Chapter 1

Function & Recursion

ฟังก์ชั่นคือส่วนของโปรแกรมที่จะทำงานก็ต่อเมื่อถูกเรียกใช้งานเท่านั้น ฟังก์ชั่นมักจะถูกใช้เป็นการทำงานเฉพาะๆ หรืองานส่วนหนึ่งๆ ซึ่งมีโอกาสถูกเรียกใช้ซ้ำ ฟังก์ชั่นจึงสามารถเข้ามาช่วยแก้ปัญหานี้ได้ โดยการสร้างฟังก์ชั่นขึ้นมาแล้วจึงเรียกใช้ฟังก์ชั่นนี้ซ้ำนั่นเอง

1.1 Function Creation

การสร้างฟังก์ชั่นสามารถทำได้โดยการระบุประเภทฟังก์ชั่น, ชื่อฟังก์ชั่น และการทำงานของฟังก์ชั่นนั้นๆ

```
type functionName() {
  //a block of code to be executed when function is called
}
```

ประเภทของฟังก์ชั่นใช้สำหรับบอกว่าฟังก์ชั่นนี้จะคืนค่าประเภทใดออกมาหลังจากจบการทำงานของฟังก์ชั่นนี้แล้ว โดยจะเหมือนกันกับประเภทของตัวแปร เพียงแต่มีเพิ่มเติมมา 1 ชนิดที่ควรต้องทราบ คือ **void** ซึ่งเป็นประเภทของฟังก์ชั่นที่ไม่มีการคืนค่าใดๆ กลับมา

```
void showHelloToTheWorld() {
  printf("Hello, world.\n");
    // no need to return anything
}
int currentYear() {
  return 2022;
    // return some integer
}
```

นอกจากนี้ ฟังก์ชั่นสามารถรับค่าพารามิเตอร์ (เปรียบเสมือนข้อมูลนำเข้าสำหรับฟังก์ชั่น) ได้โดยการใส่ประเภทตัวแปร และชื่อตัวแปรลงไปใน ( ) หลังชื่อฟังก์ชั่น

```
int sum (int a, int b) {
  return a + b;
}
```

การส่งค่าพารามิเตอร์เข้าไปยังฟังก์ชั่นต่างๆ นั้น เสมือนเป็นการสร้างตัวแปรใหม่ขึ้นมาที่มีค่าเหมือนเดิมทุกประการ เพียงแต่การเปลี่ยนแปลงตัวแปรนั้นภายในฟังก์ชั่นจะไม่ส่งผลกระทบต่อค่าเดิมของตัวแปรที่เป็นต้นแบบในการส่งเข้าไป

```
void add(int a, int b){
   a += b;
}
int main () {
   int a = 10, b = 5;
      add(a, b);
      printf("%d\n", a);
}
```

โปรแกรมด้านบนนี้เป็นโปรแกรมเพื่อบวกค่าของตัวแปร b เข้าไปในตัวแปร a แต่หากลองให้โปรแกรมนี้ทำงานแล้วผลลัพธ์จะออกมาว่าค่าของตัวแปร a ยังมีค่าเป็น 10 เหมือนเดิมไม่มีการเปลี่ยนแปลงใดๆ เพราะการบวกค่าตัวแปร a ภายในฟังก์ชั่น `add` เป็นตัวแปรคนละตัวกับตัวแปร a ภายในฟังก์ชั่น `main` นั่นเอง

เราจึงต้องใช้การ pass by reference เข้ามาช่วย ซึ่งจะเป็นการส่งตัวแปรเข้าไปจริงๆ ไม่ใช่การสร้างตัวแปรใหม่ที่มีค่าเหมือนกัน

```
void add(int &a, int &b){
   a += b;
}
int main() {
   int a = 10, b = 5;
      add(a, b);
      printf("%d\n", a);
}
```

ผลลัพธ์จากการทำงานของโปรแกรมด้านบนนี้จะได้ค่าของตัวแปร a ออกมาเป็น 15

1.2 Recursion

การเรียกซ้ำ คือการให้ฟังก์ชั่นเรียกใช้ตัวเอง เพื่อทำการย่อยปัญหาให้เล็กลง แล้วแก้ปัญหาย่อยนั้นก่อนที่จะส่งผลลัพธ์จากการแก้ปัญหาย่อยนั้นกลับไปเพื่อแก้ปัญหาที่ใหญ่ขึ้น โดยจะเรียกฟังก์ชั่นเหล่านี้ว่า ฟังก์ชั่นเวียนบังเกิด (Recursive function)

