การเขียนโปรแกรมคอมพิวเตอร์ 1

Computer Programming I

ฟังก์ชัน (โปรแกรมย่อย)

ภิญโญ แท้ประสาทสิทธิ์

Emails : pinyotae+111 at gmail dot com, pinyo at su.ac.th

Web : http://www.cs.su.ac.th/~pinyotae/compro1/

Facebook Group : ComputerProgramming@CPSU

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

สัปดาห์ที่ 10

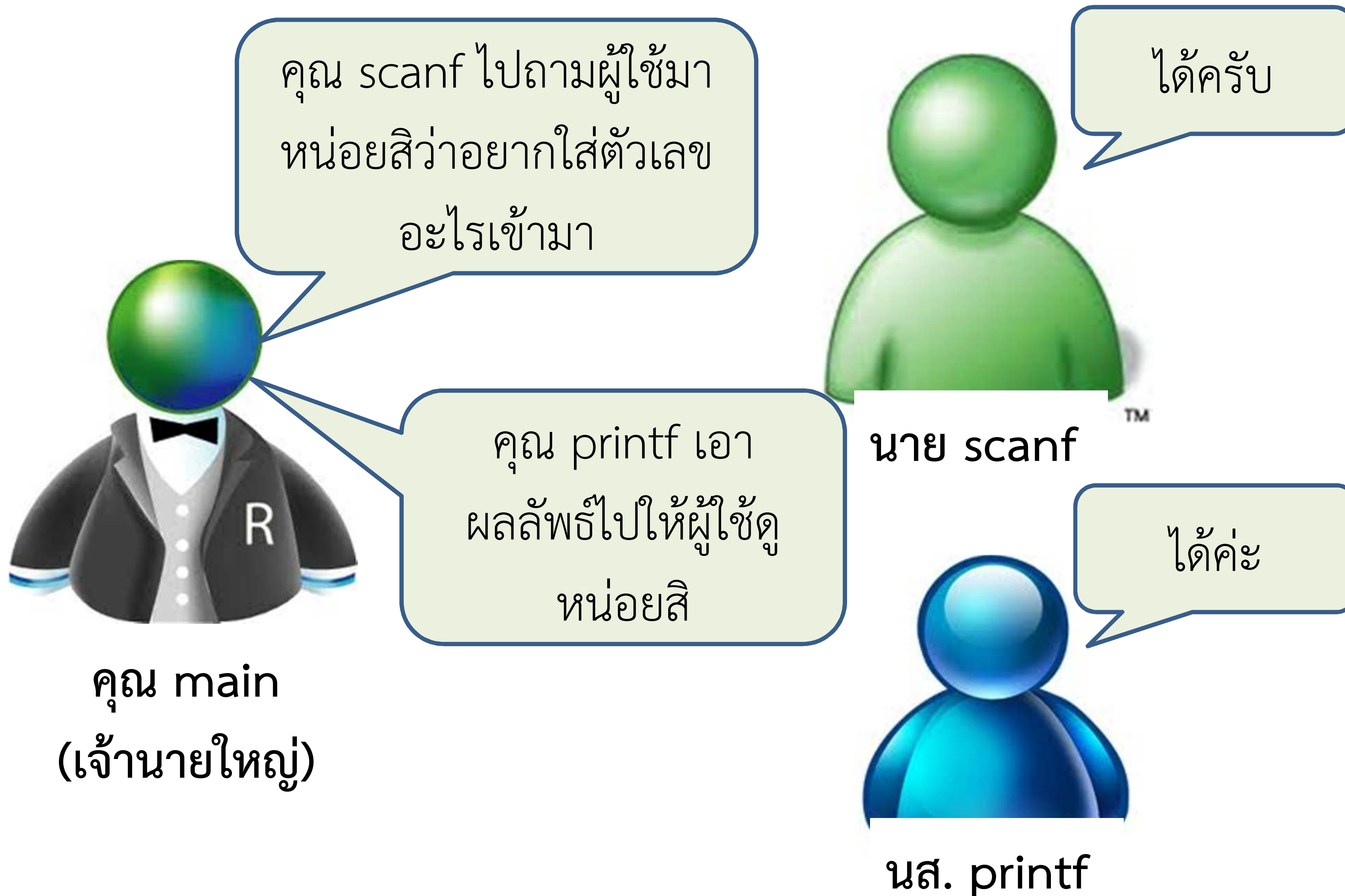
หัวข้อเนื้อหา

- ฟังก์ชัน (Function) หรือ โปรแกรมย่อยคืออะไร
- ประเภทและตัวอย่างของฟังก์ชัน
  - ฟังก์ชันมาตรฐาน (Standard function)
  - ฟังก์ชันสร้างเอง (User-defined function)
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์กับฟังก์ชันคณิตศาสตร์
- กฎเหล็กของฟังก์ชัน
  - กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน
  - ฟังก์ชันกับการรับค่า (parameter) และการส่งผลลัพธ์ให้ผู้เรียก
- การประกาศและนิยามฟังก์ชันแยกจากกัน
- ประเภทของฟังก์ชันสร้างเอง

