

# Bài 4

## ĐỆ QUY

---

Trịnh Thành Trung  
[trungtt@soict.hust.edu.vn](mailto:trungtt@soict.hust.edu.vn)



# *1*

ĐỆ QUY

---

# Khái niệm đệ quy

- Mô tả mang tính đệ quy về một đối tượng là mô tả theo cách phân tích đối tượng thành nhiều thành phần mà trong số các thành phần có thành phần mang tính chất của chính đối tượng được mô tả.
- Tức là mô tả đối tượng qua chính nó.
  - Mô tả đệ quy tập số tự nhiên  $N$  :
    - Số 1 là số tự nhiên (1-N).
    - Số tự nhiên bằng số tự nhiên cộng 1.
  - Mô tả đệ quy cấu trúc danh sách (list) kiểu  $T$  :
    - Cấu trúc rỗng là một danh sách kiểu  $T$ .
    - Ghép nối một thành phần kiểu  $T$  (nút kiểu  $T$ ) với một danh sách kiểu  $T$  ta có một danh sách kiểu  $T$ .
  - Mô tả đệ quy cây gia phả: Gia phả của một người bao gồm người đó và gia phả của cha và gia phả của mẹ

# Ví dụ

- Định nghĩa không đệ quy n!:

$$- n! = n * (n-1) * \dots * 1$$

- Định nghĩa đệ quy:

$$\begin{array}{ll} - n! = & 1 \quad \text{nếu } n=0 \\ & n * (n-1)! \quad \text{nếu } n>0 \end{array}$$

- Mã C++:

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

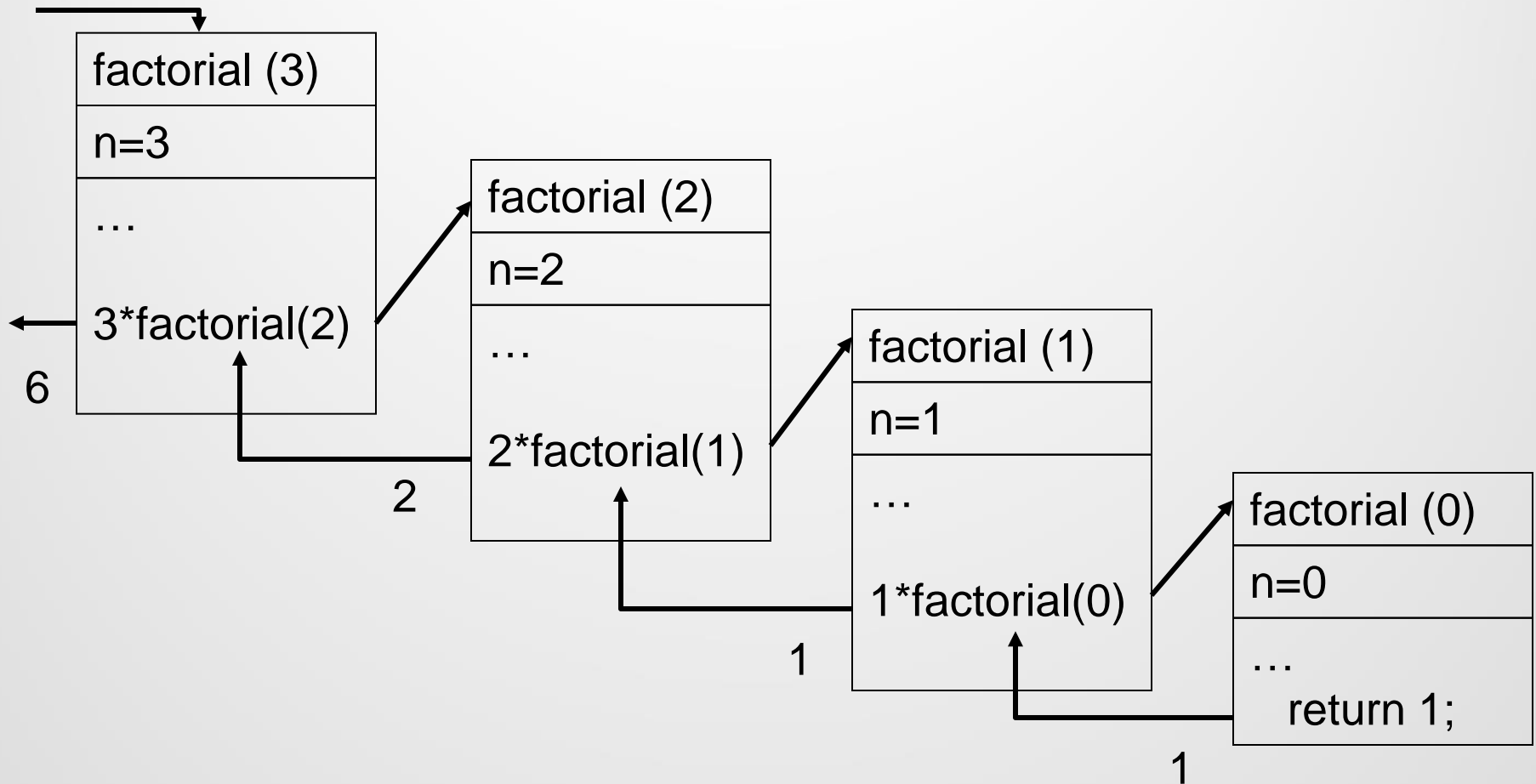
# Mô tả đệ quy

- Mô tả đệ quy gồm hai phần
  - Phần neo: trường hợp suy biến của đối tượng
  - Ví dụ: 1 là số tự nhiên, cấu trúc rỗng là danh sách kiểu T,  $0! = 1$ ,  $SM(a[x:x])$  là thao tác rỗng.
  - Phần qui nạp: mô tả đối tượng (giải thuật) thông qua chính đối tượng (giải thuật) đó một cách trực tiếp hoặc gián tiếp.

