บทที่ 6:
Recursion

สุนทรี คุ้มไพโรจน์

Recursion

- **การเวียนเกิด**
- ปรากฏการณ์ที่มีการวนกลับไปเรียกฟังก์ชันตัวเองซ้ำๆ

เปรียบเทียบการทำงาน

เขียนโปรแกรมเพื่อหาผลบวกตัวเลขจำนวนเต็มบวก $\sum_{i=1}^{n} i$

Iteration

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

Recursion

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

ให้เขียนแผนภาพแสดงการเรียกซ้ำ

ประเภทของ recursion

- Linear Recursion
- Binary Recursion
- Multiple Recursion
- Tail Recursion

Linear Recursion

นิยามแบบเวียนเกิดได้ดังนี้

เขียนโปรแกรมเพื่อหาผลคูณจำนวนเต็มบวก

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

Iteration

```
int factorial(int n) {
    int j = 1;
    for (int i = 2; i <= n; i++) {
        j = j * i;
    }
    return j;
}
```

Recursion

มีการเรียกตัวเองในฟังก์ชัน 1 ครั้ง

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

ให้เขียนแผนภาพแสดงการเรียกซ้ำ

แผนภาพการเรียกซ้ำของฟังก์ชันยกกำลัง

$3^5 = 3 \times 3^4 = 3 \times 81 = 243$

$3^4 = 3 \times 3^3 = 3 \times 27 = 81$

$3^3 = 3 \times 3^2 = 3 \times 9 = 27$

$3^2 = 3 \times 3^1 = 3 \times 3 = 9$

$3^1 = 3 \times 3^0 = 3 \times 1 = 3$

$3^0 = 1$

การคำนวณจำนวนเต็มยกกำลัง คือ

$3^0 = 1$

$3^1 = 3$

$3^2 = 3 \times 3 = 9$

$3^3 = 3 \times 3 \times 3 = 27$

$3^4 = 3 \times 3 \times 3 \times 3 = 81$

$3^5 = 3 \times 3 \times 3 \times 3 \times 3 = 243$