ฟังก์ชัน (Function) หรือ โปรแกรมย่อยคืออะไร

- ฟังก์ชันหรือโปรแกรมย่อยคือสิ่งที่สามารถถูกเรียกงานใช้ซ้ำ ๆ ได้จากโปรแกรมส่วนอื่น ๆ
  - ที่ผ่านมาเราใช้ main ซึ่งเป็นโปรแกรมหลักในการเรียกใช้ฟังก์ชัน printf และ scanf
  - สังเกตด้วยว่าใน main ของเราเรียกใช้ printf ได้หลายครั้ง เรียกใช้ scanf ได้หลายครั้งด้วย จะวนลูปเรียกใช้ซ้ำก็ยังได้
- ฟังก์ชันเป็นเหมือนพนักงานที่รับงานอย่างใดอย่างหนึ่งไปทำแทน
  - เช่น รับหน้าที่ไปหาทางเอาตัวหนังสือขึ้นบนหน้าจอคอมพิวเตอร์
  - เช่น รับหน้าที่อ่านค่าจากคีย์บอร์ดแล้วนำมาเก็บไว้ในตัวแปร x ให้เรา

มโนภาพของการเรียกใช้งานฟังก์ชัน



ประโยชน์ของฟังก์ชัน

- ทำให้เกิดการแบ่งโค้ดเป็นส่วน ๆ ที่มีเป้าหมายการทำงานชัดเจน
  - เหมือนมีพนักงานประจำตำแหน่งทำหน้าที่อย่างใดอย่างหนึ่ง
  - โค้ดจะเป็นระเบียบขึ้นมาก จากฟังก์ชัน main เราให้ฟังก์ชัน main ทำหน้าที่แบ่งงานให้พนักงานประจำตำแหน่งรับหน้าที่แทน
- การแบ่งงานจะทำให้โค้ดไม่ยุ่งเหยิงดูซับซ้อนจนเกินเหตุ
  - เปรียบเหมือนกับที่คนส่วนใหญ่ไม่ต้องปลูกข้าวและทอผ้าเอง แต่กระจายงานให้เกษตรกรและโรงงานจัดการให้ผ่านการว่าจ้าง
  - ถ้าเราต้องปลูกข้าวหรือทอผ้าเองชีวิตคงวุ่นวายมาก
- การแบ่งงานทำให้โค้ดดูเข้าใจง่าย เช่น การแบ่งงานให้พนักงานชื่อ printf ทำให้เราเข้าใจได้โดยง่ายว่าเราจะแสดงผลออกทางจอภาพ

ประเภทของฟังก์ชัน

- 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 ซึ่งเป็นฟังก์ชันภายใต้หมวดหมู่นี้
- ฟังก์ชันสร้างเอง (user-defined function) คือ ฟังก์ชันที่เราเขียนขึ้นมาเอง
  - เปรียบเหมือนกับเราฝึกพนักงานขึ้นมาเอง แทนที่จะจ้างคนนอกมาทำงานให้

เรื่องน่ารู้เกี่ยวกับฟังก์ชัน

- ฟังก์ชันตัวแรกที่เรารู้จักคือฟังก์ชันหลัก ซึ่งต้องมีชื่อตายตัวว่า main
- ฟังก์ชันมาตรฐานตัวแรก ๆ ที่เรารู้จักคือฟังก์ชัน printf และ scanf
  - ฟังก์ชันมาตรฐานจะมีชื่อที่ตายตัวเช่นกัน
- ฟังก์ชันสร้างเองจะมีการทำงานเป็นไปตามที่เราอยากให้เป็น
  - เราจะเขียนให้มันทำอะไรก็ได้ จะให้มันเรียกฟังก์ชันอื่นต่อ ๆ กันไปก็ได้
  - ฟังก์ชันสร้างเองจะเรียกฟังก์ชันมาตรฐานก็ได้
  - ชื่อของมันไม่ตายตัว ขึ้นอยู่กับเราตั้ง แต่โดยส่วนมากจะตั้งชื่อตามหน้าที่ เช่น คนมักใช้ชื่อว่า max เป็นชื่อฟังก์ชันการหาค่าสูงสุด
- อย่าไปคิดว่าตั้งชื่อเสร็จแล้วฟังก์ชันสร้างเองจะทำงานตามชื่อที่ตั้ง เราต้องเขียนโปรแกรมย่อยเองทุกอย่างเพื่อระบุงานที่มันต้องทำ

