

# บทที่ 5

## ฟังก์ชัน

สุนทรี คุ่มไพโรจน์

# ฟังก์ชัน(Function)

- ชุดคำสั่งการทำงาน ที่ถูกเขียนขึ้นเพื่อให้โปรแกรมเมอร์สามารถเรียกใช้งานได้ง่าย
- โปรแกรมเมอร์สามารถเรียกใช้ ฟังก์ชัน โดยไม่จำเป็นต้องทราบรายละเอียดการทำงานทั้งหมดของระบบ
- โปรแกรมเมอร์สามารถเรียกใช้ ฟังก์ชัน โดยทราบเพียงวิธีการใช้งาน และ ผลลัพธ์ที่ได้หลังจากการเรียกใช้ฟังก์ชันเท่านั้น
- ฟังก์ชันทำให้โปรแกรมเมอร์สามารถเขียนโปรแกรมที่มีการทำงานที่ซับซ้อนได้

# ประเภทของฟังก์ชัน

- ฟังก์ชันไลบรารีมาตรฐาน (Standard Library function)
- ฟังก์ชันที่สร้างขึ้นเอง (User Defined function)

# Standard Library Function

- ฟังก์ชันที่มีอยู่ใน **Library** สามารถเรียกใช้งานได้โดยการ **include directives** ก่อน
- **Directive** คือ สารบัญของกลุ่มฟังก์ชัน เช่น **stdio.h, conio.h, string.h, math.h** เป็นต้น
- การ **include directives** จะทำให้ **compiler** ทราบว่ามีการใช้คำสั่ง ในกลุ่มของ **directive** นั้นๆ เช่นการใช้คำสั่ง **sin( )** ที่อยู่ใน **math.h** ต้อง **include math.h** ไว้เสมอ

ดังตัวอย่างโปรแกรม **L51.C**

```
1  #include <stdio.h>
2  #include <conio.h>
3  #include <math.h>
4  int main () {
5      double rad = -1.0;
6      do {
7          printf("Sine of %f is %f\n", rad, sin(rad));
8          rad += 0.1;
9      }while (rad <= 1.0)
10     return 0;
11 }
```

# การเรียกใช้ Standard Library Function

- ต้องทราบว่าโปรแกรมนี้ต้องการทำอะไร ต้องใช้ฟังก์ชันอะไร
- ต้องทราบ **directive** ที่เป็นสารบัญของคำสั่ง
- **#include directive**
- เรียกใช้ฟังก์ชันนั้น
- ตัวอย่าง **library function**
  - ฟังก์ชันการคำนวณทางคณิตศาสตร์
    - ไฟล์ header => math.h
  - ฟังก์ชันสำหรับตัวอักษรและ **string**
    - ไฟล์ header => ctype.h  
string.h

<https://pubs.opengroup.org/onlinepubs/9699919799/>

# ตัวอย่างฟังก์ชัน

## **#include <math.h>**

- sin(var)
- cos(var)
- tan(var)
- sqrt(var)
- pow(var1,var2)
- log(var)
- log10(var)
- exp(var)
- fabs(var)

## **#include <string.h>**

- Strcpy(str1, str2)
- Strcat(dest, src)
- Strcmp(dest, src)
- Strncpy(str1,str2,n)
- Strlen(str)

## **#include <ctype.h>**

- tolower(ch)
- toupper(ch)

## ตัวอย่างโปรแกรมที่ include library (L52.c)

```
1  #include <stdio.h>
2  #include <conio.h>
3  #include <math.h>
4  #define PI  3.14
5  int main () {
6      float deg, rad;
7      printf ("Enter Degree:");
8      scanf  ("%f", &deg);
9      rad = deg * PI /180;
10     printf("sin(%.2f) = %.3f\n",deg,sin(rad));
11     return 0;
12 }
```

# User-defined function

- เนื่องจาก **standard library function** ทั้งหมด เป็นฟังก์ชันมาตรฐานที่มีเฉพาะการทำงานพื้นฐานต่างๆ เท่านั้น
- หากต้องการฟังก์ชันที่มีการทำงานเฉพาะกิจ โปรแกรมเมอร์ต้องเขียนฟังก์ชันขึ้นมาเอง



