ครั้งที่ 8
การเขียนโปรแกรมเรียกตัวเอง

Recursive หรือ Recursion

- เป็นการเขียนคำสั่งโปรแกรมที่มีการปฏิบัติงานเรียกตัวเอง
- โดยมีการเรียกการทำงาน ในลำดับก่อนมาทำงานในลำดับปัจจุบัน
- โดยจะเรียกลำดับก่อนวนซ้ำไปเรื่อยๆ จนถึงข้อมูลที่กำหนดการสิ้นสุดในการเรียกตัวเอง

รูปแบบของฟังก์ชัน

- ค่าเริ่มต้นการทำงานวนรอบ โดยทั่วไปเป็น formal parameter ที่มีการส่งผ่านค่ามาจากจุดเรียกใช้
- ค่าสิ้นสุดการทำงานเป็นค่าของข้อมูลสุดท้าย ที่ผ่านการเรียกตัวเองโดยลดค่าลงอย่างเป็นลำดับ ในการเรียกฟังก์ชันแต่ละครั้ง
- คำสั่งในการปฏิบัติงาน เรียกตัวเองเป็นการประมวลผลที่นำผลลัพธ์จากการทำงานในรอบที่แล้วมาปฏิบัติงานในรอบปัจจุบัน เพื่อส่งผลลัพธ์จากการทำงานไปยังจุดเรียกใช้

recursive

- โดยทั่วไปการเรียกตัวเองเปรียบเสมือนการทำงานที่กระทำซ้ำๆกันในรูปของคำสั่ง while, for
- แต่เราสามารถนำมาเขียนในรูปแบบที่เรียกตัวเองโดยแทนด้วยคำสั่งเงื่อนไข การกระทำซ้ำจะแทนด้วยการเรียกฟังก์ชันในลำดับก่อนทำงาน

รูปแบบของคำสั่ง

if the stopping case is reached

Return a value for the stopping case

else

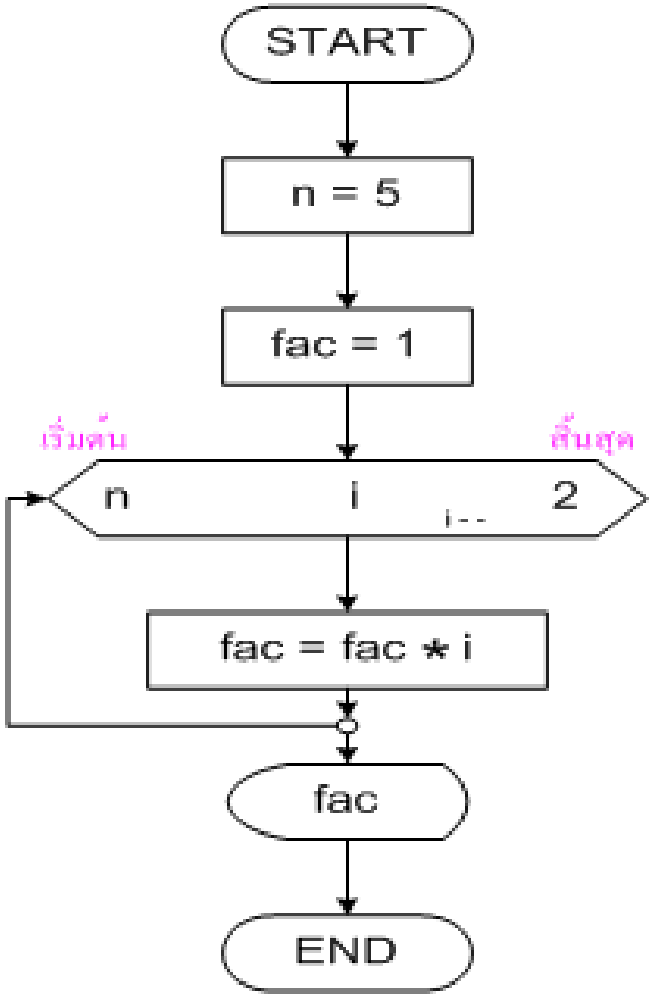
Return a value computed by calling the function again with different arguments.

ตัวอย่าง ฟังก์ชันหาค่าแฟคตอเรียล

ค่าแฟคตอเรียลเป็นการคำนวณหาผลคูณของตัวเลขที่มีการลดตัวคูณอย่างเป็นลำดับ จนกระทั่งมีค่าเท่ากับ 1 ดังนี้

10! = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1
5! = 5 * 4 * 3 * 2 * 1
4! = 4 * 3 * 2 * 1
1! = 1
0! = 1

ตัวอย่าง ฟังก์ชันหาค่าแฟคตอเรียล



n = 5

fac = 1

n	fac
5	5 * 1
4	4 * 5
3	3 * 20
2	2 * 60

ตัวอย่าง ฟังก์ชันหาค่าแฟคตอเรียล

```
int fac (int n)
{
   int sum, i;
   sum = 1;
   for (i = n; i > 1;  i - -)
         sum = sum * i;
   return sum;
}
```