โครงสร้างของฟังก์ชัน

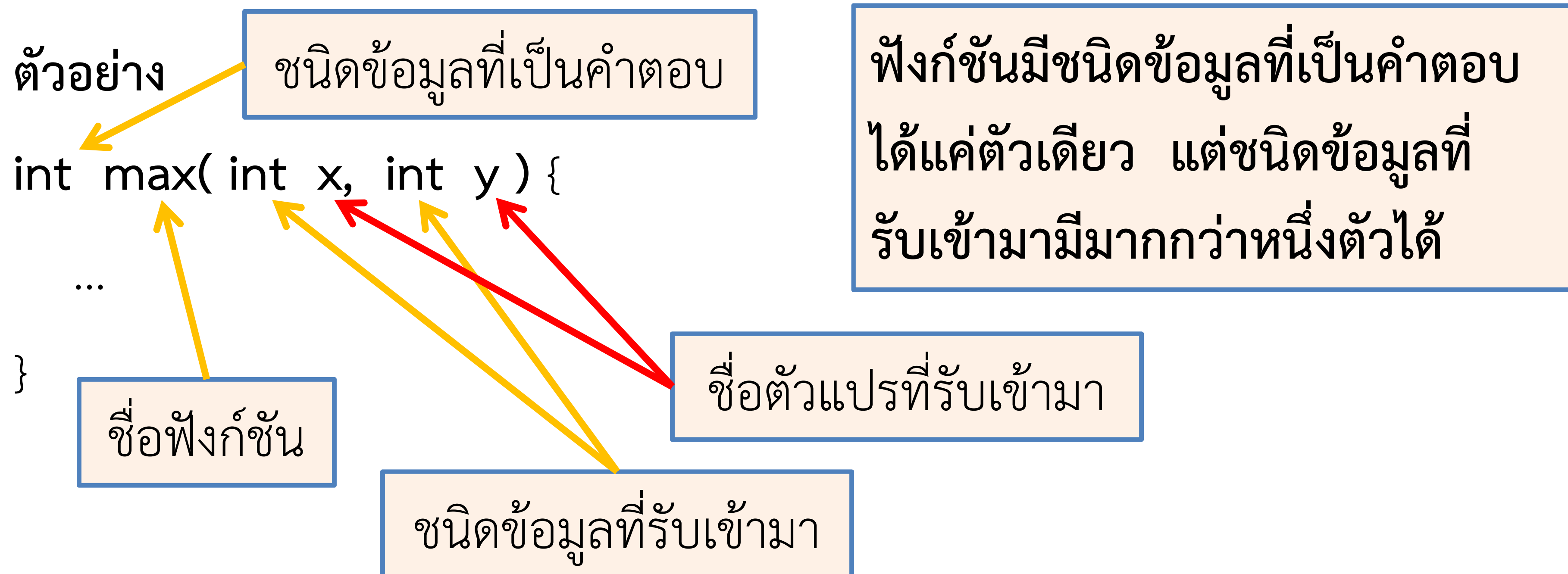
ฟังก์ชันมีโครงสร้างดังนี้

ชนิดข้อมูลที่เป็นคำตอบ ชื่อฟังก์ชัน(ชนิดข้อมูลที่รับเข้ามา ชื่อตัวแปรที่รับเข้ามา) {

....

}

ตัวอย่าง



ตัวอย่างการใช้งาน

ตัวอย่าง จงเขียนโปรแกรมที่รับค่าตัวเลขจำนวนเต็มมาสามคู่ตามลำดับดังนี้ x1 y1, x2 y2, และ x3 y3 จากนั้นให้โปรแกรมพิมพ์ค่าตัวเลขที่มีค่ามากที่สุดของแต่ละคู่ออกมาทางจอภาพ

เช่น

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

วิธีดั้งเดิมแบบแรก

```
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);
```

เราเขียนโค้ดเปรียบเทียบค่าซ้ำซากรวมสามรอบ

วิธีดั้งเดิมแบบที่สอง

```
int x1, x2, x3, y1, y2, y3;
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);
```

ถึงจะเลี่ยงการเก็บค่าผลลัพธ์ไว้ในตัวแปร เราก็ต้องเขียนโค้ดเปรียบเทียบซ้ำซากสามรอบอยู่ดี

วิธีใช้ฟังก์ชันแบบที่หนึ่ง

```
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);
}
```

เราสามารถแยกโค้ดหาค่าสูงสุดมาเป็นโปรแกรมย่อยได้

วิธีใช้ฟังก์ชันแบบที่สอง

```c
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);

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

ผลที่ได้จากฟังก์ชันจะทำตัวเหมือนตัวเลขหรือตัวอักษรตามชนิดข้อมูลของผลลัพธ์ที่ระบุไว้

เพราะฟังก์ชัน max ถูกกำหนดให้มีชนิดข้อมูลของผลลัพธ์เป็น int เราจึงเอามาใช้คู่กับ %d ของ printf ได้เลย

ตัวอย่างฟังก์ชันยอดนิยม : คำนวณพื้นที่วงกลม

ฟังก์ชันหาพื้นที่วงกลมจากค่ารัศมีที่ใส่เข้าไป