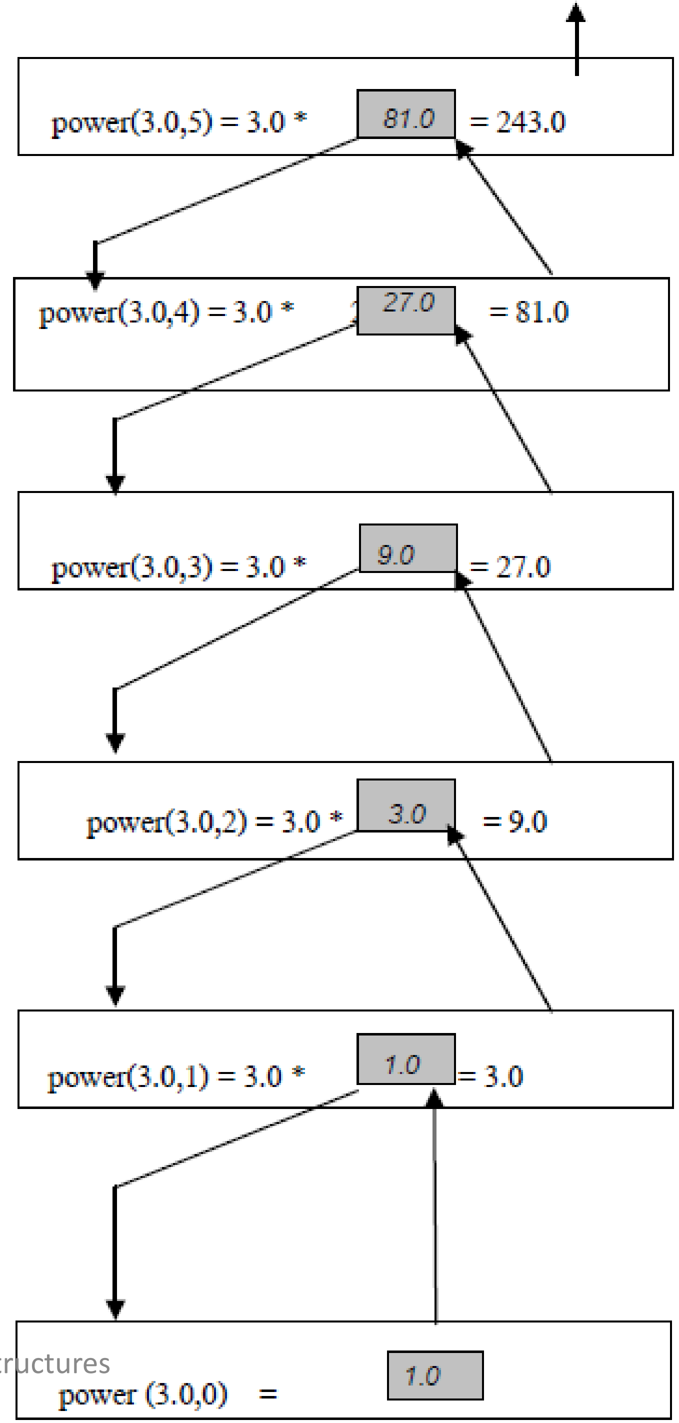
Algorithm การคำนวณเลขยกกำลัง

โดยใช้นิยามแบบเรียกซ้ำของฟังก์ชันยกกำลัง

ให้ $X^0 = 1$

$X^n = X * X^{n-1}$ สำหรับ $n > 0$

```
double power (int x, int n)
{
int result;
if ( n = = 0)
  result = 1;
else
  result : = x * power (x, n -1);
return result;
}
```



01418231 Data structures

ขั้นตอนการทำงานของแบบเวียนเกิด

- ประกอบไปด้วย 2 ขั้นตอน คือ
- Base step กรณีสิ้นสุดการเวียนเกิด ในกรณีนี้ไม่มีการเรียกซ้ำ แต่จะมีคืนค่าคงที่ออกมา เช่น return 1, return n เป็นต้น
- Induction/Recursion Step กรณีนี้เกิดจากการเรียกฟังก์ชันใช้งานตัวเอง เป็นการเรียกซ้ำตัวเองอีกครั้งอยู่ภายในตัวฟังก์ชันนั้นเอง จะเรียกซ้ำแบบนี้ไปเรื่อยๆ จนกว่าจะเข้าสู่ base step จึงจะหยุดการเรียกซ้ำ

วิธีการเขียนโปรแกรม Recursion

- การแปลงสมการคณิตศาสตร์ให้เป็นฟังก์ชันเวียนเกิดแบบง่าย
- เขียนสมการหรือฟังก์ชันทางคณิตศาสตร์

เช่น $f(n) = \sum_{i=1}^{n} i$ $\quad n = 0,1,2,3,\ldots$

- แยกสมการออกเป็น 2 กรณี (หรือมากกว่าก็ได้)
  - Base case กรณีที่สมการเท่ากับค่าของพจน์แรกเพียงพจน์เดียวซึ่งจะได้
    เช่นกำหนดให้ $f(n) = 1$ กรณี $n = 1$
  - Recursion/Induction case กรณีที่สมการอยู่ในรูปทั่วไป
    โดยสมมติว่า กรณี $f(n-1)$ เป็นจริง เมื่อเพิ่มพจน์สุดท้ายเข้าไปก็เป็นจริง
    เช่นกำหนดให้ $f(n) = f(n-1) + n$ $\quad n = 2,3,4,\ldots$
- ใช้คำสั่ง if-else เพื่อระบุเงื่อนไขของแต่ละกรณี ตัวอย่าง เช่น