การเรียกใช้

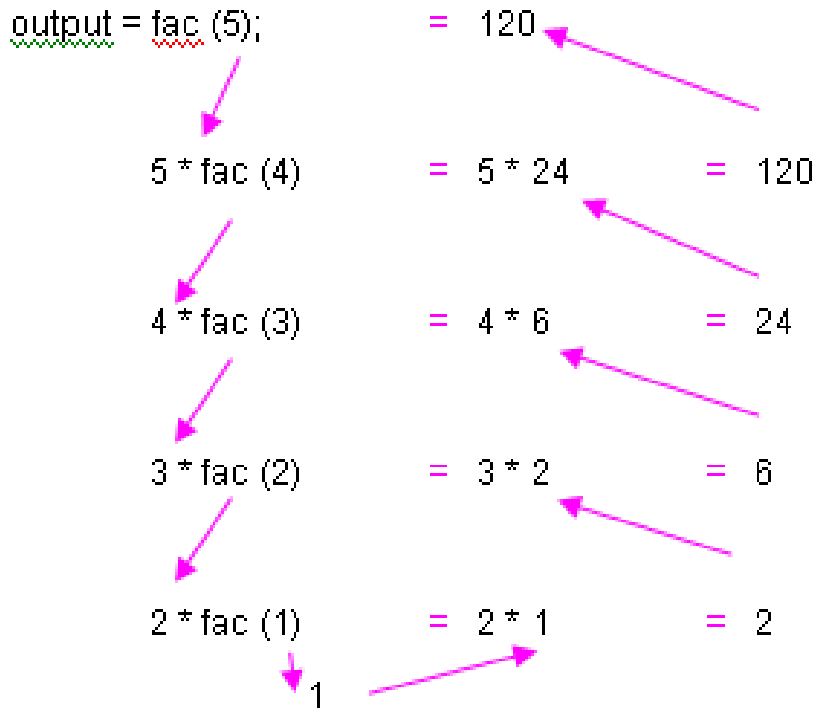
```
   output = fac (5);
```

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

ตัวอย่าง ฟังก์ชันหาค่าแฟคตอเรียล

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

การเรียกใช้ และการทำงาน



ผลการทำงาน output มีค่าเท่ากับ 120

ตัวอย่าง ฟังก์ชันหาค่ายกกำลัง

- การหาค่ายกกำลังที่มีการเรียกตัวเองที่มีการส่งผ่านค่าอาร์กิวเมนต์ 2 ค่าเพื่อปฏิบัติงาน โดยเขียนเป็นฟังก์ชันเรียกตัวเองในการหาค่า ใดๆ กำหนดให้ ตัวแปร x มีชนิดของมูลเป็น Double และ y เป็นชนิดข้อมูล int