```
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);
}
```

ฟังก์ชันคอมพิวเตอร์กับฟังก์ชันคณิตศาสตร์ (1)

- เป็นของที่มีความสัมพันธ์กันโดยตรง เช่น หากเรามองไปที่ฟังก์ชันคำนวณพื้นที่วงกลมในรูปแบบคณิตศาสตร์ เราจะได้ฟังก์ชัน

$$f(r) = \pi r^2$$

- ฟังก์ชันนี้มี r เป็นตัวแปร และ คำนวณพื้นที่วงกลมได้ผลลัพธ์ออกมาเป็นตัวเลข
- ย้อนกลับมาดูฟังก์ชันในภาษาซี

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

- เห็นได้ว่าโค้ดภาษาซีเหมือนฟังก์ชันคณิตศาสตร์ คือมีค่ารัศมีเป็นตัวแปร และคำนวณพื้นที่วงกลมเป็นผลลัพธ์มาเป็นตัวเลข (ชนิด double)

ฟังก์ชันคอมพิวเตอร์กับฟังก์ชันคณิตศาสตร์ (2)

ฟังก์ชันที่เราเห็นมาก่อนหน้า เช่น max ก็เขียนบรรยายในรูปคณิตศาสตร์ได้เหมือนกัน

- การแยกกรณีในฟังก์ชันคณิตศาสตร์จะสัมพันธ์กับการใช้ if ในภาษา C
- เงื่อนไขในฟังก์ชันคณิตศาสตร์กับ if เป็นเงื่อนไขอันเดียวกัน

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

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

คำว่า otherwise แปลว่า ‘มิฉะนั้น’ หรือ ‘หาไม่แล้ว’ ซึ่งในทางคณิตศาสตร์หรือภาษาโปรแกรมจะใส่ไว้ในอันดับสุดท้ายเพื่อบอกว่า ‘มิฉะนั้นก็จะทำ ...’ หรือ ‘มิฉะนั้นก็จะมีค่าเท่ากับ ...’

- จุดนี้ตรงกับการใช้ else ในภาษาซี

มาดูฟังก์ชันคณิตศาสตร์ที่เราอุตส่าห์ทำมาตอนต้นเทอม

<u>ฟังก์ชันปลากระป๋องแบบพื้นฐาน</u>

โจทย์ ปลากระป๋องยี่ห้อหนึ่งใช้ปลาซาร์ดีนสามตัวและมะเขือเทศสองผลเพื่อผลิตปลากระป๋องหนึ่งกระป๋อง หากโรงงานผลิตมีปลาซาร์ดีนอยู่ X ตัวและมะเขือเทศอยู่ Y ผล โรงงานจะผลิตปลากระป๋องได้ทั้งหมดกี่กระป๋อง

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

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

แต่ก่อนเราเขียนโปรแกรมว่า ...

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

    if(x / 3 < y / 2)
        printf("%d", x / 3);
    else
        printf("%d", y / 2);
}
```

ถ้าเราจัดเป็นฟังก์ชันจะได้เป็น ...

```
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));
}
```

แยกการคำนวณที่ซับซ้อนออกจากกันแล้วจะทำให้ดูออกง่ายว่าโปรแกรมคิดจะทำอะไร เพราะชื่อฟังก์ชันมักจะสื่อความหมายอยู่แล้ว

ฟังก์ชันทดสอบว่าเป็นเลขคู่หรือไม่

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

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

ฟังก์ชันทดสอบว่าเป็นเลขคี่หรือไม่

จงเขียนฟังก์ชันที่คืนผลลัพธ์เป็นเลข 1 (ค่าจริง) เมื่อตัวเลขที่รับมาเป็นอินพุตเป็นเลขคี่ และคืนเลข 0 (ค่าเท็จ) เมื่อตัวเลขที่รับมาเป็นเป็นเลขคู่

แนวคิด เราใช้วิธีกลับค่าความจริงจากฟังก์ชัน isEven ที่เขียนมาเมื่อสักครู่ได้

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

การใช้ ! เป็นการกลับค่าความจริง เลขศูนย์ซึ่งแทนค่าเท็จจะถูกกลับเป็นเลขหนึ่ง และเลขหนึ่งแทนค่าจริงจะถูกกลับเป็นเลขศูนย์

ทดลองด้วยตัวเอง

```
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));
}
```

สำคัญมาก

เอากลับไปลองทำที่บ้านด้วย

หัวข้อเนื้อหา

- ฟังก์ชัน (Function) หรือ โปรแกรมย่อยคืออะไร
- ประเภทและตัวอย่างของฟังก์ชัน
  - ฟังก์ชันมาตรฐาน (standard function)
  - ฟังก์ชันสร้างเอง (User-defined function)
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์กับฟังก์ชันคณิตศาสตร์
- **กฎเหล็กของฟังก์ชัน**
  - **กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน**
  - ฟังก์ชันกับการรับค่า (parameter) และการส่งผลลัพธ์ให้ผู้เรียก
- การประกาศและนิยามฟังก์ชันแยกจากกัน
- ประเภทของฟังก์ชันสร้างเอง

กฎของการเรียกใช้ฟังก์ชัน

- ฟังก์ชันจะถูกเรียกได้ ณ จุดเรียกใช้ที่มาหลังการประกาศฟังก์ชัน
  - ดังนั้นหากเราประกาศฟังก์ชันไว้ก่อน main ฟังก์ชันก็จะถูกเรียกใช้ได้ใน main
  - เพราะจุดเรียกใช้อยู่ภายใน main ซึ่งตามหลังฟังก์ชันที่เราประกาศไว้
- 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

