สัปดาห์ที่ 4 Functions and Program Structure - ตอนที่ 1

เรียบเรียงโดย ชาคริต วัชโรภาส

วิชา 01418113 Computer Programming

1. Function Definition (การนิยามฟังก์ชัน)

- รูปแบบของการนิยามฟังก์ชัน

```
return-value-type function-name(parameter-list)
{
   declarations

   statement
   statement
   :
}
```

- Function header and Function body
- คอมไพเลอร์ภาษา C เป็นคอมไพเลอร์ One-pass

---

2. Function Prototypes

- รูปแบบของฟังก์ชัน prototype

```
return-value-type function-name(parameter-list);
```

- มีลักษณะเหมือน function header ของ function definition
- Parameter-list สามารถละชื่อตัวแปรได้ แต่ไม่สามารถละประเภทข้อมูลได้
- **คำถาม** ทำไมเราต้องมีการใช้ function prototypes

---

3. Standard Library Functions and Math Library Functions

- ฟังก์ชัน printf, scanf และ getchar เป็นตัวอย่างของฟังก์ชันที่อยู่ใน Standard Library functions
- ฟังก์ชัน sqrt, exp และ log เป็นตัวอย่างของฟังก์ชันที่อยู่ใน Math Library functions
- หากโปรแกรมของเรามีการใช้ฟังก์ชันเหล่านี้ เวลาคอมไพล์โปรแกรม เราจำเป็นต้องคอมไพล์ (link) โปรแกรมของเราเข้ารวมกับโค้ดไบนารีของฟังก์ชันเหล่านี้ (gcc มีโค้ดไบนารีของฟังก์ชันกลุ่มนี้อยู่ในไลบรารีที่ถูกติดตั้งไว้แล้วตอนที่เรารัน pacman -S gcc-libs)
- โค้ดไบนารีของ Standard C Library functions ถูกเก็บอยู่ใน /usr/lib/libc.a
- โค้ดไบนารีของ Math Library functions ถูกเก็บอยู่ใน /usr/lib/libm.a
- ลองใช้คำสั่ง nm /usr/lib/libc.a | grep "T _"

- ปกติ gcc จะ link ฟังก์ชันใน Standard Library ให้เราเองโดยอัตโนมัติ โดยที่เราไม่ต้องระบุ option เพิ่มเติม ตอนออกคำสั่ง gcc เพื่อคอมไพล์
- แต่ฟังก์ชันใน math library จะไม่ได้ถูก link ในอัตโนมัติ โดยเราต้องระบุ -lm เพิ่มเข้าไปตอนเรียกใช้ gcc
- หากเราต้องการใช้ฟังก์ชันใน Math Library เราควรต้อง #include <math.h> ด้วย
- ตัวอย่างฟังก์ชันใน Math Library ได้แก่ sin(), cos(), sqrt(), log(), pow(), exp(), floor(), ceil(), fmod()

หมายเหตุ: MSYS2 (MinGW) จัดการไฟล์บน Windows แตกต่างจาก standard GCC

- ไฟล์ header ถูกเก็บอยู่ภายใต้ **/mingw64/x86_64-w64-mingw32/include**
- โค้ดไบนารีของ Standard C Library functions ถูกเก็บอยู่ใน **/mingw64/x86_64-w64-mingw32/lib/libmsvcrt.a**

In [1]:

```c
//%cflags: -lm

#include <stdio.h>
#include <math.h>
int main()
{  float c, d, f;

   c = 13.0;
   d = 3.0;
   f = 4.0;

   printf("         c + d * f = %.2f\n", c + d * f);
   printf("sqrt ของ c + d * f = %.2f\n", sqrt(c + d * f));
}
```

```
         c + d * f = 25.00
sqrt ของ c + d * f = 5.00
```

sqrt → ต้องมีนะ

4. Return Values

- ฟังก์ชันจะมีการส่งค่ากลับ หรือไม่ก็เป็นฟังก์ชันที่มี return type เป็น void
- เราใช้ void เพื่อบอกคอมไพเลอร์ว่าฟังก์ชันนี้ไม่มีการส่งค่าใดๆ กลับไปยังผู้เรียก
- หากเป็นฟังก์ชันที่มีการส่งค่ากลับ เราใช้คีย์เวิร์ด *return* เพื่อส่งกลับค่าข้อมูล
- ต.ย.

```
return 5;
return (x > 5);
return (myFunction());
```

- ตัวอย่างที่ 1

```
#include <stdio.h>

int isEven(int val)
{
   if (val % 2 == 0)
      return 1;
   else
      return 0;
}

int main()
{  int value;

   printf("Enter an Integer Value: ");
   scanf("%d", &value);

   if (isEven(value) == 1)
      printf("%d is an even number\n", value);
   else
      printf("%d is not an even number\n", value);
}
```

- **เมื่อโปรแกรมทำงานจนไปเจอคำสั่ง return ที่อยู่ในฟังก์ชัน ลำดับการทำงานของโปรแกรมจะกลับไปทำงานต่อจากจุดที่ฟังก์ชันนั้นถูกเรียกใช้**

- **ตัวอย่างที่ 2**

```c
#include <stdio.h>

int isOdd(int val)
{
   return val % 2;
}

int isEven(int val)
{
   return !isOdd(val);
}

int main()
{  int value;

   printf("Enter an Integer Value: ");
   scanf("%d", &value);

   if (isEven(value))
      printf("%d is an even number\n", value);
   else
      printf("%d is not an even number\n", value);
}
```

---

5. Using Functions as Parameters to Functions

- ค่าที่ได้จากการเรียกใช้ฟังก์ชันสามารถส่งผ่านไปเป็นพารามิเตอร์ของการเรียกใช้อีกฟังก์ชันได้