ตัวอย่าง ฟังก์ชันหาค่ายกกำลัง

วิธีการนั้นต้องสมมุติค่า x และ y ใดๆขึ้นมาพิจารณา

กรณีที่ x=3 และ y=3

$$3^3 = 3 * 3 * 3$$

$$3^2 = 3 * 3$$

$$3^1 = 3$$

$$3^0 = 1$$

กรณีที่ x=3 และ y=-3

$$3^{-3} = \frac{1}{3} * \frac{1}{3} * \frac{1}{3}$$

$$3^{-2} = \frac{1}{3} * \frac{1}{3}$$

$$3^{-1} = \frac{1}{3}$$

$$3^0 = 1$$

วิธีการนั้นต้องสมมุติค่า x และ y ใดๆขึ้นมาพิจารณา

กรณีที่ x=3 และ y=3

$3^3 = 3 * 3 * 3$

$3^2 = 3 * 3$

$3^1 = 3$

$3^0 = 1$

กรณีที่ x=3 และ y=-3

$3^{-3} = \frac{1}{3} * \frac{1}{3} * \frac{1}{3}$

$3^{-2} = \frac{1}{3} * \frac{1}{3}$

$3^{-1} = \frac{1}{3}$

$3^0 = 1$

y เป็นจำนวนเต็มบวก

$x^y = x * x^{y-1}$

$3^3 = 3 * 3^2$

y เป็นจำนวนเต็มลบ

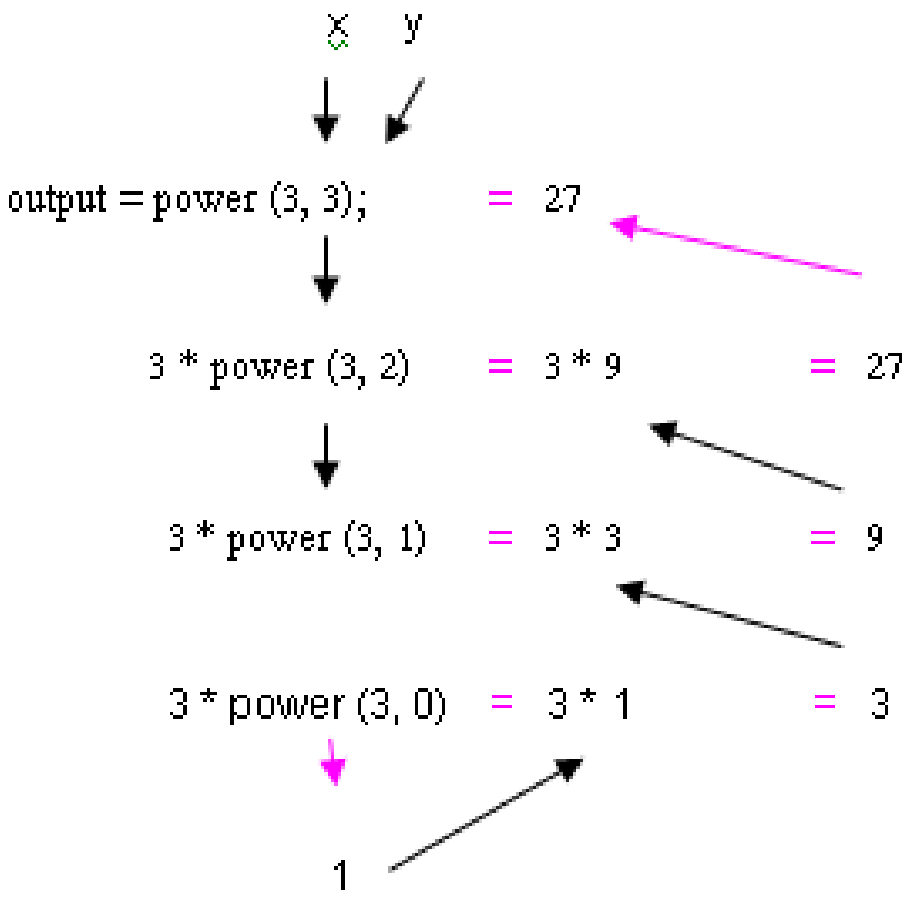
$x^y = \frac{1}{x} * x^{y+1}$

$3^{-3} = \frac{1}{3} * 3^{-2}$

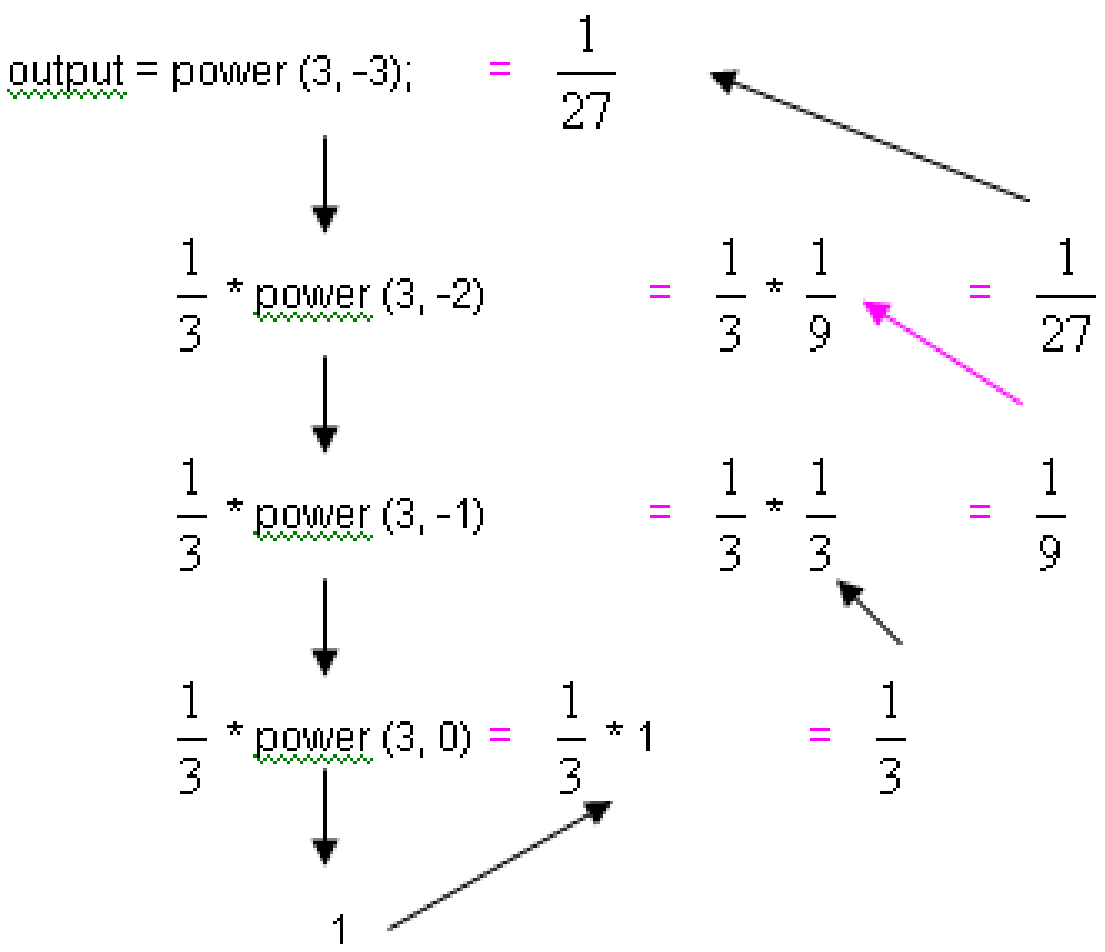
```
double power (double x, int y)
{
    if (y == 0)
          return 1;
    else if (y > 0)
         return x * power (x, y – 1);
    else
          return 1/x * power (x, y + 1);
}
```

กรณีที่ y มีค่าเป็นเลขจำนวนเต็มบวก

การเรียกใช้และการทำงาน



กรณีที่ y มีค่าเป็นเลขจำนวนเต็มลบ



ผลของการทำงาน output มีค่าเท่ากับ $\frac{1}{27}$

ตัวอย่างฟังก์ชันหาค่า Fibonacci

การหาค่า Fibonacci นั้นเราต้องกำหนดลำดับที่ต้องการ และต้องทราบค่าของลำดับ 1 และ ลำดับที่ 2 จึงจะหาค่าของลำดับอื่นๆได้ ในที่นี้ถ้าเรากำหนดให้ค่าของลำดับที่ 1 และค่าของลำดับที่ 2 มีค่าเท่ากับ 1 ดังนั้นข้อมูลในลำดับต่อๆไปมีค่าดังนี้

1 1 2 3 5 8 13 21 ....

จะเห็นได้ว่าการทำงานจะเริ่มจากลำดับที่ 3 เป็นต้นไป

โดยค่าของลำดับที่ 3 จะเกิดจากนำค่าของลำดับที่ 1 บวกกับค่าในลำดับที่ 2 และทำนองเดียวกัน ค่าในลำดับที่ 4 จะเกิดจากค่าในลำดับที่ 2 บวกกับค่าในลำดับที่ 3 กล่าวกันง่ายๆคือ นำค่าของลำดับก่อนหน้า 2 ลำดับมาบวกกันนั่นเอง