วิธีจัดวางฟังก์ชัน

เพราะจุดที่มีการเรียกฟังก์ชันต้องมาหลังจากการประกาศฟังก์ชัน

- การจัดวางที่ถูกต้องก็คือให้ประกาศฟังก์ชันที่จะถูกเรียกไว้ด้านบน และในฟังก์ชันที่ทำการเรียก (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

ฟังก์ชันกับการรับค่า (parameter)

- ฟังก์ชันสามารถรับค่าตัวเลขหรือตัวแปรได้ดังที่แสดงไว้ก่อนหน้า

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

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

- ค่าที่รับส่งกันมาระหว่างฟังก์ชันเรียกว่าพารามิเตอร์
- ในมุมมองของตัวฟังก์ชันเองค่าที่รับมาจะอยู่ในรูปของตัวแปร ในตัวอย่างนี้คือ ตัวแปร x, y และ radius

พารามิเตอร์ในมุมมองของผู้เรียกฟังก์ชัน (1)

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

พารามิเตอร์ในมุมมองของผู้เรียกฟังก์ชัน (2)

```
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

พารามิเตอร์ในมุมมองของผู้เรียกกับของฟังก์ชัน (1)

- จะเห็นได้ว่าผู้เรียกจะปฏิบัติกับพารามิเตอร์เป็นค่า ๆ หนึ่ง ซึ่งค่านั้นจะเป็นค่าคงที่หรือตัวแปรก็ได้
- ส่วนตัวฟังก์ชันจะมองพารามิเตอร์เป็นตัวแปรตลอด
- เหมือนตอนเขียนฟังก์ชันคณิตศาสตร์ เราก็เขียนติดตัวแปรตลอด

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

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

พารามิเตอร์ในมุมมองของผู้เรียกกับของฟังก์ชัน (2)

- ผู้เรียกฟังก์ชันเป็นคนส่งค่าเข้าไปเพราะต้องการผลลัพธ์ ดังนั้นผู้เรียกต้องเตรียมค่าไว้ก่อนเป็นค่าใดค่าหนึ่ง
- การเตรียมค่าพารามิเตอร์ของผู้เรียกฟังก์ชันจะกำหนดไว้เป็นค่าคงที่ตายตัวในโปรแกรม หรือให้ผู้ใช้ใส่เข้ามาในตัวแปรก็ได้
- ส่วนฟังก์ชันที่ถูกเรียกต้องเตรียมวิธีคำนวณเอาไว้เพื่อให้คำนวณผลลัพธ์จากค่าต่าง ๆ ได้อย่างถูกต้อง
  - ฟังก์ชันมองพารามิเตอร์เป็นตัวแปรตลอดทำให้มีความยืดหยุ่นในการจัดการค่าที่แตกต่างกันไป
  - ถ้าผู้เรียกใส่ค่าคงที่เข้ามา ฟังก์ชันก็เพียงนำค่าคงที่นั้นไปใส่ไว้ในตัวแปรที่เป็นพารามิเตอร์ (เช่นใส่ค่า y ให้เท่ากับ 2 ในตัวอย่างที่ผ่านมา)

ชื่อตัวแปรพารามิเตอร์ของผู้เรียกและของฟังก์ชัน

ชื่อตัวแปรพารามิเตอร์ของผู้เรียกและของฟังก์ชันไม่จำเป็นต้องเหมือนกัน

- เพราะแท้จริงแล้วผู้เรียกสนใจแค่ว่าค่าที่จะส่งไปมีค่าเท่าไหร่และชนิดข้อมูลตรงกันหรือไม่
- ฟังก์ชันที่ถูกเรียกจะดึงค่าที่ถูกส่งมาไปเก็บไว้ในตัวแปร ไม่ได้สนใจว่าค่าที่ส่งมาเป็นตัวแปรหรือค่าคงที่
- แต่ก็สนใจว่าชนิดข้อมูลตรงกันหรือเปล่า

```
void main() {
    int a, b;
    scanf("%d %d", &a, &b);
    int max_val  = max(a, b);
    printf("max value = %d", max_val);
}
```

ชื่อตัวแปรใน main กับของฟังก์ชัน max ไม่เหมือนกันก็ได้

โปแกรมนี้ให้ผลลัพธ์เป็นอย่างไร

เราส่งตัวแปร x และ y จาก main ไปให้ sub สองครั้ง

ผลลัพธ์ที่ได้จะเป็นอย่างไรและเหมือนกันทั้งสองครั้งหรือไม่

