
- Nếu sử dụng hàm đệ quy thì thông thường thời gian ta bỏ ra để giải quyết một bài toàn là ngắn hơn so với việc dùng các vòng lặp đơn thuần. Tuy nhiên thời gian chạy là không ngắn hơn, thậm chí là dài hơn nhiều!

Yêu cầu

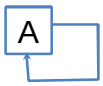
- Sau khi học xong phần này chúng phải biết được phương pháp thiết kế hàm đệ quy để giải quyết các bài toán mà cách giải mang tính đệ quy!

Nội dung

- Hàm đệ quy là gì?
- Viết hàm đệ quy như thế nào?
- Việc thực thi hàm đệ quy diễn ra như thế nào?

Hàm đệ quy (Recursive function)

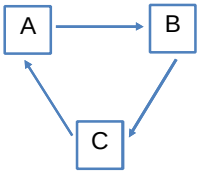
- Hàm đệ quy là hàm mà nó gọi chính nó hoặc ở trong một chu trình xoay vòng các lời gọi hàm.
- Nếu một hàm gọi chính nó, ta gọi là hàm đệ quy tức thì (immediate recursion).




```
void A(){
   A();
   return;
}
```


Hàm đệ quy (Recursive function)

- Trường hợp một hàm ở trong một chu trình các lời gọi hàm:




```
void C(){
   A();
   return;
}

void B(){
   C();
   return;
}

void A(){
   B();
   return;
}
```

Hàm đệ quy (Recursive function)

- Để giải quyết một bài toán, việc sử dụng hàm đệ quy là phương thức không hiệu quả - xét về thời gian chạy.
- Tuy nhiên việc nghiên cứu về hàm đệ quy là thú vị. Ở đây ta chỉ nghiên cứu về hàm đệ quy tức thời vì nó cũng đủ gây rất nhiều khó khăn trong quá trình lập giải thuật và cài đặt chương trình.

Viết hàm đệ quy

- Nhìn chung, một hàm đệ quy có khuôn dạng sau:

```
ReturnType Function( tham số thích hợp ) {
   if trường hợp đơn giản
      return giá trị đơn giản;   // trường hợp cơ sở, điều kiện dừng
   else
      gọi hàm với phiên bản đơn giản hơn của bài toán;
}
```

Ví dụ: tính n!

- Nhắc lại: n! = n * (n - 1)!
- Đây có thể xem là một hàm đệ quy, vì để tính được n! ta phải tính (n - 1)!
- Trường hợp đơn giản nhất: 0! = 1.

```
int Factorial(int n) {
   if (n == 0) return 1;          // Simple case: 0! = 1
   return (n * Factorial(n - 1));  // General function: n! = n * (n - 1)!
}
```

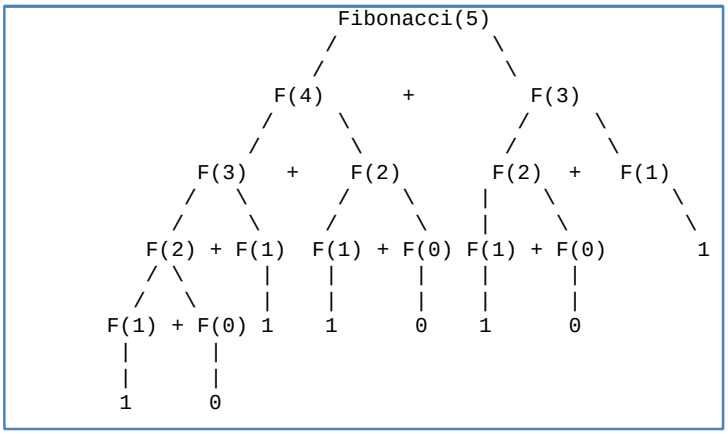

Duyệt qua trình tự thực thi đệ quy

- Cho hàm tính số Fibonacci như sau:

```
int Fibonacci(int x) {
   if (x == 0) return 0;  // Stopping conditions
   if (x == 1) return 1;
   return Fibonacci(x - 1) + Fibonacci(x - 2);
}
```