Ví dụ:

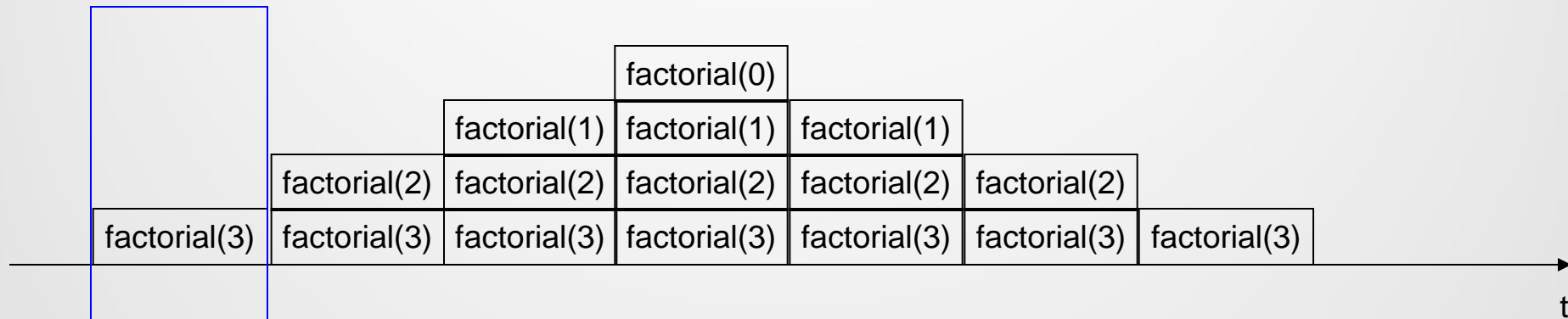
- $n! = n * (n - 1)!$
- $SM(a[m:n]) \equiv Merge(SM(a[m:(m+n) \div 2]), SM(a[(m+n) \div 2 + 1 : n]))$

# Thi hành hàm tính giai thừa

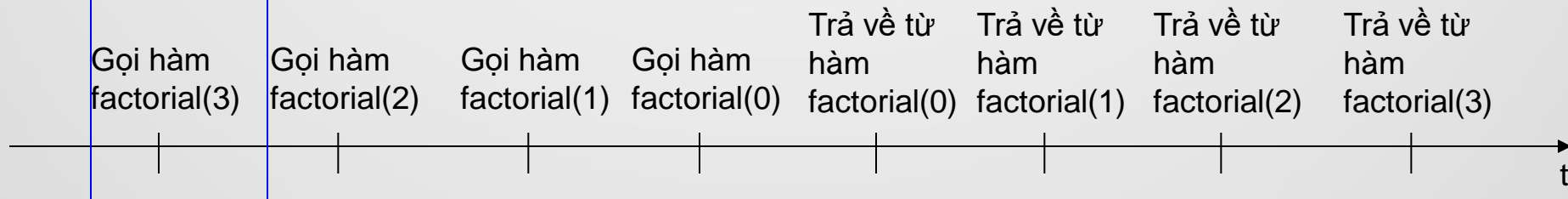


# Trạng thái hệ thống khi thi hành hàm tính giai thừa

**Stack hệ thống**



**Thời gian hệ thống**



# Phân loại

- Phân loại đệ quy
  - Đệ quy trực tiếp
    - Đệ quy tuyến tính
    - Đệ qui nhị phân
    - Đệ quy phi tuyến
  - Đệ quy gián tiếp
    - Đệ quy hỗ tương



# Đệ quy tuyến tính

- là dạng đệ quy trực tiếp đơn giản nhất có dạng

```
P( ) {  
    If (B) thực hiện S;  
    else { thực hiện S* ; gọi P }  
}
```

Với S, S\* là các thao tác không đệ quy.

- VD: Hàm FAC(n) tính số hạng n của dãy n!

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

```
KieuDuLieu TenHam(Thamso)  
{  
    if(Dieu Kien Dung)  
    {  
        ...;  
        return Gia tri tra ve;  
    }  
    ...;  
    TenHam(Thamso)  
    ...;  
}
```

Tính  $S(n) = 1/(1*2) + 1/(2*3) + \dots + 1/(n*(n+1))$

- $S(n) = 1/(1*2) + 1/(2*3) + \dots + 1/(n*(n+1))$
- $S(n) = 1/2$  khi  $n==1$   
 $= S(n-1) + 1/(n*(n+1))$

```
float S(int n) {  
    if ( n==1) return 0.5;  
    else return S(n-1)+1.0/(n*(n+1));  
}
```

# Đệ quy nhị phân

- là đệ quy trực tiếp có dạng như sau

```
P ( ) {  
    If (B) thực hiện S;  
    else {  
        thực hiện S*;  
        gọi P ; gọi P;  
    }  
}
```

Với S , S\* là các thao tác không đệ quy.

- Ví dụ: Hàm FIBO(n) tính số hạng n của dãy FIBONACCI

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

```
KieuDuLieu TenHam(Thamso)  
{  
    if(Dieu Kien Dung)  
    {  
        ...;  
        return Gia tri tra ve;  
    }  
    ...;  
    TenHam(Thamso);  
    ...;  
    TenHam(Thamso);  
    ...;  
}
```



DN:

$$\begin{aligned} H(n) &= n \text{ khi } n < 3 \\ &= 2 * H(n-1) * H(n-2) \text{ khi } n > 2 \end{aligned}$$

```
long H(int n) {  
    if (n<3) return n;  
    else return (2*H(n-1)*H(n-2));  
}
```

```
long Tong(int n) {  
    long tg=0;  
    for( int i=1; i<=n;i++)  
        tg+=H(i);  
    return tg;  
}
```

---



Khử đệ quy!!

```
long T(int n) {  
    long h1=1,h2=2,h,tg=3;  
    if( n==1) return 1;  
    else if (n==2) return 3;  
    else {  
        for(int i=3;i<=n;i++) {  
            h=2*h2*h1;  
            tg+=h;  
            h1=h2; h2=h;}  
        return tg;  
    }  
}
```

---

# Đệ quy phi tuyến

- là đệ quy trực tiếp mà lời gọi đệ quy được thực hiện bên trong vòng lặp.

```
P ( ) {  
    for (<giá trị đầu> to  
    <giá trị cuối>) {  
        thực hiện S ;  
        if (điều kiện dừng) then  
            thực hiện S*;  
        else gọi P;  
    }  
}
```