ตัวอย่างฟังก์ชันหาค่า Fibonacci

ในที่นี้ ถ้า $n = 7$

$$f(7) = f(5) + f(6)$$

$$13 = 5 + 8$$

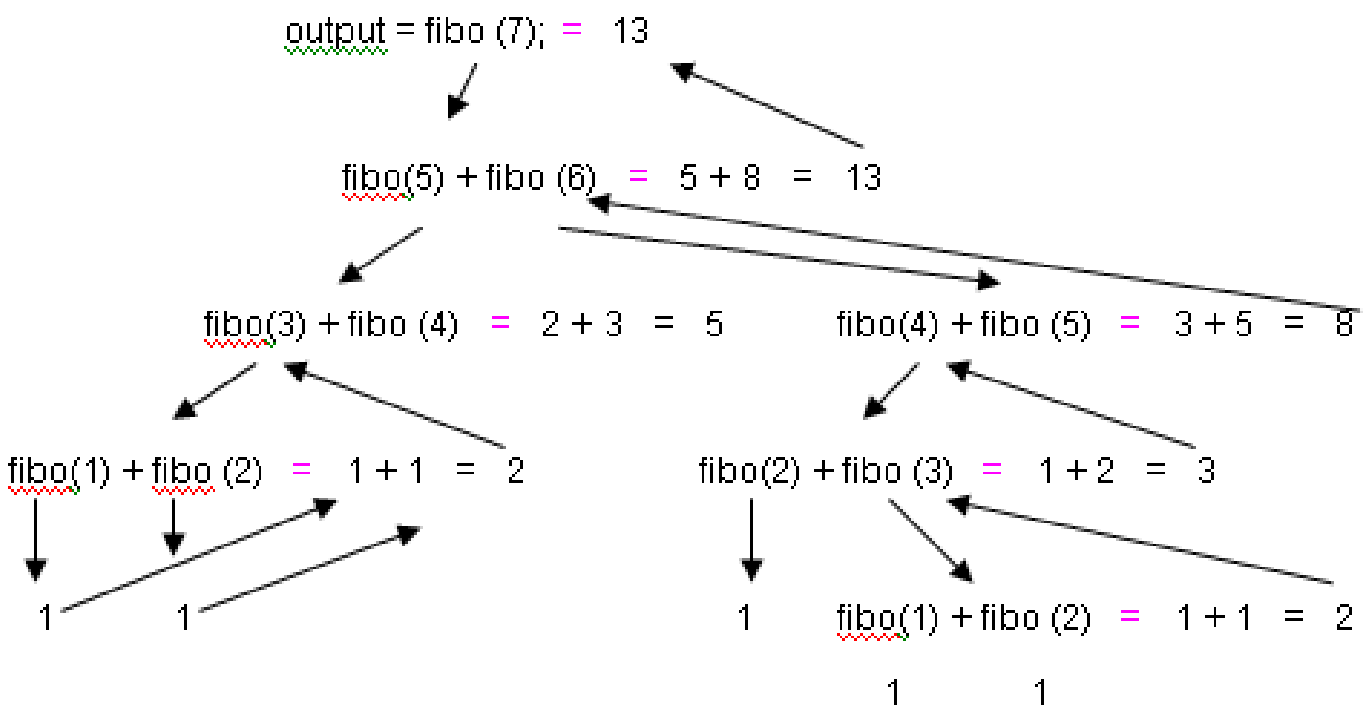
จากการคำนวณในลักษณะนี้นี่เองเราสามารถเขียนเป็นฟังก์ชันการทำงาน โดยสรุปได้ดังนี้

$$f(n) = 1 \quad \text{ถ้า } n = 1 \text{ หรือ } n = 2$$

$$= f(n-2) + f(n-1) \quad \text{ถ้า } n > 2$$

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

การเรียกใช้และการทำงาน



ตัวอย่างฟังก์ชันหาค่า Binary search

- วิธี Binary search นั้นเป็นการค้นหาที่แก้ปัญหาทำให้เราสามารถใช้เวลาเฉลี่ยในการค้นหาข้อมูลแต่ละตัวเท่าๆกัน ถึงแม้ว่าข้อมูลจะอยู่ในตำแหน่งใดก็ตามจะพบหรือไม่พบก็ตาม

Binary search

วิธีการนั้นสมมุติว่ามีข้อมูลที่เก็บในหน่วยความจำเรียงต่อๆกันดังรูป

0	1	2	3	4	5	6	7
31	63	82	25	64	77	50	10

1. นำข้อมูลมาเรียงลำดับจากน้อยไปมาก หรือจากมากไปน้อยก็ได้

0	1	2	3	4	5	6	7
10	25	31	50	63	64	77	82

2. หาข้อมูลตำแหน่งกลางของกลุ่ม

first = 0

last = 7

$$mid = \frac{0+7}{2} = 3$$

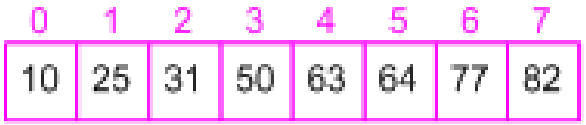
3. แบ่งกลุ่มออกเป็น 2 กลุ่ม นำค่าที่ต้องการค้นหา เปรียบเทียบกับตำแหน่งตรงกลาง ถ้าใช่ตำแหน่ง mid จะ return mid และหยุดค้นหา แต่ถ้าค่าน้อยกว่า จะพิจารณาเฉพาะกลุ่มทางซ้าย โดยกำหนดค่า first = 0 และให้ last = mid – 1 แต่ถ้าค่าที่ต้องการค้นหามากกว่าตำแหน่ง mid จะพิจารณา เฉพาะกลุ่มทางขวา โดยกำหนดให้ค่า first = mid + 1 และค่า last = 7 เป็นการกำหนดกลุ่มของข้อมูลใหม่มีขนาดเล็กลงไปเรื่อยๆ แต่ถ้าแบ่งข้อมูลจนกระทั่งไม่มีข้อมูลที่ต้องการค้นหาแล้วแสดงว่า ไม่พบข้อมูลที่ต้องการจะส่งให้ค่า first > last เราจะส่งค่า –1 กลับเพื่อให้ผู้ใช้ทราบว่าไม่พบข้อมูลนั่นเอง การทำงานจะเป็นการเรียกตัวเอง โดยกระทำซ้ำข้อ 2-3 จนพบข้อมูล หรือไม่พบข้อมูล

จากตัวอย่างนี้เรากำหนดตัวแปรต่างๆในการปฏิบัติงานดังนี้

table	เป็น array 1 มิติ เก็บข้อมูลเป็นเลขจำนวนเต็มใดๆ
target	ค่าที่ต้องการค้นหา
first	เก็บตำแหน่งแรกของกลุ่ม
last	เก็บตำแหน่งสุดท้ายของกลุ่ม
mid	เก็บตำแหน่งกลาง