```
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));
}
```

การส่งพารามิเตอร์สำคัญที่ลำดับค่าไม่ใช่ชื่อตัวแปร

- มือใหม่มักจะหลงคิดว่าต้องทำชื่อตัวแปรของผู้เรียกให้ตรงกับชื่อตัวแปรพารามิเตอร์ในฟังก์ชัน <u>แต่นั่นเป็นความคิดที่ผิด</u>
- เรื่องนี้ดูออกง่ายเพราะถ้าต้องทำชื่อให้เหมือนกัน เราก็คงจะส่งค่าคงที่ไปเป็นพารามิเตอร์ไม่ได้
- ดังนั้นเราต้องคอยดูเป็นอันดับแรกเลยว่าลำดับค่าที่ส่งไปตรงกับลำดับที่ฟังก์ชันบอกให้เป็นหรือเปล่า ชนิดข้อมูลตรงกันทุกลำดับหรือเปล่า
- ถ้าไม่ใส่ใจเรื่องลำดับ โปรแกรมมักจะผิด โดยเฉพาะตอนที่พารามิเตอร์มีชนิดข้อมูลที่เหมือนกันหมด

ฟังก์ชันที่สลับลำดับพารามิเตอร์ก็ไม่มีผล

ฟังก์ชันบางอย่างก็ให้ผลลัพธ์โดยไม่ขึ้นอยู่กับลำดับของพารามิเตอร์ เช่น

- การหาค่าสูงสุดของพารามิเตอร์ทั้งสอง

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

- การหาค่าสัมบูรณ์ของผลต่างของพารามิเตอร์ทั้งสอง

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

การเปลี่ยนค่าตัวแปรพารามิเตอร์ในฟังก์ชัน

- จากหน้าที่แล้ว เรากล่าวว่า "ฟังก์ชันที่ถูกเรียกจะดึงค่าที่ถูกส่งมาไปเก็บไว้ในตัวแปรไม่ได้สนใจว่าค่าที่ส่งมาเป็นตัวแปรหรือค่าคงที่"
  - พูดง่าย ๆ ก็คือคนเรียกใช้จะส่งค่าตัวแปรหรือค่าคงที่มาให้ มันก็เป็นเพียงค่าค่าหนึ่งที่ฟังก์ชันจะนำไปเก็บไว้ในตัวแปรของมันเอง
  - หากฟังก์ชันทำการเปลี่ยนค่าตัวแปรพารามิเตอร์ ผลก็เกิดแต่ข้างในฟังก์ชัน ไม่ได้เปลี่ยนอะไรที่ผู้เรียกเลย
- พูดง่าย ๆ ก็คือผู้เรียกฟังก์ชันไม่ได้รับผลกระทบใด ๆ จากการเปลี่ยนค่าตัวแปรพาราในฟังก์ชัน ตัวแปรที่ส่งไปมีค่าเท่าใดกลับมาก็มีค่าเท่านั้น
  - เรื่องนี้สมเหตุสมผลดี เพราะถ้าผู้เรียกส่งค่าคงที่เข้าไปจะให้ฟังก์ชันมันเปลี่ยนค่าคงที่ได้ก็คงเป็นเรื่องประหลาด

ตัวอย่างการเปลี่ยนค่าตัวแปรพารามิเตอร์

```
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);
}
```

ได้เลข 5 เหมือนตอนต้น

ได้เลข 14

การส่งผลลัพธ์ให้ผู้เรียก

- ตอนเรียกผู้ใช้ต้องให้ความสำคัญกับชื่อฟังก์ชันและค่าของพารามิเตอร์
- ส่วนตอนที่รับค่ากลับมาผู้เรียกจะมองฟังก์ชันว่าเป็นเพียงค่าคงที่
  - จะเอาค่าคงที่นั้นไปเก็บไว้ในตัวแปรก็ได้
  - จะเอาค่าคงที่นั้นไปแสดงผลก็ได้
- ผู้เขียนโปรแกรมจึงต้องเข้าใจลำดับการพิจารณา
  - เป็นไปตามลำดับการทำงานจริงของโปรแกรม คือ
    เตรียมค่าพารามิเตอร์ → เรียกฟังก์ชัน → ฟังก์ชันคืนผลลัพธ์กลับมา
  - ตอนเตรียมค่าเราเลือกก่อนว่าจะเป็นค่าคงที่ธรรมดาหรือเป็นตัวแปร
  - ตอนเรียกเรามองว่าชื่อฟังก์ชันกับลำดับพารามิเตอร์ตรงกันหรือไม่
  - ตอนกลับมาต้องดูว่าชนิดข้อมูลตรงกับตัวแปรที่จะเก็บค่าไว้หรือเปล่า

ตัวอย่างการส่งผลลัพธ์และรับค่าผลลัพธ์จากฟังก์ชัน

