



## หัวข้อ

- อัลกอริทึมในชีวิตประจำวัน
- กลยุทธ์ Divide and Conquer
- แนะนำฟังก์ชัน (Function)
- C Standard Library
- สร้างฟังก์ชันขึ้นเอง
- การใช้ฟังก์ชันกับแถวลำดับ

## ขั้นตอนการทำข้าวผัดปู

1. นำน้ำมันพืช ตั้งไฟให้ร้อน จากนั้นใส่กระเทียมเจียวให้หอม
2. จากนั้นตอกไข่ไก่ ใช้ไข่สองฟองเพื่อเป็นการเพิ่มพลังให้กับร่างกาย ผัดไข่ให้สุกใช้ไฟกลาง
3. เทข้าวสวยใช้หอมมะลิต่างดี ใส่เนื้อปูลงไปด้วย แล้วทำการผัดให้เข้ากัน
4. เปรูรสด้วยพริกไทย น้ำมันหอย ซีอิ๊วขาว น้ำตาลทราย
5. ได้ที่แล้ว ใส่ต้นหอม ผักชี โรยหน้า รับประทานใส่จาน พร้อมแต่งกวารอเสิร์ฟ
6. จากนั้นใส่จานรอเสิร์ฟ อย่าลืม ต้นหอม แต่งกวาหั่นเป็นแว่นๆ แล้วมะนาวผ่าซีก ไว้ข้างจาน เวลารับประทานก็จะเอร็ดอร่อยกับอาหารจานโปรด...

## กลยุทธ์ Divide and Conquer

### ขั้นตอนการรับบริการงานผู้ป่วยนอก

- ขั้นตอนที่ 1 : ยื่นบัตรโรงพยาบาล,บัตรทองหรือบัตรสิทธิอื่นๆ,หรือบัตรนัด และรับบัตรคิวที่จุดประชาสัมพันธ์
- ขั้นตอนที่ 2 : ห้องบัตรค้นเวชระเบียนผู้ป่วย (OPD Card) ส่งจุดซักประวัติ หน้าห้องเบอร์ 7
- ขั้นตอนที่ 3 : ซักประวัติ, ชั่งน้ำหนัก, วัดความดันโลหิต, วัดปรอท, ตามลำดับบัตรคิว
- ขั้นตอนที่ 4 : เข้ารับการตรวจจากแพทย์ ณ ห้องตรวจโรค
- ขั้นตอนที่ 5 : พบพยาบาล ให้คำแนะนำหน้าห้องตรวจโรค, รับคำแนะนำ

## แนะนำฟังก์ชัน

ฟังก์ชันเป็นกลุ่มคำสั่งที่ใช้ในการแบ่งโปรแกรมเป็นส่วนย่อย ๆ เพื่อทำงานอย่างใดอย่างหนึ่ง ฟังก์ชันนี้อาจมีการส่งข้อมูลไปและกลับระหว่างฟังก์ชันได้

รูปแบบของฟังก์ชัน

จะมีหรือไม่ขึ้นการลักษณะของฟังก์ชัน

```
ชนิดข้อมูล ชื่อฟังก์ชัน(การประกาศพารามิเตอร์)
{
    การประกาศตัวแปรเฉพาะที่
    กลุ่มคำสั่ง
    [return]
}
```

ค่าที่ส่งกลับ (ถ้ามี)

## ฟังก์ชันแรกในภาษาซี (main)

```
#include "stdio.h"
void main()
{
    int x,y,z;
    x = 100;
    y = 65;
    z = x + y;
    printf("%d",z);
}
```

Test.exe

เรียกใช้ฟังก์ชัน test()

```
#include "stdio.h"
void test()
{
    int x, y, z;
    x = 100;
    y = 65;
    z = x + y;
    printf("%d",z);
}
void main()
{
    test();
}
```