- Với S , S\* là các thao tác không đệ quy .

```
KieuDuLieu TenHam(Thamso)  
{  
    if(Dieu Kien Dung)  
    {  
        ...;  
        return Gia tri tra ve;  
    }  
    ...;  
    vonglap(dieu kien lap)  
    {  
        ...TenHam(Thamso)...;  
    }  
    return Gia tri tra ve;  
}
```

# Đệ quy phi tuyến

- Ví dụ: Cho dãy  $\{A_n\}$  xác định theo công thức truy hồi :

$$A_0 = 1 ;$$

$$A_n = n^2 A_0 + (n-1)^2 A_1 + \dots + 2^2 A_{n-2} + 1^2 A_{n-1}$$

```
int A( int n ) {  
    if ( n == 0 ) return 1 ;  
    else {  
        int tg = 0 ;  
        for (int i = 0 ; i < n ; i++ ) tg = tg +  
sqr(n-i) *A(i);  
        return ( tg ) ;  
    }  
}
```

Tính  $X_n$  với?

với:  $X_0 = 1$ ;  $X_n = canba(n) * X_0 + canba(n-1) * X_1 + \dots + canba(1) * X_{n-1}$ .

# Đệ quy gián tiếp

- Đệ quy tương hỗ: Trong đệ quy tương hỗ có 2 hàm, và trong thân của hàm này có lời gọi của hàm kia, điều kiện dừng và giá trị trả về của cả hai hàm có thể giống nhau hoặc khác nhau

```
KieuDuLieu TenHamX(Thamso)
{
    if(Dieu Kien Dung)
    {
        ...;
        return Gia tri tra ve;
    }
    ...;
    return TenHamX(Thamso) <Lien
ket hai ham> TenHamY(Thamso);
}
```

```
KieuDuLieu TenHamY(Thamso)
{
    if(Dieu Kien Dung)
    {
        ...;
        return Gia tri tra ve;
    }
    ...;
    return TenHamY(Thamso)<Lien
ket hai ham>TenHamX(Thamso);
}
```



# Ví dụ

```
long Y(int n); //prototype của  
ham y
```

```
long X(int n) {  
    if(n==0)  
        return 1;  
    else  
        return X(n-1) + Y(n-  
1);  
}
```

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

**$X(n) = 1, 2, 3, 5, 11, 41, \dots$**

**$Y(n) = 1, 1, 2, 6, 30, 330, \dots$**

```
void main(){  
    int n;  
    printf("\n Nhap n = ");  
    scanf("%d",&n);  
    printf( "\n  X = %d  ",X(n));  
    printf( "\n  Y = %d  ",Y(n));  
    getch();  
}
```

# Tìm giải thuật đệ quy

Ba bước:

## 1. Thông số hóa bài toán .

- Tổng quát hóa bài toán cụ thể cần giải thành bài toán tổng quát (một họ các bài toán chứa bài toán cần giải )
- Tìm ra các thông số cho bài toán tổng quát
  - các thông số điều khiển: các thông số mà độ lớn của chúng đặc trưng cho độ phức tạp của bài toán , và giảm đi qua mỗi lần gọi đệ quy.
  - Ví dụ
  - $n$  trong hàm  $FAC(n)$  ;
  - $a, b$  trong hàm  $USCLN(a,b)$  .

# Tìm giải thuật đệ quy

## 2. Tìm các trường hợp neo cùng giải thuật giải tương ứng

- trường hợp suy biến của bài toán tổng quát
- các trường hợp tương ứng với các giá trị biên của các biến điều khiển
- VD:  $FAC(1) = 1$
- $USCLN(a,0) = a$

## 3. Tìm giải thuật giải trong trường hợp tổng quát bằng phân rã bài toán theo kiểu đệ quy

# Tìm giải thuật đệ quy

- Phân rã bài toán tổng quát theo phương thức đệ quy
  - Tìm phương án (giải thuật ) giải bài toán trong trường hợp tổng quát phân chia nó thành các thành phần
    - giải thuật không đệ quy
    - bài toán trên nhưng có kích thước nhỏ hơn.
  - Ví dụ
$$FAC(n) = n * FAC(n - 1) .$$
$$Tmax(a[1:n]) = \max(Tmax(a[1:(n-1)]) , a[n] )$$

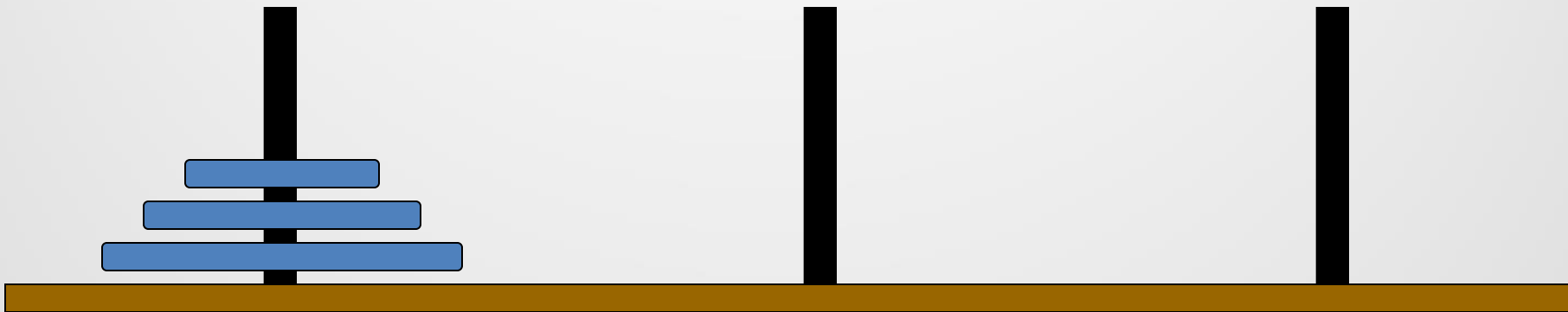


# MỘT SỐ BÀI TOÁN

- Bài toán tháp Hà Nội
- Bài toán chia phần thưởng
- Bài toán hoán vị

# Bài toán Tháp Hà Nội

- Luật:
  - Di chuyển mỗi lần một đĩa
  - Không được đặt đĩa lớn lên trên đĩa nhỏ



