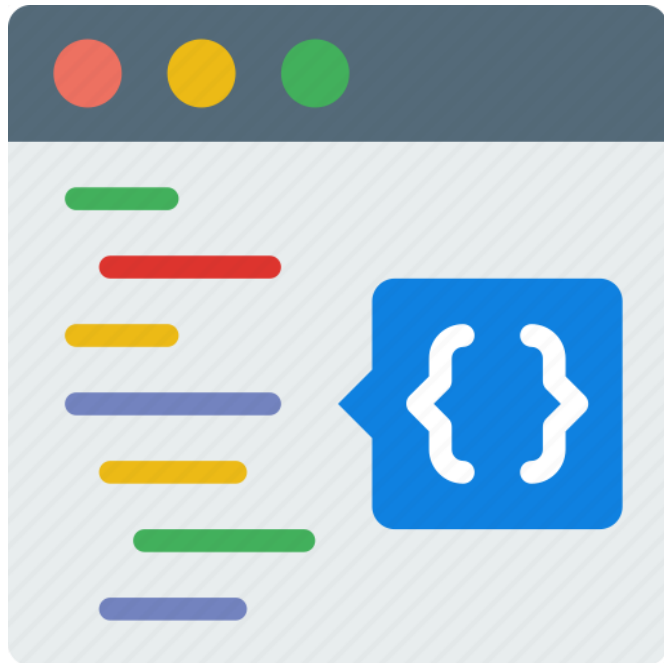




Computer Programming I: การเขียนโปรแกรมคอมพิวเตอร์ I

ฟังก์ชัน (Function; โปรแกรมย่อย)



อ.ดร.ปัญญานต์ อ้นพงษ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

aonpong_p@su.ac.th

Outline



- ฟังก์ชัน
- ประเภทและตัวอย่างของฟังก์ชัน
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์
- ข้อบังคับของการใช้ฟังก์ชันในภาษาซี
- การประกาศและนิยามฟังก์ชันแยกจากกัน
- ประเภทของฟังก์ชันสร้างเอง

ฟังก์ชัน (Function)



- โปรแกรมที่เราเขียน มักเป็นโปรแกรมขนาดเล็ก จึงไม่ค่อยมีโอกาสดูใช้ฟังก์ชันมากนัก แต่เนื่องจากในอนาคต เราจะต้องเขียนโปรแกรมที่ใหญ่ขึ้น
- ถ้าโปรแกรมที่เราเขียนมีขนาดใหญ่ การพัฒนางานก็จะยาก เราจึงสร้างโปรแกรมเป็นส่วนย่อย ๆ แล้วให้ฟังก์ชัน main เรียกฟังก์ชันย่อยมาใช้งาน
- ฟังก์ชันหรือโปรแกรมย่อยคือสิ่งที่สามารถถูกเรียกงานใช้ซ้ำ ๆ ได้จากโปรแกรมส่วนอื่น ๆ เพื่อทำงานบางอย่างให้โปรแกรมหลัก
- จริง ๆ แล้ว โปรแกรมที่เราเขียนมาตลอดมีการเรียกใช้งานฟังก์ชันมาบ้างแล้ว
 - printf, scanf เป็นฟังก์ชันสำเร็จรูปที่เราใช้งานบ่อยที่สุด

ฟังก์ชัน (Function)



- ฟังก์ชันหรือโปรแกรมย่อยคือสิ่งที่สามารถถูกเรียกงานใช้ซ้ำ ๆ ได้จากโปรแกรมส่วนอื่น ๆ เพื่อทำงานบางอย่างให้โปรแกรมหลัก
 - ให้จินตนาการว่าโปรแกรมที่เราเขียนเป็นการใช้ชีวิตประจำวันของเรา

ฟังก์ชัน (Function)



- มีอยู่วันหนึ่งนักศึกษาเดินไม่ทันระวังจนกางเกงขาดเป็นรู
 - เราจะซ่อมมันได้อย่างไร
 - ขั้นแรก ไปหาเข็มกับด้าย
 - ขั้นที่สอง ร้อยด้ายเข้าไปในรูเข็ม
 - ขั้นที่สาม ทำปมบนด้ายด้านหนึ่ง
 - ขั้นที่สี่ เย็บขอบที่ขาดให้ติดกัน
 - ...
 - ขั้นรองสุดท้าย ตัดด้ายที่เย็บเสร็จแล้วออกจากกางเกง
 - ขั้นสุดท้ายนำไปซัก ตาก



ฟังก์ชัน (Function)



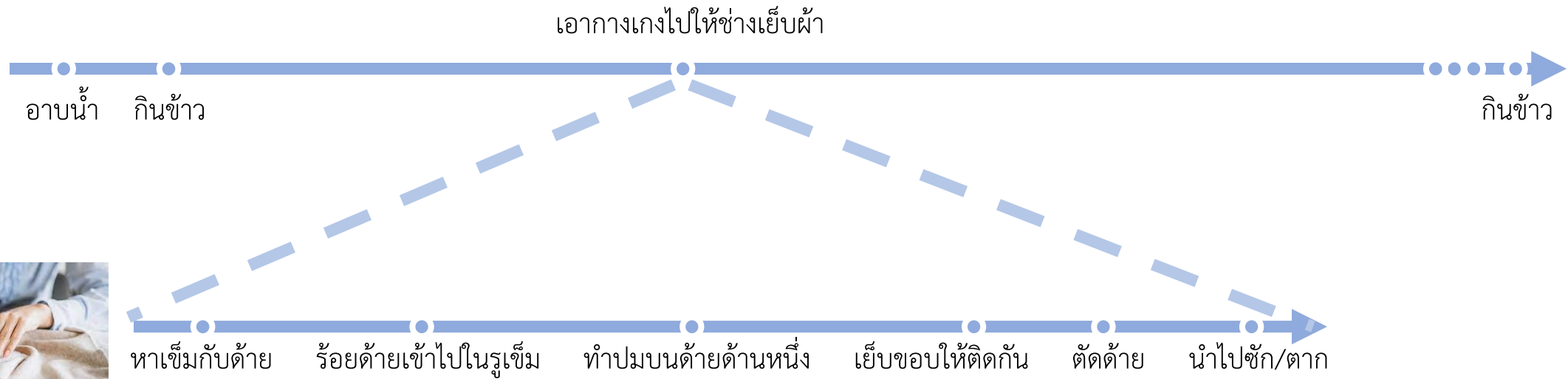
- ในหนึ่งวัน เราก็จะมีอะไรที่ต้องทำมากมาย



จะเห็นว่างานปะปนกันจนวันนี้ดูวุ่นวายมาก

ฟังก์ชัน (Function)

- ที่นี้ถ้าเราเอาไปให้ช่างเย็บผ้า...

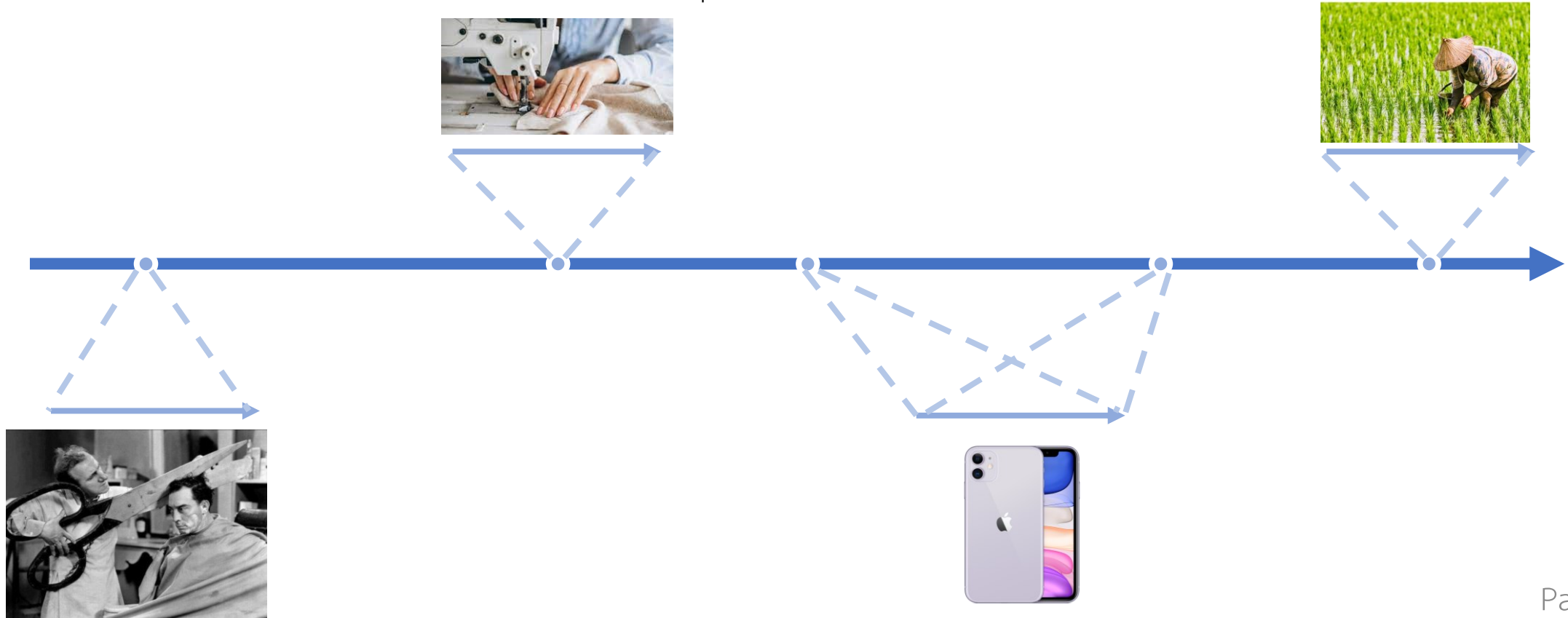


ฟังก์ชัน (Function)



