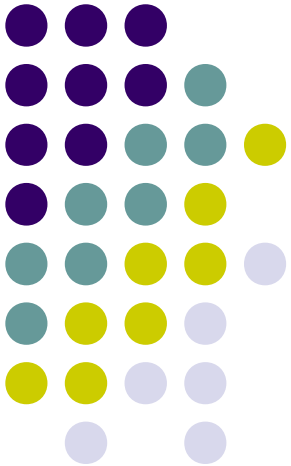
Đệ qui
Thuật toán đệ qui

recursion



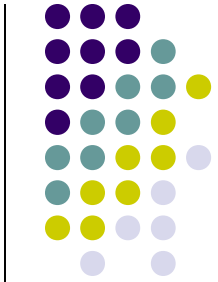
Tư tưởng đệ qui

Đệ qui

- Khái niệm
  - Từ "Đệ qui" xuất phát từ thuật ngữ "Recursion" có nghĩa là quay lại, lặp lại.
  - Một đối tượng được gọi là đệ qui nếu nó bao gồm chính nó như một bộ phận
  - hoặc nó được định nghĩa dưới dạng của chính nó.

    - Từ “đối tượng" cần hiểu theo nghĩa rộng, không nhất thiết là đối tượng hàm hoặc phương thức.

Thuật toán đệ qui



- Một thuật toán được gọi là đệ qui nếu nó giải bài toán bằng cách rút gọn liên tiếp bài toán ban đầu đến bài toán cũng tương tự như vậy nhưng có dữ liệu đầu vào nhỏ hơn.

Ví dụ :

- Định nghĩa giai thừa của 1 số nguyên n là

  n!=1*2*3*…(n-2)*(n-1)*n khi n>0

  n!=1 khi n=0
- n! có thể được định nghĩa bằng cách qui nạp như sau:
- 0! = 1,
- n! = n*(n-1)!, với mọi n > 0.

→Bài toán tính N! bây giờ thành tính N*(N-1)!

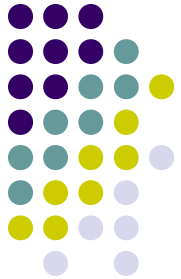
Nhận xét

- Hàm (phương thức) được gọi là đệ qui nếu trong quá trình thực hiện nó tự gọi đến chính mình.
- đệ qui trực tiếp : Lời gọi có thể nằm bên trong hàm (phương thức), khi đó hàm (phương thức).
- đệ qui gián tiếp : nếu hàm (phương thức) gọi tới hàm (phương thức) khác, mà hàm (phương thức) này lần lượt gọi tới hàm (phương thức) ban đầu.

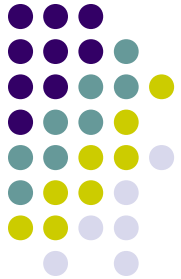
Loại đệ qui

- Trực tiếp
  - A→ A
- Gián tiếp
- A→ B
- B→A

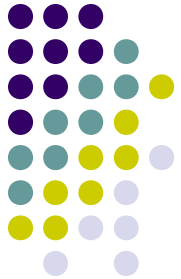
- Trực tiếp
  - A→ A→A→A→ A→A…
- Gián tiếp
- A→ B→A→B→A→B….



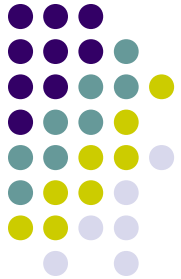
- Như đã biết, một thuật toán được đòi hỏi phải thỏa mãn các tính chất:
- Tính xác định.
- Tính hữu hạn hay tính dừng.
- Tính đúng.



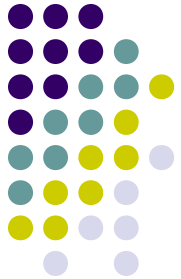
- Do tính chất của thuật toán là sau một số hữu hạn các phép toán thì thuật toán kết thúc,vì vậy thuật toán đệ qui phải gồm có 2 phần:
  - phần đệ qui
  - phần không đệ qui (*phần cơ sở* -phần làm dừng tính đệ qui)



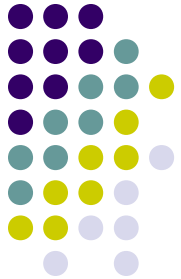
- **Phần cơ sở** gồm các trường hợp không cần thực hiện lại thuật toán, tức là các trường hợp dừng mà ta có thể trực tiếp giải quyết được bài toán (hay không có yêu cầu gọi **đệ qui**).
- Ví dụ : tính N! thì trường hợp n=1 thì không cần tính nữa→ cơ sở



- **Phần đệ quy** là phần trong thuật toán có yêu cầu gọi đệ quy, tức là yêu cầu thực hiện lại thuật toán ở cấp độ thấp hơn.
- Trong phần đệ quy, yêu cầu gọi **đệ qui** thường được đặt trong một điều kiện kiểm tra việc gọi đệ quy.
- Ví dụ : tính N! thì phần gọi đệ qui là những giá trị (N-1)! ,N-2! …. Và thường được đặt trog điều kiện N>1