Với chồng gồm  $n$  đĩa cần  $2^n - 1$  lần chuyển

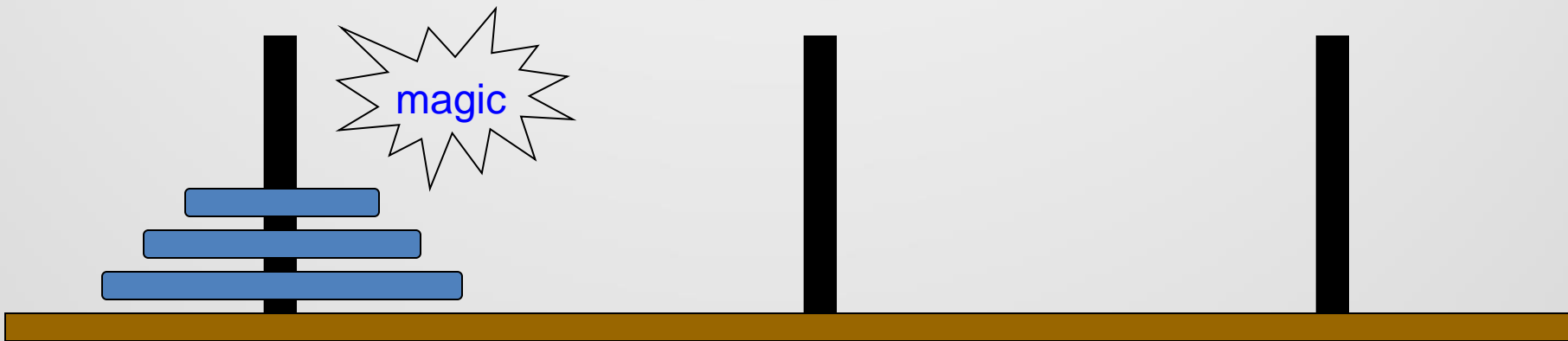
– Giả sử thời gian để chuyển 1 đĩa là  $t$  giây thì thời gian để chuyển xong chồng 64 đĩa sẽ là:

–  $T = (2^{64} - 1) * t = 1.84 * 10^{19} t$

– Với  $t = 1/100$  s thì  $T = 5.8 * 10^9$  năm = 5.8 tỷ năm .

# Bài toán Tháp Hà Nội

- Hàm đệ quy: Chuyển  $n$  đĩa từ A sang C qua trung gian B
  - Chuyển  $n-1$  đĩa trên đỉnh của cột A sang cột B
  - Chuyển 1 đĩa (cuối cùng) của cột A sang cột C
  - Chuyển  $n-1$  đĩa từ cột B sang C qua tg A



# Bài toán Tháp Hà Nội

- Giải bài toán bằng đệ quy
  - Thông số hóa bài toán
    - Xét bài toán ở mức tổng quát nhất : chuyển  $n$  ( $n \geq 0$ ) đĩa từ cột A sang cột C lấy cột B làm trung gian .
    - $THN(n, A, B, C) \rightarrow$  với 64 đĩa gọi  $THN(64, A, B, C)$
    - $n$  sẽ là thông số quyết định bài toán –  $n$  là tham số điều khiển
  - Trường hợp suy biến và cách giải
    - Với  $n = 1$  :  $THN(1, A, B, C)$ 
      - Giải thuật giải bt  $THN(1, A, B, C)$  là thực hiện chỉ 1 thao tác cơ bản : Chuyển 1 đĩa từ A sang C (ký hiệu là  $Move(A, C)$  ) .
    - $THN(1, A, B, C) \equiv \{ Move(A, C) \}$  .
    - $THN(0, A, B, C) \equiv \{ \varphi \}$



# Bài toán Tháp Hà Nội

- Phân rã bài toán

- Ta có thể phân rã bài toán TH N ( $k, A, B, C$ ) : chuyển  $k$  đĩa từ cột A sang cột C lấy cột B làm trung gian thành dãy tuần tự 3 công việc sau :

- Chuyển ( $k - 1$ ) đĩa từ cột A sang cột B lấy cột C làm trung gian :

- THN ( $k - 1, A, C, B$ ) (bài toán THN với  $n = k - 1, A = A, B = C, C = B$ )

- Chuyển 1 đĩa từ cột A sang cột C : Move ( $A, C$ ) (thao tác cơ bản).

- Chuyển ( $k - 1$ ) đĩa từ cột B sang cột C lấy cột A làm trung gian :

- THN ( $k - 1, B, A, C$ ) (bài toán THN với  $n = k - 1, A = B, B = A, C = C$ ).

- Vậy giải thuật trong trường hợp tổng quát ( $n > 1$ ) là:

THN( $n, A, B, C$ )  $\equiv$  {  
    THN ( $n - 1, A, C, B$ ) ;  
    Move ( $A, C$ ) ;  
    THN ( $n - 1, B, A, C$ ) ;  
}

# Bài toán Tháp Hà Nội – Mã C++

```
void move(int n, int A, int B, int C) {  
    if (n > 0) {  
        move(n - 1, A, B, C);  
        printf( "\n Move disk % d from %c  to % c ", n, A,C );  
        move(n - 1, B, C, A);  
    }  
}
```

# Ví dụ

- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau :

$$T=1+2+3+6+11+20+37+68+125+\dots$$

- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau :

$$T=1+2+3+7+13+23+43+79+145+\dots$$

– Biết rằng số phần tử của dãy số luôn  $\geq 5$

## Ví dụ (tiếp)

- Viết hàm đệ quy tính các phần tử của dãy số sau với  $n$  phần tử ( $n \geq 10$ ), rồi tính tổng các phần tử của dãy số

1,2,3,4,4,5,8,11,12,14,21,30,35,40,56,.....

1,2,3,4,6,9,14,21,32,48,73,110,167,252,...



