

บทที่ 5 ฟังก์ชัน Function

วัตถุประสงค์

- เขียนโปรแกรมเรียกใช้ฟังก์ชันได้
- สร้างฟังก์ชันเองได้
- เขียนการรับส่งข้อมูลระหว่างฟังก์ชันได้
- สร้างและใช้ฟังก์ชันแบบเรียกตัวเอง (**recursion**)
ได้

เนื้อหาในบทเรียน

1. ฟังก์ชันคืออะไร
2. ทำไมต้องมีฟังก์ชัน
3. ฟังก์ชันในภาษา C
4. การสร้างฟังก์ชัน
5. การเรียกใช้งานฟังก์ชัน
6. ฟังก์ชันแบบเรียกตัวเอง
7. ขอบเขตของตัวแปร

เนื้อหาในบทเรียน (ต่อ)

8. ฟังก์ชันและอาร์เรย์
9. การเรียกใช้ฟังก์ชันแบบ **call-by-value** และ **call-by-reference**
10. การส่งค่าบางอีลีเมนต์ของอาร์เรย์ให้กับฟังก์ชัน
11. การส่งค่าทุกอีลีเมนต์ของอาร์เรย์ให้กับฟังก์ชัน
12. ฟังก์ชันที่รับพารามิเตอร์เป็นอาร์เรย์
13. **String** สตริงหรือสายตัวอักษร
14. ฟังก์ชันที่เกี่ยวกับสตริง
15. อาร์เรย์ของสตริง

1. ฟังก์ชันคืออะไร

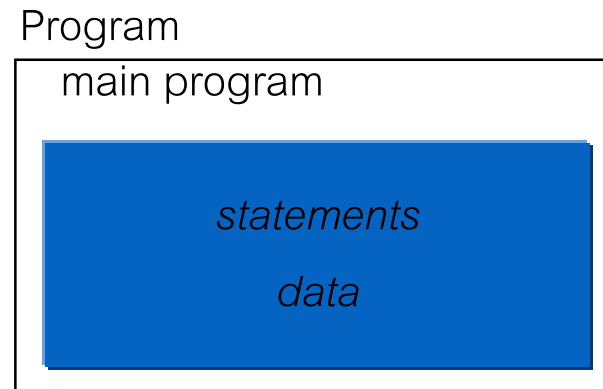
- ฟังก์ชัน(**function**) คือ ส่วนของโปรแกรมที่ทำงานเสร็จสิ้นภายในตัวเอง มีลักษณะเหมือนกับโปรแกรมย่อย ที่รวมอยู่ในโปรแกรมหลักอีกทีหนึ่ง
- ใช้หลักการ *Divide and Conquer*
 - แบ่งการทำงานออกเป็นส่วนเล็กๆ ที่ทำงานเสร็จสมบูรณ์ในตัว
 - ทดสอบและแก้ไขส่วนเล็กๆนี้
 - ประกอบส่วนเล็กๆนี้ ขึ้นมาเป็นโปรแกรมขนาดใหญ่ที่สมบูรณ์ในขั้นตอนสุดท้าย

2. ทำไมต้องมีฟังก์ชัน

- เนื่องจากโปรแกรมถูกพัฒนาขึ้นเรื่อยๆ มีความซับซ้อนและขนาดใหญ่มากขึ้น
- การเขียนโปรแกรมแบบเดิมที่ไม่มีโครงสร้าง (Unstructured Programming) มีความยุ่งยากในการพัฒนาและการแก้ไขข้อผิดพลาด
 - มีส่วนของโปรแกรมที่ทำงานลักษณะเดียวกันกระจายอยู่หลายที่
 - เมื่อแก้ไขส่วนการทำงานนั้นต้องไปแก้ไขหลายที่ในโปรแกรมจึงเกิดความผิดพลาดซ้ำซ้อนได้ง่าย
- การเขียนโปรแกรมแบบมีโครงสร้าง (Structure Programming) เหมาะสมในการพัฒนาโปรแกรมที่มีความซับซ้อนมากกว่า
- การใช้ฟังก์ชันเป็นลักษณะการเขียนโปรแกรมแบบมีโครงสร้างแบบหนึ่ง

2.1 Unstructured Programming

รูปแบบ : โปรแกรมเขียนเรียงไปเรื่อยๆตามลำดับการประมวลผล



ข้อเสีย : เมื่อโปรแกรมเริ่มมีขนาดใหญ่ ทำให้ยากต่อการดูแลหรือปรับปรุง
เปลี่ยนแปลงเช่น

- การหาจุดผิดพลาด ทำได้ยากขึ้น,
- เกิดการใช้ตัวแปรซ้ำซ้อนได้ง่ายขึ้น, ...

ตัวอย่าง 5.1 Unstructured Programming

```
#include <stdio.h>
main()
{
    int first, second, third;
    printf("\n F(X)=3X +10 if X > 0\n");
    printf("\n F(X)=10 if X <=0\n");
    printf("\n Enter 3 values\n");
    scanf("%d %d %d", &first, &second, &third);
    if (first > 0)
        printf("F(%d)is %d", first, 3*first +10);
    else
        printf("F(%d)is 10", first);
    if (second > 0)
        printf("F(%d)is %d", second, 3*second +10);
    else
        printf("F(%d)is 10", second);
    if (third > 0)
        printf("F(%d)is %d", third, 3*third +10);
    else
        printf("F(%d)is 10", third);
}
```

Code program ซ้ำซ้อน

คำนวณ first

คำนวณ second

คำนวณ third

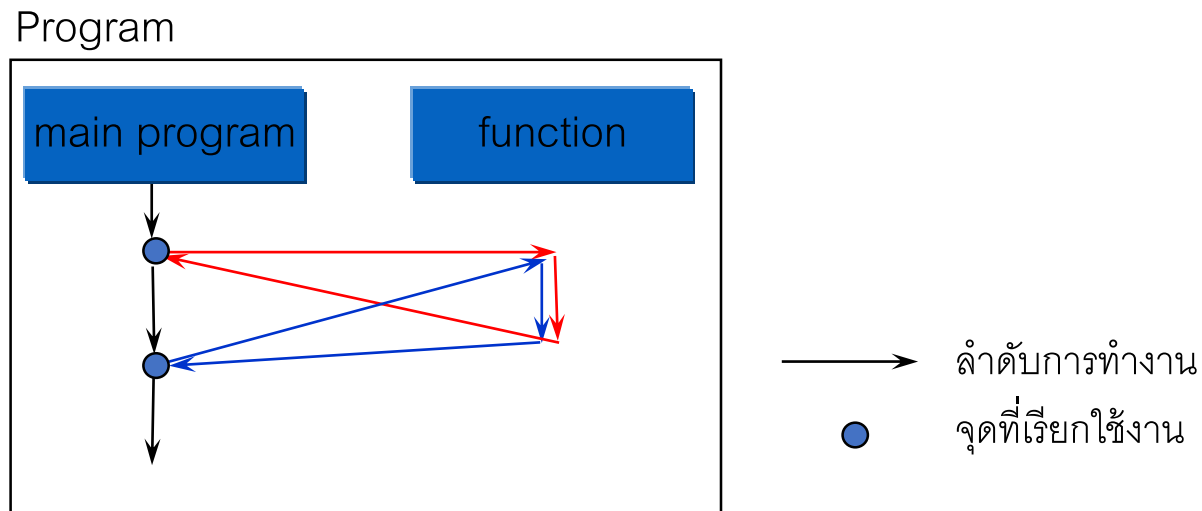


จากตัวอย่าง 5.1

- พบว่าชุดคำสั่งเดียวกัน (ส่วนเงื่อนไขของ if-else) มีการใช้งานหลายแห่งในโปรแกรม ทำให้ต้องสำเนาชุดคำสั่งนี้หลายครั้งในหนึ่งโปรแกรม
- เกิดแนวคิดที่จะรวบรวมชุดคำสั่งนี้เข้าไว้ที่เดียวกัน และเรียกใช้งานได้ตามต้องการ ซึ่งเป็นหลักการของ Structure Programming

2.2 Structural Programming (1/2)

รูปแบบ : นำชุดคำสั่งบางส่วนไปรวบรวมไว้ในฟังก์ชัน โปรแกรมหลัก(**main program**) ซึ่งเป็นส่วนที่เริ่มการทำงานของโปรแกรม จะเรียกใช้ชุดคำสั่งในฟังก์ชันนั้นมาทำงาน

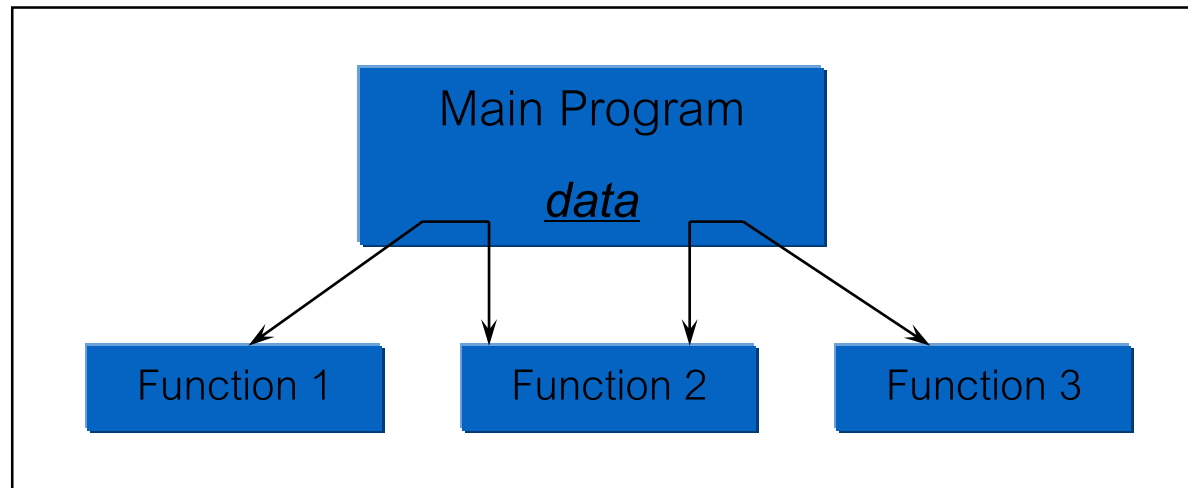


ข้อดี :

- มีการซ่อนรายละเอียดซับซ้อนไว้ในฟังก์ชัน ทำให้โปรแกรมหลักง่ายในการทำความเข้าใจ
- ลดความซ้ำซ้อนของส่วนโปรแกรมที่เหมือนกันไว้ในฟังก์ชันเดียวกัน
- การหาข้อผิดพลาดของโปรแกรมง่ายขึ้น โดยแยกหาระหว่างส่วน **main** กับฟังก์ชันอื่นๆ