Duyệt qua trình tự thực thi đệ quy

- Nếu n=5 thì dãy các lời gọi đệ quy sau sẽ được thực thi:



Tổng kết

- Ta dùng hàm đệ quy nhằm mục tiêu viết chương trình rõ ràng hơn với thời gian lập trình ít hơn. Tuy nhiên dung lượng bộ nhớ và thời gian chạy lại cao hơn phương pháp dùng vòng lặp!
- Nhớ rằng, bất kỳ hàm đệ quy nào cũng phải có trường hợp cơ sở và hàm đệ quy phải diễn tiến về trường hợp cơ sở này.

Kiểm tra kiến thức

1. Hai phần của hàm đệ quy tức thì là:
   - A. (1) Nếu vấn đề là đơn giản thì giải ra ngay, và (2) nếu vấn đề không thể được giải ra ngay thì chia nó ra thành các vấn đề nhỏ hơn để giải.
   - B. (1) Chia vấn đề thành các vấn đề nhỏ hơn, và (2) đưa ra lời giải trực tiếp cho các vấn đề khó hơn.
   - C. (1) Bỏ qua các trường hợp khó, và (2) giải các trường hợp dễ.
   - D. (1) Giải vấn đề bằng cách yêu cầu nó giải chính nó, và (2) giải các trường hợp đơn giản trong một bước.

Kiểm tra kiến thức

2. Làm sao để Sherlock Holmes điều tra một vụ sát nhân?
   - A. (1) Bắt ngay người quản gia.
   - B. (1) Tra vấn một nhân chứng, sau đó (2) tra vấn nạn nhân.
   - C. (1) Loại bỏ một nhân chứng, và (2) loại bỏ một nhân chứng khác.
   - D. (1) Khi chỉ còn một nghi can thì đó chính là tên sát nhân, (2) Xem xét các chứng cứ để loại trừ một nghi can, sau đó loại trừ các nghi can còn lại.

Kiểm tra kiến thức

3. Việc chạy hàm đệ quy tốn rất nhiều bộ nhớ vì:
   - A. Các hàm đệ quy có xu hướng khai báo rất nhiều biến.
   - B. Các lời gọi hàm trước đó vẫn còn mở khi một hàm gọi chính nó và thông tin về các lời gọi hàm trước đó vẫn còn được lưu trong stack của bộ nhớ.
   - C. Có quá nhiều phiên bản của mã lệnh của hàm được tạo ra.
   - D. Hàm đệ quy cần giá trị cực lớn cho dữ liệu của nó.

Kiểm tra kiến thức

4. Giá trị mà hàm ***mystery*** sau đây trả về sẽ là gì khi ta gọi nó với tham số có giá trị 4:
   - A. 0
   - B. 1
   - C. 4
   - D. 24

```
int mystery(int number)
{
   if ( number <= 1 )
      return 1;
   else
      return number * mystery(number - 1);
}
```

Kiểm tra kiến thức

5. Hàm ***sum*** sau đây có thực sự trả về giá trị là tổng các số từ ***a*** đến ***b*** hay không, giả sử là ***a*** < ***b***:

   A. Có

   B. Không

```
int sum(int a, int b){
   if(a==b)
      return b;
   else
      return a + sum(a + 1, b);
}
```


Kiểm tra kiến thức

6. Hãy dự đoán kết quả của chương trình sau:

   A. 2

   B. 4

   C. 16

   D. Có lỗi

```
#include <stdio.h>

int fun(int n)
{
   if (n == 4)
      return n;
   else return 2*fun(n+1);
}


int main()
{
   printf("%d ", fun(2));
   return 0;
}
```

