

2110104: Computer Programming

Function



Jessada Thutkawkorapin

DEPT. OF COMPUTER ENGINEERING
CHULALONGKORN UNIVERSITY

ทบทวนเนื้อหา

- ฟังก์ชัน
 - sin, log, setprecision, round, max
- เราเรียนรู้ที่จะใช้งานฟังก์ชัน
 - เรียนรู้ว่าฟังก์ชันจะรับค่าอะไรบ้าง
 - เรียนรู้ว่าเราสามารถคาดหวังผลจากฟังก์ชันได้
 - เรียนรู้เกี่ยวกับการคืนค่า

ทบทวนเนื้อหา (ต่อ)

- ตัวอย่าง: `setprecision(15)`
 - ฟังก์ชันนี้รับค่าหนึ่งค่า 
 - ค่าที่รับเป็นจำนวนเต็ม
 - ฟังก์ชันนี้จะทำการแสดงผลเลขทศนิยมเป็นไปตามที่เรากำหนด
 - ไม่มีการคืนค่า (เราจึงไม่เขียน `a = setprecision(15);`)
- ตัวอย่าง: `max(4,7)`
 - ฟังก์ชันนี้รับค่าสองค่า 
 - ค่าที่รับจะเป็นตัวเลข
 - ฟังก์ชันนี้จะคืนค่าที่เป็นค่าที่มากที่สุดในค่าสองค่า

ฟังก์ชันคืออะไร

คือชุดของโค้ดที่ถูกเขียนขึ้นมา โดยจะมีคุณสมบัติดังนี้

- จะทำงานก็ต่อเมื่อถูกเรียกใช้งานเท่านั้น
- สามารถเรียกใช้งานได้หลายครั้ง
- สามารถเรียกใช้ได้หลายที่
- มีหน้าที่ในตัวเองอย่างเด่นชัด
- ทำให้โปรแกรมอ่านง่ายขึ้น

องค์ประกอบของฟังก์ชันที่เราจะสร้างเอง

กำหนดชนิดตัวแปร
หากต้องการคืนค่า
void ในที่นี้คือไม่คืนค่า

ชื่อฟังก์ชัน

ส่วนที่ใช้กำหนดว่าฟังก์ชันจะรับค่าอะไรบ้าง
หากไม่สามารถกำหนดเป็น **()** ได้

```
void func_name (int var_name) {  
    // function body (code)  
}
```

ส่วนเนื้อหาของฟังก์ชันจะอยู่ภายใน { }

ตัวอย่างฟังก์ชัน

```
1      2      3  
int hello(string name, string surname) {  
4      cout << "Hello " << name << " " << surname << endl;  
5      return 0;  
}
```

- (1) ฟังก์ชันนี้จะคืนค่าเป็น **int**
- (2) ฟังก์ชันนี้ชื่อ **hello**
- (3) ฟังก์ชันนี้มีพารามิเตอร์ 2 ตัวคือ **name** และ **surname** เป็นข้อมูลประเภท **string** ทั้งคู่
- (4) พอถูกเรียกใช้จะแสดง **Hello** ตามด้วยค่าของ **name** ตามด้วยช่องว่าง แล้วตามด้วยค่าของ **surname**
- (5) คืนค่า **0** ซึ่งเป็น **int** ตามที่ประกาศไว้ที่ (1) และย้อนกลับไปทำงานต่อจากบรรทัดที่เรียก **hello**

ลำดับการทำงาน การคืนค่าและการรับค่า

```
# include <iostream>
using namespace std;
float get_mean(float x1, float x2, float x3) {
    4 return (x1+x2+x3)/3;
}
int main() {
    5 cout << get_mean(2,5,6) << endl;
}
```

Diagram illustrating the flow of execution and data flow in the provided C++ code:

- 1. Call `get_mean` (in `main`)
- 2. Pass parameters (2, 5, 6) to `get_mean`
- 3. Calculate the mean: $(2+5+6)/3$
- 4. Return the result from `get_mean`
- 5. Display the result using `cout`


จะแสดงผล

4.33333

1. เรียก `get_mean`
2. ส่งค่าไปยัง parameter
3. คำนวณค่าเฉลี่ย ในที่นี้คือ $(2+5+6)/3$
4. คืนค่าผ่านคำสั่ง `return`
5. แสดงผ่านคำสั่ง `cout` ได้ 4.33333

ลำดับการทำงาน(อีกตัวอย่าง)

```
# include <iostream>
using namespace std;
void hello(string name) {
    cout << "Hello " << name << endl;
}
int main() {
    hello( "Jenny" );
    hello( "Jisoo" );
    hello( "Rose" );
    hello( "Lisa" );
}
```

A diagram with four dashed orange arrows originates from the 'name' parameter in the 'cout' statement of the 'hello' function. Each arrow points to one of the four 'hello' function calls in the 'main' function: 'hello("Jenny")', 'hello("Jisoo")', 'hello("Rose")', and 'hello("Lisa")'. This illustrates how the function is called multiple times with different arguments.