## แนะนำฟังก์ชัน

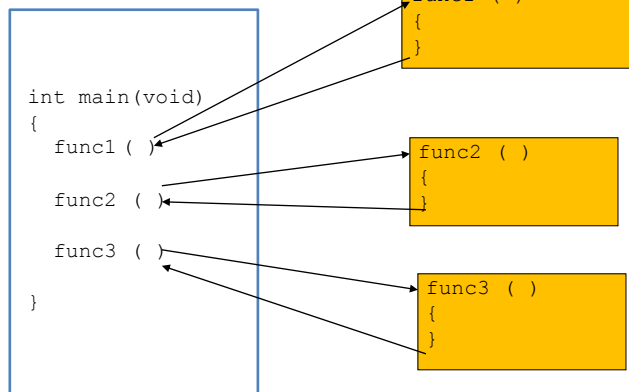
- เป็นเทคนิคในการเขียนโปรแกรม
- มองภาพรวม วิเคราะห์แล้วแตกปัญหาเป็นส่วน ๆ
- ลำดับขั้นตอนในแต่ละส่วน
- แต่ละส่วนเรียกว่าโมดูล ในภาษาซี เรียกว่า ฟังก์ชัน (Function)
- กลไกการไหล (Flow) เมื่อมีการเรียกใช้โมดูลเป็นไปโดยอัตโนมัติ
  - เรียกใช้->คืนกลับ
- เมื่อต้องการติดต่อโมดูล
  - เมื่อจะใช้โมดูลให้เรียกชื่อโมดูล
  - เมื่อโมดูลทำงานเสร็จสิ้นจะคืนกลับสู่จุดที่เรียกใช้

## โมดูลหรือฟังก์ชัน

- ลักษณะของโมดูล
  - มีหน้าที่การทำงาน หรือมีเป้าหมาย
  - สามารถรับข้อมูลจากภายนอกเข้าไปทำงานได้
  - สามารถส่งข้อมูลกลับจากการทำงานได้
- เมื่อมีหลาย ๆ โมดูล แต่ละโมดูลมีหน้าที่ของใครของมัน
- เมื่อนำมาประกอบกันเป็นโปรแกรม จะทำให้มองดูเป็นสักระยะเป็นส่วนลดความซับซ้อน ทำความเข้าใจได้ง่าย แยกกันไปเขียนได้