Kiểm tra kiến thức

7. Hàm *fun*(4,3) sau đây trả về kết quả gì?

A. 13
B. 12
C. 9
D. 10

```
int fun(int x, int y)
{
  if (x == 0)
    return y;
  return fun(x - 1,  x + y);
}
```


Kiểm tra kiến thức

8. Hàm *fun* sau đây muốn tính gì?

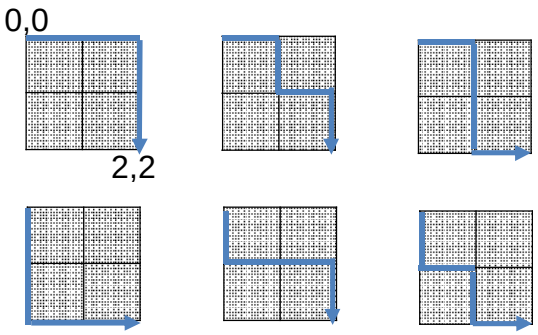
A. x+y
B. x+x*y
C. x*y
D. x^y

```
int fun(int x, int y)
{
    if (y == 0)   return 0;
    return (x + fun(x, y-1));
}
```

Bài tập tổng kết

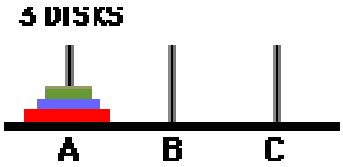
1. Hãy viết hàm đệ quy int2bin(int n) để in ra số nhị phân tương ứng với số thập phân n.
2. Hãy viết hàm đệ quy gdc(int m, int n) để trả về giá trị là ước số chung lớn nhất của hai số m, n.
3. Ví dụ trong hình dưới biểu diễn một lưới kích thước 2x2. Chúng ta bắt đầu từ điểm có toạ độ (0, 0) và tìm đường đi đến điểm có toạ độ (2,2) - từ điểm trên cùng bên trái đến điểm dưới cùng bên phải. Tại mỗi điểm, chỉ có thể có 2 bước đi: đi xuống hoặc đi sang phải. Với lưới 2x2 thì ta có 6 con đường đi có thể từ (0,0) đến (2,2). Với điểm dưới cùng bên phải là (x,y), x>=0, y>=0, hãy lập trình để tính ra có bao nhiêu con đường đi có thể từ (0,0) đến nó.

Bài tập tổng kết



Bài tập tổng kết

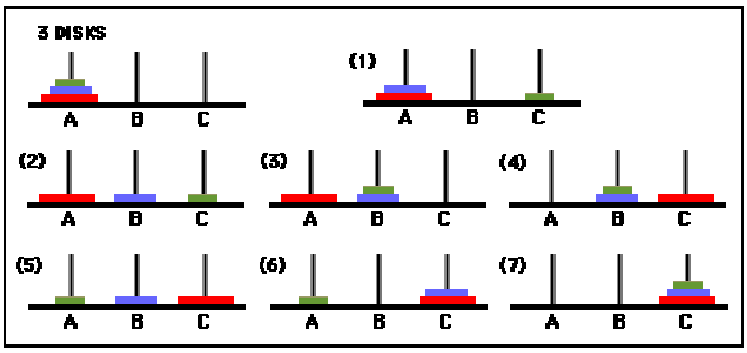
4.Bài toán Tháp Hà Nội: Có 3 cây cọc và một chồng n đĩa có kích thước từ nhỏ đến lớn (không trùng kích thước). Các đĩa đều có lỗ bên trong để chồng vào các cây cọc. Ban đầu n đĩa được chồng vào cây cọc thứ nhất, đĩa lớn nhất ở dưới, và kích thước đĩa giảm dần từ dưới lên:



- Ví dụ của bài toán Tháp Hà Nội với n=3

Bài tập tổng kết

- Nhiệm vụ của chúng ta là phải di chuyển n cái đĩa từ cọc A sang cọc C với ràng buộc là: Có một cái cọc trung gian là B, mỗi lần chỉ được di chuyển 1 cái đĩa từ một cọc đến cọc khác và đĩa to hơn không được xếp lên trên đĩa nhỏ hơn.
- Ví dụ dưới đây cho thấy cách di chuyển 3 cái đĩa từ A sang C:



Bài tập tổng kết

- **Yêu cầu**: Hãy viết chương trình chỉ ra cách giải bài toán Tháp Hà Nội với n nhập từ bàn phím.

CT101 – Lập trình căn bản

Khoa CNTT&TT – ĐHCT