# 2

KHỬ ĐỆ QUY

---

# Tổng quan về khử đệ quy

- Đệ quy
  - Ưu điểm: gọn gàng, dễ hiểu, dễ viết code
  - Nhược điểm: tốn không gian nhớ và thời gian xử lý.
- Mọi giải thuật đệ quy đều có thể thay thế bằng một giải thuật không đệ quy.
- Sơ đồ để xây dựng chương trình cho một bài toán khó khi ta không tìm được giải thuật không đệ quy thường là:
  - Dùng quan niệm đệ quy để tìm giải thuật cho bài toán .
  - Mã hóa giải thuật đệ quy.
  - Khử đệ quy để có được một chương trình không đệ quy .
- Tuy nhiên: khử đệ quy không phải bao giờ cũng dễ => trong nhiều trường hợp ta cũng phải chấp nhận sử dụng chương trình đệ quy

# Khử đệ quy bằng vòng lặp

- Giải thuật hồi qui thường gặp

$$f(n) = C \text{ khi } n = n_0 \text{ (C là một hằng)}$$

$$= g(n, f(n-1)) \text{ khi } n > n_0$$

- Ví dụ:

- Hàm giai thừa  $FAC(n) = n! = 1$  khi  $n=0$

$$= n * FAC(n-1) \text{ khi } n > 0$$

- Tổng n số đầu tiên của dãy đan dấu sau :

$$S_n = 1 - 3 + 5 - 7 \dots + (-1)^{(n+1)} * (2n-1)$$

$$S(k) = 1 \text{ khi } k = 1$$

$$= S(k-1) + (-1)^{(k+1)} * (2*k-1) \text{ với } k > 1$$

# KHỬ ĐỆ QUY BẰNG VÒNG LẶP

- Giải thuật đệ quy tính giá trị  $f(n)$   
 $f(n) \equiv \text{if}(n == n_0) \text{ return } C;$   
 $\text{else return } (g(n, f(n-1)));$
  - Giải thuật lặp tính giá trị  $f(n)$   
 $K = n_0; F := C;$   
 $\{ F = f(n_0) \}$   
 $\text{While}( k < n )\{$   
     $k += 1;$   
     $F = g(k, F);$   
 $\}$   
 $\text{return } F;$
-



- Khử đệ quy với hàm tính giai thừa

```
int FAC ( int n ) {  
    int k = 0;  
    int F = 1;  
    while ( k < n ) F = ++k * F;  
    return (F);  
}
```

- Khử đệ quy với hàm tính S(n)

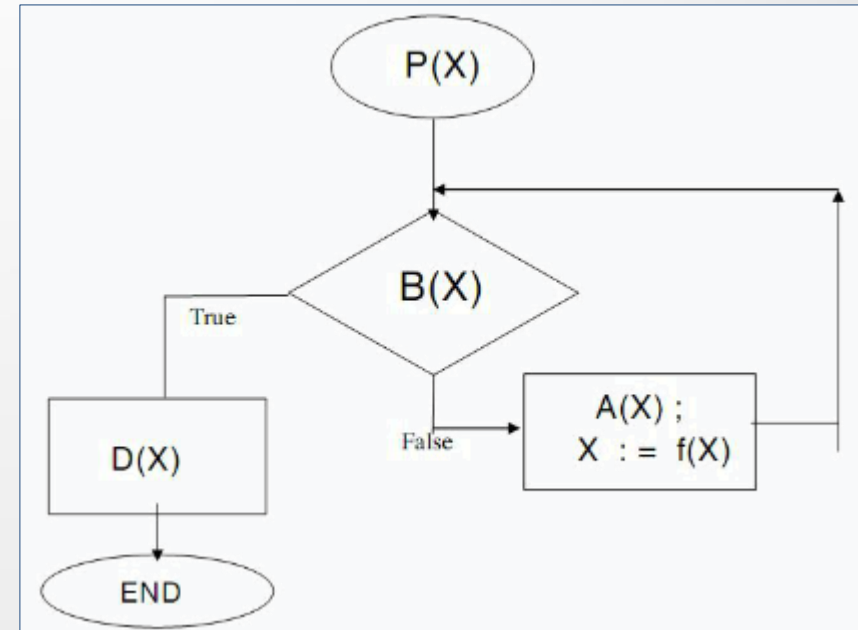
```
int S ( int n ) {  
    int k = 1 , tg = 1 ;  
    while ( k < n ) {  
        k ++ ;  
        if (k%2 == 1)  tg + = 2 * k -1;  
        else tg -= 2 * k + 1 ;  
    }  
    return ( tg ) ;  
}
```

---

# Các thủ tục đệ quy dạng đệ quy đuôi

- Xét thủ tục P dạng :

```
P(X)  $\equiv$     if B(X) then D(X)
            else {
                A(X) ;
                P(f(X)) ;
            }
```



- Trong đó:

- $X$  là tập biến (một hoặc một bộ nhiều biến)
- $P(X)$  là thủ tục đệ quy phụ thuộc  $X$
- $A(X)$ ;  $D(X)$  là các thao tác không đệ quy
- $f(X)$  là hàm biến đổi  $X$

- Xét quá trình thi hành  $P(X)$  :
  - gọi  $P_0$  là lần gọi  $P$  thứ 0 (đầu tiên)  $P(X)$
  - $P_1$  là lần gọi  $P$  thứ 1 (lần 2)  $P(f(X))$
  - $P_i$  là lần gọi  $P$  thứ  $i$  (lần  $i + 1$ )  $P(f(f(...f(X)...) )$
  - (  $P(f_i(X))$  hợp  $i$  lần hàm  $f$  )
- Gọi  $P_i$  nếu  $B(f_i(X))$ 
  - (false) { A và gọi  $P_{i+1}$  }
  - (true) { D }
- Giả sử  $P$  được gọi đúng  $n + 1$  lần . Khi đó ở trong lần gọi cuối cùng (thứ  $n$  )  $P_n$  thì  $B(f_n(X)) = \text{true}$  , lệnh D được thi hành và chấm dứt thao tác gọi thủ tục  $P$