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

Exercise

เขียนฟังก์ชัน Recursion และ แผนภาพแสดงการเรียกซ้ำของ การเรียก a(5)

ฟังก์ชัน $a_n = a_{n-1} + 3$ and $a_1 = 2$

Sequence a = {2, 5, 8,...}

Tail Recursion

- เป็นการส่งผลลัพธ์เป็น พารามิเตอร์ไปพร้อมเสร็จสรรพ
- ดังนั้นเมื่อถึงฟังก์ชันเรียกซ้ำถึงที่สุดแล้ว ก็สามารถตอบได้เลย
- ไม่ต้องเปลือง Stack หรือเนื้อที่ในหน่วยความจำ
- เปรียบเทียบ Linear Recursion กับ Tail Recursion ดังนี้

Fr(3)=fr(2)+3=6
Fr(2)=fr(1)+2=3
Fr(1)=fr(0)+1=1
Fr(0)=0

Ft(3,0)
Ft(2,3)
Ft(1,5)
Ft(0,6)

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

```
int ft(int n,int sum){
    if (n<=0)
        return sum;
    else
        return ft(n-1, n+sum);
}
```

- เขียนฟังก์ชัน Tail recursion ของ สมการ

$$a_n = a_{n-1} + 3 \text{ and } a_1 = 2$$

และเขียนแผนภาพแสดงการเรียกซ้ำของ Tail Recursion ของการเรียกฟังก์ชัน a(5,2) คำนวณค่า a_5

Sequence => 2, 5, 8, 11, 14

a_5

Result ของ sequence

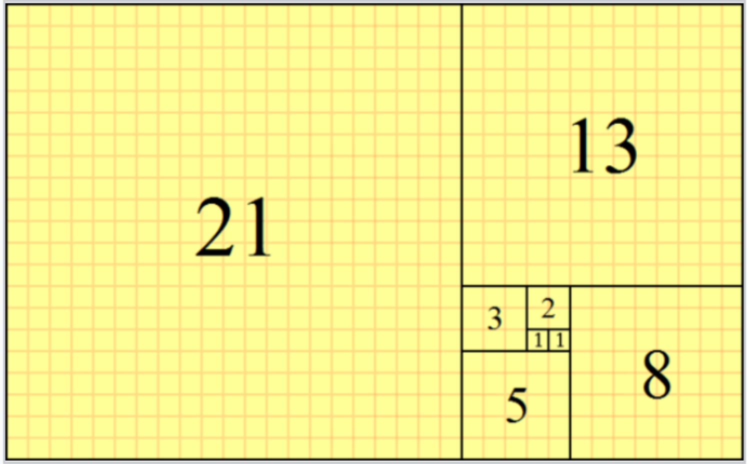
Binary Recursion

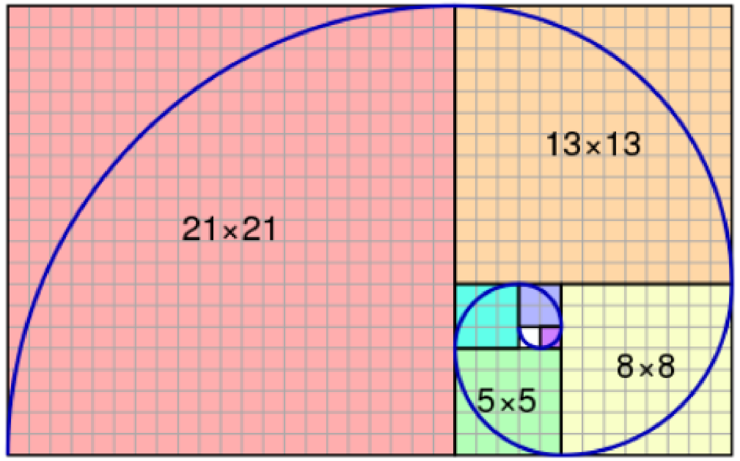
- เป็นการเขียนฟังก์ชันเพื่อเรียกตัวเอง โดยทำการเรียกตัวเอง 2 ครั้งในฟังก์ชันตัวเอง
- ตัวอย่างเช่น ลำดับฟิโบนักซี
  หากเขียนให้อยู่ในรูปของสัญลักษณ์ ลำดับ F_n ของจำนวนฟีโบนัชชีนิยามขึ้นด้วยความสัมพันธ์เวียนเกิดดังนี้