การเรียกใช้

```
loc = binSearch (table, 25, 0, 7);
```



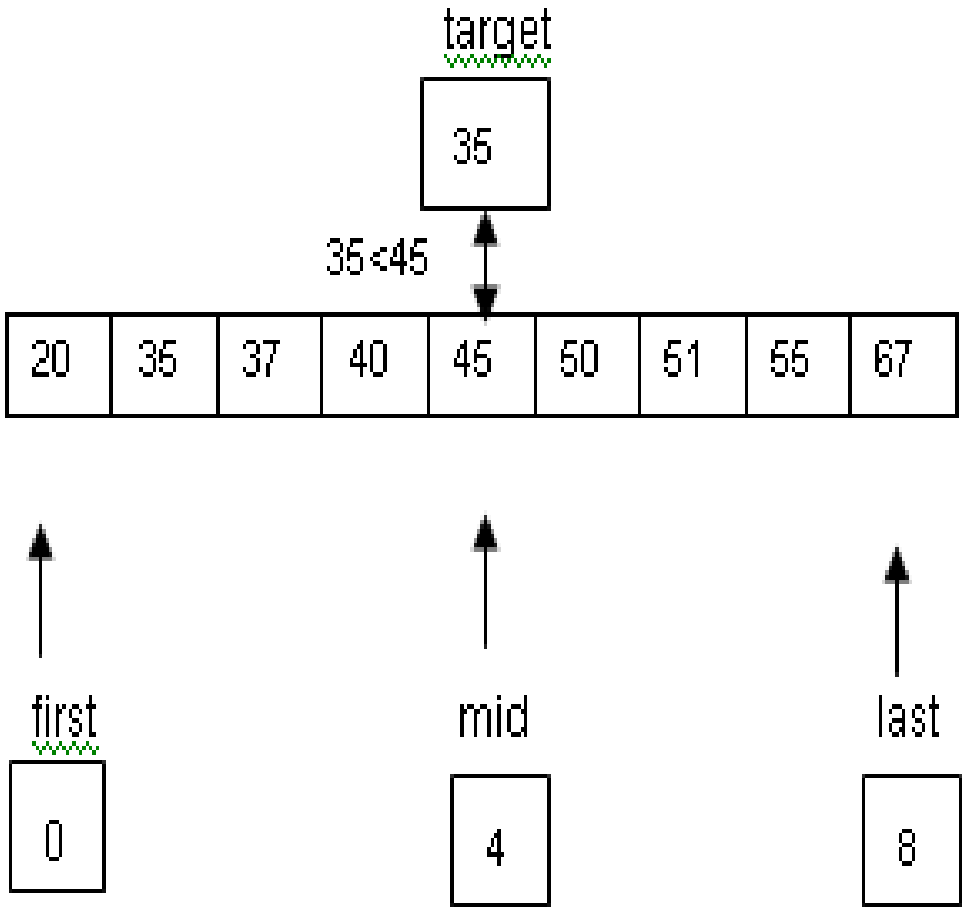
```
//Searches for target in elements first through last of array
//Precondition : The elements of table are sorted & first and last are defined.
//Postcondition: IF target is in the array, return its position; otherwise, returns –1.

int binSearch (int table[], int target, int first, int last)
{
        int mid;
        mid = (first + last)/2;
        if (first > last)              //หาไม่พบ
                return -1;
        else if (target = = table[mid])  //ค้นพบในตำแหน่งกลาง
                return mid;
        else if (target < table[mid])
                return binSearch (table, target, first, mid - 1);
        else
                return binSearch (table, target, mid + 1, last);
}
```

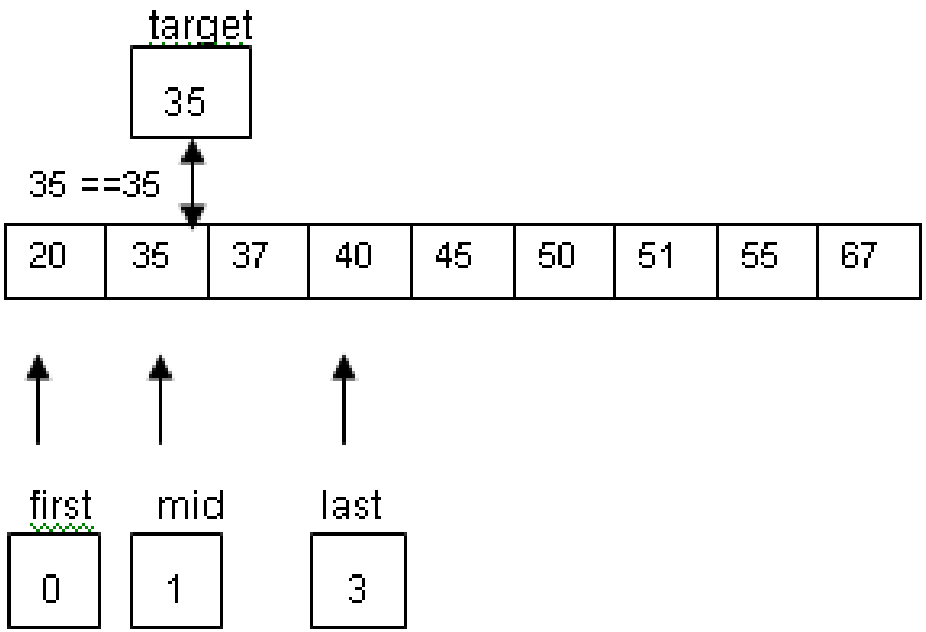
การทำงาน

- การทำงานจะมีเรียกตัวเองโดยมีการเปลี่ยนแปลงค่าของ first และ last ในการทำงานแต่ละรอบ
- โดยแบ่งกลุ่มของข้อมูลออกเป็นสองส่วน
- จนกระทั่งพบค่าที่ต้องการอยู่ในตำแหน่งกลางของกลุ่ม

การทำงานของฟังก์ชันแสดงได้จากรูปภาพต่อไปนี้ โดยการทำงานในรอบแรกจะพิจารณาข้อมูลทั้งหมดก่อน