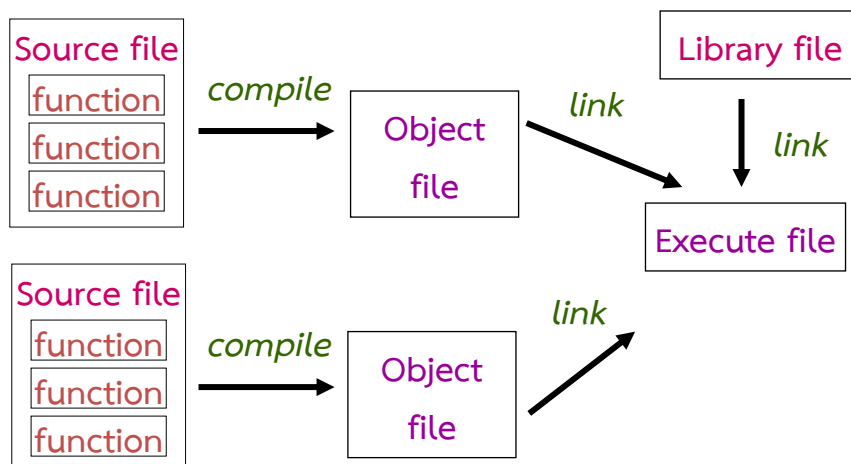
## ฟังก์ชันในภาษาซี

Main program

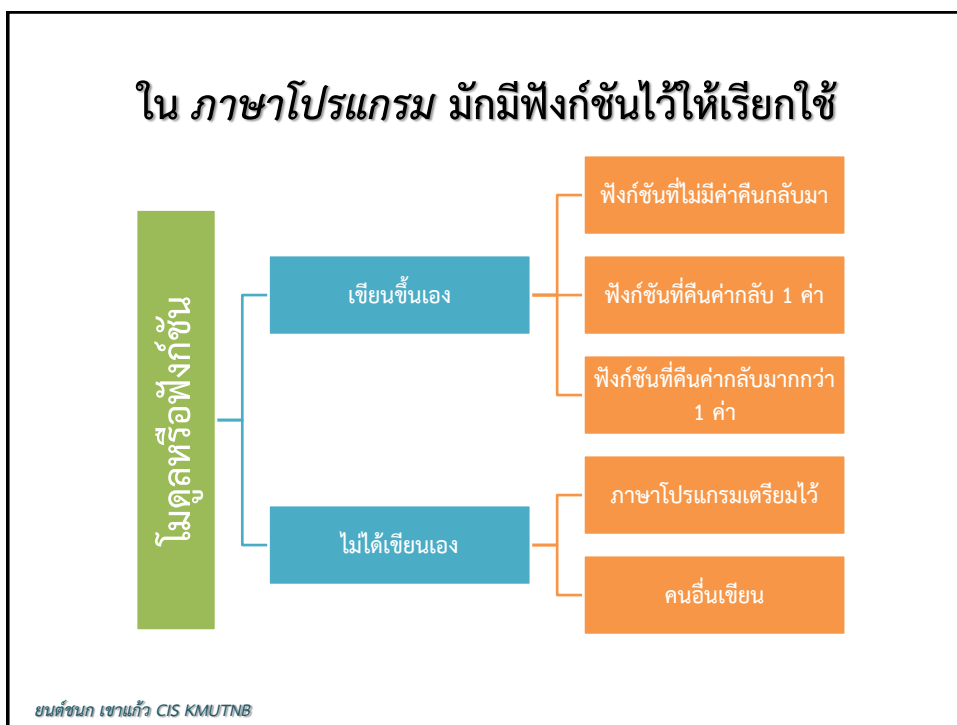
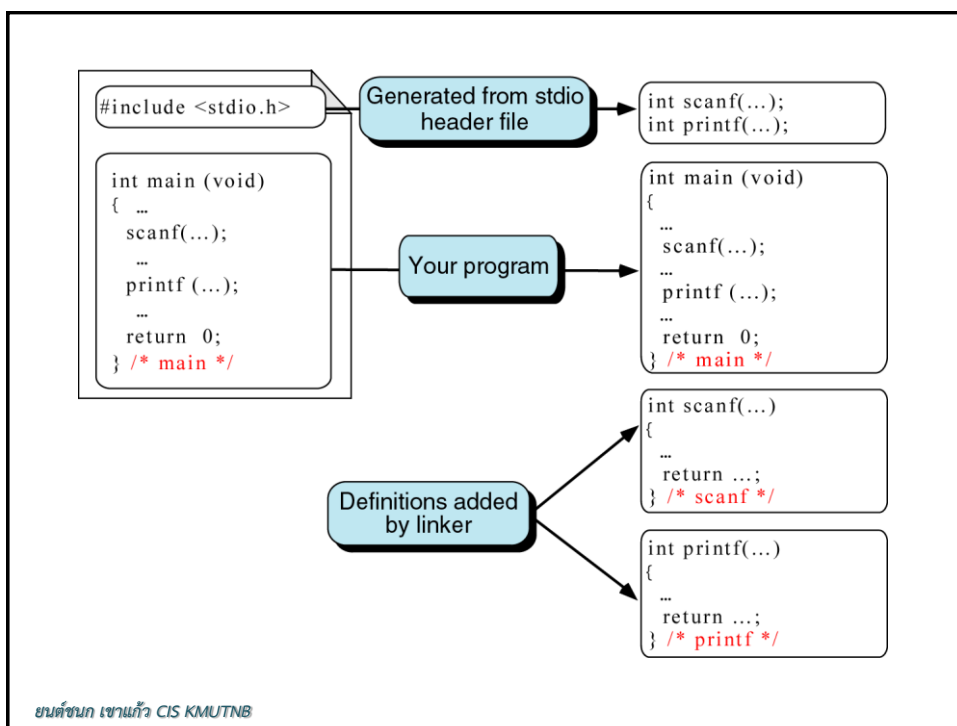