- Sơ đồ thực hiện giải thuật trên bằng vòng lặp

```
While ( !B(X) ) {
```

```
    A(X) ;
```

```
    X = f(X) ;
```

```
}
```

```
D(X) ;
```

# Ví dụ: Tìm ước chung lớn nhất

- Giải thuật đệ quy

```
int USCLN(int m , int n) {  
    if (n == 0) return m;  
    else USCLN(n, m % n);  
}
```

- $X$  là  $(m, n)$
- $P(X)$  là  $USCLN(m, n)$
- $B(X)$  là  $n == 0$
- $D(X)$  là lệnh `return m`
- $A(X)$  là lệnh rỗng
- $f(X)$  là  $f(m, n) = (n, m \bmod n)$

- Khử đệ quy

```
int USCLN(int m , int n )  
{  
    while(n != 0 ) {  
        int sd = m % n ;  
        m = n ;  
        n = sd ;  
    }  
    return (m) ;  
}
```

## Ví dụ:

Đổi 1 số nguyên không âm  $Y$  ở cơ số 10 sang dạng cơ số  $k$  ( $2 \leq k \leq 9$ )

–Dùng mảng  $A[n]$

–Convert( $x, m$ ) để tạo dãy giá trị:  $A[0], A[1], \dots, A[m]$

–Giải thuật

```
Convert( $n, m$ )  $\equiv$  if ( $n \neq 0$ ) {  
     $A[m] = n \% k$  ;  
    Convert( $n/k, m - 1$ ) ;  
}
```

–Trong ví dụ này ta có

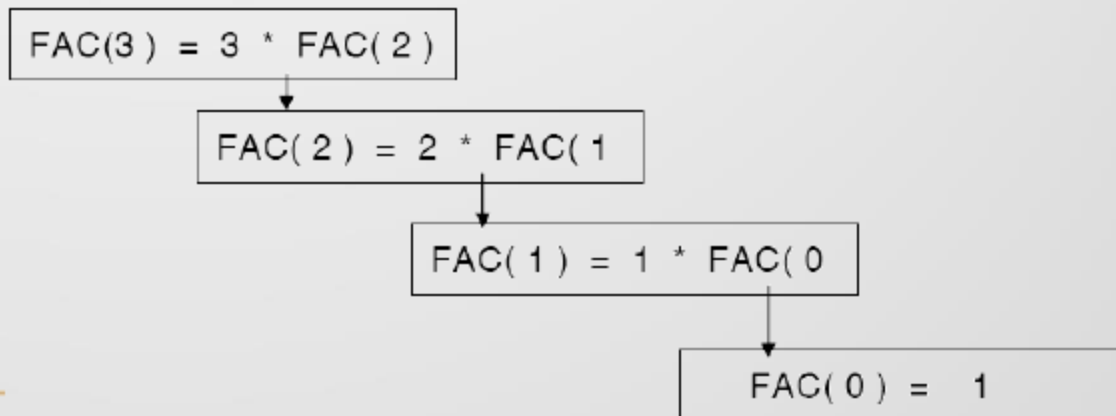
- $X$  là ( $n, m$ ) ;
- $B(X)$  là biểu thức boolean  $\text{not}(n \neq 0)$
- $A(X)$  là lệnh gán  $A[m] := n \% k$  ;
- $f(X)$  là hàm  $f(n, m) = (n/k, m - 1)$  ;
- $D(X)$  là lệnh rỗng

–Đoạn lệnh lặp tương ứng với thủ tục Convert( $x, m$ ) là:

```
While ( $n \neq 0$ ) {  
     $A[m] = n \% k$  ;           //  $A(X)$   
     $n = n / k$  ;              //  $X := f(X)$   
     $m = m - 1$  ;  
}
```

# Cơ chế thực hiện đệ quy

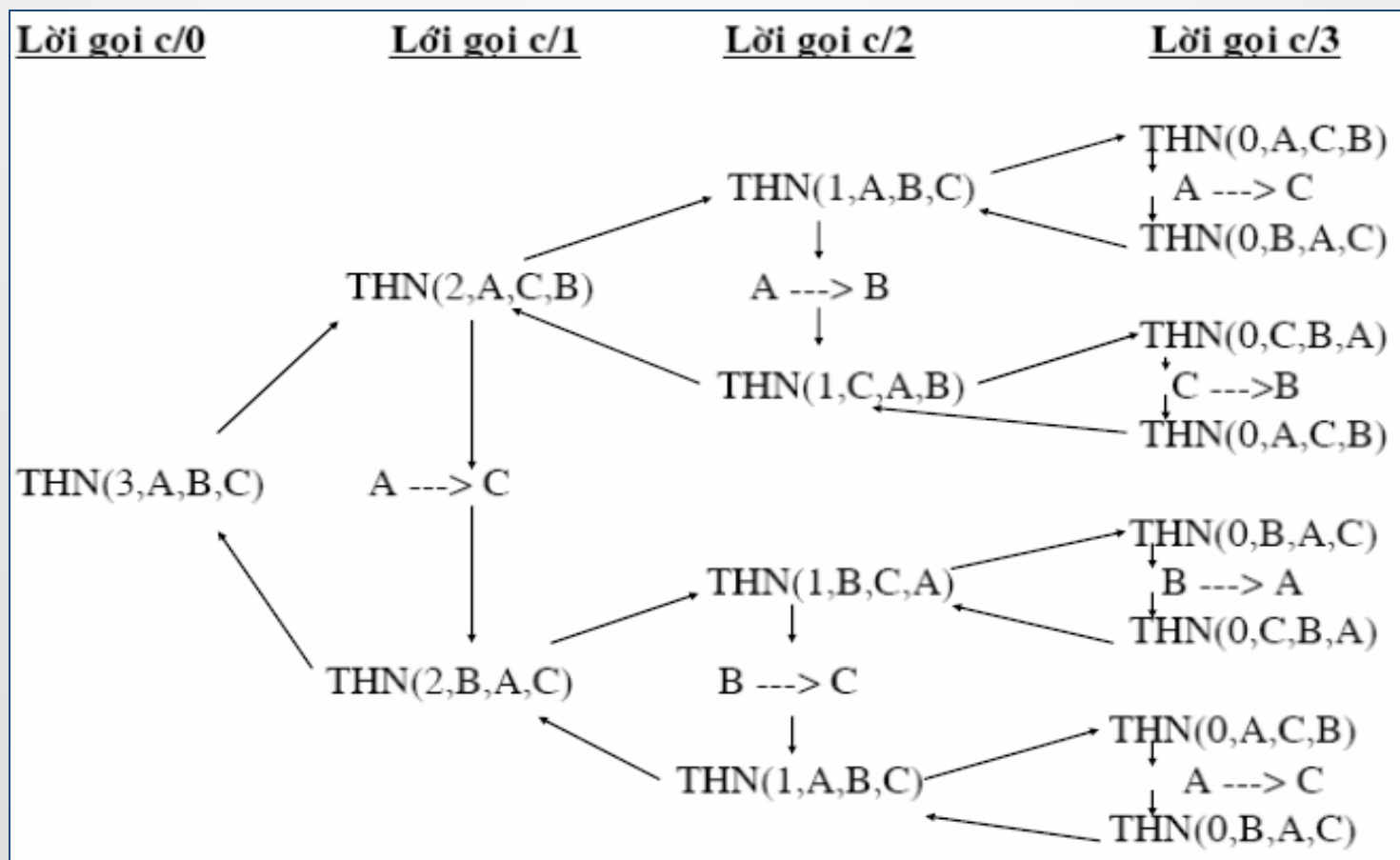
- Trạng thái của tiến trình xử lý một giải thuật: nội dung các biến và lệnh cần thực hiện kế tiếp.
- Với tiến trình xử lý một giải thuật đệ quy ở từng thời điểm thực hiện, cần lưu trữ cả các trạng thái xử lý đang còn dang dở
- Xét giải thuật giai thừa  
 $FAC(n) \equiv \text{if}(n = 0) \text{ then return } 1;$   
 $\text{else return } (n * FAC(n - 1));$
- Sơ đồ thực hiện