- นอกจากนี้ ในความเป็นจริง ชีวิตเราในแต่ละวันก็ไม่ได้ทำทุกอย่างคนเดียว

คน/อุปกรณ์พวกนี้ทำให้การชีวิตโดยรวมเป็นระเบียบมากขึ้น



ฟังก์ชัน (Function)

- กลับมาที่โปรแกรมภาษาซี
- ฟังก์ชันเป็นเหมือนเครื่องจักรที่รับงานอย่างใดอย่างหนึ่งไปทำแทนเรา (เราจะโปรแกรมมันเอาไว้ว่า ถ้ารับค่า[วัตถุดิบ] เข้ามาแล้วจะต้องดำเนินการอย่างไรกับวัตถุดิบเหล่านั้นบ้าง)
 - เช่น รับหน้าที่ไปหาทางเอาตัวหนังสือขึ้นบนหน้าจอคอมพิวเตอร์
 - เช่น รับหน้าที่อ่านค่าจากคีย์บอร์ดแล้วนำมาเก็บไว้ในตัวแปร x ให้เรา
 - เช่น การหาค่ามากที่สุด น้อยที่สุดในกลุ่มข้อมูล เป็นต้น

ฟังก์ชัน (Function)



ข้อดีของฟังก์ชัน

- ทำให้เกิดการแบ่งโค้ดเป็นส่วน ๆ ที่มีเป้าหมายการทำงานชัดเจนเหมือนมีพนักงานประจำตำแหน่งทำหน้าที่อย่างใดอย่างหนึ่ง
- โค้ดจะเป็นระเบียบขึ้นมาก จากฟังก์ชัน main เราให้ฟังก์ชัน main (ชีวิตเราใน1วัน) ทำหน้าที่แบ่งงานให้ฟังก์ชันย่อย (พนักงานหรืออุปกรณ์) รับหน้าที่แทน
- การแบ่งงานจะทำให้โค้ดไม่ยุ่งเหยิงและแก้ปัญหได้ง่าย
- การแบ่งงานทำให้โค้ดดูเข้าใจง่าย เช่น เพียงแค่เห็นเครื่องมือชื่อ printf เราก็จะเข้าใจได้ทันทีว่าเป็นส่วนสำหรับแสดงผลออกทางจอภาพ

Outline



- ฟังก์ชัน
- **ประเภทและตัวอย่างของฟังก์ชัน**
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์
- ข้อบังคับของการใช้ฟังก์ชันในภาษาซี
- การประกาศและนิยามฟังก์ชันแยกจากกัน
- ประเภทของฟังก์ชันสร้างเอง

ประเภทและตัวอย่างของฟังก์ชัน

ฟังก์ชันมีอยู่สองประเภทคือ ฟังก์ชันมาตรฐานและฟังก์ชันสร้างเอง

- ฟังก์ชันมาตรฐานคือฟังก์ชันที่ได้มากจากการ `#include`
 - การ `#include` มีหลายแบบที่เป็นไปได้ ไม่ว่าจะเป็น `#include <stdio.h>`, `#include <stdlib.h>`, `#include <math.h>` และ `#include <string.h>`
 - การ `#include` จะทำให้โปรแกรมเราสามารถเรียกใช้โปรแกรมย่อยที่อยู่ในหมวดหมู่ที่เรา `#include` เข้ามาได้ เช่น เมื่อเรา `#include <stdio.h>` เราก็จะเรียกใช้ `printf` และ `scanf` ได้ (ซึ่งเป็นฟังก์ชันภายใต้หมวดหมู่นี้)
- ฟังก์ชันสร้างเอง (user-defined function) คือ ฟังก์ชันที่เราเขียนขึ้นมาเองเปรียบเหมือนกับเราออกแบบเครื่องจักรขึ้นมาเอง แทนที่จะไปซื้อเครื่องจักรมาใช้

ประเภทและตัวอย่างของฟังก์ชัน

สิ่งที่เราได้ทำไปในทุกสัปดาห์ มีหลายส่วนที่เป็นฟังก์ชันอยู่แล้ว (แต่คนอาจไม่รู้)

- ฟังก์ชันตัวแรกที่เราารู้จักคือฟังก์ชันหลัก ซึ่งต้องมีชื่อตายตัวว่า main
- ฟังก์ชันมาตรฐานตัวแรก ๆ ที่เรารู้จักคือฟังก์ชัน printf และ scanf
 - ฟังก์ชันมาตรฐานจะมีชื่อที่ตายตัวเช่นกัน
- ฟังก์ชันสร้างเองจะมีความทำงานเป็นไปตามที่เราอยากให้เป็น
 - เราจะเขียนให้มันทำอะไรก็ได้ จะให้มันเรียกฟังก์ชันอื่นต่อ ๆ กันไปก็ได้
 - ฟังก์ชันสร้างเองจะเรียกฟังก์ชันมาตรฐานก็ได้
 - ชื่อของมันไม่ตายตัว ขึ้นอยู่กับเราตั้ง แต่โดยส่วนมากจะตั้งชื่อตามหน้าที่ เช่น คนมักใช้ชื่อว่า max เป็นชื่อฟังก์ชันการหาค่าสูงสุด (เหมือนตั้งชื่อโพลเดอร์)

Outline

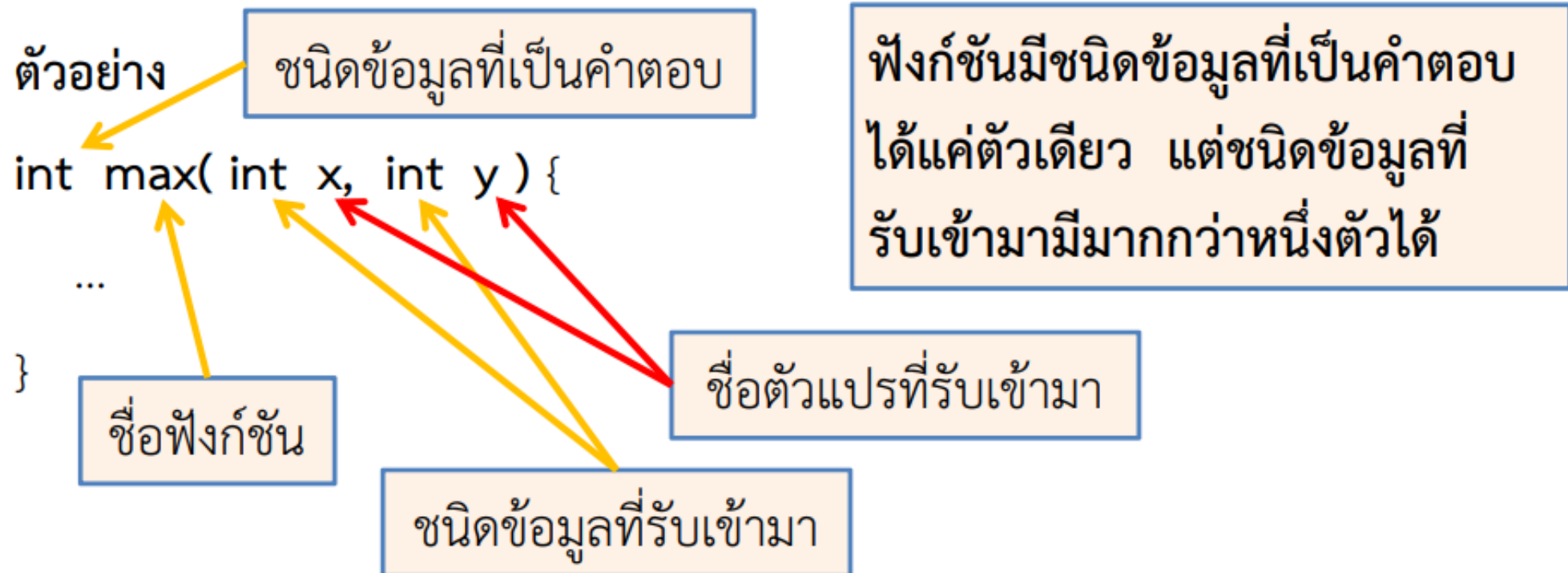


- ฟังก์ชัน
- ประเภทและตัวอย่างของฟังก์ชัน
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์
- ข้อบังคับของการใช้ฟังก์ชันในภาษาซี
- การประกาศและนิยามฟังก์ชันแยกจากกัน
- ประเภทของฟังก์ชันสร้างเอง

โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ชนิดของข้อมูลคำตอบ ชื่อฟังก์ชัน (ชนิดข้อมูลที่รับเข้า ชื่อตัวแปรที่รับเข้า, ...){
...ชุดคำสั่งภายในฟังก์ชัน...