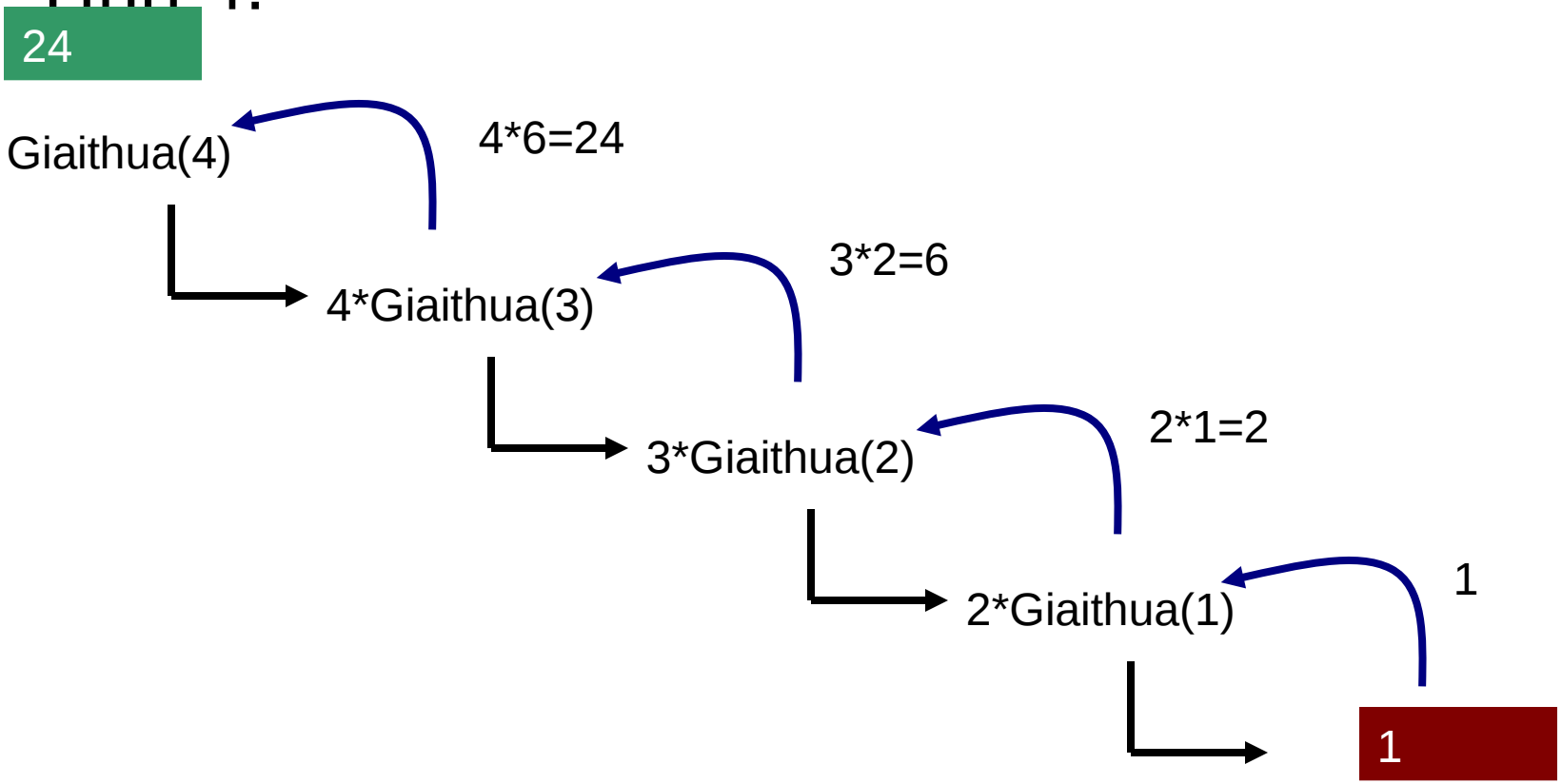


- **<u>Thuật toán đệ quy tính giai thừa của một số tự nhiên</u>.**
- Input : số tự nhiên n.
- Output : giaithua (n) bằng n!.
- Thuật toán :
- 1. if (n=1) || (n=0) return 1 //n!= 1
- 2. return giaithua(n-1) * n; // Tính (n-1)! rồi nhân với n

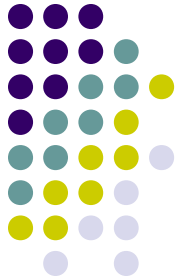


- Int Giaithua(int n){
- If(n==1)||(n==0) { return 1;}
- Return(Giaithua(n-1)*n);
- }

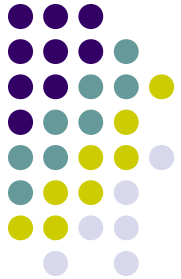
- Tính 4!