2.2 Structure Programming (2/2)

การทำงานของ **main** เรียกใช้งานฟังก์ชันโดยอาจมีการส่งค่าให้กับการเรียกแต่ละครั้ง
เราอาจมองได้ว่า **main** เป็นตัวประสานงานระหว่างหลายๆ ฟังก์ชันและเป็นส่วน
คอยควบคุมข้อมูลหลัก



สรุป การโปรแกรมแบบมีโครงสร้าง ประกอบด้วยหนึ่งโปรแกรมหลักและหลายส่วนย่อย
ที่เรียกว่าฟังก์ชัน

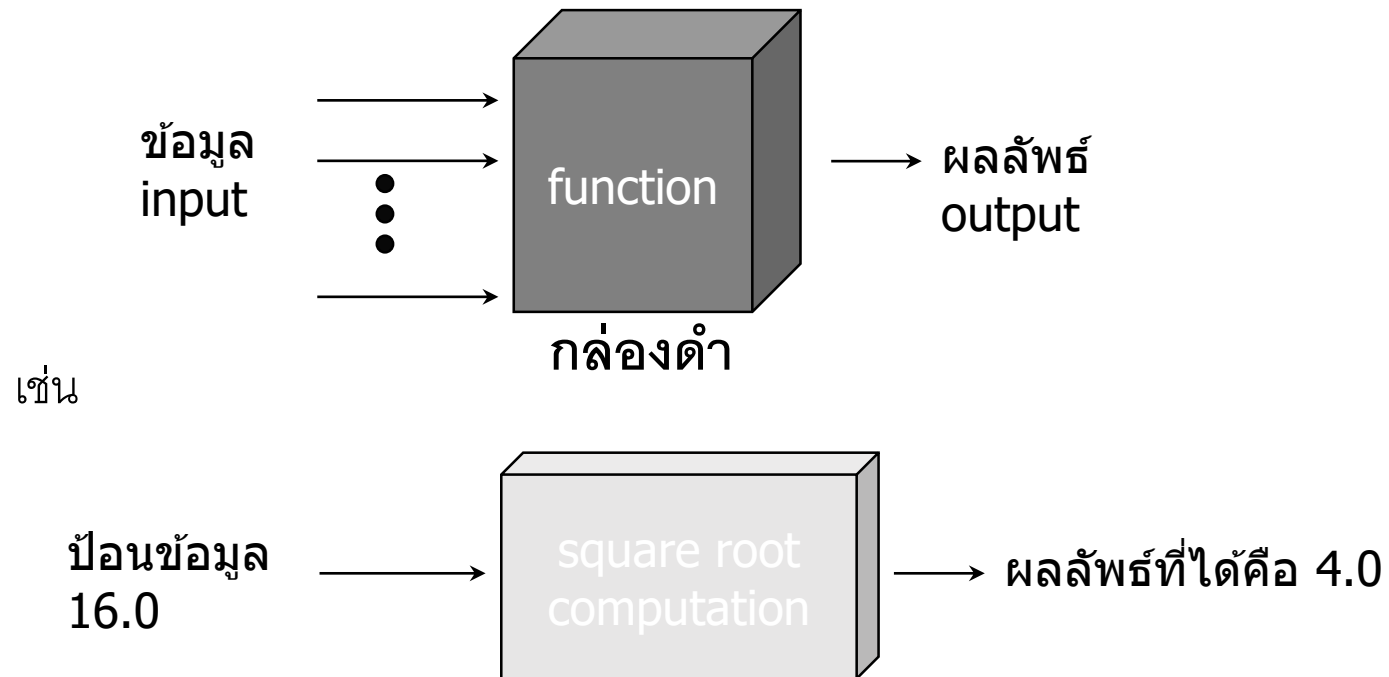
ตัวอย่าง 5.2 Structure Programming

```
#include <stdio.h>
void get_Fx(int x);
main()
{
    int first, second, third;
    printf("\n F(X)=3X +10 if X > 0\n");
    printf("\n F(X)=10 if X <=0\n");
    printf("\n Enter 3 values\n");
    scanf("%d %d %d", &first, &second, &third);
    get_Fx(first);
    get_Fx(second);
    get_Fx(third);
}
void get_Fx(int x)
{
    if (x > 0)
        printf("F(%d)is %d", x, (3*x)+10);
    else
        printf("F(%d)is 10", x);
}
```

- รวมการทำงานแบบเดียวกันไว้ด้วยกัน
- เปลี่ยนแปลงตัวแปร x ในการเรียกใช้งานแต่ละครั้ง

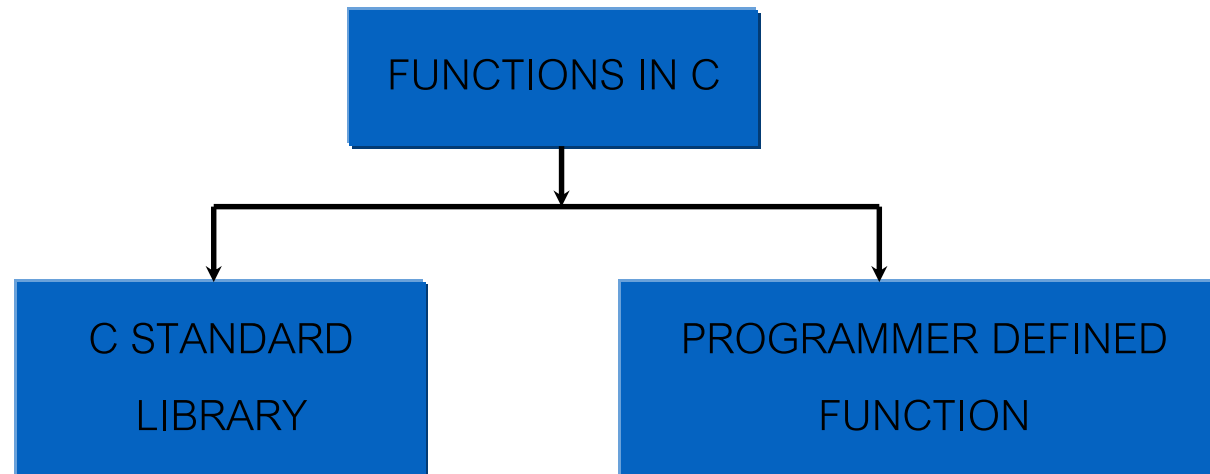
3. ฟังก์ชันในภาษา C

- ฟังก์ชัน คือ ชุดของคำสั่งที่ทำงานอย่างใดอย่างหนึ่งและมีชื่อเรียก ซึ่งส่วนอื่นของโปรแกรมสามารถเรียกใช้งานฟังก์ชันได้
- ในการเรียกใช้ฟังก์ชัน ไม่จำเป็นต้องทราบว่าภายในฟังก์ชันทำงานอย่างไร เปรียบได้กับกล่องดำที่มองไม่เห็นภายใน แต่ต้องทราบว่าต้องป้อนข้อมูลอะไรให้ และได้รับผลลัพธ์อะไรกลับมา



3. ฟังก์ชันในภาษา C (ต่อ)

- โปรแกรมภาษา C ประกอบไปด้วยหนึ่งฟังก์ชันหรือมากกว่า (ต้องมี **main**)
- แต่ละฟังก์ชันประกอบไปด้วยหนึ่งคำสั่ง (**statement**) หรือมากกว่า
- ภาษา C แบ่งฟังก์ชันเป็น 2 แบบ
 - ฟังก์ชันมาตรฐานใน C
 - ฟังก์ชันที่สร้างโดยผู้เขียนโปรแกรม



ฟังก์ชันมาตรฐานใน C (C Standard Library)

- ฟังก์ชันที่ผู้ผลิต **C compiler** เป็นผู้เขียนขึ้น โดยเก็บไว้ใน **C library**
- ประกอบด้วยฟังก์ชันเกี่ยวกับ
 - disk I/O (input/output), standard I/O ex. `printf()`, `scanf()`, ...
 - string manipulation ex. `strlen()`, `strcpy()`, ...
 - mathematics ex. `sqrt()`, `sin()`, `cos()`, ...
 - etc..
- สามารถเรียกใช้งานได้เลย แต่ต้องมีการ **include header file** ที่นิยามฟังก์ชันนั้นๆไว้ เช่น
จะใช้ `printf()`, `scanf()` ต้องเขียน **`#include <stdio.h>`**
จะใช้ `sqrt()`, `sin()`, `cos()` ต้องเขียน **`#include <math.h>`**
etc.

ตัวอย่าง 5.3 การใช้งานฟังก์ชันมาตรฐานใน C

โปรแกรมหาค่ารากที่สอง โดยใช้ฟังก์ชัน `sqrt()` ใน `math.h`

ต้นแบบ(prototype): **`double sqrt(double num)`**

ให้ค่ารากที่สองของ `num`

```
#include <stdio.h>
```

```
#include <math.h>
```

```
int main()
```

```
{    printf("%.2f\n", sqrt(16.0));
```

```
    printf("%.2f\n", sqrt(sqrt(16.0)));
```

```
}
```

4.00

2.00

ตัวอย่าง 5.4 การใช้วงฟังก์ชันมาตรฐานใน C

โปรแกรมแสดง 10^y ยกกำลัง 1 ถึง 5 โดยเรียกใช้ฟังก์ชัน `pow()` ใน `math.h`

ต้นแบบ `double pow(double base, double exp)`

ให้ค่า base^{exp}

```
#include <stdio.h>
#include <math.h>
int main()
{
    double x=10.0, y =1.0;

    do {
        printf("%.2f\n", pow(x,y));
        y++;
    } while (y < 6);
}
```

10.00

100.00

1000.00

10000.00

100000.00

หารากที่ 2 => ยกกำลัง 0.5

หารากที่ 5 => ยกกำลัง ?