```
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);
}
```

เพราะว่า findCircleArea แท้จริงคือ double ดังนั้นเราจึงควรใช้ตัวแปรประเภท double มาเก็บผลลัพธ์ไว้ ถ้าใช้ area2 มาเก็บผลลัพธ์จะคลาดเคลื่อน

ลำดับการคำนวณที่ชวนงง

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

ลำดับการทำงานที่แท้จริงก็คือว่า ฟังก์ชัน findCircleArea จะรับพารามิเตอร์ไปคิดผลลัพธ์ก่อน จากนั้นจึงค่อยคืนผลลัพธ์มาไว้ที่ area1

- เรื่องนี้ทำให้บางคนชอบเขียนอะไรผิด ๆ ว่า

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

  ข้างบนนี้ผิดเพราะการกำหนดค่าตัวแปรนั้นมีกฎเหล็กอยู่ว่า ตัวแปรที่เราจะเปลี่ยนค่าจะต้องอยู่ทางด้านขวาของเครื่องหมายเท่ากับ
- ลำดับการคิดเกิดขึ้นทางด้านขวาของเครื่องหมายเท่ากับก่อนเสมอ เช่น int x = a + b; สังเกตว่าแต่ก่อนเราก็คิดคำนวณทางขวาให้เสร็จก่อน

หัวข้อเนื้อหา

- ฟังก์ชัน (Function) หรือ โปรแกรมย่อยคืออะไร
- ประเภทและตัวอย่างของฟังก์ชัน
  - ฟังก์ชันมาตรฐาน (standard function)
  - ฟังก์ชันสร้างเอง (User-defined function)
- โครงสร้างของฟังก์ชันและตัวอย่างการใช้งาน
- ฟังก์ชันคอมพิวเตอร์กับฟังก์ชันคณิตศาสตร์
- กฎเหล็กของฟังก์ชัน
  - กฎของการเรียกใช้ฟังก์ชัน และวิธีจัดวางฟังก์ชัน
  - ฟังก์ชันกับการรับค่า (parameter) และการส่งผลลัพธ์ให้ผู้เรียก
- **การประกาศและนิยามฟังก์ชันแยกจากกัน**
- ประเภทของฟังก์ชันสร้างเอง

การประกาศและนิยามฟังก์ชัน

- แท้จริงแล้วคำว่าการประกาศ (declare) กับการนิยาม (define) ฟังก์ชันเป็นสิ่งที่แตกต่างกัน
- ในตัวอย่างก่อนหน้าทั้งหมดเราทำการ declare และ define พร้อมกัน
- แล้วอะไรที่เรียกว่า declare อะไรที่เรียกว่า define
  - การระบุชนิดข้อมูล ชื่อฟังก์ชัน และ พารามิเตอร์ คือการ declare (คือการประกาศว่ามีฟังก์ชันนี้อยู่)
  - การระบุว่าฟังก์ชันทำงานอย่างไร (โค้ดข้างในฟังก์ชัน) คือการ define (การนิยามคือการระบุรายละเอียดทางการคำนวณ)

ต้นแบบฟังก์ชัน

- 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);
- **จุดแตกต่างจากเดิมมีเพียงว่า เราไม่เปิดวงเล็บปีกกาเพื่อเขียนวิธีคำนวณ แต่เราปิดการประกาศด้วยเซมิโคลอนทันที**

การประกาศและนิยามฟังก์ชันแยกออกจากกัน

ทำได้ตามขั้นตอนดังนี้

1. ให้เราระบุต้นแบบไว้ด้านบนก่อน
2. ณ จุดที่เราต้องการนิยามฟังก์ชัน ให้เราเขียนซ้ำต้นแบบอีกครั้ง แต่
   i. แทนที่เราจะปิดต้นแบบด้วยเซมิโคลอน เราจะเปิดวงเล็บปีกกา
   ii. เราระบุวิธีคำนวณของฟังก์ชันไว้ภายในวงเล็บปีกกาเหมือนกับที่เราทำมาก่อนหน้านี้

ตัวอย่าง

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

prototype

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

Definition

ผลที่ได้จะเหมือนกับว่าเราเขียนทุกอย่างเหมือนเดิม แต่มี function prototype เพิ่มขึ้นมาจากเดิม

แล้วจะเขียนแยกกันไปทำไมเนี่ย?

ประโยชน์ของการประกาศและนิยามฟังก์ชันแยกกัน

1. ทำให้เราสามารถนิยามฟังก์ชันไว้ด้านใต้ของจุดที่ทำการเรียกใช้ได้

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

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

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

2. สามารถประกาศ prototype กับนิยามตัวฟังก์ชันไว้คนละไฟล์ได้

ประโยชน์ที่สำคัญที่สุดของการประกาศและนิยามแยกกัน

1. กระจายการใช้งานฟังก์ชันไปหลาย ๆ ไฟล์ได้
   ด้วยการเขียนเฉพาะ prototype ใหม่ ไม่ต้องเขียนนิยามใหม่
2. ทำให้ฟังก์ชันสองอันเรียกใช้งานกันเองได้ เพราะจุดเรียกใช้อยู่ภายใต้การประกาศฟังก์ชันด้วยกันทั้งคู่

```
int  A(int x);
int  B(int y);