$$F_n = F_{n-1} + F_{n-2}$$

โดยกำหนดค่าเริ่มแรกให้ [1]

$$F_0 = 0; \; F_1 = 1$$





Fibonacci

Iteration

```
int fibonacci(int n) {
   if (n == 1)
      return 0;
   else if (n == 2)
      return 1;
   else {
      int a = 1, b = 1, i, c;
      for (i = 3; i <= n; i++) {
         c = a + b;
         a = b;
         b = c;
      }
      return a;
   }
}
```

Recursion

```
int fibonacci(int n) {
   if (n == 1)
      return 0;
   else if (n == 2)
      return 1;
   else
      return fibonacci(n - 1) + fibonacci(n - 2);
}
```

ให้เขียนแผนภาพแสดงการเรียกซ้ำ

Multiple recursion

- เป็นการเขียนฟังก์ชันเพื่อเรียกตัวเอง โดยทำการเรียกตัวเองหลายครั้งในฟังก์ชันตัวเอง
- ตัวอย่างให้แสดงผลความน่าจะเป็นในการผสมตัวอักษรทั้งหมด จากตัวอักษรชุด

  เช่น ABC
  คำตอบที่ได้คือ ABC ACB BAC BCA CAB CBA
- ตัวอย่างเลข Ackermann

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

เขียนโปรแกรมเรียกและตัวฟังก์ชัน Ackermann ให้เขียนแผนภาพแสดงการเรียกซ้ำ A(1,2)

Greatest Common Division(GCD)

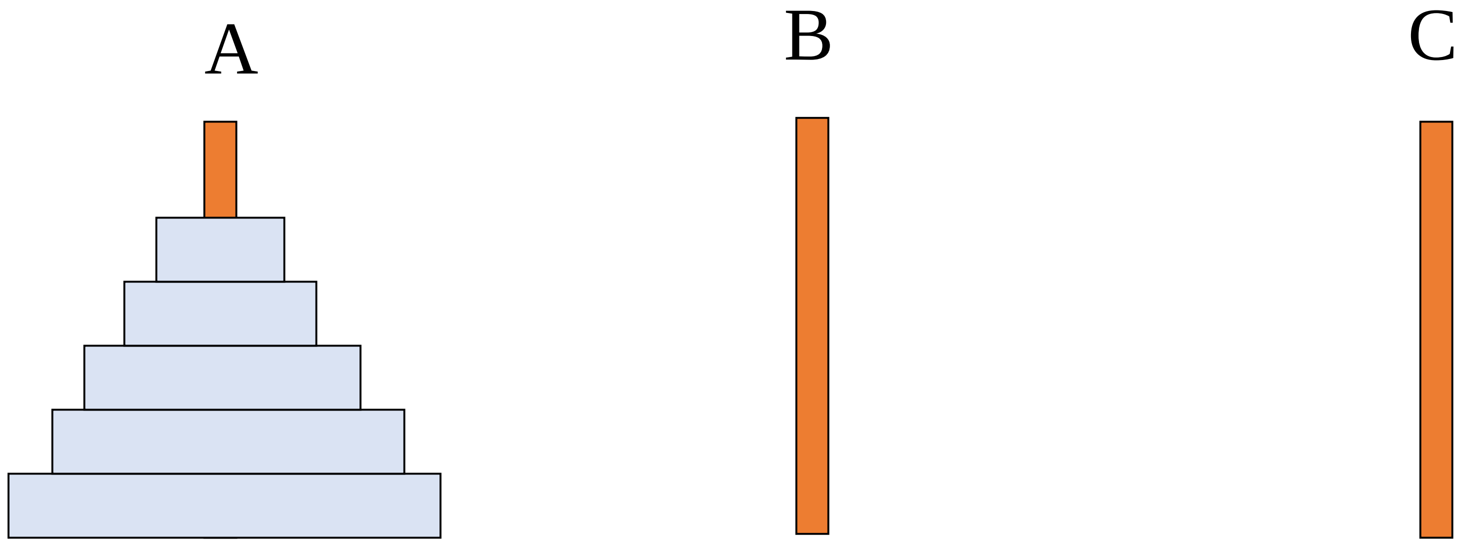
- การหาตัวหารร่วมมาก (ห.ร.ม.)
- gcd ของจำนวนเต็มซึ่งไม่เป็น 0 พร้อมกัน คือจำนวนเต็มที่มากที่สุดที่**หาร**ทั้งสองจำนวนลงตัว
- เช่น gcd ของ 10 กับ 25 คือ 5
- มีประโยชน์ในการทำเศษส่วนให้เป็นเศษส่วนอย่างต่ำ
- เขียนฟังก์ชันการหา gcd ของจำนวนเต็มที่ไม่เป็นค่าลบของจำนวน 2 จำนวน gcd(a,b) โดยใช้ Euclid's algorithm