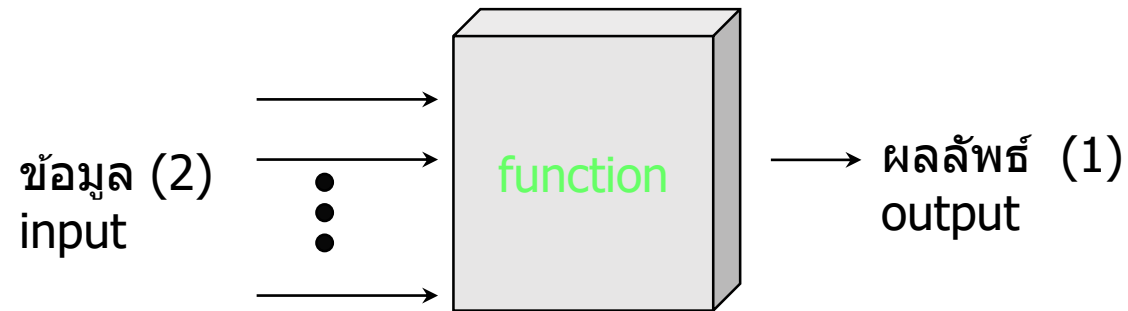
4. การสร้างฟังก์ชัน

- การสร้างฟังก์ชันมี 2 ส่วนคือ
 1. ต้นแบบฟังก์ชัน (function prototype)
 2. นิยามฟังก์ชัน (function definition)
- ต้นแบบฟังก์ชันจะมีหรือไม่ ขึ้นกับตำแหน่งที่เรานิยามฟังก์ชันในโปรแกรม
 - ถ้าฟังก์ชันนิยามก่อน **main** ไม่จำเป็นต้องมีต้นแบบฟังก์ชัน
 - ถ้าฟังก์ชันนิยามหลัง **main** จำเป็นต้องมีการประกาศต้นแบบฟังก์ชันก่อนฟังก์ชัน **main**

4.1 Function Prototype

ต้นแบบของฟังก์ชันเป็นตัวบอกให้คอมไพเลอร์รู้ถึงสิ่งเหล่านี้

1. ชนิดข้อมูลที่จะส่งค่ากลับ (1)
2. จำนวนพารามิเตอร์ที่ฟังก์ชันต้องการ (2)
3. ชนิดของพารามิเตอร์แต่ละตัว รวมทั้งลำดับของพารามิเตอร์เหล่านั้น



ข้อสังเกต ฟังก์ชันอาจจะไม่มีการส่งค่ากลับ และไม่ต้องการข้อมูล input

4.1 Function prototype (ต่อ)

- รูปแบบ

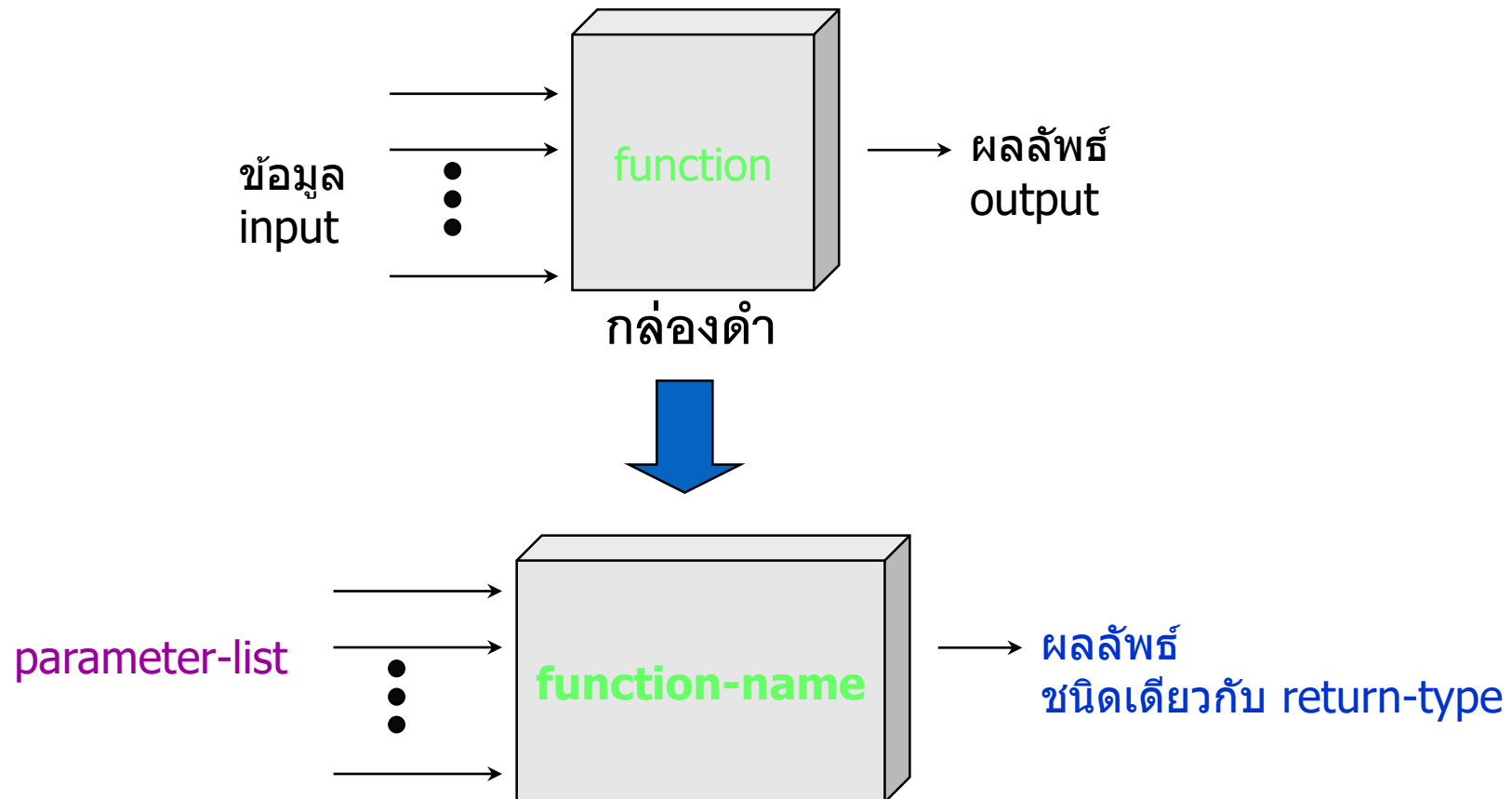
```
return-type function-name (parameter-list) ;
```

- **return-type** หรือชนิดข้อมูลที่จะส่งค่ากลับได้แก่ void, int, double, char, ...
- **function-name** ชื่อของฟังก์ชัน
- **parameter-list** จำนวนพารามิเตอร์ที่ฟังก์ชันต้องการ
 - แต่ละพารามิเตอร์ประกอบด้วย ชนิดตัวแปรและชื่อตัวแปร
 - แต่ละพารามิเตอร์แยกด้วยเครื่องหมาย “,”

ตัวอย่าง **int** add (int a, int b) ;

4.1 Function prototype (ต่อ)

- เทียบกับตอนเป็นกล่องดำ



4.2 นิยามฟังก์ชัน (function definition)

- นิยามฟังก์ชันเป็นการเขียนรายละเอียดการทำงานของฟังก์ชันนั้นๆ
- ประกอบด้วยส่วนของ **header** และ **algorithm**
 - **header** จะมีการเขียนเหมือน ต้นแบบฟังก์ชัน แต่ไม่มี ;
 - **algorithm** เขียนอยู่ภายใน **{ }**
- รูปแบบ

```
return-type function-name (parameter-list)
{
    รายละเอียดการทำงาน
}
```

ถ้า **return-type** ไม่ใช่ **void** ต้องมีการส่งค่ากลับ
โดยใช้คำสั่ง **return** ตามด้วยค่าที่จะส่งกลับ

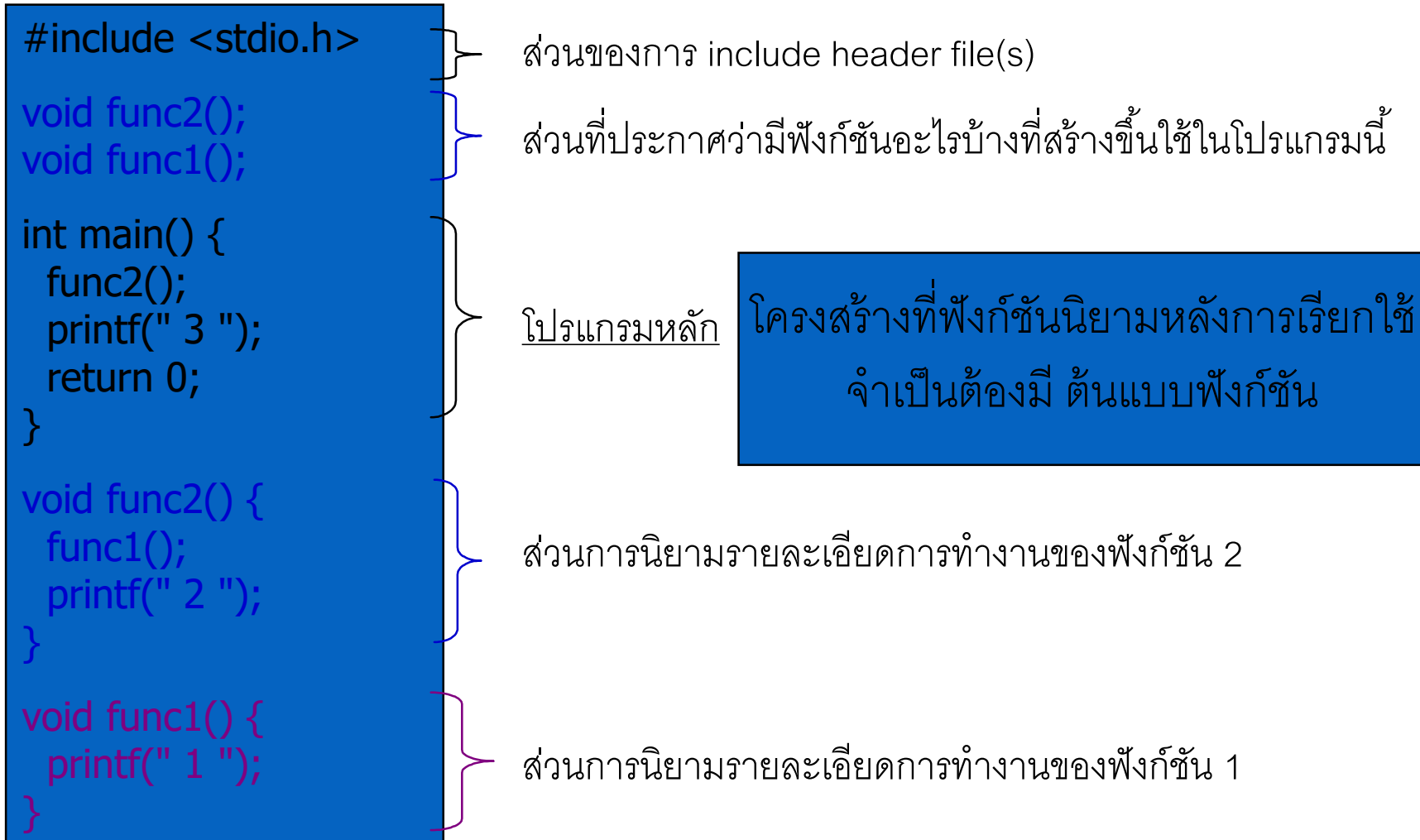
- ตัวอย่างฟังก์ชันที่มี **return**

```
int add (int a, int b)
{
    int result;
    result = a+b;
    return result;
}
```

- ตัวอย่างฟังก์ชันที่ไม่มี **return**

```
void checkresult (int result)
{
    if(result < 0)
        printf("result is  
negative");
    if(result == 0)
        printf("result is  
zero");
    if(result > 0)
        printf("result is  
positive")
}
```