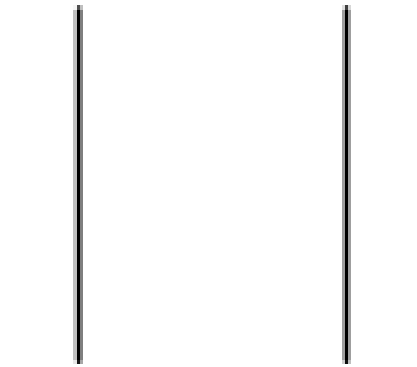
การทำงานในรอบต่อๆไปจะมีการปรับเปลี่ยนกลุ่มของข้อมูลโดยแบ่งครึ่งจากข้อมูลเดิม และกระทำในลักษณะเดียวกัน



ถ้าข้อมูลที่ต้องการค้นหาอยู่ในตำแหน่ง mid จะหยุดการกระทำซ้ำและส่งตำแหน่งของข้อมูลที่ต้องการค้นหากลับมายังจุดเรียกใช้

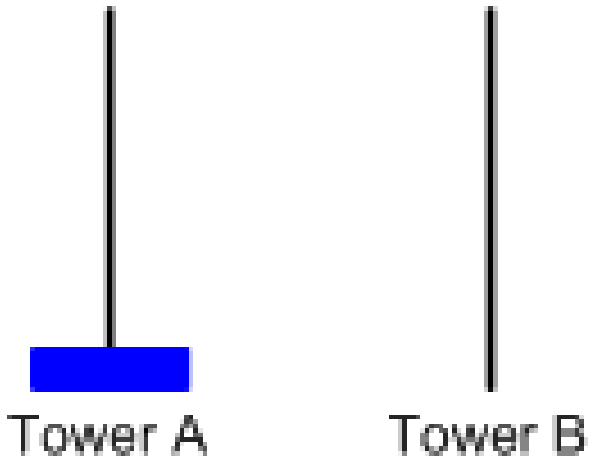
ตัวอย่าง ฟังก์ชันหอคอยฮานอย

- The Tower of Hanoi หรือ หอคอยฮานอย เป็นตัวอย่างของการปฏิบัติงานในลักษณะของการเรียกตัวเองที่ไม่มีการคำนวณหรือส่งผ่านค่าใดๆกลับ แต่เป็นการปฏิบัติงานที่กระทำในลักษณะที่เป็นขั้นตอนที่คล้ายๆกันเป็นการย้ายสิ่งของจากหอหนึ่งไปยังอีกหอหนึ่ง



สมมุติว่าต้องการย้ายสิ่งของมีด้วยกันหลายขนาดจากหอ A ไปยังหอ B โดยสิ่งของมีหลายขนาดแตกต่างกัน เรานำสิ่งของมาซ้อนทับกัน โดยสิ่งของขนาดเล็กต้องอยู่บนขนาดใหญ่เท่านั้น สมมุติว่าสิ่งของทุกชิ้นมีช่องตรงกลาง ให้ผู้ย้ายสามารถใส่หรือนำออกในหอต่างๆได้ การย้ายสิ่งของกำหนดให้ย้ายได้ครั้งละ 1 ชิ้น เท่านั้น การย้ายสิ่งของนั้นวิธีการขึ้นอยู่กับจำนวนชิ้นที่มีอยู่ในหอ

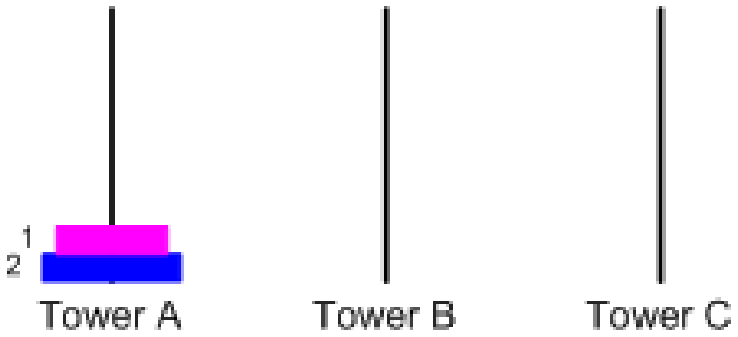
กรณีที่ 1 มีสิ่งของ 1 ชิ้นที่หอ A ต้องการย้ายไปที่ หอ B



เราสามารถนำสิ่งของจากหอ A ย้ายไปที่หอ B ได้ทันที

```
n = 1;
cout <<"Move 1 from A to B";
```

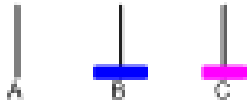
กรณีที่ 2 มีสิ่งของ 2 ชิ้น ขนาดไม่เท่ากัน ต้องการย้ายจากหอ A ไปที่ หอ B



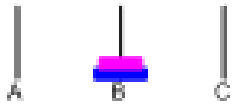
การย้ายสิ่งของไม่สามารถกระทำได้โดยตรงเพราะขนาดของสิ่งของไม่เท่ากัน และข้อกำหนด ตกลงไว้ว่าสิ่งของชิ้นเล็กต้องอยู่บนชิ้นใหญ่เสมอ ดังนั้นเราต้องกำหนดหอ C มาช่วยในการย้าย และมีกระบวนการย้ายแบ่งเป็น 3 ขั้นตอนดังนี้

Move 1 from A to C

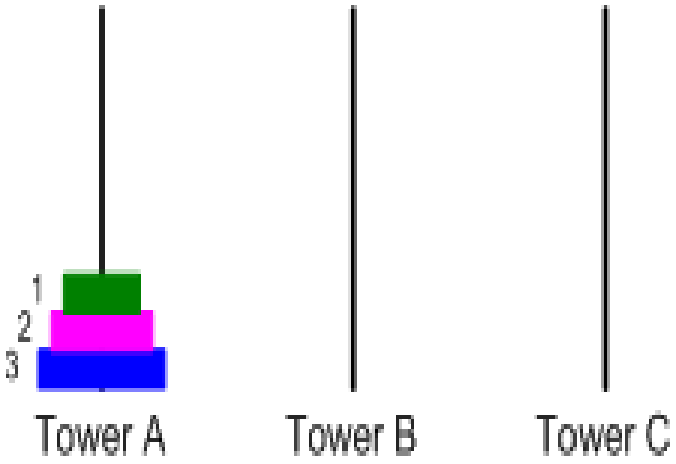
Move 2 from A to B



Move 1 from C to B



กรณีที่ 3 มีสิ่งของ 3 ชิ้นขนาดไม่เท่ากัน ต้องการย้ายจากหอ A ไปไว้ที่หอ B



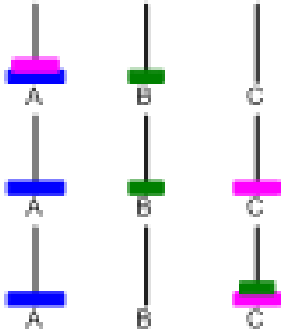
เราจะแบ่งขั้นตอนในการย้ายสิ่งของออกเป็น 3 ขั้นตอนใหญ่แต่เป็น 7 ขั้นตอนย่อยด้วยกัน