ยนต์ชนก เขานแก้ว CIS KMUTNB

## การเขียนโปรแกรมภาษาซี



ยนต์ชนก เขานแก้ว CIS KMUTNB



## ยกตัวอย่างฟังก์ชันที่คืนค่า

ฟังก์ชันสร้างทหารในเกม Clash of Clans เมื่อมีการทำงานสำเร็จเราก็จะได้ทหารออกมาใช้งานได้



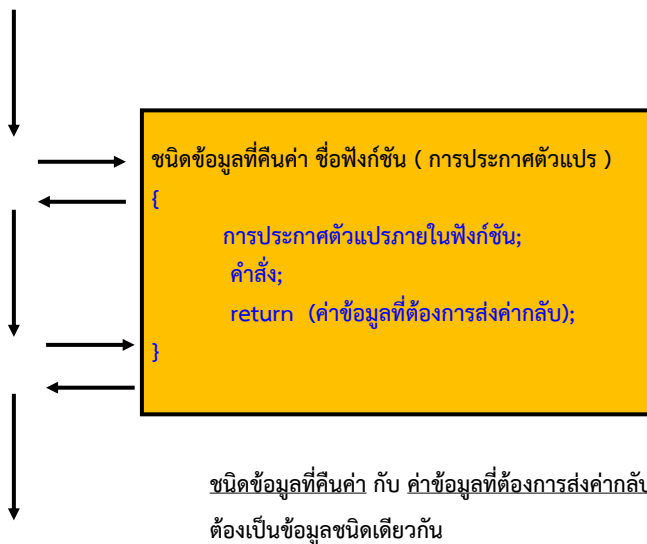
## ยกตัวอย่างฟังก์ชันที่ไม่คืนค่า

ฟังก์ชันอัปเกรดบ้อมในเกม Clash of Clans เมื่อมีการทำงานสำเร็จเราก็จะไม่ได้อะไรออกมา (แต่จะได้บ้อมใหม่ที่มีความสามารถมากขึ้น)