ตัวอย่าง 5.5 โครงสร้างโดยรวมของโปรแกรม 1



ตัวอย่าง 5.6 โครงสร้างโดยรวมของโปรแกรม 2

```
#include <stdio.h>

void func1() {
    printf("1");
}

void func2() {
    func1();
    printf("2");
}

int main() {
    func2();
    printf("3");
    return 0;
}
```

ส่วนของการ include header file(s)

ส่วนการนิยามรายละเอียดการทำงานของฟังก์ชัน 1

ส่วนการนิยามรายละเอียดการทำงานของฟังก์ชัน 2

โปรแกรมหลัก

โครงสร้างที่ฟังก์ชันนิยามก่อนเรียกใช้
ไม่จำเป็นต้องมี ต้นแบบฟังก์ชัน

ตัวอย่าง 5.7 ชื่อต่างๆเกี่ยวกับฟังก์ชัน

```
#include <stdio.h>
int square(int x);
int main()
{
    int a=2,b;
    b = square(a);
    printf("b = %d \n",b);
    return 0;
}

int square(int x)
{
    int y;
    y = x*x;
    return y;
}
```

ตัวอย่าง 5.7 ชื่อต่างๆเกี่ยวกับฟังก์ชัน

```
#include <stdio.h>
```

```
int square(int x);
```

```
int main()
```

```
{
```

```
    int a=2,b;
```

```
    b = square(a);
```

```
    printf("b = %d \n",b);
```

```
    return 0;
```

```
}
```

```
int square(int x)
```

```
{
```

```
    int y;
```

```
    y = x*x;
```

```
    return y;
```

```
}
```

ต้นแบบฟังก์ชัน

function prototype

อากิวเมนต์

เรียกใช้งานฟังก์ชัน

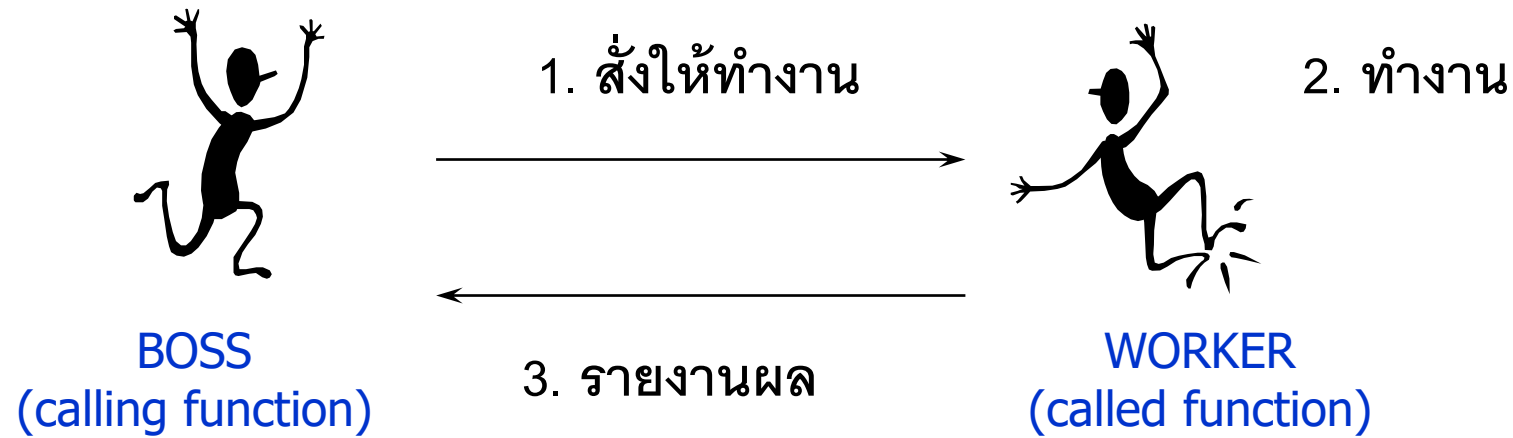
call function

พารามิเตอร์

นิยามฟังก์ชัน

function definition

5. การเรียกใช้งานฟังก์ชัน



BOSS อาจเป็นโปรแกรม main หรือเป็นฟังก์ชันอื่น

WORKER คือฟังก์ชันที่ถูกเรียกใช้งาน

5.1 ฟังก์ชันแบ่งตามการรับส่งข้อมูล

แบบที่ 1 - ฟังก์ชันที่ไม่รับผ่านค่า และไม่ส่งผ่านค่ากลับ

void func1();

แบบที่ 2 - ฟังก์ชันที่มีการรับผ่านค่า แต่ไม่ส่งผ่านค่ากลับ

void func2(int a);

แบบที่ 3 - ฟังก์ชันที่มีการรับผ่านค่า และส่งผ่านค่ากลับ

int func3(int a);

แบบที่ 4 - ฟังก์ชันที่ไม่รับผ่านค่า แต่มีการส่งผ่านค่ากลับ

int func4();

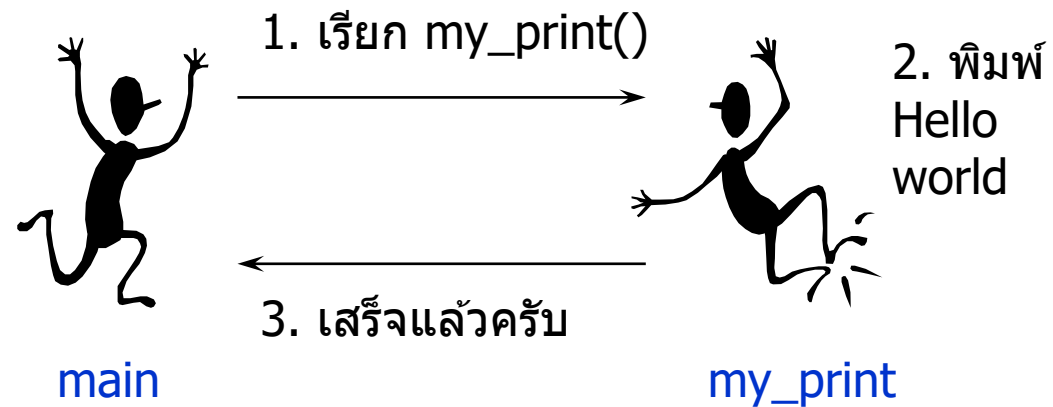
ตัวอย่าง 5.8

- ฟังก์ชันที่ไม่รับผ่านค่า และไม่ส่งผ่านค่ากลับ

```
#include <stdio.h>
void my_print();

void main()
{
    my_print();
}

void my_print()
{
    printf("Hello world");
}
```



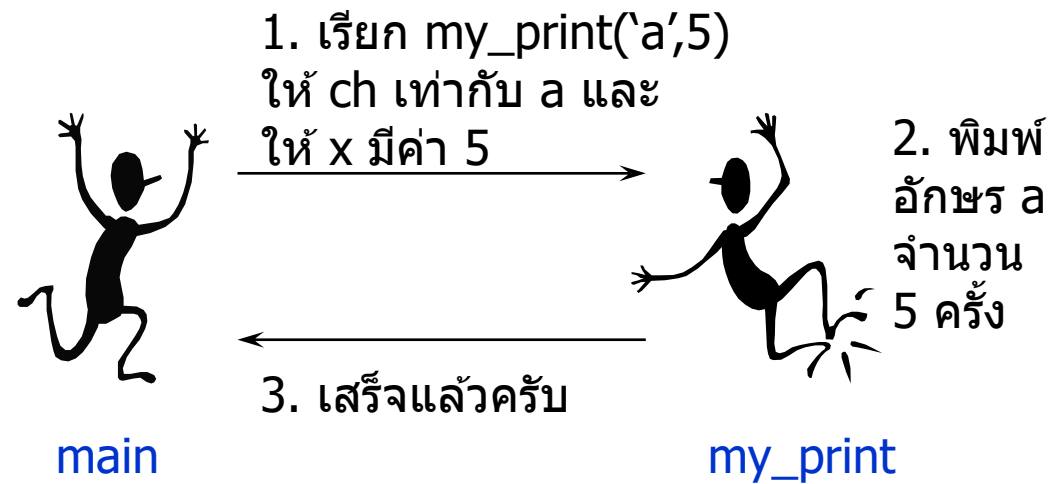
ตัวอย่าง 5.9

- ฟังก์ชันที่มีการรับผ่านค่า แต่ไม่ส่งผ่านค่ากลับ

```
#include <stdio.h>
void my_print(char ch, int x);

void main()
{
    my_print('a', 5);
}

void my_print(char ch, int x)
{
    while (x > 0)
    {
        printf("%c", ch);
        x--;
    }
}
```



Note :

ฟังก์ชันต้องการพารามิเตอร์ 2 ตัว ตอนเรียกใช้งานต้องให้อาภิวกเมนต์ให้ถูกต้องตามจำนวนและชนิดที่ต้องการ

ตัวอย่าง 5.10

- ฟังก์ชันที่มีการรับผ่านค่า และส่งผ่านค่ากลับ

```
#include <stdio.h>

char my_print(int x);

void main()
{
    char ch;
    ch = my_print(5);
    printf("%c", ch);
}

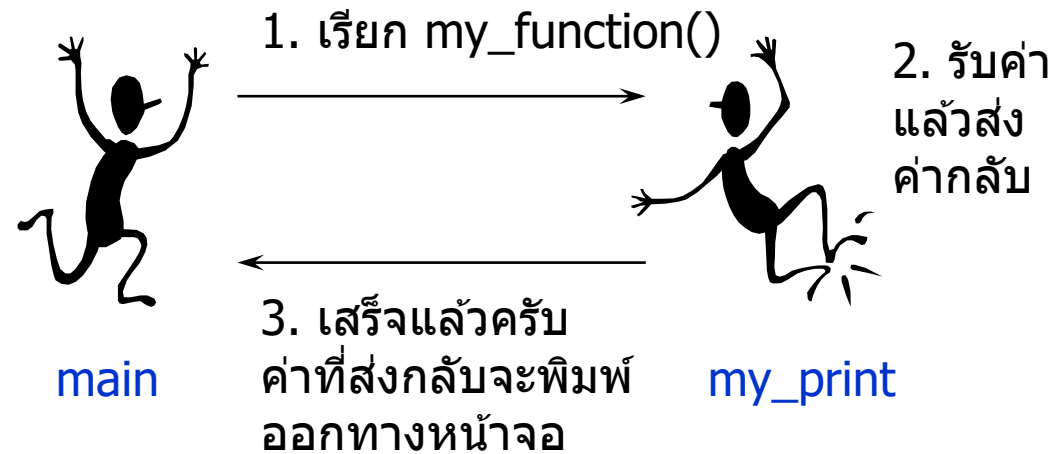
char my_print(int x)
{
    char lch;
    scanf("%c", &lch);
    while (x > 0)
    {
        printf("%c", lch);
        x--;
    }
    return lch;
}
```



ตัวอย่าง 5.11

- ฟังก์ชันที่ไม่รับผ่านค่า แต่มีการส่งผ่านค่ากลับ

```
#include <stdio.h>
int my_function();
void main()
{
    printf("%d", my_function());
}
int my_function()
{
    int a;
    scanf("%d", &a);
    return a*5;
}
```