Move 1 from A to B

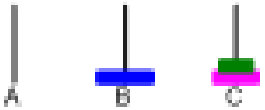
Move 2 from A to C

Move 1 from B to C

Move 2 from A to C



Move 3 from A to B

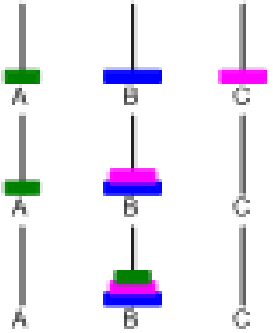


Move 1 from C to A

Move 2 from C to B

Move 1 from A to B

Move 2 from C to B



```
void tower (char from, char too, char temp, int n)
{
        if (n = = 1)
                cout << "Move Object 1 from tower"  << from
                        << "to tower" << too << endl;
        else
        {
                tower (from, temp, too, n – 1);
                cout <<"Move Object" <<n <<"from tower" <<from
                        <<"to tower" <<too <<endl;
                tower (temp, too, from, n – 1);
        }
}
```

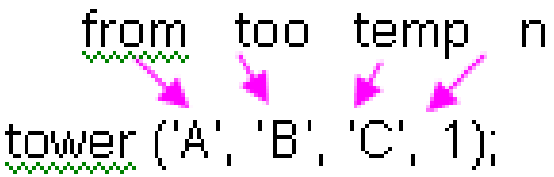
ในที่นี้ n คือจำนวนสิ่งของที่ต้องการย้าย

from คือหอที่มีสิ่งของอยู่จำนวน n ชิ้นโดยชิ้นเล็กซ้อนทับชิ้นใหญ่

too คือหอที่ต้องการย้ายสิ่งของไปเก็บไว้

temp คือหอที่ใช้สำหรับรับฝากสินค้าไว้เพื่อขนย้าย

ในกรณีที่ n = 1



ผล Move Object 1 from tower A to tower B

ในกรณีที่ n = 2

tower ('A', 'B', 'C', 2);

จะกระทำ 3 คำสั่งดังนี้

1. tower (from, temp, too, n – 1); ①

   tower ('A', 'C', 'B', 1);

   ผล Move Object 1 from tower A to tower C

2. พิมพ์ ②

   tower ('A', 'B', 'C', 2);

   ผล Move Object 2 from tower A to tower B

3. tower ('C', 'B', 'A', 1); ③

   ผล Move Object 1 from tower C to tower B

ในกรณีที่ n = 3

```
tower ('A', 'B', 'C', 3);
```

จะกระทำ 3 ขั้นตอนหลักเหมือนกัน แต่มีการเรียกตัวเองเพื่อปฏิบัติงานในลักษณะซ้ำกัน ผลของการทำงาน จะปฏิบัติงานทั้งหมด 7 ขั้นตอนด้วยกัน ดังนี้

ผลของการทำงาน

Move disk 1 from tower A to tower C

Move disk 2 from tower A to tower B

Move disk 1 from tower C to tower B

Move disk 3 from tower A to tower C

Move disk 1 from tower B to tower A

Move disk 2 from tower B to tower C

Move disk 1 from tower A to tower C

ตัวอย่างโปรแกรม reverse

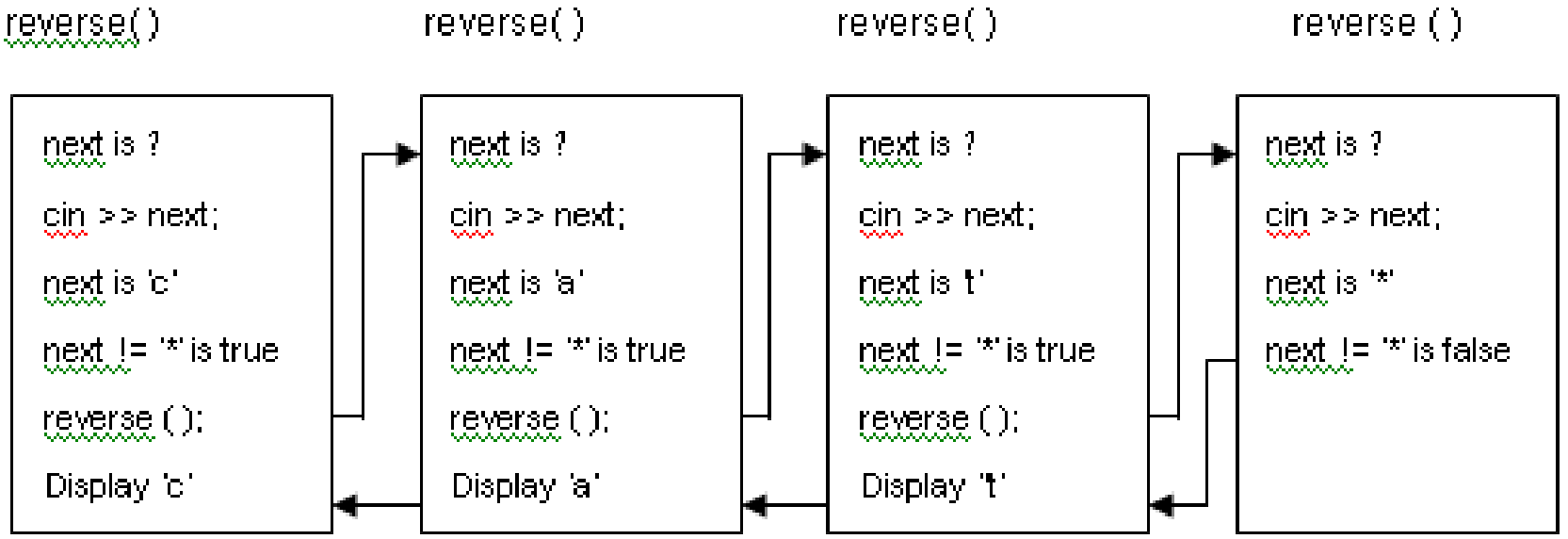
```
#include<iostream>
using namespace std;
// Function prototype
void reverse( );
int main ( )
{
    reverse( );
   cout << endl;
   return 0;
}
```

```
void reverse( )
{
    char next;
    cout << “Next character
                or * to stop: ”;
    cin >> next;
    if (next  !=  ‘*’ )
    {
          reverse ( );
           cout << next;
    }
}
```