$$\gcd(a,0) = a$$
$$\gcd(a,b) = \gcd(b, a \bmod b),$$

where

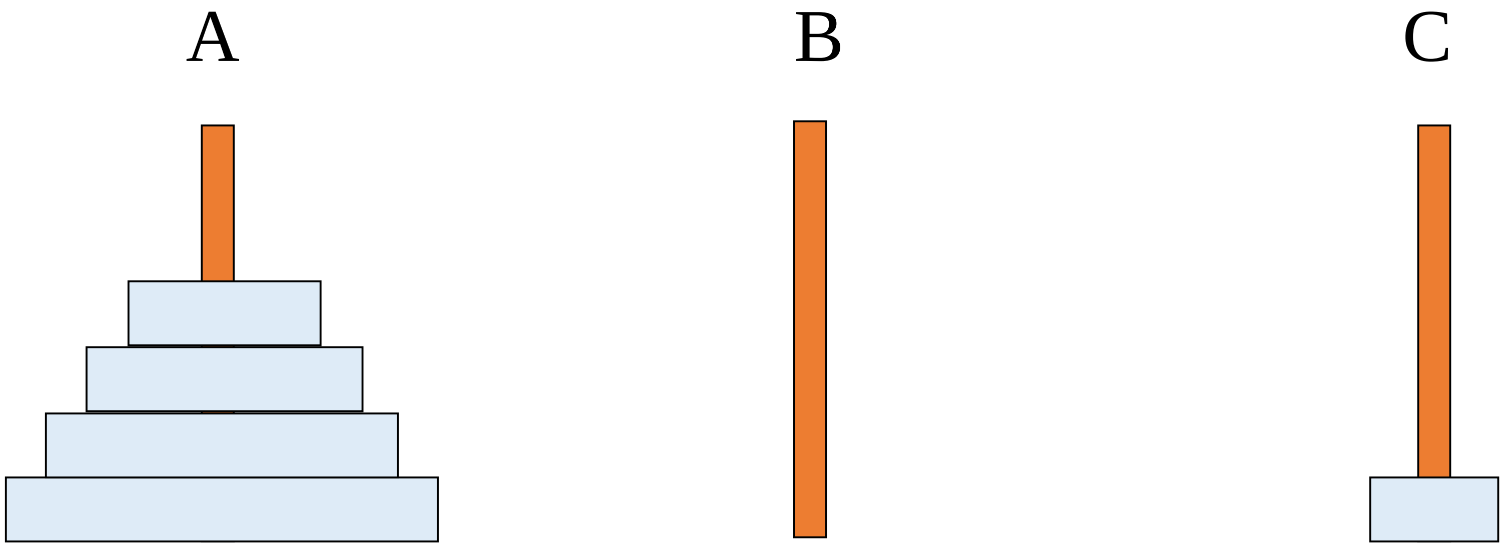
$$a \bmod b = a - b \left\lfloor \frac{a}{b} \right\rfloor .$$