}



โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (1) จงเขียนโปรแกรมที่รับค่าตัวเลขจำนวนเต็มมาสามคู่ตามลำดับดังนี้ x_1 y_1 , x_2 y_2 , และ x_3 y_3 จากนั้นให้โปรแกรมพิมพ์ค่าตัวเลขที่มีค่ามากที่สุดของแต่ละคู่ออกมาทางจอภาพ

ข้อมูลเข้า	ผลลัพธ์	ข้อมูลเข้า	ผลลัพธ์
5 9	9	9 5	9
0 -5	0	7 7	7
2 1	2	2 0	2
5 0	5	0 2	2
4 -1	4	5 7	7
-5 -1	-1	-2 0	0

โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (1): วิธีที่ 1 แบบดั้งเดิม

จะเห็นว่ามีการทำงานเหมือนกัน 3 ครั้ง

เปรียบเทียบค่าของคู่ที่ 1

เปรียบเทียบค่าของคู่ที่ 2

เปรียบเทียบค่าของคู่ที่ 3

```
#include <stdio.h>

void main()
{
    int x1, x2, x3, y1, y2, y3, result;
    scanf("%d %d", &x1, &y1);
    scanf("%d %d", &x2, &y2);
    scanf("%d %d", &x3, &y3);

    if(x1 > y1) result = x1;
    else result = y1;
    printf("%d\n", result);

    if(x2 > y2) result = x2;
    else result = y2;
    printf("%d\n", result);

    if(x3 > y3) result = x3;
    else result = y3;
    printf("%d\n", result);
}
```

โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (1): วิธีที่ 2 แบบดั้งเดิม แต่เลี่ยงการเก็บค่า
สังเกตว่า แม้จะเลี่ยงการเก็บค่า
ก็ยังคงมีการทำงานเหมือนกัน 3 ครั้งอยู่ดี

เปรียบเทียบค่าของคู่ที่ 1

เปรียบเทียบค่าของคู่ที่ 2

เปรียบเทียบค่าของคู่ที่ 3

```
#include <stdio.h>

void main()
{
    int x1, x2, x3, y1, y2, y3, result;
    scanf("%d %d", &x1, &y1);
    scanf("%d %d", &x2, &y2);
    scanf("%d %d", &x3, &y3);

    if(x1 > y1) printf("%d\n", x1);
    else printf("%d\n", y1);

    if(x2 > y2) printf("%d\n", x2);
    else printf("%d\n", y2);

    if(x3 > y3) printf("%d\n", x3);
    else printf("%d\n", y3);
}
```

โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (1): วิธีที่ 3 สร้างฟังก์ชัน

ไวยากรณ์ของฟังก์ชันก็คือ รับค่าเข้ามา แล้วคืนค่าออกไป



โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (1): วิธีที่ 3 สร้างฟังก์ชัน

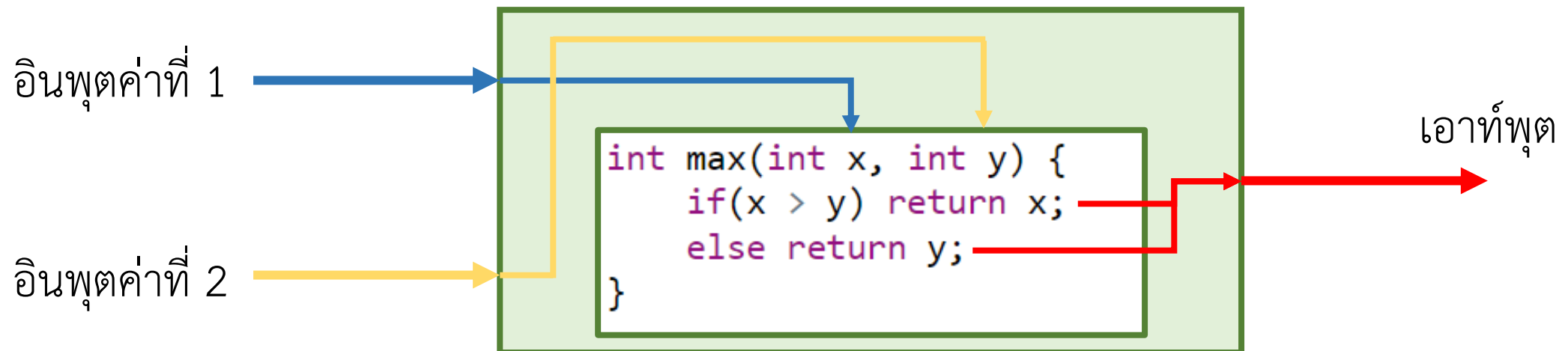
จากโปรแกรมที่สร้างโดยวิธีที่ 1 และ 2 จะพบว่า การหาค่าที่มากกว่า เป็นการใช้เทคนิคเดียวกันซ้ำๆ ดังนั้นเราสามารถเขียนฟังก์ชันได้ดังนี้

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}
```

โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (1): วิธีที่ 3 สร้างฟังก์ชัน

ถ้านำฟังก์ชันหาค่ามากที่สุดมาเขียน ก็จะได้กลไกประมาณนี้



โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน



ตัวอย่าง (1): วิธีที่ 3 สร้างฟังก์ชัน

พอเรานำมาเขียนในฟังก์ชัน main ก็จะต้องเรียบเรียงน้อยกว่าเดิม สังเกตว่าฟังก์ชัน max ไม่ได้เขียนในฟังก์ชัน main และนี่เป็นครั้งแรกที่เราทำแบบนี้

```
#include<stdio.h>

int max(int x, int y) {
    if(x > y) return x;
    else return y;
}
```

```
void main() {
    int x1, x2, x3, y1, y2, y3, result;
    scanf("%d %d", &x1, &y1);
    scanf("%d %d", &x2, &y2);
    scanf("%d %d", &x3, &y3);

    result = max(x1, y1);
    printf("%d\n", result);

    result = max(x2, y2);
    printf("%d\n", result);

    result = max(x3, y3);
    printf("%d\n", result);
}
```

โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (1): วิธีที่ 4 สร้างฟังก์ชัน แบบเลี้ยงตัวแปร

แบบนี้จะทำให้โค้ดดูเรียบร่อยกว่าเดิมอย่างชัดเจน นอกจากนี้ถ้าพบว่ามีคามผิดพลาดในส่วนการหาค่า max หรือจะเปลี่ยนวิธีการหา เราก็แก้ไขในฟังก์ชันที่เดียว ก็จะส่งผลกับทั้งโปรแกรม

```
#include<stdio.h>

int max(int x, int y) {
    if(x > y) return x;
    else return y;
}
```

ผลที่ได้จากฟังก์ชันจะทำตัวเหมือนตัวเลขหรือตัวอักษรตามชนิดข้อมูลของผลลัพธ์ที่ระบุไว้

```
void main() {
    int x1, x2, x3, y1, y2, y3, result;
    scanf("%d %d", &x1, &y1);
    scanf("%d %d", &x2, &y2);
    scanf("%d %d", &x3, &y3);

    printf("%d\n", max(x1, y1));
    printf("%d\n", max(x2, y2));
    printf("%d\n", max(x3, y3));
}
```

โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (2) คำนวณพื้นที่วงกลม จงเขียนโปรแกรมหาพื้นที่วงกลมจากค่ารัศมีที่ใส่เข้าไป เมื่อสมการหาพื้นที่วงกลมคือ $area_{circle}(r) = \pi r^2$ โดยใช้ฟังก์ชัน

ตอนนี้ยังมีฟังก์ชันมาให้ลองทำความเข้าใจก่อน ให้นักศึกษาลองเขียนฟังก์ชัน main ที่รองรับกับฟังก์ชันนี้ด้วยตัวเอง