# โครงสร้างโปรแกรมภาษา C

```
1  #include <file.h>           //preprocessor directive
2  type function_name(type);    //function prototype
3  type variable                //global variable
4  int main(){
5      type y;                  //local variable
6      type x;                  //local variable
7      statement-1;
8      ...
9      y=function_name(x);
10     statement-n;
11     return 0;
12 }
13 type function_name(type variable){
14     type var;                 //local variable
15     statement-1;
16     ...
17     statement-n;
18     return(var);
19 }
```

ส่วนหัวโปรแกรม

ฟังก์ชัน main ()

ฟังก์ชันย่อย

# ข้อกำหนดการเขียน user defined function

- ต้องมีการประกาศ **function prototype** ที่ต้นโปรแกรมเสมอ  
จึงจะเรียกใช้งาน **function** นั้นๆ ได้  
(เป็นการบอก **compiler** ว่าคำสั่งดังกล่าวคือฟังก์ชัน ไม่ใช่ **syntax error**)
- ต้องมีการเขียนฟังก์ชันตามโครงสร้างที่ได้ประกาศไว้ใน **function prototype** เท่านั้น
- การเขียน **user defined function** มี 2 รูปแบบ
  - สร้าง **function( )** ก่อน **main()**  
**main()** สามารถเรียกใช้งาน **function( )** ที่สร้างขึ้นได้
  - สร้าง **function( )** ต่อจาก **main( )**  
ต้องประกาศ **function prototype** ก่อนเพื่อให้ **main( )** รู้ว่ามีฟังก์ชันที่สร้างขึ้น

# Function Prototype

- การประกาศการใช้งานฟังก์ชัน

`type function_name(type1, type2,...,typen);`

<code>type</code>	คือ ชนิดของฟังก์ชัน ว่าฟังก์ชันที่ทำการสร้างจะส่งข้อมูลชนิดใดกลับ
<code>function_name</code>	คือ ชื่อฟังก์ชันที่จะสร้างขึ้น
<code>type_1...type_n</code>	คือ ชนิดของข้อมูลที่ส่งให้ฟังก์ชัน

# การสร้างฟังก์ชันก่อน main ()

```
1  #include <file.h>
2  type variable //global variable
3  type function_name(type variable){
4      type var;
5      statement-1;
6      ...
7      statement-n;
8      return(var) ;
9  }
10 int main(){
11     type y; //local variable
12     type x;
13     statement-1;
14     ...
15     y=function_name(x) ;
16     statement-n;
17     return 0;
18 }
```

# การสร้างฟังก์ชันต่อจาก main ()

```
1  #include <file.h>
2  type function_name(type) ;
3  type variable
4  int main() {
5      type y;
6      type x;
7      statement-1;
8      ...
9      y=function_name(x) ;
10     statement-n;
11     return 0;
12 }
13 type function_name(type variable) {
14     type var;
15     statement-1;
16     ...
17     statement-n;
18     return(var) ;
19 }
```

# ตัวอย่างโปรแกรม (L54.c)

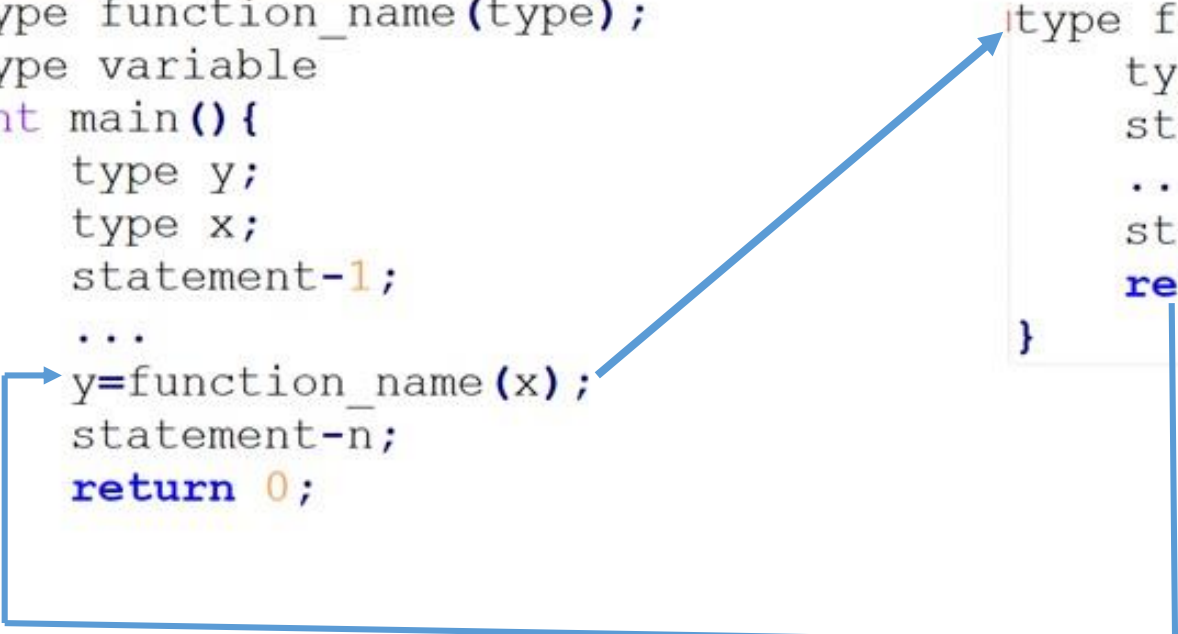
```
1  #include <stdio.h>
2  #include <conio.h>
3  void write_line (int n){
4      int i;
5      char c='-';
6      for (i=0;i<n;i++)
7          printf("%c",c);
8  }
9  int main() {
10     int length=9;
11     write_line(length);
12     printf("\n|   CS231   |\n");
13     write_line(length);
14     return 0;
15 }
```

```
1  #include <stdio.h>
2  #include <conio.h>
3  void write_line (int);
4  int main() {
5      int length=9;
6      write_line(length);
7      printf("\n|   CS231   |\n");
8      write_line(length);
9      return 0;
10 }
11 void write_line (int n){
12     int i;
13     char c='-';
14     for (i=0;i<n;i++)
15         printf("%c",c);
16 }
```