Tower of Hanoi Problem



ปัญหาของการย้ายแผ่นจานขนาดต่างๆ จากหลักหนึ่ง(A) ไปยังอีกหลักหนึ่ง(C) โดยมีหลักสำหรับวางซ้อนจานอยู่ 3 หลัก

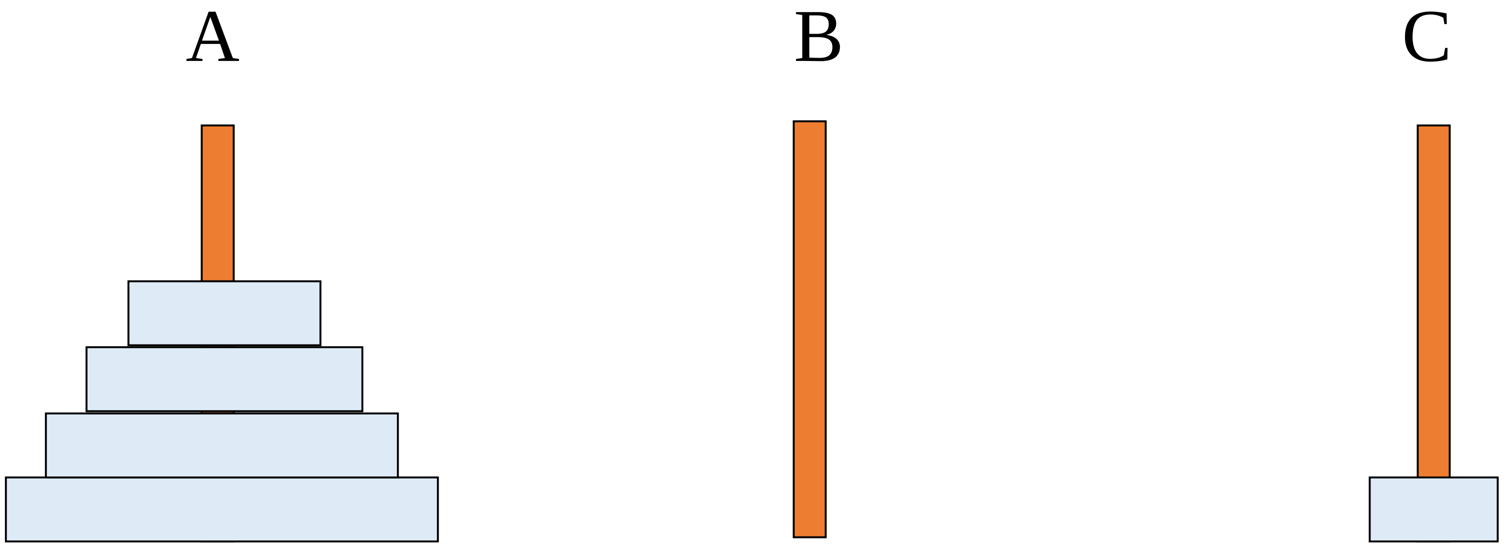
Tower of Hanoi Problem



เงื่อนไขในการเคลื่อนย้ายแผ่นจาน

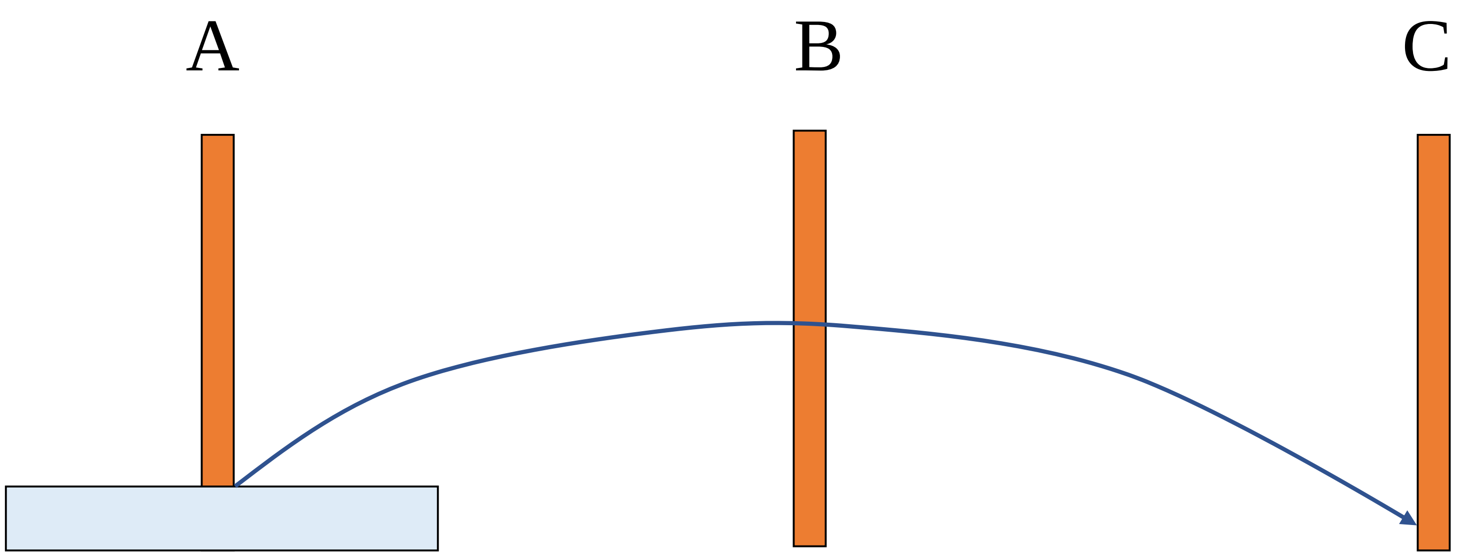
1. สามารถเคลื่อนย้ายแผ่นจานได้ครั้งละ 1 แผ่นเท่านั้น
2. เคลื่อนได้เฉพาะแผ่นจานที่อยู่บนสุดเท่านั้น โดยไม่ขัดกับเงื่อนไขในข้อ 3
3. แผ่นจานที่มีขนาดเล็กสามารถวางซ้อนบนแผ่นจานที่มีขนาดใหญ่กว่าได้ แต่แผ่นจานที่มีขนาดใหญ่จะวางซ้อนบนแผ่นจานที่มีขนาดเล็กกว่าไม่ได้

Tower of Hanoi Problem



ให้คำสั่ง MOVE(N,A,B,C) เป็นการเคลื่อนย้ายแผ่นจานจำนวน N แผ่น จากตำแหน่ง A ไปยังตำแหน่ง C (ส่วนตำแหน่ง B คือตำแหน่งที่ว่างอยู่ใช้สำหรับวางแผ่นจานชั่วคราว)

Tower of Hanoi Problem



กรณี N = 1

ถ้าต้องการเคลื่อนย้ายแผ่นจานตำแหน่ง A ไปยังตำแหน่ง C สามารถทำได้ทันที

เขียนคำสั่งได้ดังนี้ MOVE(1,A,B,C)