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

ถ้าเป็นแบบข้างล่างนี้ จะมีแค่ฟังก์ชันเดียวที่จะเรียกฟังก์ชันอีกอันได้

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

ประเภทของฟังก์ชันสร้างเอง

1. ฟังก์ชันที่รับค่าเข้ามาและคำนวณผลลัพธ์กลับไปให้ผู้เรียก
2. ฟังก์ชันที่ไม่รับค่าเข้ามา แต่คำนวณผลลัพธ์กลับไปให้ผู้เรียก

3. ฟังก์ชันที่ไม่รับค่าเข้ามา และไม่คำนวณผลลัพธ์กับไปให้ผู้เรียก (subroutine หรือ procedure อีกแบบหนึ่ง)
4. ฟังก์ชันที่รับค่าเข้ามา แต่ไม่คำนวณผลลัพธ์กลับไปให้ผู้เรียก (subroutine หรือ procedure แบบหนึ่ง)

ตัวอย่างฟังก์ชันที่ไม่รับค่าเข้ามาแต่คำนวณผลกลับไป

ฟังก์ชันที่ไม่รับค่าเข้ามา แต่คำนวณผลลัพธ์กลับไปให้ผู้เรียก

```
int random() { … }
```

ตรงที่เป็นพารามิเตอร์จะว่าง แต่ก็ต้องมีวงเล็บไว้

(ไม่อย่างนั้นเราจะแยกไม่ออกเลยว่ามันเป็นฟังก์ชันหรือตัวแปรธรรมดา)

```
int random { … } แบบนี้ผิด
```

ถ้าเราจะเขียน prototype ก็จะยิ่งวุ่นเข้าไปอีก

```
int random; ทีนี้หน้าตากลายเป็นตัวแปร int ธรรมดา
```

ตรงวงเล็บเปล่าเราสามารถใส่คำว่า void เข้าไปก็ได้

```
int random( void ) { … }
```

เทคนิคบางอย่างเกี่ยวกับการส่งข้อมูลเข้าให้ฟังก์ชัน

- ในบางทีเราไม่ส่งข้อมูลเข้าให้กับฟังก์ชันผ่านพารามิเตอร์
- เราอาจจะใช้ตัวแปรแบบ global ในการส่งข้อมูลเข้าให้กับฟังก์ชัน

```
int array[10];

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

    return sum / 10;
}
```

ตัวอย่างฟังก์ชันที่ไม่รับค่าเข้ามาและไม่คำนวณผลกลับไป

- เช่น ฟังก์ชันที่มีการทำงานและแสดงผลลัพธ์แล้วเสร็จในฟังก์ชัน
- เราเห็นมาก่อนหน้าแล้วในฟังก์ชัน void main( ) { … }
- นั่นคือ หากเราไม่ต้องการคืนผลลัพธ์กลับไปหาผู้เรียก
  → เราใช้ void มาเป็นชนิดข้อมูลของผลลัพธ์ฟังก์ชัน
  void ตรงนี้แปลว่า ‘ไม่มี’

```
void main() { … }
```

- สรุปได้ว่าเราสามารถใช้ void ได้อย่างน้อยสองแบบในการกำหนดชนิดข้อมูล คือ ใช้ระบุว่า
  - ฟังก์ชันไม่มีพารามิเตอร์
  - ฟังก์ชันไม่มีผลลัพธ์คืนกลับไปให้ผู้เรียก

ตัวอย่างฟังก์ชันที่รับค่าเข้ามาแต่ไม่คำนวณผลกลับไป

เช่น ฟังก์ชันที่มีการทำงานและแสดงผลลัพธ์แล้วเสร็จในฟังก์ชัน

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

ฟังก์ชันที่เขียนผลลัพธ์ลงไฟล์บางทีก็เป็นจำพวกนี้เหมือนกัน

หรือฟังก์ชันที่ส่งผลลัพธ์กลับไปผ่าน

เรื่องน่ารู้เกี่ยวกับฟังก์ชันที่ไม่คืนผลลัพธ์กลับไป

- บางภาษาเช่น ภาษาเบสิค และภาษาปาสคาล จะเรียกฟังก์ชันที่ไม่คืนผลลัพธ์กลับไปว่าเป็น subroutine หรือ procedure ซึ่งแปลว่า ขั้นตอนการทำงานย่อย หรือ ขั้นตอนการทำงานทั่วไป
- เพราะจากตัวคณิตศาสตร์ ฟังก์ชันจะมีการคำนวณคำตอบกลับมาเสมอ
- บางภาษาเคร่งครัดกับสิ่งที่เป็นไปในคณิตศาสตร์จึงเรียกโปรแกรมย่อยที่ไม่คืนผลลัพธ์กลับไปเป็นชื่ออื่นแทน (subroutine หรือ procedure)
- แต่ทางภาษาซี และภาษาจำนวนมากไม่สนใจที่จุดนี้และเรียกโปรแกรมย่อยทุกอย่างเป็นฟังก์ชันไปหมด
  - ขอเพียงโปรแกรมเมอร์เปลี่ยนชนิดข้อมูลให้ถูกก็พอ
  - ภาษาที่ให้ความสนใจในจุดนี้จะมีคำสำคัญมาแยกประเภทโปรแกรมย่อย