

เมทีอดแบบเรียกซ้ำ (recursive)

ความสัมพันธ์เวียนเกิด (Recurrences)

- การเขียนความสัมพันธ์ของจำนวนเต็มในลำดับ

$$0, 1, 2, 3, 4, \dots \quad a_n = a_{n-1} + 1 \quad \text{เมื่อ } n > 0, a_0 = 0$$

$$3, 5, 7, 9, 11, \dots \quad a_n = a_{n-1} + 2 \quad \text{เมื่อ } n > 0, a_0 = 3$$

$$0, 1, 3, 6, 10, 15, \dots \quad a_n = a_{n-1} + n \quad \text{เมื่อ } n > 0, a_0 = 0$$

$$0, 1, 1, 2, 3, 5, 8, \dots \quad f_n = f_{n-1} + f_{n-2} \quad \text{เมื่อ } n > 1, f_0 = 0, f_1 = 1$$

2110101 วิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย (15/06/52)

2

$$a_n = 0, 1, 3, 6, 10, 15, \dots$$

- รู้ว่า $a_n = (0+1+2+\dots+n)$

```
public static int a(int n) {  
    int s = 0;  
    for (int i = 0; i <= n; i++)  
        s += i;  
    return s;  
}
```

แบบที่ 1

- รู้ว่า $a_n = a_{n-1} + n$ เมื่อ $n > 0, a_0 = 0$

```
public static int a(int n) {  
    if (n <= 0)  
        return 0;  
    else  
        return a(n-1) + n;  
}
```

แบบที่ 2

2110101 วิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย (15/06/52)

3

$$f_n = 0, 1, 1, 2, 3, 5, 8, 13, \dots$$

```
public static int fib(int n) {  
    if (n < 2) return n;  
    int[] f = new int[n + 1];  
    f[0] = 0; f[1] = 1;  
    for (int i = 2; i <= n; i++)  
        f[i] = f[i-1] + f[i-2];  
    return f[n];  
}
```

แบบที่ 1

```
public static int fib(int n) {  
    if (n < 2) return n;  
    int fn = 0, fn1 = 0, fn2 = 1;  
    for (int i = 2; i <= n; i++) {  
  
    }  
    return fn;  
}
```

แบบที่ 2

2110101 วิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย (15/06/52)

4

$$f_n = 0, 1, 1, 2, 3, 5, 8, 13, \dots$$

```
public static int fib(int n) {  
    if (n < 2) return n;  
    int fn2 = fib(n-2);  
    int fn1 = fib(n-1);  
    return fn1 + fn2;  
}
```

แบบที่ 3

```
public static int fib(int n) {  
    if (n < 2) return n;  
    return fib(n-1) + fib(n-2);  
}
```

2110101 วิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย (15/06/52)

5

เมทีอดหาค่า $n!$

```
public static int fac(int n) {  
    int fac = 1;  
    for (int i = 1; i <= n; i++) {  
        fac = fac * i;  
    }  
    return fac;  
}
```

แบบที่ 1

$$n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$$

```
public static int fac(int n) {  
    if (n == 0) return 1;  
    int f = fac(n-1);  
    return n * f;  
}
```

กรณีเล็กสุด

ตามนิยามของแฟกทอเรียล

แบบที่ 2

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n \geq 1 \end{cases}$$

2110101 วิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย (15/06/52)

6

โปรแกรมแบบ recursive

- โปรแกรมที่มีการเรียกตัวเอง
- การทำงานแบ่งเป็นกรณี ๆ
 - กรณีพื้นฐาน ทำเสร็จได้ทันที
 - กรณีอื่น เรียกตัวเอง โดยขนาดของปัญหาต้องเล็กลง
- การเรียกแต่ละครั้ง จะมีการสร้างตัวแปรในเมธอดชุดใหม่ในระบบ