```
double findCircleArea(double radius) {  
    const double PI = 3.1415926535;  
    return PI * radius * radius;  
}
```


โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน

ตัวอย่าง (2) คำนวณพื้นที่วงกลม จงเขียนโปรแกรมหาพื้นที่วงกลมจากค่ารัศมีที่ใส่เข้าไป เมื่อสมการหาพื้นที่วงกลมคือ $area_{circle}(r) = \pi r^2$ โดยใช้ฟังก์ชัน

ตอนนี้ยังมีฟังก์ชันมาให้ลองทำความเข้าใจก่อน ให้นักศึกษาลองเขียนฟังก์ชัน main ที่รองรับกับฟังก์ชันนี้ด้วยตัวเอง

```
double findCircleArea(double radius) {  
    const double PI = 3.1415926535;  
    return PI * radius * radius;  
}
```

```
void main() {  
    double radius;  
    scanf("%lf", &radius);  
    double circleArea = findCircleArea(radius);  
    printf("Circle area = %lf\n", circleArea);  
}
```

Outline



- ฟังก์ชัน
- ประเภทและตัวอย่างของฟังก์ชัน
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- **ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์**
- ข้อบังคับของการใช้ฟังก์ชันในภาษาซี
- การประกาศและนิยามฟังก์ชันแยกจากกัน
- ประเภทของฟังก์ชันสร้างเอง

ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์

- เป็นของที่มีความสัมพันธ์กันโดยตรง เช่น จากตัวอย่างที่ 2 หากเรามองไปที่ฟังก์ชันคำนวณพื้นที่วงกลมในรูปแบบคณิตศาสตร์ เราจะได้ฟังก์ชัน

$$area_{circle}(r) = \pi r^2$$

- ฟังก์ชันนี้มี r เป็นตัวแปร และ คำนวณพื้นที่วงกลมให้ผลลัพธ์ออกมาเป็นตัวเลข
- ถ้ากลับมาพิจารณาฟังก์ชันในภาษาซี จะเห็นได้ว่าจริงๆ แล้วมันคือสิ่งเดียวกัน (คือมีค่ารัศมีเป็นตัวแปรและคำนวณพื้นที่วงกลมเป็นผลลัพธ์มาเป็นตัวเลข (ชนิด double))

```
double findCircleArea(double radius) {  
    const double PI = 3.1415926535;  
    return PI * radius * radius;  
}
```

ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์

- ฟังก์ชันที่หาค่าสูงสุดในตัวอย่างที่ 1 ก็ไม่ต่างกัน เราสามารถเขียนบรรยายในรูปคณิตศาสตร์ได้เหมือนกัน
 - การแยกกรณีในฟังก์ชันคณิตศาสตร์จะสัมพันธ์กับการใช้ if ในภาษาซี
 - เงื่อนไขในฟังก์ชันคณิตศาสตร์กับ if เป็นเงื่อนไขชุดเดียวกัน

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}
```

$$f(x, y) = \begin{cases} x & ; x > y \\ y & ; otherwise \end{cases}$$

ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์

ตัวอย่าง (3) ฟังก์ชันปลากระป๋องแบบพื้นฐาน

โจทย์ ปลากระป๋องยี่ห้อหนึ่งใช้ปลาซาร์ดีนสามตัวและมะเขือเทศสองผลเพื่อผลิตปลากระป๋องหนึ่งกระป๋อง หากโรงงานผลิตมีปลาซาร์ดีนอยู่ X ตัวและมะเขือเทศอยู่ Y ผล โรงงานจะผลิตปลากระป๋องได้ทั้งหมดกี่กระป๋อง

วิเคราะห์ ถ้านำโจทย์มาเขียนเป็นสมการทางคณิตศาสตร์ จะได้ดังนี้

$$f(x, y) = \begin{cases} \frac{x}{3}; & \frac{x}{3} < \frac{y}{2} \\ \frac{y}{2}; & \text{otherwise} \end{cases}$$

$$f(x, y) = \begin{cases} \frac{x}{3}; & \frac{x}{3} < \frac{y}{2} \\ \frac{y}{2}; & \frac{y}{2} \leq \frac{x}{3} \end{cases}$$

ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์

ตัวอย่าง (3) ฟังก์ชันปลากระป๋องแบบพื้นฐาน (วิธีที่ 1 แบบเดิมๆ)

```
#include<stdio.h>

void main() {
    int x, y;
    scanf("%d %d", &x, &y);
    if(x / 3 < y / 2)
        printf("%d", x / 3);
    else
        printf("%d", y / 2);
}
```

ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์



ตัวอย่าง (3) ฟังก์ชันปลากะป๋องแบบพื้นฐาน (วิธีที่ 2 ใช้ฟังก์ชัน)

```
#include<stdio.h>

int findCans(int x, int y) {
    if(x / 3 < y / 2)
        return x / 3;
    else
        return y / 2;
}

void main() {
    int x, y;
    scanf("%d %d", &x, &y);
    printf("%d", findCans(x, y));
}
```

ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์



ตัวอย่าง (4) เป็นเลขคู่หรือไม่

จงเขียนฟังก์ชันที่คืนผลลัพธ์เป็นเลข 1 (ค่าจริง) เมื่อตัวเลขที่รับมาเป็นอินพุตเป็นเลขคู่ และคืนเลข 0 (ค่าเท็จ) เมื่อตัวเลขที่รับมาเป็นเลขคี่

ฟังก์ชันคอมพิวเตอรืและฟังก์ชันคณิตศาสตร์

ตัวอย่าง (4) เป็นเลขคู่หรือไม่

จงเขียนฟังก์ชันที่คืนผลลัพธ์เป็นเลข 1 (ค่าจริง) เมื่อตัวเลขที่รับมาเป็นอินพุตเป็นเลขคู่ และคืนเลข 0 (ค่าเท็จ) เมื่อตัวเลขที่รับมาเป็นเลขคี่

วิธีการหนึ่งที่เป็นไปได้ มีใครเขียนได้ไม่เหมือนวิธีนี้บ้าง?

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

ฟังก์ชันคอมพิวเตอรืและฟังก์ชันคณิตศาสตร์

ตัวอย่าง (5) เป็นเลขคี่หรือไม่

จงเขียนฟังก์ชันเพิ่มเติมจากโปรแกรมในข้อ 4 ที่คืนผลลัพธ์เป็นเลข 1 (ค่าจริง) เมื่อตัวเลขที่รับมาเป็นอินพุตเป็นเลขคี่ และคืนเลข 0 (ค่าเท็จ) เมื่อตัวเลขที่รับมาเป็นเลขคู่

แนวคิด เราใช้วิธีกลับค่าความจริงจากฟังก์ชัน isEven ที่เขียนมาเมื่อสักครู่นี้ได้

```
int isEven(int x){  
    if(x % 2 == 0)  
        return 1;  
    else  
        return 0;  
}  
int isOdd(int x) {  
    return !isEven(x);  
}
```

ฟังก์ชันคอมพิวเตอรืและฟังก์ชันคณิตศาสตร์

ลองนำตัวอย่าง (4) และ (5) มารวมกันเป็นโปรแกรมพร้อมใช้งาน

แนะนำให้นำไปลองทำดังนี้

- (1) ลองแก้ไขขั้นตอนใน isEven ให้ทำงานอย่างอื่นดู
- (2) ลองแก้ไขขั้นตอนใน isOdd ให้ทำงานโดยไม่พึ่งฟังก์ชัน isEven
- (3) ลองสลับค่า isEven และ isOdd ใน main และดูความแตกต่าง

```
#include<stdio.h>

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

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

void main() {
    int x;
    scanf("%d", &x);
    printf("isEven = %d\n", isEven(x));
    printf("isOdd = %d\n", isOdd(x));
}
```

Outline



- ฟังก์ชัน
- ประเภทและตัวอย่างของฟังก์ชัน
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์
- **ข้อบังคับของการใช้ฟังก์ชันในภาษาซี**
- การประกาศและนิยามฟังก์ชันแยกจากกัน
- ประเภทของฟังก์ชันสร้างเอง

Outline



- ข้อบังคับของการใช้ฟังก์ชันในภาษาซี
 - กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน
 - ฟังก์ชันกับการรับค่า (parameter) และการส่งผลลัพธ์ให้ผู้เรียก