In [2]:

```c
#include <stdio.h>

long triple(long val)
{
   return 3 * val;
}

long square(long val)
{
   return val * val;
}

long cube(long val)
{
   return val * val * val;
}

int main()
{  long answer, my_value;

   my_value = 2l;
   answer = triple( square( cube( my_value) ) );

   printf("Answer = %ld\n", answer);
}
```

```
Answer = 192
```

6. Recursion (การเรียกซ้ำ)

- ฟังก์ชันสามารถเรียกใช้ตนเองได้ ซึ่งเรียกว่า recursion
- การเรียกตนเองสามารถอยู่ในรูป direct และ indirect
- สิ่งที่ควรรู้คือ เมื่อฟังก์ชันมีการเรียกใช้ตนเอง โปรแกรมจะสำเนาค่าตัวแปรต่างๆ ภายในฟังก์ชัน (เช่น ตัวแปร local) ขึ้นมาอีกชุด โดยเป็นอิสระจากการถูกเรียกในครั้งก่อนหน้า
- ต.ย.
  - Factorial ของเลขจำนวนเต็มบวก n ซึ่งปกติเขียนอยู่ในรูป n! (อ่านว่า "n factorial") เป็นผลคูณของ n * (n-1) * (n-2) * ... * 1
  - ในทางคณิตศาสตร์ เรานิยาม

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

In [3]:

```c
#include <stdio.h>

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

int main()
{  long n;

   n = 6;
   printf("%ld! = %ld\n", n, factorial(n));
}
```

```
6! = 720
```

ตัวอย่าง Fibonacci Series

- 1, 1, 2, 3, 5, 8, 13, 21, 34 ...
- Fibonacci Series เป็นอนุกรมที่เลขสองจำนวนแรกมีค่าเป็น 1 แล้วค่าของเลขถัดไปจะมีค่าเป็นผลรวมของสองจำนวนก่อนหน้า
- **ลองเขียนโปรแกรมเพื่อพิมพ์อนุกรมนี้ออกมา**
- ให้เขียนทั้งโปรแกรมที่มีการ*เรียกซ้ำ* และอีกโปรแกรมเป็นการ*วนซ้ำ*

- sum.c, triangle.c, hi5.c

6.1 Local Variables

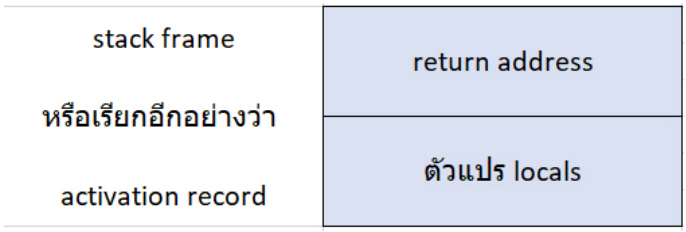
- ตัวแปรที่ถูกประกาศไว้อยู่ภายใน *block*, ภายในตัวฟังก์ชัน รวมถึงตัวแปรพารามิเตอร์ของฟังก์ชันจะเป็นตัวแปร local ซึ่งสามารถถูกใช้งานได้เพียงภายในฟังก์ชันที่ถูกประกาศไว้

---

7. Function Call และ Stack Frames

- เพื่อให้โปรแกรมสามารถจัดเก็บตัวแปรเป็นจำนวนมากในขณะที่โปรแกรมทำงานได้ ฟังก์ชันจะจัดทำ stack frame เพื่อจัดเก็บข้อมูลของตัวแปร parameter และตัวแปร local ที่อยู่ภายในฟังก์ชัน
- เมื่อฟังก์ชันถูกเรียกใช้งาน stack frame จะถูกสร้างขึ้น (1 stack frame ต่อการเรียกใช้ฟังก์ชัน 1 ครั้ง)
- return address เป็นตำแหน่งของคำสั่งที่จะถูกทำงานหลังจากที่ฟังก์ชันที่ถูกเรียกทำงานเสร็จ
- เมื่อฟังก์ชันทำงานเสร็จ (สิ้นสุดฟังก์ชันหรือเจอคำสั่ง return) stack frame จะถูกทำลาย โดยที่ลำดับการทำงานของโปรแกรมจะกลับไปทำคำสั่ง ณ ตำแหน่ง return address

7.1 ส่วนประกอบหลักของ Stack Frames



7.2 ลองไล่โค้ดเพื่อหาผลลัพธ์จากโค้ดข้างล่างนี้

- ตัวอย่างที่ 1 [Stack Frame Visualization (https://drive.google.com/file/d/14Egjik7m7HN3aV9O7mdGeqyKfgd-iDge/view?usp=sharing)](https://drive.google.com/file/d/14Egjik7m7HN3aV9O7mdGeqyKfgd-iDge/view?usp=sharing)

```
int bar(int a, int b)
{  int x, y;

   x = 444;
   y = a + b;
   return y;
}

int main()
{  int result;

   result = 1;
   result = bar(111, 222);
   printf("%d\n", result);
}
```

- ตัวอย่างที่ 2

```
int foo(int x)
{
   if (x > 1)
      return x % 8 + 10 * foo(x / 8);
   else
      return x;
}

int main()
{
   printf("%d\n", foo(88));
   return 0;
}
```

- ตัวอย่างที่ 3

```
void bar(int a, int b)
{
   if (b%2 == a && b >= 2) {
      printf("hi");
      bar(1-a, b-4);
   }
   else if (b > 0) {
      printf("ha");
      bar(a, b-1);
   }
}

int main()
{
   bar(1, 10);
   return 0;
}
```