## องค์ประกอบของฟังก์ชัน รูปแบบที่ 1

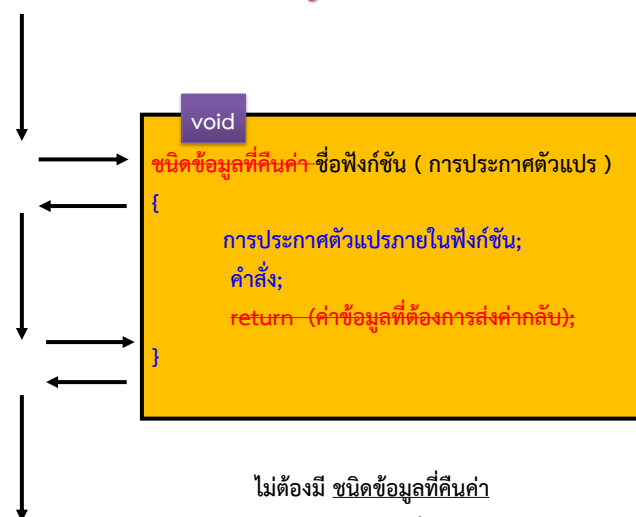
1. ส่วนหัว
2. ส่วนตัว
3. ส่วนคืนกลับ



ยนต์ชนก เขานแก้ว CIS KMUTNB

## องค์ประกอบของฟังก์ชัน รูปแบบที่ 2

1. ส่วนหัว
2. ส่วนตัว
3. ส่วนคืนกลับ

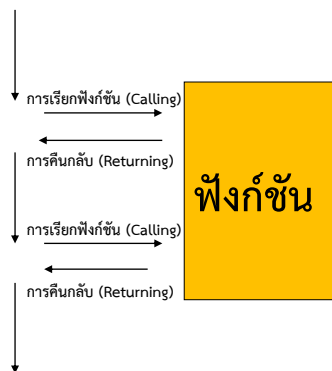


ยนต์ชนก เขานแก้ว CIS KMUTNB



## องค์ประกอบของการไหล (Flow)

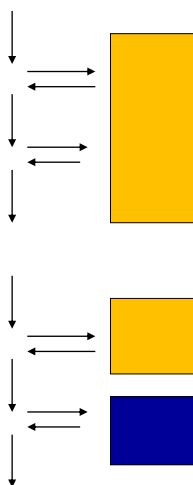
1. การเรียกฟังก์ชัน (Calling)
2. การคืนกลับ (Returning)



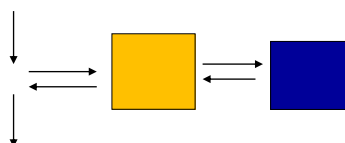
ฟังก์ชันเดิมสามารถเรียกได้หลายครั้ง

ยนต์ชนก เขานแก้ว CIS KMUTNB

## รูปแบบการเรียกใช้ฟังก์ชัน



- 1 ฟังก์ชันเรียกใช้ได้หลายครั้ง
- มีฟังก์ชันให้เรียกใช้หลายฟังก์ชัน
- ฟังก์ชันเรียกใช้ฟังก์ชันอื่นได้
- ทุกการเรียกใช้จะคืนกลับสู่จุดที่เรียก



ยนต์ชนก เขานแก้ว CIS KMUTNB

## โพรโทไทป์ (Prototype)

ไม่ว่าจะเป็นฟังก์ชันจากคลังหรือฟังก์ชันที่เขียนขึ้นเองเมื่อจะเรียกใช้ฟังก์ชันให้ดูที่ โพรโทไทป์ ซึ่งโพรโทไทป์จะมีส่วนประกอบหลัก 3 ส่วนของ function

header

**type** **function\_name** (**param 1, ...,param n**)

ชื่อฟังก์ชัน(function name)

พารามิเตอร์ (parameter list)

ชนิดของข้อมูลที่ส่งกลับ (function return type)

## โพรโทไทป์ (Prototype)

- โพรโทไทป์ไม่มีการประกาศชื่อตัวแปร มีแต่การเขียนประเภทของตัวแปรไว้ภายใน
- เป็นการช่วยให้คอมไพเลอร์สามารถตรวจสอบจำนวนของตัวแปรประเภทของตัวแปร ประเภทของการคืนค่า ภายในโปรแกรมว่ามีการเรียกใช้งานสิ่งต่าง ๆ เกี่ยวกับฟังก์ชันนั้นถูกต้องหรือไม่
- อาจแยกส่วนโปรโตไทป์ไปเขียนไว้ในอินคลูชไฟล์ได้

# C Standard Library

[https://en.wikipedia.org/wiki/C\\_standard\\_library](https://en.wikipedia.org/wiki/C_standard_library)

- ctype.h: character manipulation
- math.h: mathematical functions
- stdio.h: standard input/output
- stdlib.h: random numbers, memory handling
- string.h: string manipulation
- time.h: date/time functions

math.h			
ตัวอย่างฟังก์ชัน	ความหมาย	ชนิดของตัวแปรเข้า	ชนิดของผลลัพธ์
abs(x)	ค่าสัมบูรณ์ของจำนวนเต็ม x	int	int
labs(x)	ค่าสัมบูรณ์ของจำนวนเต็มความจุสูง x	long	long
fabs(x)	ค่าสัมบูรณ์ของจำนวนจริง x	double	double
atan(x)	ค่า arctangent ของ x	double	double
sin(x)	ค่า sine ของ x	double	double
cos(x)	ค่า cosine ของ x	double	double
pow(x, y)	ค่า x ยกกำลัง y	double	double
fmod(x, y)	ค่าเศษจากการหาร x/y แบบจำนวนจริง	double	double