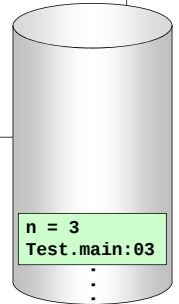
```
public static void main(String[] args) {
    jeng3(10);
}
public static void jeng3(int n) {
    System.out.println(n);
    jeng3(n-1);
}
```

ลองสังเกตุว่า เกิดอะไรขึ้น

ตัวแปรระหว่างการเรียกซ้ำ

```
01: public class Test {
02:     public static void main(String[] args) {
03:         System.out.println(fac(3));
04:     }
05:     public static int fac(int n) {
06:         if (n == 0) return 1;
07:         int f = fac(n-1);
08:         return n * f;
09:     }
10: }
```

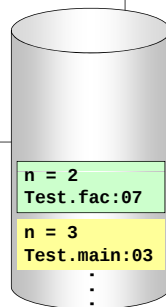
เมื่อระบบเรียกเมธอด fac
ระบบจะสร้างตัวแปรของ fac



ตัวแปรระหว่างการเรียกซ้ำ

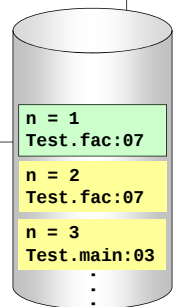
```
01: public class Test {
02:     public static void main(String[] args) {
03:         System.out.println(fac(3));
04:     }
05:     public static int fac(int n) {
06:         if (n == 0) return 1;
07:         int f = fac(n-1);
08:         return n * f;
09:     }
10: }
```

เมื่อเรียกเมธอด fac อีก
ระบบจะสร้างตัวแปรชุดใหม่ของ fac
ชุดใหม่ทับชุดเก่า
(จะนำตัวแปรชุดเก่ากลับมาใช้ใหม่
เมื่อ return)



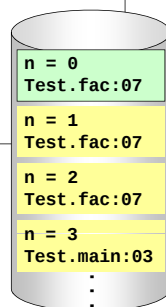
ตัวแปรระหว่างการเรียกซ้ำ

```
01: public class Test {
02:     public static void main(String[] args) {
03:         System.out.println(fac(3));
04:     }
05:     public static int fac(int n) {
06:         if (n == 0) return 1;
07:         int f = fac(n-1);
08:         return n * f;
09:     }
10: }
```



ตัวแปรระหว่างการเรียกซ้ำ

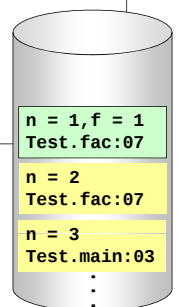
```
01: public class Test {
02:     public static void main(String[] args) {
03:         System.out.println(fac(3));
04:     }
05:     public static int fac(int n) {
06:         if (n == 0) return 1;
07:         int f = fac(n-1);
08:         return n * f;
09:     }
10: }
```



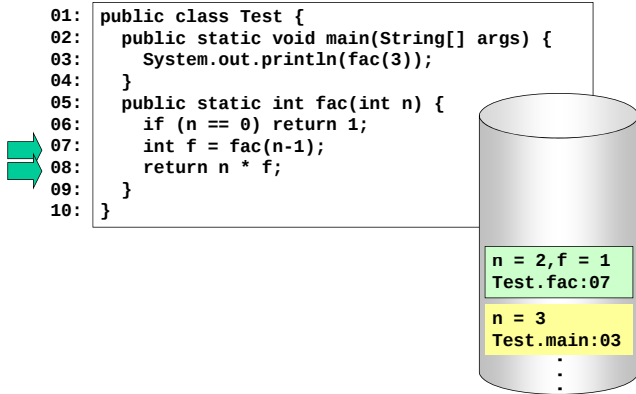
ตัวแปรระหว่างการเรียกซ้ำ

```
01: public class Test {
02:     public static void main(String[] args) {
03:         System.out.println(fac(3));
04:     }
05:     public static int fac(int n) {
06:         if (n == 0) return 1;
07:         int f = fac(n-1);
08:         return n * f;
09:     }
10: }
```