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

- ฟังก์ชันจะถูกเรียกได้ ณ จุดเรียกใช้ ที่มาหลังการประกาศฟังก์ชัน (เหมือนตัวแปร ที่เรียกใช้ได้หลังจากประกาศแล้วเท่านั้น)
 - ดังนั้นหากเราประกาศฟังก์ชันไว้ก่อน main ฟังก์ชันก็จะถูกเรียกใช้ได้ใน main (เหมือนตัวอย่างที่ได้แสดงให้เห็นไปก่อนหน้านี้)
- ฟังก์ชันสามารถถูกเรียกต่อ ๆ กันได้
 - ถ้า main เรียกฟังก์ชัน A ฟังก์ชัน A จะเรียกฟังก์ชัน B ต่ออีกทอดก็ได้
 - ฟังก์ชัน B จะเรียกฟังก์ชันอื่น ๆ ต่อไปอีกก็ได้
 - ฟังก์ชันอันหนึ่ง (ทั้ง main และฟังก์ชันย่อย) เรียกฟังก์ชันอื่นมากกว่าหนึ่งก็ได้
 - เช่น main จะเรียกทั้ง scanf และ printf ก็ได้
 - ฟังก์ชันย่อยจะเรียกฟังก์ชันมาตรฐานก็ได้

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

- เพราะจุดที่มีการเรียกฟังก์ชันต้องมาหลังจากการประกาศฟังก์ชัน
 - การจัดวางที่ถูกต้องก็คือให้ประกาศฟังก์ชันที่จะถูกเรียกไว้ด้านบนและในฟังก์ชันที่ทำการเรียก (เช่น ฟังก์ชัน main และ isOdd) ไว้หลังฟังก์ชันที่จะถูกเรียก
 - สังเกตว่า main เรียกทั้ง isEven และ isOdd ดังนั้นเราจึงต้องประกาศ isEven และ isOdd ไว้ก่อน ส่วนการเรียกใช้ใน main จะมาหลังสุด
 - isOdd เรียก isEven ดังนั้นเราต้องประกาศ isEven ไว้ก่อน
 - ส่วนการเรียกใช้ใน isOdd จะมาหลังสุด

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน



```
#include<stdio.h>

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

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

void main() {
    int x;
    scanf("%d", &x);
    printf("isEven = %d\n", isEven(x));
    printf("isOdd = %d\n", isOdd(x));
}
```

```
15
isEven = 0
isOdd = 1
```

```
#include<stdio.h>

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

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

void main() {
    int x;
    scanf("%d", &x);
    printf("isEven = %d\n", isEven(x));
    printf("isOdd = %d\n", isOdd(x));
}
```

```
main.c: In function 'isOdd':
main.c:5:13: warning: implicit declaration of function 'isEven' [-Wimplicit-declaration]
    5 |         return !isEven(x);
      |                ^~~~~~
```


กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

- ฟังก์ชันสามารถรับค่าตัวเลขหรือตัวแปรก็ได้ดังที่แสดงไว้ก่อนหน้านี้
- ค่าที่รับส่งกันไปมาระหว่างฟังก์ชัน เรียกว่าพารามิเตอร์ (Parameter)

```
int isEven(int x){  
    if(x % 2 == 0)  
        return 1;  
    else  
        return 0;  
}  
int isOdd(int x) {  
    return !isEven(x);  
}
```

```
double findCircleArea(double radius) {  
    const double PI = 3.1415926535;  
    return PI * radius * radius;  
}
```

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}
```

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

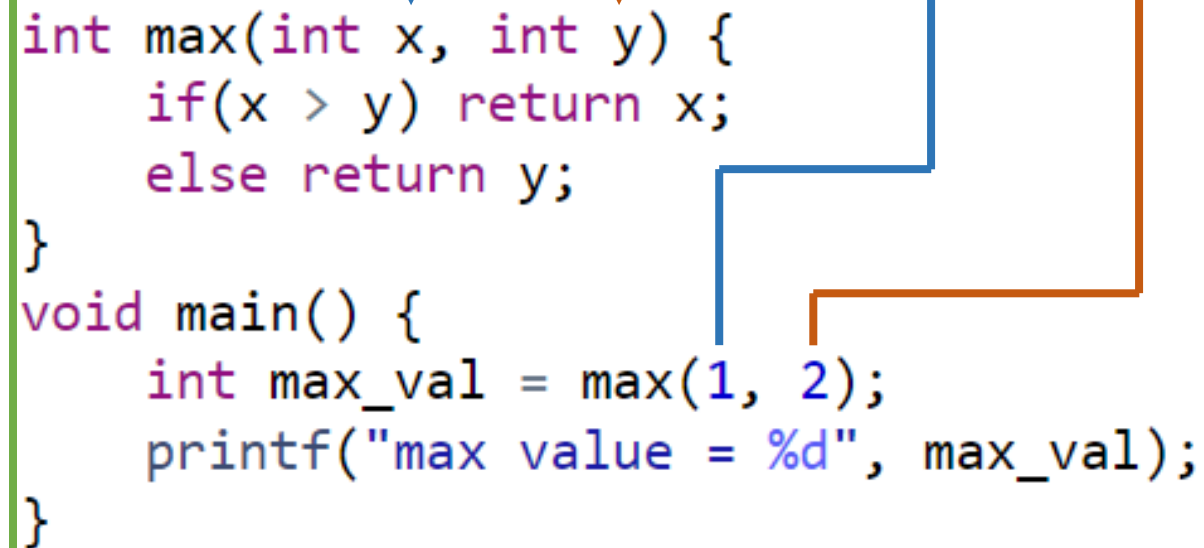
ในมุมมองของฟังก์ชันและผู้เรียกฟังก์ชัน จะมองพารามิเตอร์เป็นคนละแบบกัน

- สังเกตตัวอย่างนี้

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}  
  
void main() {  
    int max_val = max(1, 2);  
    printf("max value = %d", max_val);  
}
```

- ในตัวอย่างนี้ฟังก์ชัน main มองพารามิเตอร์เป็นตัวเลขธรรมดา
- แต่ฟังก์ชัน max มองพารามิเตอร์เป็นตัวแปรตลอดเวลา

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน



```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}  
void main() {  
    int max_val = max(1, 2);  
    printf("max value = %d", max_val);  
}
```

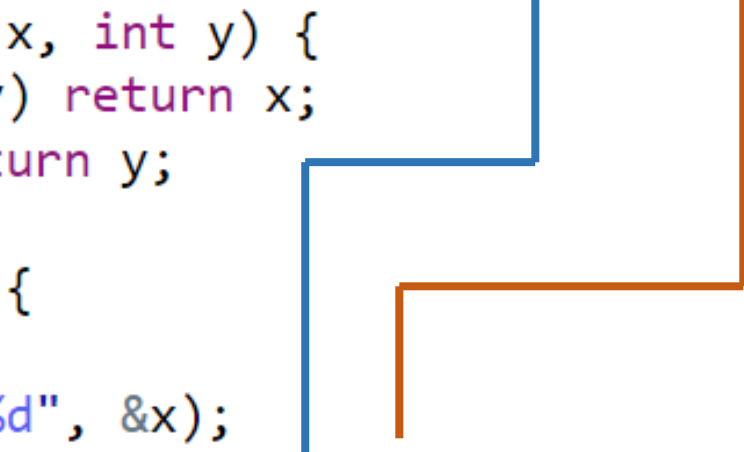
The diagram illustrates the execution flow between two functions. A blue line starts from the `main` function, goes up, and then branches into two paths. One path leads to the `max` function, and the other path leads to the `printf` statement in `main`. An orange line starts from the `max` function, goes up, and then branches into two paths. One path leads to the `main` function, and the other path leads to the `printf` statement in `main`. This visualizes the call stack and the return flow between the functions.

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}  
  
void main() {  
    int x;  
    scanf("%d", &x);  
    int max_val = max(x, 2);  
    printf("max value = %d", max_val);  
}
```

- ในตัวอย่างนี้ฟังก์ชัน main มองพารามิเตอร์ x เป็นตัวแปร ส่วนพารามิเตอร์อีกตัวเป็นตัวเลขธรรมดา
- แต่ฟังก์ชัน max มองพารามิเตอร์เป็นตัวแปรตลอดเวลา

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน



```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}  
void main() {  
    int x;  
    scanf("%d", &x);  
    int max_val = max(x, 2);  
    printf("max value = %d", max_val);  
}
```

The diagram illustrates the flow of control between the `main` function and the `max` function. A blue line starts at the `scanf` call in `main`, goes up, and then right to enter the `max` function at its opening curly brace. An orange line starts at the `printf` call in `main`, goes up, and then right to enter the `max` function at its opening curly brace. Both lines exit the `max` function at its closing curly brace and return to the `main` function. The blue line returns to the line following `scanf`, and the orange line returns to the line following `printf`.

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

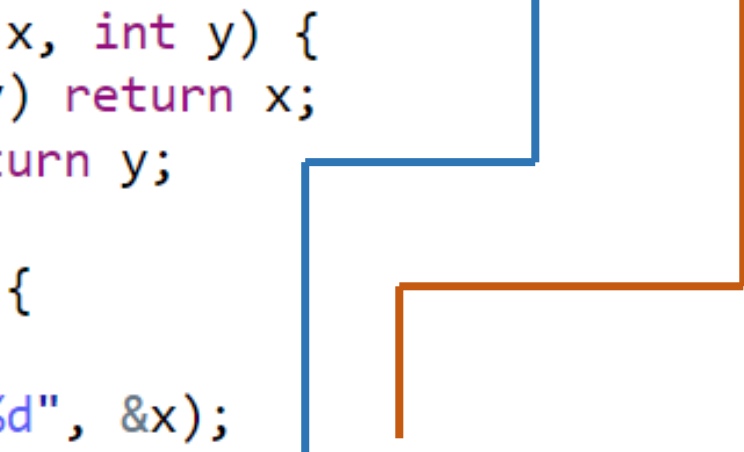
- จะเห็นได้ว่าผู้เรียกจะปฏิบัติกับพารามิเตอร์เป็นค่า ๆ หนึ่ง ซึ่งค่านั้นจะเป็นค่าคงที่หรือตัวแปรก็ได้
- ส่วนตัวฟังก์ชันจะมองพารามิเตอร์เป็นตัวแปรเสมอ
- เหมือนตอนเขียนฟังก์ชันคณิตศาสตร์ เราก็เขียนติดตัวแปรเอาไว้เสมอ เช่น

$$f(x, y) = \begin{cases} \frac{x}{3}; & \frac{x}{3} < \frac{y}{2} \\ \frac{y}{2}; & \frac{y}{2} \leq \frac{x}{3} \end{cases}$$

- แต่เมื่อเราจะหาค่าฟังก์ชัน เราจะพิจารณาเป็นค่าที่ใส่เข้าไป เช่น $f(300, 100)$ หรือ $f(200, 100)$ เป็นต้น

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}  
void main() {  
    int x;  
    scanf("%d", &x);  
    int max_val = max(x, 2);  
    printf("max value = %d", max_val);  
}
```



ถ้าเปลี่ยนตัว x ตัวนี้เป็นตัวแปรอื่นจะมีปัญหามั้ย?

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

- ชื่อของตัวแปรพารามิเตอร์ของผู้เรียกและของฟังก์ชันไม่จำเป็นต้องเหมือนกัน
 - เพราะแท้จริงแล้วผู้เรียกสนใจแค่ค่าที่จะส่งไปมีค่าเท่าไรและชนิดข้อมูลตรงกันหรือไม่
 - ฟังก์ชันที่ถูกเรียกจะดึงค่าที่ถูกส่งมาไปเก็บไว้ในตัวแปร
- ไม่ได้สนใจว่าค่าที่ส่งมาเป็นตัวแปรหรือค่าคงที่ แต่ก็สนใจว่าชนิดข้อมูลตรงกันหรือไม่

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}  
void main() {  
    int a, b;  
    scanf("%d %d", &a, &b);  
    int max_val = max(a, b);  
    printf("max value = %d", max_val);  
}
```


กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

ตัวอย่าง (6) โปรแกรมต่อไปนี้ให้ผลลัพธ์อย่างไร เนื่องจากการเรียกใช้ฟังก์ชันสองครั้ง ให้พิจารณาคำตอบของฟังก์ชันเหมือนกันทั้งสองครั้งหรือไม่

```
int sub(int x, int y) {  
    return x - y;  
}  
void main() {  
    int x = 5;  
    int y = 2;  
    printf("%d\n", sub(x, y));  
    printf("%d\n", sub(y, x));  
}
```

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

ตัวอย่าง (6) คำตอบ ไม่เหมือนกัน เรียกฟังก์ชันครั้งแรกได้ 3 เรียกครั้งที่ 2 ได้ -3

สรุปได้ว่า การส่งพารามิเตอร์สำคัญที่ลำดับค่า ไม่ใช่ชื่อตัวแปร

- ความผิดพลาดที่มักพบเห็นได้คือผู้เขียนเข้าใจผิดว่าชื่อพารามิเตอร์กับตัวแปรจะต้องเหมือนกัน
- จริงๆแล้วไม่ต้องเหมือนกันก็ได้ สามารถตั้งชื่อแปลกๆหรือแม้กระทั่งส่งตัวเลขเข้าไปเลยก็ได้
- แต่ลำดับที่ส่งเข้าไปและประเภทตัวแปรจะต้องถูกต้อง
- ถ้าลำดับผิด โปรแกรมมักจะให้ผลผิดพลาด
- ถ้าประเภทตัวแปรผิด โปรแกรมจะแจ้ง Error

```
int sub(int x, int y) {  
    return x - y;  
}  
void main() {  
    int x = 5;  
    int y = 2;  
    printf("%d\n", sub(x, y));  
    printf("%d\n", sub(y, x));  
}
```

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

แต่ก็มีบางฟังก์ชันที่ลำดับค่าไม่มีผล แต่การที่ลำดับค่าไม่มีผลนั้นเกิดขึ้นจากกระบวนการภายในฟังก์ชันที่เราเขียน ไม่ใช่ธรรมชาติของฟังก์ชัน เช่น

ฟังก์ชันหาค่าสูงสุด

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}
```

ฟังก์ชันหาค่าสัมบูรณ์ เป็นต้น

```
int absoluteDiff(int x, int y) {  
    if(x > y) return x - y;  
    else return y - x;  
}
```

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

ฟังก์ชัน ไม่สามารถเปลี่ยนตัวแปรที่อยู่นอกฟังก์ชันได้

- “ฟังก์ชันที่ถูกเรียกจะดึงค่าที่ถูกส่งมาไปเก็บไว้ในตัวแปรไม่ได้สนใจว่าค่าที่ส่งมาเป็นตัวแปรหรือค่าคงที่”
- กล่าวคือ ฟังก์ชันจะทำสำเนาพารามิเตอร์มาใช้ภายใน
 - หากฟังก์ชันทำการเปลี่ยนค่าตัวแปรพารามิเตอร์ ผลก็จะเกิดเพียงข้างในฟังก์ชัน ไม่ได้เปลี่ยนอะไรที่ผู้เรียก
 - นั่นคือผู้เรียกฟังก์ชันไม่ได้รับผลกระทบใด ๆ จากการเปลี่ยนค่าตัวแปรพารามิเตอร์ในฟังก์ชัน

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน



ตัวอย่าง (7) โปรแกรมนี้ให้ผลลัพธ์อะไร

```
#include<stdio.h>

int add_mult(int x, int y) {
    x = x + y;
    x = x * y;
    return x;
}

void main() {
    int x, y, result;
    x = 5; y = 2;
    result = add_mult(x, y);
    printf("%d\n", x);
    printf("%d\n", result);
}
```

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน



ตัวอย่าง (7) โปรแกรมนี้ให้ผลลัพธ์อะไร

```
#include<stdio.h>

int add_mult(int x, int y) {
    x = x + y;
    x = x * y;
    return x;
}

void main() {
    int x, y, result;
    x = 5; y = 2;
    result = add_mult(x, y);
    printf("%d\n", x);
    printf("%d\n", result);
}
```

5
14

กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

การส่งผลลัพธ์ให้ผู้เรียก

- การรับค่ากลับมาจากฟังก์ชัน ผู้เรียกจะมองฟังก์ชันว่าเป็นเพียงค่าคงที่
 - จะเอาค่าคงที่นั้นไปเก็บไว้ในตัวแปรก็ได้
 - จะเอาค่าคงที่นั้นไปแสดงผลก็ได้
- ผู้เขียนโปรแกรมจึงต้องเข้าใจลำดับการพิจารณา เป็นไปตามลำดับการทำงานจริงของโปรแกรมคือ
 1. เตรียมค่าพารามิเตอร์ เป็นค่าคงที่ธรรมดาหรือเป็นตัวแปร
 2. เรียกฟังก์ชัน ชื่อฟังก์ชันกับลำดับพารามิเตอร์ตรงกันหรือไม่
 3. ฟังก์ชันคืนผลลัพธ์กลับมา ชนิดข้อมูลตรงกับตัวแปรที่จะเก็บค่าไว้หรือเปล่า



กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน



เพราะว่า findCircleArea แท้จริงคือ double ดังนั้น
เราจึงควรใช้ตัวแปรประเภท double มาเก็บผลลัพธ์
ไว้ ถ้าใช้ area2 มาเก็บผลลัพธ์จะคลาดเคลื่อน

```
#include<stdio.h>

double findCircleArea(double radius) {
    const double PI = 3.1415926535;
    return PI * radius * radius;
}

void main() {
    double radius = 1.2;
    double area1 = findCircleArea(radius);
    int area2 = findCircleArea(radius);
}
```


กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน

เรื่องที่น่าจะทำให้สับสน

```
double radius = 1.2;  
double area1 = findCircleArea(radius);
```