# การเรียกใช้งานของ User Defined Function

```
#include <file.h>
type function_name(type);
type variable
int main(){
    type y;
    type x;
    statement-1;
    ...
    y=function_name(x);
    statement-n;
    return 0;
}
```

```
type function_name(type variable){
    type var;
    statement-1;
    ...
    statement-n;
    return(var);
}
```



# การรับค่าและส่งค่าจากฟังก์ชัน

- ฟังก์ชันที่ไม่มีการรับส่งค่า
- ฟังก์ชันที่มีการรับค่า แต่ไม่ส่งค่ากลับ
- ฟังก์ชันที่มีการรับค่า และมีการส่งค่ากลับ
- ฟังก์ชันที่ไม่มีการรับค่า แต่มีการส่งค่ากลับ



# ฟังก์ชันที่ไม่มีการรับส่งค่า

- เป็นฟังก์ชันที่ไม่มีการรับค่าเข้ามาในฟังก์ชัน และ ไม่มีการส่งค่ากลับออกไปจากฟังก์ชัน

```
1  #include <file.h>
2  void function_name(void) {
3      statement-1;
4      statement-2;
5      ...
6      statement-n;
7  }
8  int main() {
9      statement-1;
10     ...
11     function_name();
12     statement-n;
13     return 0;
14 }
```

# ฟังก์ชันที่มีการรับค่า แต่ไม่ส่งค่ากลับ

- เป็นฟังก์ชันที่มีการรับค่าเข้ามาในฟังก์ชัน แต่ไม่มีการส่งค่ากลับไปออกไปจากฟังก์ชัน

```
1  #include <file.h>
2  void function_name(type varRe) {
3      statement-1;
4      statement-2;
5      ...
6      statement-n;
7  }
8  int main() {
9      statement-1;
10     ...
11     function_name(varSe);
12     statement-n;
13     return 0;
14 }
```

# ฟังก์ชันที่มีการรับค่า และมีการส่งค่ากลับ

- เป็นฟังก์ชันที่มีการรับค่าเข้ามาในฟังก์ชัน และมีการส่งค่ากลับออกไปจากฟังก์ชัน

```
1  #include <file.h>
2  type function_name(type);
3  type variable
4  □int main(){
5      type y;
6      type x;
7      statement-1;
8      ...
9      y=function_name(x);
10     statement-n;
11     return 0;
12 }
13 □type function_name(type variable){
14     type var;
15     statement-1;
16     ...
17     statement-n;
18     return(var);
19 }
```

# ฟังก์ชันที่ไม่มีการรับค่า แต่มีการส่งค่ากลับ

- เป็นฟังก์ชันที่ไม่มีการรับค่าเข้ามาในฟังก์ชัน แต่มีการส่งค่ากลับออกไปจากฟังก์ชัน

```
1  #include <file.h>
2  type function_name () {
3      type var;
4      statement-1;
5      ...
6      statement-n;
7      return (var) ;
8  }
9  int main () {
10     type y;
11     statement-1;
12     ...
13     y=function_name () ;
14     statement-n;
15     return 0;
16 }
```