ยนต์ชนก เขานแก้ว CIS KMUTNB

# C library function pow()

cppreference.com

Create account

Search

Page

Discussion

View

Edit

History

C

Numerics

Common mathematical functions

## pow, powf, powl

Defined in header <math.h>

float powf( float base, float exponent ); (1) (since C99)

double pow( double base, double exponent ); (2)

long double powl( long double base, long double exponent ); (3) (since C99)

Defined in header <tgmath.h>

#define pow( base, exponent ) (4) (since C99)

1-3) Computes the value of base raised to the power exponent.

4) Type-generic macro: If any argument has type `long double`, `powl` is called. Otherwise, if any argument has integer type or has type `double`, `pow` is called. Otherwise, `powf` is called. If at least one argument is complex or imaginary, then the macro invokes the corresponding complex function (`cpowf`, `cpow`, `cpowl`).

### Parameters

**base** - base as floating point value

**exponent** - exponent as floating point value

### Return value

If no errors occur, base raised to the power of exponent ( $base^{exponent}$ ) is returned.

If a domain error occurs, an implementation-defined value is returned (NaN where supported).

If a pole error or a range error due to overflow occurs, `±HUGE_VAL`, `±HUGE_VALF`, or `±HUGE_VALL` is returned.

If a range error occurs due to underflow, the correct result (after rounding) is returned.

# C library function sqrt()

cppreference.com

Create account

Search

Page

Discussion

View

Edit

History

C

Numerics

Common mathematical functions

## sqrt, sqrtf, sqrtl

Defined in header <math.h>

float sqrtf( float arg ); (1) (since C99)

double sqrt( double arg ); (2)

long double sqrtl( long double arg ); (3) (since C99)

Defined in header <tgmath.h>

#define sqrt( arg ) (4) (since C99)

1-3) Computes square root of arg.

4) Type-generic macro: If arg has type `long double`, `sqrtl` is called. Otherwise, if arg has integer type or the type `double`, `sqrt` is called. Otherwise, `sqrtf` is called. If arg is complex or imaginary, then the macro invokes the corresponding complex function (`csqrtf`, `csqrt`, `csqrtl`).

### Parameters

**arg** - floating point value

### Return value

If no errors occur, square root of arg ( $\sqrt{arg}$ ), is returned.

If a domain error occurs, an implementation-defined value is returned (NaN where supported).

If a range error occurs due to underflow, the correct result (after rounding) is returned.

# ฟังก์ชันมาตรฐาน

การใช้งานฟังก์ชันมาตรฐาน ต้องรู้ว่าจะรับข้อมูลเข้าเป็นอะไรและคืนค่าประเภทใดออกมา

ตัวอย่าง ฟังก์ชันทางคณิตศาสตร์ `#include <math.h>`

ฟังก์ชัน `sin(x)`, `cos(x)`, `tan(x)` หาค่าของฟังก์ชันโดยรับค่ามุมเป็นเรเดียน

ฟังก์ชัน `sqrt(x)` หาค่ารากที่สองของ  $x$

ฟังก์ชัน `exp(x)` ฟังก์ชันหาค่า  $e^x$  โดย  $e$  มีค่าประมาณ 2.718282

ฟังก์ชัน `pow(x,y)` ใช้หาค่า  $x^y$  โดย  $x$  เป็นค่าคงที่หรือตัวแปรที่มีค่ามากกว่า 0 ส่วน  $y$  เป็นค่ายกกำลัง