5.2 สรุปการเรียกใช้งานฟังก์ชัน

- ฟังก์ชันที่ไม่มีการส่งค่ากลับ สามารถเรียกชื่อฟังก์ชันได้เลย โดยต้องให้อากิวเมนต์ให้ถูกต้อง

เช่น `my_print();` [จาก ex.5.8], `my_print('a',5);` [จาก ex.5.9]

ex. 5.8

```
void my_print()
{
    printf("Hello world");
}
```

ex. 5.9

```
void my_print(char ch, int x)
{
    while (x > 0)
    {
        printf("%c", ch);
        x--;
    }
}
```

5.2 สรุปการเรียกใช้งานฟังก์ชัน (ต่อ)

- ฟังก์ชันที่มีการส่งค่ากลับ ให้มองเสมือนว่าเป็นตัวแปรตัวหนึ่งที่มีชนิดเดียวกับชนิดของข้อมูลที่ส่งค่ากลับ วิธีเรียกใช้งาน

- เอาตัวแปรมารับค่า เช่น `ch = my_print(5);` [จาก ex.5.10]
- ใช้เป็นส่วนหนึ่งในนิพจน์ เช่น `printf("%d", my_function());` [จาก ex.5.11]
- ใช้เป็นอาร์กิวเมนต์ของฟังก์ชันอื่น เช่น มีฟังก์ชัน `int add(int a, int b);`
`int x = add(add(1,2),4);`

คำนวณ `add(1,2)` ก่อน แล้วผลลัพธ์ที่ได้ ให้เป็นอาร์กิวเมนต์ของ `add(... ,4)`

โจทย์ตัวอย่าง

จงเขียนโปรโตไทป์ให้ฟังก์ชันชื่อ **script** ที่มีพารามิเตอร์ 3 ตัว

- 1 จำนวนช่องว่างที่ต้องการให้แสดงเมื่อเริ่มบรรทัด
- 2 ตัวอักษรที่ต้องการให้แสดงหลังจากช่องว่างในพารามิเตอร์แรก
- 3 จำนวนครั้งที่ต้องการให้แสดงตัวอักษรในพารามิเตอร์ที่ 2

โดยฟังก์ชัน **script** ไม่มีการส่งค่ากลับ

```
void script(int space, char c,  
int time);
```

โจทย์ตัวอย่าง (ต่อ)

- เขียนรายละเอียดการทำงานของฟังก์ชัน **script**
 - ฟังก์ชัน **script** มีพารามิเตอร์ 3 ตัว
 - 1 จำนวนช่องว่างที่ต้องการให้แสดงเมื่อเริ่มบรรทัด
 - 2 ตัวอักษรที่ต้องการให้แสดงหลังจากช่องว่างในพารามิเตอร์แรก
 - 3 จำนวนครั้งที่ต้องการให้แสดงตัวอักษรในพารามิเตอร์ที่ 2
- โดยฟังก์ชัน **script** ไม่มีการส่งค่ากลับ

โจทย์ตัวอย่าง (ต่อ)

- เขียนรายละเอียดการทำงานของฟังก์ชัน **script**

```
void script(int space, char c, int time)  
{   int i;  
    for(i=0; i< space; i++)  
        printf(" ");  
    for(i=0; i< time; i++)  
        printf("%c", c);  
}
```

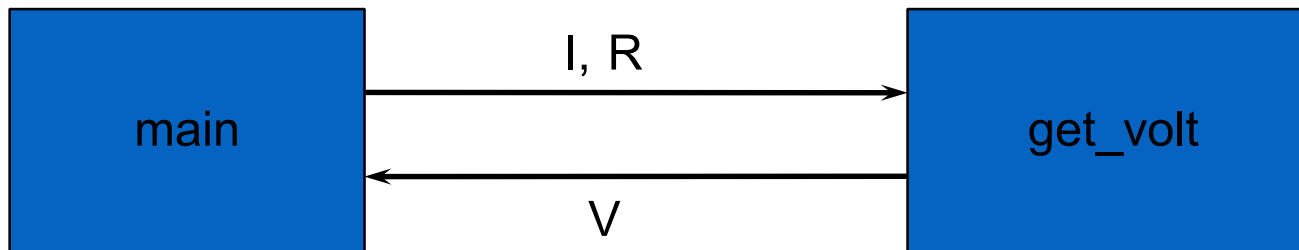
ตัวอย่าง 5.16

จงเขียนโปรแกรมสำหรับหาค่าศักย์ไฟฟ้าซึ่งมีสมการดังนี้

$$V = I * R$$

โดยที่ V คือ ค่าศักย์ไฟฟ้า , I คือ ค่ากระแสไฟฟ้า ส่วน R คือ ค่าความต้านทาน และทั้งสามค่านี้เป็นจำนวนจริง โดยกำหนดให้ส่วนที่คำนวณค่า V อยู่ในฟังก์ชัน `get_volt` สำหรับส่วนที่รับค่า I และ R จากผู้ใช้งานรวม

ทั้งส่วนที่แสดงผลลัพธ์ของค่า V ให้อยู่ในฟังก์ชัน `main`



```
#include <stdio.h>
float get_volt(float I, float R);

int main()
{
    float i, r, v;
    printf("Enter I  :");
    scanf("%f", &i);
    printf("Enter R  :");
    scanf("%f", &r);
    v = get_volt(i, r);
    printf("Volt =%f \n", v);
    return 0;
}

float get_volt(float I, float R)
{
    float V = I*R;
    return V;
}
```

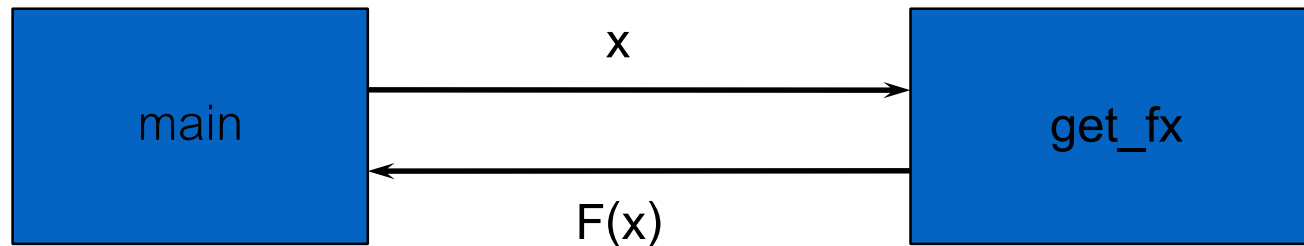

ตัวอย่าง 5.17

จงเขียนโปรแกรมสำหรับหาค่า $F(x)$ ซึ่งมีสมการดังนี้

$$F(x) = x^2 + 2x + 1 \quad \text{ถ้า } x \text{ มีค่าไม่ต่ำกว่า } 0$$
$$= 0 \quad \text{ถ้า } x \text{ มีค่าต่ำกว่า } 0$$

กำหนดให้ส่วนที่ใช้ในการคำนวณค่า $F(x)$ อยู่ในฟังก์ชัน `get_fx`

สำหรับส่วนที่รับค่า x จากผู้ใช้ และแสดงผลลัพธ์ของค่า $F(x)$ อยู่ในฟังก์ชัน `main` (x และ $F(x)$ เป็นจำนวนเต็ม)



```

#include <stdio.h>

int get_fx(int a);

int main()
{
    int x, fx;
    printf("Enter x :");
    scanf("%d", &x);
    fx = get_fx(x);
    printf("f(%d)=%d \n", x, fx);
    return 0;
}

int get_fx(int a)
{
    if(a>=0)
        return (a*a)+(2*a)+1;
    else
        return 0;
}

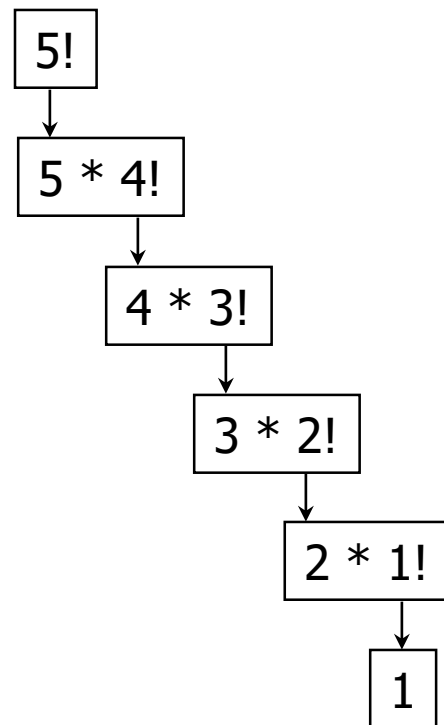
```

6. ฟังก์ชันแบบเรียกตัวเอง (recursive function)

ฟังก์ชันที่มีการเรียกตัวเองโดยให้พารามิเตอร์ที่แตกต่างกันออกไปเช่น การหา

Factorial หรือการหา **Fibonacci**

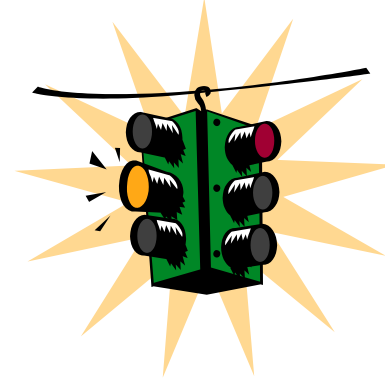
$$n! = n * (n-1)! \quad \Rightarrow \quad \text{factorial}(n) = n * \text{factorial}(n-1)$$



```
#include<stdio.h>
int factorial(int x);
int main()
{
    int y = factorial(3);
    printf("3! = %d", y);
    return 0;
}
int factorial(int x)
{
    if(x <= 1)
        return 1;
    else
        return x* factorial(x-1);
}
```

ฟังก์ชันหาค่าแฟกทอเรียลแบบเรียกตัวเอง

```
#include<stdio.h>
int factorial(int x);
int main()
{
    int y = factorial(3);
    printf("3! = %d", y);
    return 0;
}
int factorial(int x)
{
    if(x <= 1)
        return 1;
    else
        return x * factorial(x-1);
}
```



ข้อควรระวัง :

ฟังก์ชันแบบเรียกตัวเอง จำเป็นจะต้องมี
if statement เพื่อใช้ในการตัดสินใจว่าฟังก์ชัน
จะเรียกตัวเองต่อไป หรือ หยุดเพื่อส่งค่ากลับ

ตัวอย่าง 5.13

- จงเขียนฟังก์ชันสำหรับหาค่า **$F(x)$** ซึ่งมีสมการดังนี้

$$F(x) = 0 \quad \text{ถ้า } x \text{ เท่ากับ } 0$$

$$= 1 \quad \text{ถ้า } x \text{ เท่ากับ } 1$$

$$= F(x-1) + F(x-2) \quad \text{ถ้า } x \text{ มากกว่า } 1$$

- กำหนดให้ **x** และ **$F(x)$** เป็นจำนวนเต็ม และให้ตั้งชื่อฟังก์ชันว่า **fib**

(อนุกรม **Fibonacci**)