มีคำตอบเป็น move A to C

Tower of Hanoi Problem

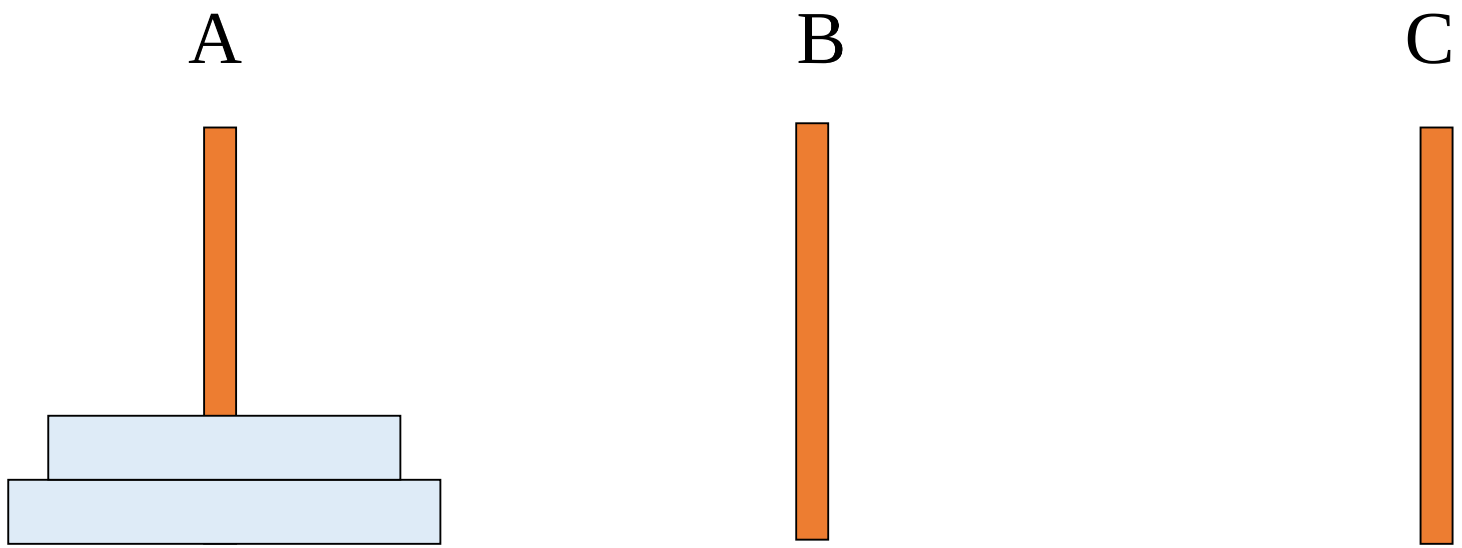
กรณี N = 2 จะต้องย้ายแผ่นจาน 3 ครั้งดังนี้

การเคลื่อนย้ายครั้งที่ 1 MOVE(1,A,C,B) มีคำตอบเป็น move A to B

การเคลื่อนย้ายครั้งที่ 2 MOVE(1,A,B,C) มีคำตอบเป็น move A to C

การเคลื่อนย้ายครั้งที่ 3 MOVE(1,B,A,C) มีคำตอบเป็น move B to C

ให้วาดภาพประกอบการเคลื่อนย้ายแต่ละครั้ง



Tower of Hanoi Problem

- ให้เขียนคำสั่งของการย้ายจานและวาดภาพประกอบ กรณีที่ N=3

Tower of Hanoi Problem

ขั้นตอนการแก้ปัญหา

กรณี Base Case : เมื่อ N = 1 (เหลือจานใบเดียวในเสา(A))

move จานใบเดียวนี้จาก เสาต้นทาง ไปยังเสาปลายทาง

กรณี Recursive Case : (มีจำนวน N จานในเสา(A))

move N – 1 จาน จาก เสาต้นทาง ไปยัง เสาสำรอง

move จานใบที่ N จากเสาต้นทาง ไปยัง เสาปลายทาง

move N – 1 จาน จาก เสาสำรอง ไปยัง เสาปลายทาง