Xét thủ tục đệ quy tháp Hà Nội THN (n , A , B , C)

THN (n : integer ; A , B , C : char)  $\equiv$  {  
 if (n > 0 ) then { THN(n-1,A ,C ,B) ;  
 Move(A, C) ; THN(n-1,B,A,C) ;} }

–Sơ đồ thực hiện THN(3,A,B,C)





# Nhận xét

- Lời gọi đệ quy sinh ra lời gọi đệ quy mới cho đến khi gặp trường hợp suy biến (neo)
- Ở mỗi lần gọi, phải lưu trữ thông tin trạng thái con dang dở của tiến trình xử lý ở thời điểm gọi. Số trạng thái này bằng số lần gọi chưa được hoàn tất.
- Khi thực hiện xong (hoàn tất) một lần gọi, cần khôi phục lại toàn bộ thông tin trạng thái trước khi gọi .
- Lệnh gọi cuối cùng (ứng với trường hợp neo) sẽ được hoàn tất đầu tiên
- Cấu trúc dữ liệu cho phép lưu trữ dãy thông tin thỏa 3 yêu cầu trên là cấu trúc lưu trữ thỏa mãn LIFO (Last In First Out => Cấu trúc stack)

# Khử đệ quy bằng Stack

- Để thực hiện một chương trình con đệ quy thì hệ thống phải tổ chức vùng lưu trữ thỏa qui tắc LIFO (vùng Stack).
- Vậy ta chủ động tạo ra cấu trúc dữ liệu stack đặc dụng cho từng chương trình con đệ quy cụ thể.

# Đệ quy chỉ có một lệnh gọi trực tiếp

- Đệ quy có dạng sau:

```
P(X)  $\equiv$  if C(X) then D(X)
      else begin
          A(X) ;
          P(f(X)) ;
          B(X) ;
      end ;
```

- X là một biến đơn hoặc biến véc tơ.
- C(X) là một biểu thức boolean của X .
- A(X) , B(X) , D(X): không đệ quy
- f(X) là hàm của X

- Giải thuật thực hiện  $P(X)$  với việc sử dụng Stack có dạng :

```
P(X)  $\equiv$  {  
    Create_Stack (S) ; ( tạo stack S )  
    While(not(C(X))) do begin  
        A(X) ;  
        Push(S,X) ; ( cất giá trị X vào stack S )  
        X := f(X) ;  
    end ;  
    D(X) ;  
    While(not(EmptyS(S))) do begin  
        POP(S,X) ; ( lấy dữ liệu từ S )  
        B(X) ;  
    end ;  
}
```

# Ví dụ

Ví dụ: Thủ tục đệ quy chuyển biểu diễn số từ cơ số thập phân sang nhị phân có dạng :

```
Binary(m)  $\equiv$  if ( m > 0 )  
then begin
```

```
    Binary( m / 2 ) ;
```

```
    write( m % 2 ) ;
```

```
end;
```

- Trong trường hợp này :
  - X là m .
  - P(X) là Binary(m) .
  - A(X) ; D(X) là lệnh rỗng .
  - B(X) là lệnh Write(m % 2) ;
  - C(X) là ( m <= 0 ) .
  - f(X) = f(m) = m / 2

- Giải thuật thực hiện Binary(m) không đệ quy là:

```
Binary (m )  $\equiv$ { Create_Stack (S);
```

```
While ( m > 0 ) do begin
```

```
    sdu = m % 2 ;
```

```
    Push(S,sdu) ;
```

```
    m = m / 2 ;
```

```
end;
```

```
While( not(EmptyS(S)) do begin
```

```
    POP(S,sdu) ;
```

```
    Write(sdu) ;
```

```
end;
```

```
}
```

# Thủ tục đệ quy với 2 lần gọi đệ quy

- **Đệ quy có dạng sau**

```
P(X)  $\equiv$  if C(X) then D(X)
else begin
    A(X); P(f(X));
    B(X); P(g(X));
end;
```