- **Fibonacci numbers: 1 1 2 3 5 8 13 21 34 55 89 144 233 ...**

```
int fib(int x)
{
    if(x == 0)
        return 0;
    else if(x == 1)
        return 1;
    else if(x > 1)
        return fib(x-1) + fib(x-2);
}
```

7. ขอบเขตของตัวแปร

ตัวแปรภายใน (**local variable**) – ตัวแปรที่ประกาศในบล็อก (ภายในเครื่องหมาย { }) ของฟังก์ชันใดๆ จะรู้จักเฉพาะในฟังก์ชันนั้นๆ

ตัวแปรภายนอก (**global variable**) - ตัวแปรที่ประกาศนอกฟังก์ชันใดๆ (รวมถึงฟังก์ชัน **main**) จะรู้จักในทุกฟังก์ชันที่เขียนถัดจากการประกาศตัวแปร

ตัวอย่าง 5.14 ขอบเขตของตัวแปรภายในฟังก์ชัน

```
#include <stdio.h>
void my_func();
int main()
{
    double x = 1.1;
    printf("In main, x = %.2f \n", x);
    my_func();
    printf("In main, x = %.2f \n", x);
    return 0;
}
void my_func()
{
    double x;
    x = 2.5;
    printf("In my_func, x = %.2f \n", x);
}
```

ผลลัพธ์

In main, x = 1.10

In my_func, x = 2.50

In main, x = 1.10

ตัวอย่าง 5.15 ขอบเขตของตัวแปรภายนอกฟังก์ชัน

```
#include <stdio.h>
double x;
void my_func();
int main()
{
    x = 1.1;
    printf("In main, x = %.2f \n", x);
    my_func();
    printf("In main, x = %.2f \n", x);
    return 0;
}
void my_func()
{
    x = 2.5;
    printf("In my_func, x = %.2f \n", x);
}
```

ผลลัพธ์

In main, x = 1.10

In my_func, x = 2.50

In main, x = 2.50

สรุปขอบเขตของตัวแปร

```
#define MAX 950
void one(int anarg,
        double second)
{
    int onelocal;
    ...
}

#define LIMIT 200
int fun_two(int one, char anarg)
{
    int localvar;
    ...
}

int main(void)
{
    int localvar;
    ...
}
```

ตัวแปร	รู้จักใน one	รู้จักใน fun_two	รู้จักใน main
MAX	✓	✓	✓
anarg(int)	✓	✗	✗
second	✓	✗	✗
onelocal	✓	✗	✗
LIMIT	✗	✓	✓
one (parameter)	✗	✓	✗
anarg(char)	✗	✓	✗
localvar(fun_two)	✗	✓	✗
localvar(main)	✗	✗	✓

ฟังก์ชัน	รู้จักใน one	รู้จักใน fun_two	รู้จักใน main
one (function)	✓	✗	✓
fun_two	✗	✓	✓

แบบฝึกหัดฟังก์ชัน

1. เราจะประกาศฟังก์ชันที่ไม่มีการส่งค่ากลับได้อย่างไร
2. ฟังก์ชันโปรโตไทป์คืออะไร มีไว้เพื่ออะไร
3. จากความยาว **3** ด้านของสามเหลี่ยมใดๆ จงเขียนฟังก์ชันในการหาพื้นที่ของสามเหลี่ยม โดยมีสูตรดังนี้

โดย

ให้เขียนโปรแกรมรับค่าความยาวทั้ง **3** ด้านของสามเหลี่ยม $\frac{a + b + c}{2}$ (**a, b, c**) แล้วคำนวณพื้นที่ (**A**) จากนั้นแสดงค่า **A, a, b, c** ออกทางหน้าจอ

แบบฝึกหัดฟังก์ชัน (ต่อ)

- เขียนโปรแกรมรับตัวเลขจำนวนเต็มเข้ามา 1 ค่า แล้วเขียนฟังก์ชัน `sum_digit` เพื่อหาผลรวมของตัวเลขแต่ละหลัก แล้วแสดงค่าออกทางหน้าจอ เช่น ค่า 443 คำนวณ `sum_digit (4+4+3)` ได้ 11
- เขียนฟังก์ชันในการคิดเกรดนักศึกษา โดยรับคะแนนเข้ามาแล้วคำนวณเกรดที่ได้ส่งกลับไปยังโปรแกรมหลัก โดยมีเกณฑ์ดังนี้
 $90-100 = A$, $80-89 = B$, $70-79 = C$, $60-69 = D$, ไม่ถึง 60 = F

ฟังก์ชันและอาร์เรย์

การส่งผ่านข้อมูลของอาร์เรย์ไปยังฟังก์ชันสามารถแบ่งได้เป็น 2 ลักษณะคือ

- การส่งผ่านค่าอีลีเมนต์อาร์เรย์ให้กับฟังก์ชัน เป็นการเรียกใช้ฟังก์ชันแบบ **Call-by-value**
- การส่งอาร์เรย์ทุกอีลีเมนต์ให้กับฟังก์ชัน เป็นการเรียกใช้ฟังก์ชันแบบ **Call-by-reference**

การเรียกใช้แบบ Call-by-value

- ใช้วิธีการส่งค่าของตัวแปร (value) ให้กับฟังก์ชัน โดยผ่านพารามิเตอร์
- ไม่สามารถแก้ไขค่าของอาร์กิวเมนต์(หรือพารามิเตอร์) ภายในฟังก์ชันได้ = *การแก้ไขค่าต่างๆในฟังก์ชัน ไม่มีผลต่อตัวแปรที่ส่งค่ามา*
- ใช้กับฟังก์ชันที่ประกาศรับพารามิเตอร์เป็นตัวแปรธรรมดา (int, float, char,...) ที่ไม่ใช่อาร์เรย์
- เช่น **void triple(int x)**
 { x=x*3; printf("x = %d",x); }

....

```
int x=5, y[2]={10,11};  
triple(x); triple(y[0]);
```

triple(x) ส่งค่า **5** ให้กับฟังก์ชัน ในฟังก์ชัน **x** เริ่มต้นเป็น **5** และถูกทำให้กลายเป็น **15** หลังจบฟังก์ชัน ค่า **x** นอกฟังก์ชัน ไม่เปลี่ยนแปลง

triple(y[0]) ส่งค่า **10** ให้กับฟังก์ชัน ในฟังก์ชัน **x** เริ่มต้นเป็น **10** และถูกทำให้กลายเป็น **30** หลังจบฟังก์ชัน ค่า **y[0]** ยังเหมือนเดิม

การเรียกใช้แบบ Call-by-reference

- ใช้วิธีการส่งค่า แอดเดรส (Address)*** ของตัวแปรไปให้ฟังก์ชัน
 - ใช้กับฟังก์ชันที่รับค่าเข้า(พารามิเตอร์) เป็น **อาร์เรย์**
 - สามารถแก้ไขค่าของอาร์กิวเมนต์ภายในฟังก์ชันได้ = *การแก้ไขค่าตัวแปรอาร์เรย์ ภายในฟังก์ชัน มีผลการเปลี่ยนแปลงต่อตัวแปรที่ส่งค่ามา เพราะ การมีการจัดการค่าของหน่วยความจำในตำแหน่งเดียวกัน*
- ***แอดเดรส (Address) คือ ค่าที่ใช้อ้างอิงถึงตัวข้อมูลภายในหน่วยความจำ เหมือนกับหมายเลขบ้านเลขที่***

ฟังก์ชันที่มีการรับค่าเข้าเป็นอาร์เรย์

- ฟังก์ชันสามารถที่จะรับค่าเข้าเป็นอาร์เรย์ได้ ซึ่งรูปแบบของการเขียนต้นแบบของฟังก์ชันเป็นดังนี้

ชนิดข้อมูล ชื่อฟังก์ชัน(ชนิดข้อมูล ชื่อตัวแปร[ขนาดอาร์เรย์]);

- ในกรณีฟังก์ชันมีการรับค่าเข้าเป็นอาร์เรย์ 1 มิติ อาจจะไม่ต้องกำหนดขนาดของอาร์เรย์ก็ได้
- ตัวอย่างเช่น

```
int sum_arr(int num[10]);  
void print_arr(int a[5]);  
float average(int num[]);
```

การส่งผ่านค่าอีลีเมนต์อาร์เรย์ให้กับฟังก์ชัน

- หากฟังก์ชัน `my_func` มีต้นแบบของฟังก์ชันดังนี้

```
void my_func(int x);
```

- และใน `main` ได้มีการประกาศตัวแปรอาร์เรย์ชื่อว่า `num`

```
int num[10];
```

- การส่งอีลีเมนต์ที่ 0 ของอาร์เรย์ `num` ไปเป็นอาร์กิวเมนต์ของฟังก์ชัน `my_func` สามารถเขียนได้ดังนี้

```
my_func(num[0]);
```

ตัวอย่างการส่งค่าแต่ละอีลีเมนต์ในอาร์เรย์ให้กับฟังก์ชัน

- ไฟล์ arrayex6.c

```
#include <stdio.h>
void check_val(int x);
int main()
{
    int i,a[3]={2,-1,5};
    for(i=0;i<3;i++)
        check_val( a[i] );
    return 0;
}
void check_val(int x)
{
    if(x >= 0)
        printf("%d : Positive\n",x);
    else
        printf("%d: Negative\n",x);
}
```

2 : Positive
-1 : Negative
5 : Positive

ตัวอย่างการส่งค่าบางอีลีเมนต์ของอาร์เรย์ให้กับฟังก์ชัน

Value is 2

- ไฟล์ arrayex7.c

```
#include <stdio.h>
void showVal(int val);    /* function prototype */
void main()
{
    int nums[5] = {2, 18, 1, 27, 16};
    showVal(nums[0]);
}

void showVal(int val)    /* show 1 value */
{
    printf("Value is %d\n", val);
}
```

การส่งอาร์เรย์ทุกอีลีเมนต์ของอาร์เรย์ให้กับฟังก์ชัน

- การส่งอาร์เรย์ในกรณีนี้ ใช้แค่ชื่อตัวแปรอาร์เรย์เท่านั้น เช่น หากใน **main** มีการประกาศอาร์เรย์ดังนี้

```
int num[10];
```

- และฟังก์ชัน **print_array** มีต้นแบบฟังก์ชันดังนี้

```
void print_array(int a[10]);
```

- การส่งอาร์เรย์ **num** ทุกอีลีเมนต์ไปให้ฟังก์ชัน **print_array** สามารถเขียน **การเรียกใช้ฟังก์ชัน** ได้ดังนี้