เมื่อนำโปรแกรมไป run จะมีการแสดงหน้าจอเพื่อให้ผู้ใช้ป้อนข้อมูล ดังนี้

```
Next character or * to stop:    c
Next character or * to stop:    a
Next character or * to stop:    t
Next character or * to stop:    *
tac
```

โดยผู้ใช้จะป้อนข้อมูลครั้งละ 1 ตัวอักษรไปเรื่อยๆ ต้องการหยุดป้อนโดยกดแป้นเครื่องหมาย * การทำงานจะพิมพ์ข้อความย้อนกลับจากตัวสุดท้ายจนถึงตัวแรก



ตัวอย่าง โปรแกรมหาค่าหารร่วมมาก

```
#include<iostream>
using namespace std;
int gcd(int, int);
int main( )
{
    int m, n;
    cout << "Enter two positive
                integer: " ;
    cin >> m >> n;
    cout << endl << "Their
          greatest common
          divisor is "
          << gcd(m, n) << endl;
return 0;
}
```

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

เมื่อนำโปรแกรมไปทำงานจะได้ผลลัพธ์ดังนี้

```
Enter two positive integer: 24 84
Their greatest common divisor is
    12
```

ตัวอย่าง ฟังก์ชันหาผลรวมของข้อมูล

ฟังก์ชันต่อไปนี้เป็นการหาผลรวมของเลข 1+2+3+..+10 โดยสมมุติว่าข้อมูลมีการเก็บในตัวแปรอาเรย์ 1 มิติชื่อ x ดังนี้

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

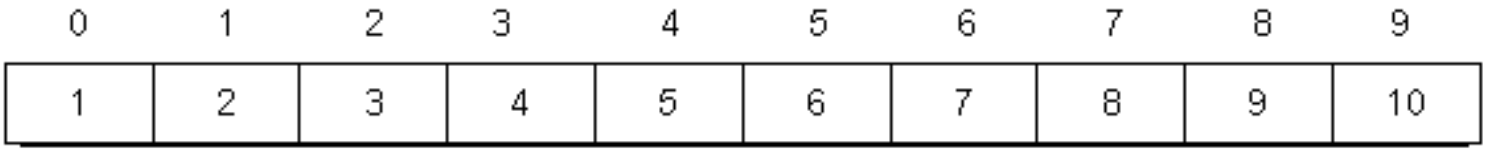
การหาค่าผลรวมเราจะสร้างฟังก์ชันชื่อ findSum โดยส่งผ่านค่าของกลุ่มข้อมูล x และขนาดของข้อมูล ไปยังฟังก์ชันเพื่อส่งค่าผลรวมของข้อมูลทั้งหมดกลับมายังจุดเรียกใช้ ซึ่งการทำงานจะเป็นไปในลักษณะเรียกตัวเอง

ตัวอย่าง ฟังก์ชันหาผลรวมของข้อมูล

```
// Finds the sum of integers in an n-element array
int findSum(int x[ ], int n)
{
    if (n == 1)
        return x[0];                          // stopping case
    else
        return x[n-1] + findSum(x, n-1);      // recursive step
}
```

ผลของการเรียกใช้จะเป็นดังนี้

```
Recursive sum is 55
Calculated sum is  55
```



การบ้าน 1

พิจารณาฟังก์ชันต่อไปนี้

```
double    test5(int a , int b)
        {
                double sum = 1.0 ;
                for (int i = 3 ; i <  b ; i++)
                {
                        sum = sum / (a*a) ;
                }
}
```

จงเขียนฟังก์ชันข้างต้นนี้ใหม่แบบเรียกตัวเอง

ถ้ามีการเรียกใช้คำสั่ง `cout<<test5(2,6)<<endl;`

ผลของการทำงานจะพิมพ์ค่า.....ออกทางจอภาพ

การบ้าน 2

จงเขียนฟังก์ชันเรียกตัวเองชื่อ SUM เพื่อให้สามารถทำงานได้ดังการทำงานต่อไปนี้

$$1 + 2/3 + 3/5 + 4/7 + \ldots n/(n+2)$$

ถ้าผู้ใช้เรียกใช้

SUM(1) ผลของการทำงานคือ 1

SUM(2) ผลของการทำงานคือ 1 + (2/3)

SUM(3) ผลของการทำงานคือ 1 + (2/3) + (3/5)

SUM(n) ผลของการทำงานคือ 1 + (2/3) + ... + n/(n+2)

การบ้าน 3

อาจารย์อุไรต้องการทราบว่าถ้าฝากเงินกำหนดเงินต้น A บาท
เป็นจำนวน Y ปี โดยอัตราดอกเบี้ยร้อยละ K
สมมุติว่าอัตราดอกเบี้ยเท่ากันทุกปี โดยฝากทบต้นจนกระทั่งครบกำหนดจะได้
เงินทั้งหมดเป็นเงินเท่าใด ท่านเขียนฟังก์ชันเรียกตัวเองให้อาจารย์หน่อย
สมมุติว่าเรียกใช้ดังนี้

```
cout<<FindMoney(1000, 4, 4 );
```

ผลการทำงานเป็นอย่างไร