# ตัวแปรและขอบเขตของการใช้งานสำหรับฟังก์ชัน

- ตัวแปร **global**

เป็นตัวแปรที่ฟังก์ชันใดก็สามารถเรียกใช้ได้โดยการประกาศตัวแปร  
ต่อจาก **preprocessor directive**

- ตัวแปร **local**

เป็นตัวแปรที่สามารถเรียกใช้ได้เฉพาะภายในฟังก์ชันที่ประกาศตัวแปรนั้น  
โดยจะประกาศตัวแปรภายในแต่ละฟังก์ชัน

# Global variable vs. Local variable (L55.c)

```
1  #include <stdio.h>
2  #include <conio.h>
3  int n;           //global variable
4  void test(void);
5  int main() {
6      n=1;
7      printf("main: before test:n=%d\n",n);
8      test();
9      printf("main: after  test:n=%d\n",n);
10     return 0;
11 }
12 void test(){
13     n=100;
14     printf("test: change n    :n=%d\n",n);
15 }
```

```
1  #include <stdio.h>
2  #include <conio.h>
3  void test(void);
4  int main() {
5      int n=1; //local variable
6      printf("main: before test:n=%d\n",n);
7      test();
8      printf("main: after  test:n=%d\n",n);
9      return 0;
10 }
11 void test(){
12     int n=100; //local variable
13     printf("test: local n    :n=%d\n",n);
14 }
```

# การส่งค่าตัวแปร

- การส่งค่าตัวแปรมี 2 ชนิดคือ
- การส่งค่าที่เก็บอยู่ในตัวแปรให้กับฟังก์ชัน (**pass by value**)
- การส่งค่า **address** ของตัวแปรให้กับฟังก์ชัน (**pass by reference**)

# Pass by value

- เป็นการส่งค่าที่เก็บอยู่ในตัวแปร(argument)เข้าสู่ฟังก์ชัน
- การเปลี่ยนแปลงค่าต่าง ๆ ของพารามิเตอร์ จะไม่เปลี่ยนแปลงค่าของตัวแปรใน **main( )**
- ฟังก์ชันสามารถคืนค่า (return) ค่าได้เพียง 1 ค่า หรือเราอาจจะเลือกไม่คืนค่าก็ได้

ตัวอย่าง

- **int plus(int a, int b)**
- **void write\_line(int count)**
- หากต้องการให้ฟังก์ชันมีการเปลี่ยนแปลงค่า และต้องการคืนค่ากลับมายังฟังก์ชันที่เรียกใช้ **มากกว่า 1 ค่า** จะต้องนำพอยน์เตอร์เข้ามาช่วย

```
1  #include <stdio.h>
2  void pass_value (int value) {
3      value=value+1;
4      printf("pass_value: value=%d\n",value);
5  }
6  int main() {
7      int i=10;
8      printf("\n In main: before call function i=%d\n",i);
9      pass_value(i);
10     printf("\n In main: after call function i=%d\n",i);
11     return 0;
12 }
```



# Pass by Reference

- เป็นการส่งค่า **address** ของตัวแปร(**parameter**)เข้าสู่ฟังก์ชัน  
เช่น

`callFunc(&a);`

- การเปลี่ยนแปลงค่าของพารามิเตอร์ในฟังก์ชัน จะส่งผลไปยังตัวแปรใน **main( )**
- การกำหนดพารามิเตอร์ในฟังก์ชันจะมีเครื่องหมาย **\*** หน้าตัวแปรเสมอ  
เช่น

`int callFunc(int *a)`

`void max(int *a, int *b)`

## ตัวอย่างโปรแกรม (L56-1.c)

```
1  #include <stdio.h>
2  void pass_ref (int *ref) {
3      *ref = *ref +1;
4      printf("pass_ref: *ref=%d\n",*ref) ;
5  }
6  int main() {
7      int i=10;
8      printf("\n In main: before call function i=%d\n",i);
9      pass_ref(&i);
10     printf("\n In main: after call function  i=%d\n",i);
11     return 0;
12 }
```

ตัวอย่าง โปรแกรมตัวอย่างการสลับค่าตัวแปร 2 ตัวโดยผ่านฟังก์ชัน  
เป็นการ pass by reference