เมื่อ return
ระบบลบตัวแปรของเมธอดออก
และนำตัวแปรชุดล่างสุดมาใช้ต่อ



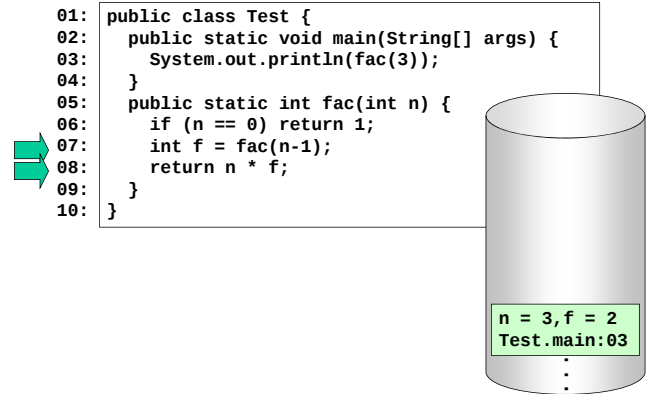
ตัวแปรระหว่างการเรียกซ้ำ



2110101 วิศวกรรมคอมพิวเตอร์ จุฬาฯ (15/06/52)

13

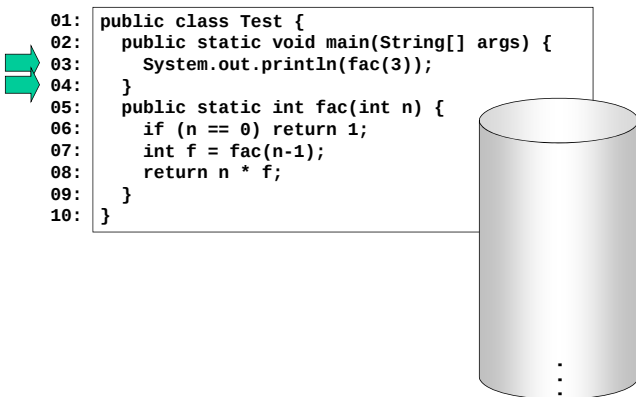
ตัวแปรระหว่างการเรียกซ้ำ



2110101 วิศวกรรมคอมพิวเตอร์ จุฬาฯ (15/06/52)

14

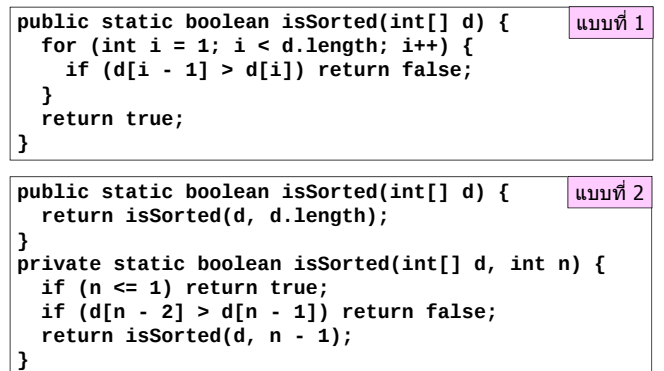
ตัวแปรระหว่างการเรียกซ้ำ



2110101 วิศวกรรมคอมพิวเตอร์ จุฬาฯ (15/06/52)

15

เมท็อดตรวจสอบว่าอาร์เรย์เรียงลำดับหรือไม่



2110101 วิศวกรรมคอมพิวเตอร์ จุฬาฯ (15/06/52)

16

การคำนวณ $a^k \bmod m$

- $a^k \bmod m$ เป็นการคำนวณที่ใช้บ่อยในการเข้ารหัสลับ
- ตัวอย่างที่ 1 : $2^{20} \% 31 = ?$
 - คำนวณ 2^{20} ได้ 1048576 จากนั้น $\% 31$ ได้ 1
- ตัวอย่างที่ 2 : $2^{101} \% 31 = ?$
 - คำนวณ 2^{101} ได้ 2535301200456458802993406410752 จากนั้น $\% 31$ ได้ 2
 - ทำอีกแบบ :

$$a^k \% m = \begin{cases} 1 & k = 0 \\ (a^{\lfloor k/2 \rfloor} \% m)^2 \% m & k \text{ is even} \\ a(a^{\lfloor k/2 \rfloor} \% m)^2 \% m & k \text{ is odd} \end{cases}$$