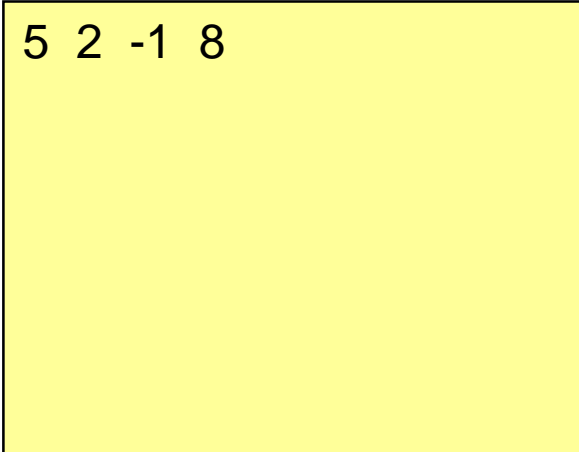
```
print_array(num);
```

ตัวอย่างการส่งอาร์เรย์ทุกอีลีเมนต์ให้กับฟังก์ชัน

- ไฟล์ arrayex8.c

```
#include <stdio.h>
void print_array(int a[4]);
int main() {
    int num[4] = {5,2,-1,8};
    print_array(num);
    return 0;
}

void print_array(int a[4])
{
    int i;
    for (i =0; i<4; i++ )
        printf("%d", a[i]);
}
```



5 2 -1 8

ตัวอย่าง การส่งอาร์เรย์ทุกอีลีเมนต์ให้กับฟังก์ชัน

- ไฟล์ arrayex9.c

The maximum value is 27

```
#include <stdio.h>
void find_max(int vals[5]);    /* function prototype */
void main()
{
    int nums[5]={2, 18, 1, 27, 16};
    find_max(nums);
}

void find_max(int vals[5])    /* find the maximum value */
{
    int i, max =vals[0];
    for (i=1; i < 5; ++i)
        if (max < vals[i]) max = vals[i];

    printf("The maximum value is %d\n" , max);
}
```

โจทย์ฝึกสมอง 1

- จงเขียนโปรแกรมที่รับจำนวนเต็มจากผู้ใช้ 10 ตัว และจำนวนเต็มบวก **N** จากนั้นให้นับว่าในจำนวนเต็มทั้ง 10 ตัวนี้ มีตัวประกอบของ **N** อยู่กี่ตัว กำหนดให้ส่วนที่รับค่าจากผู้ใช้และส่วนที่แสดงผลลัพธ์อยู่ใน **main** และให้ส่วนที่นับตัวประกอบอยู่ในฟังก์ชันชื่อ **count_factor**




```

#include <stdio.h>
int count_factor(int n, int x[10]);
int main()
{
    int i, num[10], num_factor, N;
    printf("N = "); scanf("%d", &N);
    for(i=0; i<10; ++i)
    {
        printf("Enter integer %d: ", i+1);
        scanf("%d", &num[i]);
    }
    num_factor = count_factor(N, num);
    printf("Found factor of %d : %d", N, num_factor);
    return 0;
}

int count_factor(int n, int x[10])
{
    int i, count = 0;
    for(i=0; i<10; ++i)
    {
        if ( n%x[i]==0 ) count++;
    }
    return count;
}

```

N = 40

Enter integer 1: 2

Enter integer 2: 15

Enter integer 3: 20

Enter integer 4: 10

Enter integer 5: 3

Enter integer 6: 1

Enter integer 7: 5

Enter integer 8: 4

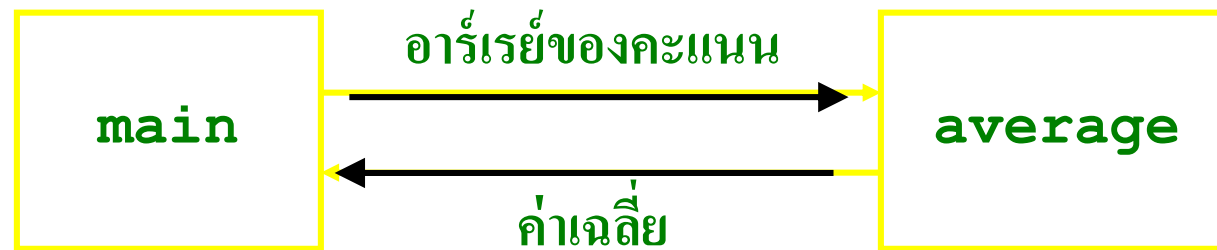
Enter integer 9: 25

Enter integer 10: 30

Found factor of 40 : 6

โจทย์ฝึกสมอง 2

- จงเขียนโปรแกรมสำหรับรับค่าคะแนนของนักศึกษาจำนวน **N** คน (**$N < 100$**) และให้พิมพ์ค่าเฉลี่ยของคะแนนทั้งหมด โดยกำหนดให้ส่วนรับคะแนนจากผู้ใช้และส่วนที่แสดงค่าเฉลี่ยอยู่ใน **main** สำหรับส่วนที่คำนวณค่าเฉลี่ยให้อยู่ฟังก์ชันชื่อ **average** โดยให้ส่งอาร์เรย์ที่เก็บคะแนนจาก **main** มาให้ฟังก์ชัน **average** (คะแนนสามารถเป็นทศนิยมได้)



```

#include <stdio.h>
#define MAX 100
float average(float s[MAX],int N);
int main()
{ int i,N; float score[10],avg_score;
  printf("Number of students N = "); scanf("%d",&N);
  for(i=0;i<N;++i)
  { printf("Enter score %d : ",i+1);
    scanf("%f",&score[i]);
  }
  avg_score =average(score, N);
  printf("Average score is %.2f\n",avg_score);
  return 0;
}
float average(float s[MAX], int N)
{ float total=0.0, avg;
  int i;
  for(i=0; i<N; i++)
    total+= s[i];
  avg = total/N;
  return avg;
}

```

```

Number of students N = 10
Enter score 1: 10
Enter score 2: 20
Enter score 3: 30
Enter score 4: 40
Enter score 5: 50
Enter score 6: 60
Enter score 7: 70
Enter score 8: 80
Enter score 9: 90
Enter score 10: 100
Average score is : 55.00

```

การแก้ไขค่าของอาร์เรย์ภายในฟังก์ชัน

ฟังก์ชันที่มีการรับพารามิเตอร์เป็นอาร์เรย์ หากมีการแก้ไขค่าภายในอาร์เรย์ดังกล่าว จะทำให้อาร์เรย์ที่ถูกส่งมาเป็นอาร์กิวเมนต์ นั้น มีการเปลี่ยนแปลงด้วย

➔ **Call-by-reference** การส่ง ชื่อของอาร์เรย์ให้กับฟังก์ชัน เป็นการส่งตำแหน่งเริ่มต้นของตัวแปรอาร์เรย์ ทำให้ฟังก์ชันสามารถเข้าถึงข้อมูลในอาร์เรย์ได้โดยตรง และเข้าไปแก้ไขข้อมูลของอาร์เรย์นั้นได้

ตัวอย่างการแก้ไขข้อมูลอาร์เรย์ในฟังก์ชัน

- ไฟล์ arrayex10.c

```
#include <stdio.h>
void edit_arr(int a[4]);
int main() {
    int i, num[4] = {2, 5};
    printf("Before: ");
    for(i=0; i<4; i++) printf("%d ", num[i]);
    printf("\nAfter: ");
    edit_arr(num);
    for(i=0; i<4; i++) printf("%d ", num[i]);
    return 0;
}
void edit_arr(int a[4]) {
    int i;
    for(i=0; i<4; i++) a[i] = a[i]*a[i]+1;
}
```

Before:	2	5	0	0
After:	5	26	1	1

ตัวอย่างฟังก์ชันรับอาร์เรย์สองมิติ (Matrix)

- เมตริกซ์

```
#include <stdio.h>
#define MX 10
void inputMatrix(int m[MX][MX], int row, int col)
{   int i,j;
    for(i=0; i<row; i++)
        for(j=0; j<col; j++)
            {   printf("Matrix [%d][%d] : ",i+1,j+1);
                scanf("%d",&m[i][j]);
            }
}
int sumMatrix(int m[MX][MX], int row, int col)
{   int i,j,sum=0;
    for(i=0; i<row; i++)
        for(j=0; j<col; j++)    sum+=m[i][j];

    return sum;
}
```

ตัวอย่างฟังก์ชันรับอาร์เรย์สองมิติ (Matrix)

```
void showMatrix(int m[MX][MX])
{   int i,j;
    for(i=0; i<row; i++)
    {   for(j=0; j<col; j++) printf("%d\t", m[i][j]);
        printf("\n");
    }
}

int main() {
    int mat[MX][MX]={0}, r, c;
    printf("Matrix dimension: ");
    scanf("%d %d", &r, &c);   showMatrix(mat, r, c);

    inputMatrix(mat, r, c);   showMatrix(mat, r, c);
    printf("Sum all elements = %d\n", sumMatrix(mat, r, c));
    return 0;
}
```

Matrix dimension : 2 3

0 0 0

0 0 0

Matrix[1][1] : 1

Matrix[1][2] : 2

Matrix[1][3] : 3

Matrix[2][1] : 4

Matrix[2][2] : 5

Matrix[2][3] : 6

1 2 3

4 5 6

Sum all elements = 21

สายตัวอักษร (String)

- สตริง หรือ สายตัวอักษร คือ ข้อมูลที่ประกอบไปด้วยค่าคงที่ชนิดตัวอักษรเรียงต่อเนื่องกันไป โดยมีจุดสิ้นสุดที่ตัวอักษร Null Character ('\0') ซึ่งมีค่ารหัสเป็น 0 เป็นตัวบ่งบอกจุดสิ้นสุดของstring (หรือเรียกว่า end character)
- การเก็บข้อมูลสตริง ใช้ตัวแปรชนิด อาร์เรย์ของตัวอักษร จึงจัดเป็นโครงสร้างของอาร์เรย์ 1 มิติ

char stringVariable[length_of_string]

การประกาศตัวแปรชนิดสตริง

ทำได้เช่นเดียวกับการประกาศตัวแปรอาร์เรย์ 1 มิติ ในการ

กำหนดตัวสตริงและมีการกำหนดค่าเริ่มต้นไปพร้อมกันนั้น อาจ

ไม่ต้องใส่จำนวนข้อมูลภายในวงเล็บบอกจำนวนสมาชิกก็ได้

เช่น

```
char str1[] = { 'A' , 'B' , 'C' , 'D' , '\0' } ;
```

```
char str2[10] = { 'A' , 'B' , 'C' , 'D' , '\0' } ;
```

```
char str3[] = "ABCD" ;
```

ตัวอย่างการประกาศที่ผิด

```
char str4[2] = { 'A' , 'B' , 'C' , 'D' , '\0' } ;
```