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

```
void swap (int *, int *);
```

```
void main ( ) {
```

```
    int x = 5, y = 10;
```

```
    printf("Before swap : x = %d y = %d\n", x, y);
```

```
    swap ( &x, &y);          /* Pass address of x and y to swap( ) */
```

```
    printf("After swap : x = %d y = %d\n", x, y);
```

```
}
```

```
void swap (int *px, int *py) {  
    int temp;  
    temp = *px;          /* Keep x value to temp */  
    *px   = *py;          /* Assign y value to x */  
    *py   = temp;         /* Assign old x value to y */  
}
```

ตัวอย่าง โปรแกรมเพื่อรับข้อมูลจำนวนจริง 3 จำนวน จากผู้ใช้  
และหาค่าเฉลี่ยของค่าที่รับเข้ามาทั้งหมด เป็นการ pass by reference

วิเคราะห์ input-process-output

- มี 3 งานย่อย
- รับข้อมูล 3 จำนวน
  - หาค่าเฉลี่ย
  - แสดงผลลัพธ์

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

```
void readData(int, double *);
```

```
void calAverage(double, double, double, double *);
```

```
void printData(double, double, double, double);
```

```
void main( ) {  
    double x1, x2, x3, average;  
    readData(1, &x1);  
    readData(2, &x2);  
    readData(3, &x3);  
    calAverage(x1, x2, x3, &average);  
    printData(x1, x2, x3, average);  
}
```

```
void readData(int seq, double *px) {  
    printf("Enter value %d : ", seq);  
    scanf("%lf", px);  
}  
  
void calAverage(double a1, double a2, double a3, double *pAvg) {  
    *pAvg = (a1 + a2 + a3)/3.0;  
}  
  
void printData(double b1, double b2, double b3, double avg) {  
    printf("Average of %.2lf, %.2lf, %.2lf is %.2lf", b1, b2, b3, avg);  
}
```

ตัวอย่าง โปรแกรมเพื่อรับข้อมูลจำนวนจริง 3 จำนวนจากผู้ใช้  
และหาค่าเฉลี่ยของค่าที่รับเข้ามาทั้งหมด Pass by reference  
แต่เขียนในอีกลักษณะหนึ่ง

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

```
void readData(double *, double *, double *);
```

```
void calAverage(double *, double *, double *, double *);
```

```
void printData(double *, double *, double *, double *);
```

```
void main( ) {  
    double x1, x2, x3, average;  
    readData(&x1, &x2, &x3);  
    calAverage(&x1, &x2, &x3, &average);  
    printData(&x1, &x2, &x3, &average);  
}
```

```
void readData(double *px1, double *px2, double *px3) {  
    printf("Enter value 1 : ");  
    scanf("%lf", px1);  
    printf("Enter value 2 : ");  
    scanf("%lf", px2);  
    printf("Enter value 3 : ");  
    scanf("%lf", px3);  
}
```

```
void calAverage(double *pa1, double *pa2, double *pa3, double *pAvg) {  
    *pAvg = (*pa1 + *pa2 + *pa3)/3.0;  
}  
  
void printData(double *pb1, double *pb2, double *pb3, double *pAvg) {  
    printf("Average of %.2lf, %.2lf, %.2lf is %.2lf",  
          *pb1, *pb2, *pb3, *pAvg);  
}
```



# ตัวอย่าง

เขียนโปรแกรมเพื่อคำนวณพื้นที่ของสี่เหลี่ยมรูปหนึ่ง

โดยรับข้อมูลความกว้างและความยาวของรูปสี่เหลี่ยมจากผู้ใช้ กำหนดให้ใช้ฟังก์ชันเพื่อคำนวณพื้นที่ของรูปสี่เหลี่ยม

ดังโปรโตไทป์

```
void calRecArea(float, float, float *);
```

โดยที่

พารามิเตอร์ตัวแรกคือความกว้าง

พารามิเตอร์ตัวที่ 2 คือความยาว

และพารามิเตอร์ตัวที่ 3 คือพื้นที่ของรูปสี่เหลี่ยม

```
#include <stdio.h>

void calRecArea(float w, float l, float *pArea) {
    *pArea = w * l;
}

void main( ) {
    float width, length, area;
    printf("Enter width : ");
    scanf("%f", &width);
    printf("Enter length : ");
    scanf("%f", &length);
    calRecArea(width, length, &area);
    printf("Rectangle area is %.2f", area);
}
```