2110101 วิศวกรรมคอมพิวเตอร์ จุฬาฯ (15/06/52)

17

$$\begin{aligned} 2^{60} \bmod 10 &= 4^2 \bmod 10 = 6 \\ 2^{30} \bmod 10 &= 8^2 \bmod 10 = 4 \\ 2^{15} \bmod 10 &= 2 \times 8^2 \bmod 10 = 8 \\ 2^7 \bmod 10 &= 2 \times 8^2 \bmod 10 = 8 \\ 2^3 \bmod 10 &= 2 \times 2^2 \bmod 10 = 8 \\ 2^1 \bmod 10 &= 2 \times 1^2 \bmod 10 = 2 \\ 2^0 \bmod 10 &= 1 \end{aligned}$$

2110101 วิศวกรรมคอมพิวเตอร์ จุฬาฯ (15/06/52)

18

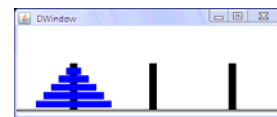
การคำนวณ $a^k \bmod m$

$$a^k \bmod m = \begin{cases} 1 & k = 0 \\ (a^{\lfloor k/2 \rfloor} \bmod m)^2 \bmod m & k \text{ is even} \\ a(a^{\lfloor k/2 \rfloor} \bmod m)^2 \bmod m & k \text{ is odd} \end{cases}$$

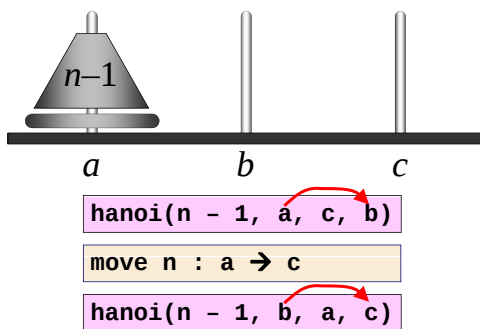
```
public static int powerMod(int a, int k, int m) {
    if (k == 0) return 1;
    int p = powerMod(a, k / 2, m);
    p = (p * p) % m;
    if ((k % 2) == 1) p = (a * p) % m;    // p is odd
    return p;
}
```

ปัญหา Tower of Hanoi

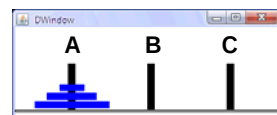
- มีเส้า 3 ต้น
- ใส่จานกลมมีรูตรงกลางขนาดต่างกัน n ใบ ในเส้าต้นซ้ายกองเรียงตามขนาด ใบใหญ่สุดอยู่ล่างสุด
- ต้องการ
 - ย้ายจานทุกใบจากเส้าซ้ายไปเส้าที่เส้าขวา
 - ย้ายได้ครั้งละใบ
 - อนุญาตให้ใบเล็กทับใบใหญ่ได้เท่านั้น
- ย้ายอย่างไร ?



hanoi(n , a , b , c)



โปรแกรมแก้ปัญหา Tower of Hanoi



```
move 1 : A -> C
move 2 : A -> B
move 1 : C -> B
move 3 : A -> C
move 1 : B -> A
move 2 : B -> C
move 1 : A -> C
```

```
public class HanoiTower {
    public static void main(String[] args) {
        hanoi(3, "A", "B", "C");
    }
    private static void hanoi(int n,
                               String from, String tmp, String to) {
        if (n == 0) return;
        hanoi(n - 1, from, to, tmp);
        System.out.println("move " + n + " : " + from +
                           " -> " + to);
        hanoi(n - 1, tmp, from, to);
    }
}
```