จะแสดงผล

```
Hello Jenny
Hello Jisoo
Hello Rose
Hello Lisa
```

1. ค่า "Jenny" ถูกส่งไปให้ตัวแปร name -> ทำการแสดงผล
2. แล้วกลับมาเรียก hello("Jisoo") -> ส่งค่า "Jisoo" ให้ตัวแปร name -> ทำการแสดงผล
3. แล้วกลับมาเรียก hello("Rose") -> ซ้ำไปเรื่อยๆ

ลองเขียนโค้ดเล่นๆ

```
#include <iostream>
using namespace std;
int to_sec(int h, int m, int s) {
    //fill in your code here
}
int main() {
    int h1,m1,s1, h2,m2,s2;
    cin >> h1 >> m1 >> s1;
    cin >> h2 >> m2 >> s2;
    cout << to_sec(h1, m1, s1) << endl;
    cout << to_sec(h2, m2, s2) << endl;
}
```

← เติมโค้ดตรงนี้

ถ้าทดสอบด้วย

```
2 10 20
4 0 0
```

จะแสดงผล

```
7820
14400
```

ลองไล่โค้ด (เฉลยอยู่หน้าถัดไป ห้ามแอบดู)

```
# include <iostream>
using namespace std;

void f1(int a) {
    int b = 3;
    cout << "In f1, a b: " << a << " " << b << endl;
}

int main() {
    int b = 4;
    f1(7);
    cout << "After f1, b: " << b << endl;
}
```

จะแสดงผล

In f1, a b: **x y**
After f1, b: **z**

x = ?
y = ?
z = ?

ตัวแปรภายในฟังก์ชันใดเป็นของฟังก์ชันนั้น

```
# include <iostream>
using namespace std;

void f1(int a) {
    int b = 3;
    cout << "In f1, a b: " << a << " " << b << endl;
}

int main() {
    int b = 4;
    f1(7);
    cout << "After f1, b: " << b << endl;
}
```

การแก้ไขตัวแปร b ใน f1 ไม่มีผลต่อค่า b ใน main

จะแสดงผล

In f1, a b: 7 3
After f1, b: 4

ฟังก์ชันหนึ่งเรียกอีกฟังก์ชันหนึ่งได้

```
# include <iostream>
using namespace std;

void hello(string name){
    cout << "Hello " << name << endl;
}

void hello_all(){
    hello("Jenny");
    hello("Jisoo");
    hello("Rose");
    hello("Lisa");
}

int main() {
    hello_all();
}
```

จะแสดงผล

```
Hello Jenny
Hello Jisoo
Hello Rose
Hello Lisa
```


ลองเขียนโค้ดเล่นๆ

```
#include <iostream>
using namespace std;
int to_sec(int h, int m, int s) {
    // fill the code you did from earlier ← แก้โค้ดตรงนี้
}
int diff(int h1, int m1, int s1, int h2, int m2, int s2) {
    // return the difference in seconds
    //      between h1:m1:s1 and h2:m2:s2
    // try to solve this by calling to_sec
    // fill the code here ← แก้โค้ดตรงนี้
}
int main() {
    int h1,m1,s1, h2,m2,s2;
    cin >> h1 >> m1 >> s1;
    cin >> h2 >> m2 >> s2;
    cout << diff(h1, m1, s1, h2, m2, s2) << endl;
}
```

ถ้าทดสอบด้วย

```
2 10 20
4 0 0
```

จะแสดงผล

```
6580
```

การเสร็จการทำงานของฟังก์ชัน

End of
function

```
int func1() {  
    std::cout << "part 1";  
}
```

หากถูกเรียกจะแสดงผล

part 1

เมื่อโค้ดทำงานมาถึง **}** ก็จะเป็นการสิ้นสุดการทำงานของฟังก์ชันโดยอัตโนมัติและ
ย้อนกลับไปหาฟังก์ชันที่เรียกใช้ func1

การเสร็จการทำงานของฟังก์ชัน

End of
function

```
int func2() {  
    std::cout << "part 2";  
    return 1;  
    std::cout << "part 3";  
}
```

หากถูกเรียกจะแสดงผล

part 2

เมื่อโค้ดทำงานมาถึง **return 1;** ฟังก์ชันก็จะคืนค่า 1 และย้อนกลับไปทำงานต่อจากบรรทัดที่เรียกใช้งาน func2

การ break ออกจาก loop หลายๆชั้น

```
...  
for (...) {  
    ...  
    for (...) {  
        ...  
        if (condition) {  
            ...  
            break;  
        }  
    }  
    ...  
}
```

break จะกระโดดออกมา
จาก loop ที่ break นั้นอยู่

```
...  
for (...) {  
    ...  
    for (...) {  
        ...  
        if (condition) {  
            ...  
            อยากออกไปนอกสุด;  
        }  
    }  
    ...  
}
```

ถ้าต้องการให้ break กระโดด
ออกมามวงนอก ๆ จะทำอย่างไร?

การ break ออกไปหลายๆชั้นด้วยตัวแปรเสริม



```
...  
to_outer = false;  
for (...) {  
    ...  
    for (...) {  
        ...  
        if (condition) {  
            ...  
            to_outer = true;  
            break;  
        }  
    }  
    if (to_outer) break;  
    ...  
}  
...
```

การ break ออกไปหลายๆชั้นด้วยการแยก ออกเป็นฟังก์ชัน

```
...  
for (...) {  
    ...  
    for (...) {  
        ...  
        if (condition) {  
            ...  
            อยากออกไปนอกสุด;  
        }  
    }  
    ...  
}
```



```
void func() {  
    for (...) {  
        ...  
        for (...) {  
            ...  
            if (condition) {  
                ...  
                return;  
            }  
        }  
        ...  
    }  
}  
  
int main() {  
    ...  
    func();  
    ...  
}
```