จากตัวอย่างโปรแกรม ลำดับการทำงานที่แท้จริงคือ ฟังก์ชัน findCircleArea จะรับพารามิเตอร์ไปคิดผลลัพธ์ก่อน จากนั้นจึงค่อยคืนผลลัพธ์มาไว้ที่ area1

แต่เรามักสับสนกับการเขียนฟังก์ชันทางคณิตศาสตร์ ที่ลำดับการเขียนจะไม่เหมือนกับภาษาซี

```
double area1;  
findCircleArea(radius) = area1;
```

*เหมือนกับที่คณิตศาสตร์ เขียนว่า $1+1=x$ ได้ แต่ภาษาซีเขียนไม่ได้ ต้องเป็น $x=1+1$;

Outline

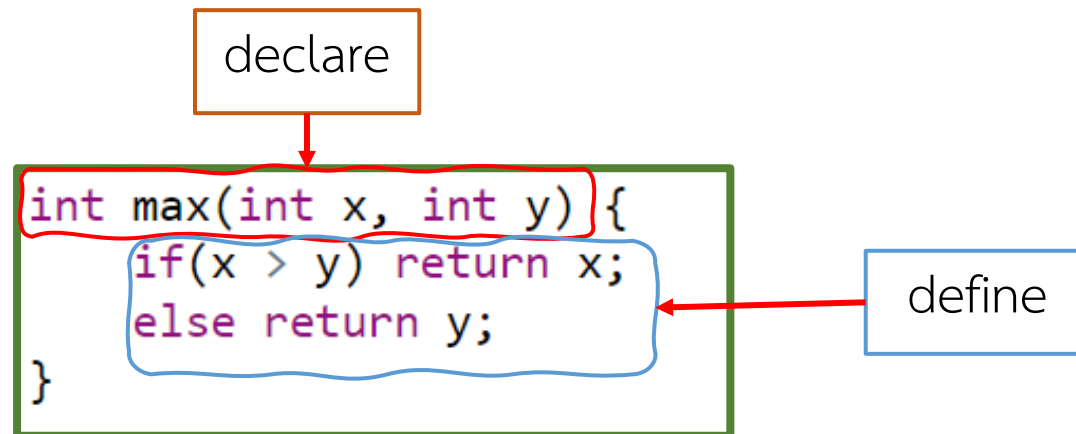


- ฟังก์ชัน
- ประเภทและตัวอย่างของฟังก์ชัน
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์
- ข้อบังคับของการใช้ฟังก์ชันในภาษาซี
- **การประกาศและนิยามฟังก์ชันแยกจากกัน**
- ประเภทของฟังก์ชันสร้างเอง

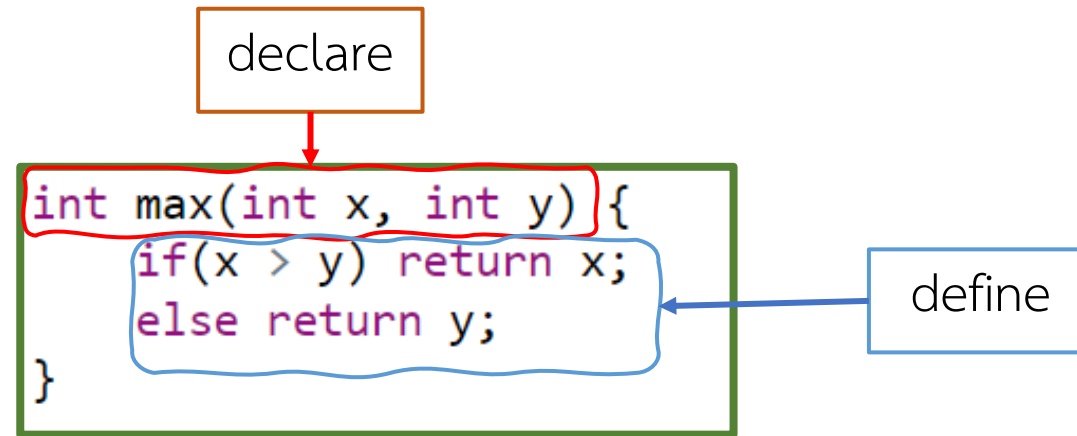
การประกาศและนิยามฟังก์ชันแยกจากกัน

คำว่าประกาศ (declare) กับการนิยาม (define) ฟังก์ชันเป็นสิ่งที่แตกต่างกัน

- ในตัวอย่างก่อนหน้านี้ทั้งหมดเราทำการ declare และ define พร้อมกัน
- อะไรที่เรียกว่า declare อะไรที่เรียกว่า define
 - การระบุชนิดข้อมูล ชื่อฟังก์ชัน และ พารามิเตอร์ คือการ declare (คือการประกาศว่ามีฟังก์ชันนี้อยู่)
 - การระบุว่าฟังก์ชันทำงานอย่างไร (โค้ดข้างในฟังก์ชัน) คือการ define (การนิยามคือการระบุรายละเอียดทางการคำนวณ)



การประกาศและนิยามฟังก์ชันแยกจากกัน



จากภาพ เราสามารถแยกการ declare และ define ออกจากกันได้

แต่ก่อนที่จะดูว่ามันทำอย่างไร มาดูข้อดีของการแยกทั้งสองออกจากกันก่อน

การประกาศและนิยามฟังก์ชันแยกจากกัน

ประโยชน์ของการแยกการ declare และ define

1. พิจารณาลักษณะของฟังก์ชันต่อไปนี้

```
int A(int x) { ... B(5); ... }  
int B(int y) { ... A(0); ... }
```

อย่างที่เรารู้ว่า การจะใช้ฟังก์ชัน จะต้องมีการประกาศฟังก์ชันก่อน (เหมือนกับตัวแปร) ดังนั้นในกรอบนี้ไม่สามารถทำได้ เพราะฟังก์ชัน A เรียกใช้ฟังก์ชัน B ซึ่งยังไม่เคยถูกประกาศมาก่อน

การประกาศและนิยามฟังก์ชันแยกจากกัน

ประโยชน์ของการแยกการ declare และ define

2. ในการเขียนโปรแกรมภาษาซี ภาพรวมทั้งหมดของโปรแกรมจะอยู่ในฟังก์ชัน main

ถ้าเราเขียนฟังก์ชันแบบรวมการ declare และ define จะทำให้ main ถูกบังคับให้ไปอยู่ล่างสุดของโปรแกรม (มีฉะนั้นจะกลายเป็น main เรียกใช้ฟังก์ชันก่อนมีการประกาศ) ซึ่งในบางกรณีการให้ main ไปอยู่ท้ายสุดก็อาจทำให้หายากกว่าการเปิดโค้ดขึ้นมาแล้วเจอเลย

นอกจากนี้การ declare ฟังก์ชันในตอนต้นยังทำให้เรารู้ด้วยว่ามีฟังก์ชันใดพร้อมใช้งานในตัวโค้ดด้วย ทำให้สามารถส่งต่อโค้ดให้นักพัฒนารุ่นต่อไปได้ง่าย

การประกาศและนิยามฟังก์ชันแยกจากกัน

ประโยชน์ของการแยกการ declare และ define

3. สามารถเขียนโค้ดแยกไฟล์กันได้ คือสามารถ include ไฟล์อื่นเพื่อเรียกใช้ฟังก์ชันในไฟล์นั้นได้เลย (ตอนสอบใช้ไม่ได้) วิธีการนี้จำเป็นต้องมีการ declare ฟังก์ชันที่จะเรียกใช้ไว้ในไฟล์หลัก

การประกาศและนิยามฟังก์ชันแยกจากกัน

วิธีการเขียนโค้ดเพื่อ declare

เราเรียกฟังก์ชันที่ถูกประกาศไว้ว่า ต้นแบบของฟังก์ชัน (function prototype)

- วิธีประกาศต้นแบบก็คือให้เราระบุชนิดข้อมูล ชื่อฟังก์ชัน และ พารามิเตอร์ จากนั้นให้ปิดการประกาศด้วยเครื่องหมาย semicolon เลย เช่น

```
int max( int x, int y );  
double findCircleArea( double radius );  
int add_mult( int x, int y );
```

- จุดแตกต่างจากการเขียนแบบรวมกันคือเราไม่เปิดวงเล็บปีกกาเพื่อเขียนวิธีคำนวณ แต่เราปิดการประกาศด้วยเซมิโคลอนทันที

การประกาศและนิยามฟังก์ชันแยกจากกัน

วิธีการเขียนโค้ดเพื่อ define

เขียนเหมือนแบบ declare และ define รวมกัน ไม่มีข้อแตกต่าง แต่จะเขียนตรงไหนก็ได้หลังจากบรรทัดที่ทำการ declare ไปแล้ว จะก่อน main หรือหลัง main หรือไปอยู่ใต้ฟังก์ชันที่ต้องเรียกใช้ก็ได้