# Thủ tục đệ quy với 2 lần gọi đệ quy

- Thuật toán khử đệ quy tương ứng với thủ tục đệ quy

```
P(X) ≡ {  
    Create_Stack (S) ;  
    Push (S, (X,1)) ;  
    Repeat  
        While ( not C(X) ) do begin  
            A(X) ;  
            Push (S, (X,2)) ;  
            X := f(X) ;  
  
        end;  
        D(X) ;  
        POP (S, (X,k)) ;  
        if ( k <> 1 ) then begin  
            B(X) ;  
            X := g(X) ;  
  
        end;  
    until ( k = 1 ) ;  
}
```

# Khử đệ quy thủ tục Tháp Hà Nội .

- **Dạng đệ quy**

```
THN(n,X,Y,Z)  $\equiv$  if(n > 0)
{
    THN (n - 1, X, Z, Y);
    Move (X, Z );
    THN (n - 1, Y, X, Z);
}
```

- **Trong đó**

- Biến X là bộ (n,X,Y,Z)
- C(X) là  $n \leq 0$
- D(X), A(X) là rỗng
- B(X) = B(n,X,Y,Z) là move(X, Z)
- $f(X) = f(n,X,Y,Z) = (n-1,X,Z,Y)$
- $g(X) = g(n,X,Y,Z) = (n-1,Y,X,Z)$

- **Giải thuật không đệ quy tương đương là**

```
THN {
    Create_Stack (S) ;
    Push (S ,(n,X,Y,Z,1)) ;
    Repeat
        While (n > 0) do begin
            Push (S ,(n,X,Y,Z,2)) ;
            n = n - 1;
            Swap (Y,Z) ;
        end ;
        Pop (S,(n,X,Y,Z,k)) ;
        if ( k <> 1 ) then begin
            Move (X ,Z ) ;
            n = n - 1 ;
            Swap (X,Y) ;
        end ;
    until ( k == 1 ) ;
}
```



# Ví dụ khử đệ quy

- Cho dãy số : 1,2,3,7,14,27,55,110,219....
- Viết hàm đệ quy tính số hạng thứ  $n$  của dãy số ( $n > 2$  nhập từ bàn phím), rồi tính tổng các số hạng của dãy
- Như trên, nhưng không dùng đệ quy
- Công thức tổng quát
$$S(n) = n, \text{ khi } n < 4$$
$$= S(n-1) + S(n-2) + 2 * S(n-3), \text{ khi } n > 3$$

# DÙNG ĐỆ QUY

```
int S( int n) {
    if (n<4)
        return n;
    else
        return S(n-1)+S(n-2)+S(n-3) * 2;
}
int main() {
    int n,i,Tong=0;
    printf("\n Vao n :");
    scanf("%d",&n);
    for(i=1;i<=n;i++) Tong+=S(i);
    printf("\n Tong day so = %d",Tong);
}
```

---

# KHÔNG DÙNG ĐỆ QUY

```
int main() {  
    int i,n,T=6; F1,F2,F3,F;  
    printf("\n Vao n : "); scanf("%d",&n);  
    if (n==1) T=1;  
    else if (n==2) T=3;  
    else if (n==3) T=6;  
    else {  
        F1=1; F2=2; F3=3;  
        for(i=4;i<=n;i++) {  
            F=2*F1+F2+F3;  
            T+=F;  
            F1=F2; F2=F3; F3=F; }  
        printf("\n Tong = %d ",T);  
    }  
}
```

---

# KHÔNG DÙNG ĐỆ QUY

```
int main() {
    int i,n,T=6, F[4]={1,2,3,7};
    printf("\n Vao n : "); scanf("%d",&n);
    if (n==0) T=1;
    else if (n==1) T=3;
    else if (n==2) T=6;
    else {
        for(i=3;i<=n;i++) {
            F[i%4] =F[(i-1)%4]+F[(i-2)%4] +2 *
F[(i-3)%4];
            T+=F[i%4];
        }
    }
    printf("\n Tong = %d ",T);
    getch();
}
```

---

# DÒNG ĐỆ QUY

```
int H(int n) {  
    if (n<5) return n;  
    else  
        return (H(n-4) + H(n-3) - H(n-2) +  
                H(n-1));  
}  
long Tong(int n) {  
    long t=0;  
    for(i=1;i<=n;i++) t+=H(i);  
    return t;  }
```

# KHÔNG DÙNG ĐỆ QUY

```
long Tong(int n)
{
    int i,h1=1,h2=2,h3=3,h4=4,h;
    long t=10;
    if(n==1) return 1;
    else if(n==2) return 3;
    else if(n==3) return 6;
    else if(n==4) return 10;
    else {
        for(i=5;i<=n;i++) {
            h=h1+h2-h3+h4;
            t+=h; h1=h2; h2=h3; h3=h4; h4=h;
        }
    }
    return t;
}
```

---

# Ví dụ

- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau :

$$T=1+2+3+6+11+20+37+68+125+\dots$$

- Sau đó viết chương trình dưới dạng không đệ quy

- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau :

$$T=1+2+3+7+13+23+43+79+145+\dots$$

- Sau đó viết chương trình dưới dạng không đệ quy.  
Biết rằng số phần tử của dãy số luôn  $\geq 4$

## Ví dụ (tiếp)

- Viết hàm đệ quy tính các phần tử của dãy số sau với  $n$  phần tử ( $n \geq 10$ ), rồi tính tổng các phần tử của dãy số

1,2,3,4,4,5,8,11,12,14,21,30,35,40,56,.....

- Sau đó viết lại toàn bộ chương trình tính tổng dãy số trên mà không dùng đệ quy

1,2,3,4,6,9,14,21,32,48,73,110,167,252,...



- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau:

$$T=1+2+3+6+11+20+37+68+125+\dots$$

- Viết hàm đệ quy tính giá trị các phần tử rồi tính tổng của dãy số sau :

$$T=1+2+3+7+13+23+43+79+145+\dots$$

....

- Biết rằng số phần tử của dãy số luôn  $\geq 4$

- Viết hàm đệ quy tính các phần tử của dãy số sau với  $n$  phần tử ( $n \geq 10$ ), rồi tính tổng các phần tử của dãy số

1,2,3,4,4,5,8,11,12,14,21,30,35,40,56,.....

1,2,3,4,6,9,14,21,32,48,73,110,167,252,...

**Khử đệ quy các chương trình trên**