24
Giaithua(4)
4*6=24
4*Giaithua(3)
3*2=6
3*Giaithua(2)
2*1=2
2*Giaithua(1)
1
1

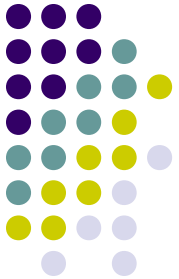
Trình bày thuật toán đệ quy



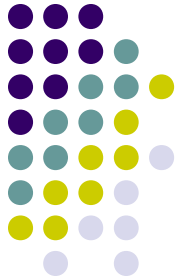
- Đặt tên cho thuật toán có đi kèm các tham biến chính liên quan đến bài toán (khai báo phương thức hay hàm và tham số trong chương trình).



- Ví dụ 2: Định nghĩa dãy số Fibonacci { f1, f2, . . ., fn, ... } :
- f0 = 1,
- f1 = 1,
- fn = fn-1 + fn-2 , với mọi n > 1.

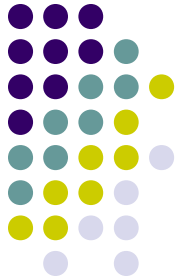


- Tính ước số chung lớn nhất của 2 số tự nhiên a và b, ký hiệu là USCLN(a,b).
- Từ các tính chất dưới đây (cho các số nguyên tùy ý) của phép tính ước số chung lớn nhất:
- USCLN(a, 0) = USCLN(0, a) = a,
- USCLN(a, b) = USCLN(a-b, b), và
- USCLN(a, b) = USCLN(a, b-a)

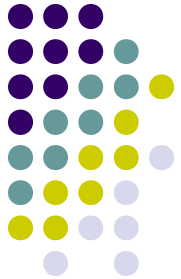


- **<u>Thuật toán đệ quy tính số hạng thứ n của dãy số Fibonacci</u>.**
- Input : số nguyên dương n.
- Output : F (n) bằng số hạng thứ n của dãy Fibonacci.
- Thuật toán :
- 1. if n=0 or n=1 then
- F := 1;
- 2. if n > 1 then
- F := F(n-1) + F(n-2)
- { tức là tính F(n-1) và F(n-2) rồi tính tổng số của các giá trị nầy để gán cho F}
- 3. Output F.

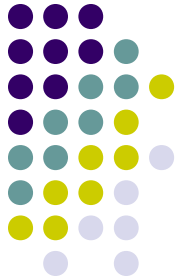
code cho fibonaci bằng đệ qui



- Static Int Fibo(int n){
- if ((n==0) || (n==1)) return 1;
- if (n > 1)
- return (Fibo(n-1) + Fibo(n-2));
- }

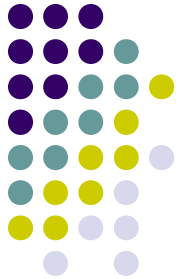


- **Thuật toán tính USCLN của a và b: USCLN(a,b)**
- If (a = 0) or (b = 0) then
- 	USCLN := a+b;
- Else If (a > b) then
- 	USCLN := **USCLN(a-b, b)**;
- Else
- 	USCLN := **USCLN(a, b -a)**;



- - Ưu điểm của thuật toán đệ qui là ngắn gọn. Nó có khả năng định nghĩa một tập hợp rất lớn các đối tượng bằng một số các câu lệnh hữu hạn.
- Khuyết điểm: không xác định được bước lặp, bước thực hiện của thuật giải
- Nếu có thể thay thế thuật toán đệ qui bằng một thuật toán khác không tự gọi chúng (không đệ qui) thuật toán này được gọi là thuật toán *khử đệ qui*.
- chỉ *khi nào không thể dùng thuật toán lặp ta mới nên sử dụng thuật toán đệ qui*

Khử đệ qui dùng lặp

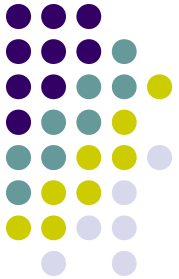


- **Thuật toán lặp tính số hạng thứ n của dãy số Fibonacci.**
- *Input* : số nguyên dương n.
- *Output* : F (n) bằng số hạng thứ n của dãy Fibonacci.
- *Thuật toán* :
- 1. a := 1
- 2. F := 1
- 3. for i:=3 to n do
- begin
- temp := a + F;
- a := F;
- F := temp;
- end;
- 4. Output F.

Khử đệ qui dùng lặp

- **<u>Thuật toán lặp tính giai thừa của một số tự nhiên</u>.**
- *Input* : số tự nhiên n.
- *Output* : F (n) bằng n!.
- *Thuật toán* :
- 1. F := 1
- 2. for i := 2 to n do
- F := F * i
- 3. Output F.

Khử đệ qui- stack



- Lưu dữ liệu vào stack lúc gọi và lấy ra thực hiện, tính toán lúc hàm thủ tục trả về
- Mô tả quá trình hoạt động của đệ qui