Recursive function อาจจะดูเป็นเรื่องยากที่จะเข้าใจ แต่หากลองฝึกทำโจทย์บ่อยๆ จะเข้าใจได้มากยิ่งขึ้น และนอกจากนี้ recursive function ยังเป็นพื้นฐานสำหรับเรื่องต่างๆ อีกมากมายที่เราจะได้เรียนภายในค่าย

```
int sum(int k) {
   if (k == 0) {
        return 0;
      }
      return k + sum(k - 1);
}
```

ฟังก์ชั่นด้านบนนี้เป็น recursive function เพื่อหาผลรวมของจำนวนตั้งแต่ 0 ถึง k เช่น หากเมื่อเรียกใช้ **sum(5)** สิ่งที่เกิดขึ้นจะเป็นขั้นตอน ดังนี้

- sum(5)
- 5 + sum(4)
- 5 + (4 + sum(3))
- 5 + (4 + (3 + sum(2)))
- 5 + (4 + (3 + (2 + sum(1))))
- 5 + (4 + (3 + (2 + (1 + sum(0)))))
- 5 + (4 + (3 + (2 + (1 + 0))))

จะเห็นได้ว่า ฟังก์ชั่น `sum` ได้ทำการเรียกตัวเองซ้ำ แต่มีการเปลี่ยนแปลงค่าพารามิเตอร์ที่ลดน้อยลงเรื่อยๆ จน `k = 0` จึงไม่มีการเรียกตัวเองซ้ำอีกแล้ว

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

ฟังก์ชั่นด้านบนนี้เป็นฟังก์ชั่นหาค่า fibonacci ลำดับที่ k เช่น เมื่อต้องการหาค่า fibonacci ลำดับที่ 4 จะมีขั้นตอนดังนี้

- **fibonacci(4)**
- **fibonacci(2)** + fibonacci(3);
- (**fibonacci(0)** + fibonacci(1)) + fibonacci(3)
- (0 + **fibonacci(1)**) + fibonacci(3)
- (0 + 1) + **fibonacci(3)**
- (0 + 1) + (**fibonacci(1)** + fibonacci(2))
- (0 + 1) + (1 + **fibonacci(2)**)
- (0 + 1) + (1 + (**fibonacci(0)** + fibonacci(1)))
- (0 + 1) + (1 + (0 + **fibonacci(1)**))
- (0 + 1) + (1 + (0 + 1))

Recursive function ที่ยกมาข้างต้นเป็นตัวอย่างพื้นฐานเท่านั้น ยังมีอีกมากมายที่ซับซ้อนมากกว่านี้ เช่น โปรแกรมการเรียงสับเปลี่ยน และการจัดหมู่เป็นต้น

```
1       void permute(int state){
2         if (state == n){
3              for (int i = 0; i < n; i++) {
4                   printf("%d ",currentPermutation[i]);
5                }
6                printf("\n");
7              return ;
8           }
9           for (int i = 1; i <= n; i++) {
10            if (!isused[i]) {
11                 isused[i] = true;
12                   currentPermutation[state] = i;
13                   permute(state + 1);
14                   isused[i] = false;
15               }
16          }
17      }
18
```

โปรแกรมข้างต้นเป็นโปรแกรมแสดงผลการเรียงสับเปลี่ยนของตัวเลขตั้งแต่ 1 จนถึง n ออกมา บรรทัดละ 1 ลำดับ

```
1       void permute(int state, int lastUsed){
2         if (state == n){
3              for (int i = 0; i < n; i++) {
4                   printf("%d ",currentPermutation[i]);
5                }
6                printf("\n");
7              return ;
8           }
9           for (int i = lastUsed + 1; i <= n; i++) {
10            if (!isused[i]) {
11                 isused[i] = true;
12                   currentPermutation[state] = i;
13                   permute(state + 1, i);
14                   isused[i] = false;
15               }
16          }
17      }
18
```