สังเกตว่า การเขียนแยก declare และ define ก็คือการเพิ่มบรรทัด declare มา 1 บรรทัด ก่อนที่เราจะเรียกใช้ฟังก์ชันเท่านั้นเอง

การประกาศและนิยามฟังก์ชันแยกจากกัน

เมื่อเราต้องทำการเขียนโค้ดแบบแยกการ declare และ define โค้ดที่ได้จะมาในทรงนี้

Declare; prototype

```
int max( int x, int y );
```

```
//Whatever
```

Define; definition

```
int max(int x, int y) {  
    if(x > y) return x;  
    else return y;  
}
```

การประกาศและนิยามฟังก์ชันแยกจากกัน

ตัวอย่างโค้ดที่ใช้ได้จริง โค้ดสำหรับหาค่า max ระหว่าง x และ y

```
#include <stdio.h>

int max( int x, int y );
void main() {
    int x, y;
    scanf("%d%d", &x, &y);
    printf("%d\n", max(x, y));
}

int max(int x, int y) {
    if(x > y) return x;
    else return y;
}
```

Outline



- ฟังก์ชัน
- ประเภทและตัวอย่างของฟังก์ชัน
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์
- ข้อบังคับของการใช้ฟังก์ชันในภาษาซี
- การประกาศและนิยามฟังก์ชันแยกจากกัน
- **ประเภทของฟังก์ชันสร้างเอง**

ประเภทของฟังก์ชันสร้างเอง

1. ฟังก์ชันที่รับค่าเข้ามาและคำนวณผลลัพธ์กลับไปให้ผู้เรียก
2. **ฟังก์ชันที่ไม่รับค่าเข้ามา แต่คำนวณผลลัพธ์กลับไปให้ผู้เรียก**
3. ฟังก์ชันที่ไม่รับค่าเข้ามา และไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก (subroutine หรือ procedure อีกแบบหนึ่ง)
4. ฟังก์ชันที่รับค่าเข้ามา แต่ไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก (subroutine หรือ procedure แบบหนึ่ง)

ประเภทของฟังก์ชันสร้างเอง

2. ฟังก์ชันที่ไม่รับค่าเข้ามาและคำนวณผลลัพธ์กลับไปให้ผู้เรียก

- เขียนได้ดังนี้

```
int random() { ... }
```

- ย้ำว่าถ้าไม่มีวงเล็บจะผิด

```
int random { ... }
```

- ถ้าจะเขียน prototype เขียนแบบตรงตัวแปรธรรมดา

```
int random;
```

- ถ้ากลัวสับสน สร้างวงเล็บ แล้วใส่ void ไว้ก็ได้

```
int random(void);
```

```
int random(void) {...}
```

ประเภทของฟังก์ชันสร้างเอง

เทคนิคบางอย่างเกี่ยวกับการส่งข้อมูลให้ฟังก์ชัน

- ในบางครั้งเราไม่ต้องส่งค่าผ่านพารามิเตอร์เข้าไปในฟังก์ชัน แต่ใช้ตัวแปร global ในการส่งข้อมูลเข้าไปในฟังก์ชันก็ได้

```
int array[10];

float average() {
    float sum = 0;
    int i;
    for(i = 0; i < 10; ++i){
        sum += array[i];
    }
    return sum / 10;
}

void main(){...}
```

ประเภทของฟังก์ชันสร้างเอง

1. ฟังก์ชันที่รับค่าเข้ามาและคำนวณผลลัพธ์กลับไปให้ผู้เรียก
2. ฟังก์ชันที่ไม่รับค่าเข้ามา แต่คำนวณผลลัพธ์กลับไปให้ผู้เรียก
3. **ฟังก์ชันที่ไม่รับค่าเข้ามา และไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก (subroutine หรือ procedure อีกแบบหนึ่ง)**
4. ฟังก์ชันที่รับค่าเข้ามา แต่ไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก (subroutine หรือ procedure แบบหนึ่ง)

ประเภทของฟังก์ชันสร้างเอง

3. ฟังก์ชันที่ไม่รับค่าเข้ามา และไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก (subroutine หรือ procedure อีกแบบหนึ่ง)

- เช่น ฟังก์ชันที่มีการทำงานและแสดงผลลัพธ์แล้วเสร็จในฟังก์ชัน
- เราเห็นมาก่อนหน้าแล้วในฟังก์ชัน `void main() { ... }`
- นั่นคือ หากเราไม่ต้องการคืนผลลัพธ์กลับไปหาผู้เรียก เราจะใช้ `void` เป็นชนิดข้อมูลของผลลัพธ์ฟังก์ชัน
- `void` แปลว่า ‘ว่างเปล่า’ นั่นเอง

ประเภทของฟังก์ชันสร้างเอง

สรุปได้ว่าเราสามารถใช้ void ได้อย่างน้อยสองแบบในการกำหนดชนิดข้อมูล
คือ ใช้ระบุว่า

- ฟังก์ชันไม่มีพารามิเตอร์
- ฟังก์ชันไม่มีผลลัพธ์คืนกลับไปให้ผู้เรียก

ประเภทของฟังก์ชันสร้างเอง

1. ฟังก์ชันที่รับค่าเข้ามาและคำนวณผลลัพธ์กลับไปให้ผู้เรียก
2. ฟังก์ชันที่ไม่รับค่าเข้ามา แต่คำนวณผลลัพธ์กลับไปให้ผู้เรียก
3. ฟังก์ชันที่ไม่รับค่าเข้ามา และไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก
(subroutine หรือ procedure อีกแบบหนึ่ง)
4. **ฟังก์ชันที่รับค่าเข้ามา แต่ไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก
(subroutine หรือ procedure แบบหนึ่ง)**

ประเภทของฟังก์ชันสร้างเอง

4. ฟังก์ชันที่รับค่าเข้ามา แต่ไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก (subroutine หรือ procedure แบบหนึ่ง)

- เช่น ฟังก์ชันที่มีการทำงานและแสดงผลลัพธ์แล้วเสร็จในฟังก์ชัน

```
void order_numbers(int x, int y) {  
    if(x < y) {  
        printf("%d %d", x, y);  
    } else {  
        printf("%d %d", y, x);  
    }  
}
```

Tips ท้ายคาบ



Tips: เรื่องน่ารู้เกี่ยวกับฟังก์ชันที่ไม่คืนผลลัพธ์กลับไป

บางภาษาเช่น ภาษาเบสิก และภาษาปาสคาล จะเรียกฟังก์ชันที่ไม่คืนผลลัพธ์ กลับไปว่าเป็น subroutine หรือ procedure ซึ่งแปลว่า ขั้นตอนการทำงานย่อย หรือ ขั้นตอนการทำงานทั่วไปเพราะจากตัวคณิตศาสตร์ ฟังก์ชันจะมีการคำนวณคำตอบ กลับมาเสมอ

บางภาษาเคร่งครัดกับสิ่งที่ไปในคณิตศาสตร์จึงเรียกโปรแกรมย่อยที่ไม่คืนผลลัพธ์กลับไปเป็นชื่ออื่นแทน (subroutine หรือ procedure) แต่ทางภาษาซี และ ภาษาจำนวนมากไม่สนใจที่จุดนี้และเรียกโปรแกรมย่อยทุกอย่างเป็นฟังก์ชันไปหมด

- ในระดับของเรา ขอเพียงโปรแกรมเมอร์เปลี่ยนชนิดข้อมูลให้ถูกก็พอ
- ภาษาที่ให้ความสนใจในจุดนี้จะมีคำสำคัญมาแยกประเภทโปรแกรมย่อย

สรุป



- ฟังก์ชัน (โปรแกรมย่อย ทำให้โปรแกรมเป็นระเบียบ พัฒนาได้ง่ายเมื่อโค้ดยาว)
- ประเภทและตัวอย่างของฟังก์ชัน (ฟังก์ชันมาตรฐาน/ฟังก์ชันสร้างเอง)
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน (มีหลายส่วน สรุปตรงนี้ไม่พอ กลับไปทบทวน)
- ฟังก์ชันคอมพิวเตอร์และฟังก์ชันคณิตศาสตร์ (เป็นสิ่งเดียวกัน แต่อย่าสับสน Syntax)
- ข้อบังคับของการใช้ฟังก์ชันในภาษาซี (รายละเอียดและวิธีการใช้ฟังก์ชันในรูปแบบต่างๆ)
- การประกาศและนิยามฟังก์ชันแยกจากกัน (ทำให้ไขว้ฟังก์ชันได้/เป็นระเบียบ/เรียกข้ามไฟล์)
- ประเภทของฟังก์ชันสร้างเอง (มี 4 แบบ รับและคืน, ไม่รับและคืน, ไม่รับไม่คืน และ รับและไม่คืน)