ตัวอย่างของสตริง

```
char a[7] = "Hello!";
```

จากตัวอย่าง "Hello!" คือ ค่าคงที่สตริง ซึ่งจะเขียนล้อมไว้ด้วยเครื่องหมาย " ค่าคงที่ "Hello!" ถูกกำหนดให้เป็นค่าเริ่มต้นแก่ตัวแปร **a** ซึ่งเป็นอาร์เรย์ขนาด 7 หน่วย หลังจากการกำหนดค่าแล้ว ในตัวแปร **a** จะมีลักษณะ ดังนี้

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
'H'	'e'	'l'	'l'	'o'	'!'	'\0'

ตัวอย่างของสตริง

- คอมไพเลอร์จะกำหนดจำนวนพื้นที่ตัวอักษรจากจำนวนตัวอักษรในค่าคงที่ของสตริงรวมกับ `'\0'`
- หรืออาจกำหนดพื้นที่ตัวอักษรให้มากกว่าความยาวของสตริงที่ใช้กำหนดเป็นค่าเริ่มต้นได้ เช่น

```
char a[10] = "Hello!";
```

- มีการเก็บข้อมูล ดังรูป

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
'H'	'e'	'l'	'l'	'o'	'!'	'\0'	0	0	0

- $a[6] = 0$ หรือ `'\0'` , $a[6]$ เป็น end character ของสตริง ส่วน $a[7] = a[8] = a[9] = 0$ ถูกเซ็ตเป็นศูนย์อัตโนมัติ เพราะ เป็น element ส่วนที่เหลือ ที่ยังไม่ได้การกำหนดค่าเริ่มต้นให้

ตัวอย่าง : การแสดงค่าของสตริง

- ใช้ **%s** ใน printf

- ไฟล์ arrayex17.c

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char name[5] = { 'J', 'o', 'h', 'n', '\0' } ;
```

```
    char surname[8] = { 'F', '\0', 'K', 'E', 'N' } ;
```

```
    printf("Name: %s \n", name);
```

```
    printf("Surname: %s \n", surname);
```

```
    printf("String constant: %s \n", "any text");
```

```
    return 0;
```

```
}
```

Name: John

Surname: F

String constant: any text

ตัวอย่างที่ : การรับค่าของสตริง

- ใช้ %s ใน scanf (รับข้อความ ไม่รวม space หรือ newline)
- ไฟล์ arrayex18.c

```
#include <stdio.h>
int main() {
    char n[10],s[10];
    printf("Enter name: ");
    scanf("%s",&n);    // or scanf("%s",n);

    printf("Enter surname : ");
    scanf("%s" ,&s);    // or scanf("%s",s);

    printf("Hello %s %s\n" , n, s);
    return 0;
}
```

Enter name: John F.
Enter surname: Kennedy
Hello John Kennedy

ตัวอย่างที่ : การรับค่าของสตริง ที่ละบรรทัด

- ใช้ฟังก์ชัน **gets** (รับข้อความทั้งบรรทัด รวมช่องว่าง หรือ space)
- ไฟล์ arrayex18.c

```
#include <stdio.h>
int main() {
    char n[20],c[20];
    printf("Enter name and surname: ");
    gets(n);

    printf("City : ");
    gets(c);


    printf("Hello %s %s\n", n, c);
    return 0;
}
```

Enter name and surname: John F. Kennedy
City : Washington D.C.
Hello John F. Kennedy Washington D.C.

การกำหนดค่าให้กับตัวแปรชนิดสตริง

การกำหนดค่าของอาร์เรย์ให้เป็นข้อความ จะไม่สามารถใช้ โอเปอเรเตอร์ = ได้โดยตรง เช่น

```
char a[20];  
a = "Hello World!";
```



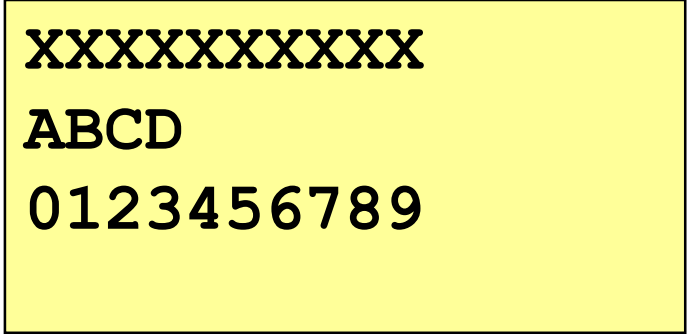
ถ้าต้องการเก็บข้อความใดๆ ลงในอาร์เรย์ ต้องใช้วิธีการเรียกใช้ฟังก์ชัน เช่น ฟังก์ชันมาตรฐาน `strcpy()` ที่นิยามไว้ใน `<string.h>`

ตัวอย่างที่ 19 : การกำหนดค่าของสตริงโดยใช้ฟังก์ชัน strcpy()

```
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20] = "XXXXXXXXXX"; // 10 of X

    printf("%s\n", a);
    strcpy(a, "ABCD");
    printf("%s\n", a);
    strcpy(a, "0123456789");
    printf("%s\n", a);

    return 0;
}
```



XXXXXXXXXX
ABCD
0123456789

ฟังก์ชันมาตรฐานที่นิยามใน <string.h>

- `strcpy (char str1[], char str2[])`

คัดลอกข้อความจาก `str2` ไปใส่ใน `str1` (ตัวแรกจะถูกเปลี่ยนค่าไป ตัวที่สองค่าคงเดิม) ***string copy***

- `int strlen (char str1[])`

ฟังก์ชัน `return` ความยาวสายอักขระ = จำนวนตัวอักษรก่อนถึง `'\0'` ตัวแรก
string length

- `strcat(char str1[], char str2[])`

นำข้อมูลจากสายอักขระ `str2` ไปใส่ต่อท้ายสายอักขระ `str1` (ตัวแรกจะถูกเปลี่ยนค่าไป ตัวที่สองค่าคงเดิม) ***string concatenation***

ฟังก์ชันมาตรฐานที่นิยามใน <string.h>

- **int strcmp (char str1[], char str2[])**

เทียบค่าสตริงทั้งสอง โดยค่าที่ return จะมีค่า

- น้อยกว่า 0 (**-1,-2, ...**) เมื่อ str1[] น้อยกว่า str2[] เช่น
"abc" < "xyz", "a1" < "a2"
- เท่ากับ 0 เมื่อ str1[] เท่ากับ str2[]
- มากกว่า 0 (**1,2,...**) เมื่อ str1[] มากกว่า str2[] เช่น
"apples" > "apple", "a" > "A", "rat" > "cat",
"cat" > "Rat"

ตัวอย่าง การใช้ฟังก์ชันเกี่ยวกับสตริง

- ไฟล์ arrayex20.c

```
#include <stdio.h>
#include <string.h>
int main()
{
    char name[20];
    int len;
    strcpy(name, "David Young" );
    len = strlen(name);
    printf("Length of %s is %d\n",name, len);
    return 0;
}
```

Length of David Young is : 11

ตัวอย่าง การใช้ฟังก์ชันเกี่ยวกับสตริง

- ไฟล์ arrayex21.c

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[] = "CAN";
    int i, len = strlen(str);

    printf("%s\n",str);
    for(i=0; i<len; i++)
        str[i] = str[i] + 1;

    printf("%s\n",str);
    return 0;
}
```



CAN
DBO

แบบฝึกหัดการใช้ฟังก์ชันเกี่ยวกับสตริง

- จงเขียนโปรแกรม นับจำนวนตัวเลข(**digit**)ที่ปรากฏ
ถ้าข้อความเป็น “END” ให้จบการทำงาน

```
#include <stdio.h>
#include <string.h>
#define MX 101 //max characters in string
int main()
{ char line[MX]=""; int d, i;
  gets(line);
  do
  { for(i=0, d=0; i<strlen(line); i++)
    { if(line[i]>='0' && line[i]<='9') d++;
    }
    printf("%d\n", d);
    gets(line);
  } while( strcmp(line, "END") != 0 );
  return 0;
}
```

```
Input line 1
1
Another line
0
123 9345
7
Date:11/12/2013
8
ENDING
0
END
```

อาร์เรย์ของสตริง

- เนื่องจาก **String** จัดเป็นอาร์เรย์ 1 มิติ

```
char name[6] = "Nesta";
```

- ดังนั้นอาร์เรย์ของ **String** จึงต้องมีรูปแบบเป็นอาร์เรย์ 2 มิติ เช่น

```
char names[2][8] = { "Nesta", "Maldini" };
```

names[0]	N	e	s	t	a	\0	\0	\0
names[1]	M	a	l	d	i	n	i	\0

โจทย์ฝึกสมองที่ 4 : ไฟล์ arrayprac4.c

- จงเขียนโปรแกรมสำหรับรับข้อความจากผู้ใช้ 4 ข้อความ โดยแต่ละข้อความมีความยาวไม่ถึง 20 ตัวอักษร จากนั้นพิมพ์ค่าความยาวของแต่ละข้อความออกทางหน้าจอ

(ฟังก์ชันที่ใช้ในการหาความยาวของข้อความคือ `strlen`)

ตัวอย่างผลการรันเป็นดังนี้

```
Enter text 1: Maths
Enter text 2: Physics
Enter text 3: Chemistry
Enter text 4: Biology
Maths: 5
Physics: 7
Chemistry: 9
Biology: 7
```

```
//arrayprac4.c
```

```
#include <stdio.h>
#include <string.h>
int main()
```

```
{
```

```
    char texts[4][20];
```

```
    int i;
```

```
    for(i=0;i<4;i++) {
```

```
        printf("Enter text %d : " ,i+1 );
```

```
        scanf("%s",&texts[i]);
```

```
    }
```

```
    for(i=0;i<4;i++)
```

```
        printf("%s:  %d\n" ,texts[i],strlen(texts[i]);
```

```
    return 0;
```

```
}
```

```
Enter text 1: Maths
```

```
Enter text 2: Physics
```

```
Enter text 3: Chemistry
```

```
Enter text 4: Biology
```

```
Maths: 5
```

```
Physics: 7
```

```
Chemistry: 9
```

```
Biology: 7
```


แบบฝึกหัดท้ายบท

- จงเขียนโปรแกรมคำนวณผลรวมจำนวนเต็มจาก **1** ถึง **n** โดยสร้างฟังก์ชัน **int summ(int n)** สำหรับการคำนวณ (ทดลองเขียนเวอร์ชันการใช้ วนซ้ำ กับ เวอร์ชันการใช้ **recursive function**)
- จงเขียนฟังก์ชันสำหรับการบวกเมตริกส์ ที่มีขนาดเท่ากัน ให้ฟังก์ชันรับอาร์เรย์ **2 มิติ** (เมตริกส์) สามตัว และ จำนวนแถวและหลักของเมตริกส์
- จงเขียนฟังก์ชันแก้ไขค่าของทุก **element** ในอาร์เรย์ให้เป็นสัมบูรณ์ (**absolute**) หรือค่าบวกเสมอ โปรดใช้ **void absolute (int array[100], int n)**

อาร์เรย์มีความยาวไม่เกิน **100**

และค่า **n** คือจำนวน **element** ของอาร์เรย์