ฟังก์ชัน `log(x)` , `log10(x)` หาค่า  $\log$  ฐาน  $n$  และค่า  $\log$  ฐาน 10

## โปรแกรมหาค่าจากฟังก์ชันตรีโกณมิติ

```
#include "stdio.h"
```

```
#include "math.h"
```

```
main()
```

```
{
```

```
double r, pi = 3.14159;
```

```
r = pi/180;
```

```
printf("%f\n",sin(r));
```

```
printf("%f\n",cos(r));
```

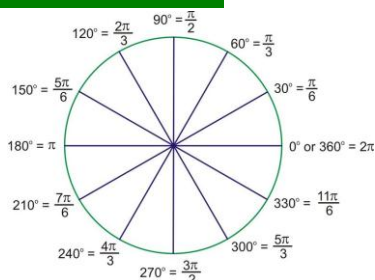
```
printf("%f\n",tan(r));
```

```
}
```

อาจใช้ `M_PI`

360 องศา =  $2\pi$

$x$  องศา =  $(\pi/180)*x$



ผลการรัน

0.017452

0.999848

0.017455

## สร้างฟังก์ชันเอง

ประเภทของฟังก์ชันที่สร้างขึ้นเอง ถ้าหากแยกตามประเภทการส่งค่าไปกลับจะแยกได้ดังนี้

- ฟังก์ชันที่ไม่มีการส่งค่าไปและไม่รับค่ากลับ
- ฟังก์ชันที่มีการส่งค่าไปแต่ไม่มีการรับค่ากลับ
- ฟังก์ชันที่มีการส่งค่าไปและรับค่ากลับ

## ฟังก์ชันที่ไม่มีทั้งการส่งค่าไปและไม่รับค่ากลับ

ฟังก์ชันประเภทนี้จะให้ชนิดของข้อมูลเป็น void และภายในวงเล็บใช้ void เพื่อบอกว่าไม่มีการส่งและรับค่า และจะไม่ใช้คำสั่ง return

รูปแบบของฟังก์ชัน

```
void func_name(void)
{
    local variable declaration;
    statement();
}
```

```
void one() {
    int a = 5, b = 7;
    printf("A = %d, B = %d\n", a, b);
}
```

ตัวอย่างโปรแกรม สร้างฟังก์ชันมาสองฟังก์ชัน ไม่มีการส่งและไม่รับค่า

```
#include <stdio.h>

void one(void);
void two(void);

void main(void)
{
    clrscr( );
    one( );
    two( );
}

void one( ) {
    int a = 5, b = 7;
    printf("A = %d, B = %d\n",a,b);
}

void two( ) {
    float p = 4.5, q = 3.5;
    printf("P = %6.2f, Q = %6.2f\n",p,q);
}
```

มีการประกาศ  
Prototype

จะสังเกตเห็นว่าไม่มีพารามิเตอร์ ไม่มี return

## ฟังก์ชันที่มีการส่งค่าไปแต่ไม่มีการรับค่ากลับ

ฟังก์ชันประเภทนี้จะให้ชนิดของข้อมูลเป็น void ภายในวงเล็บจะมีอาร์กิวเมนต์ และ จะไม่ใช่คำสั่ง return

รูปแบบของฟังก์ชัน

```
void ADD(int a, int b)
{
    int x;
    x = a+b;
    printf("sum = %d",x);
}
```

```
void funct_name(type1 arg1, type2 arg2,.....,typeN argN)
{
    local variable declaration;
    statement();
}
```

## ฟังก์ชันที่มีการส่งค่าไปและรับค่ากลับ

ฟังก์ชันประเภทนี้จะให้ชนิดของข้อมูลเป็นประเภทที่จะส่งกลับ ภายในวงเล็บจะมีอาร์กิวเมนต์ และจะใช้คำสั่ง `return` ส่งค่ากลับมาให้ฟังก์ชัน

```
int ADD(int a, int b){
    int x;
    x = a+b;
    return x;
}
```

รูปแบบของฟังก์ชัน

ข้อมูล

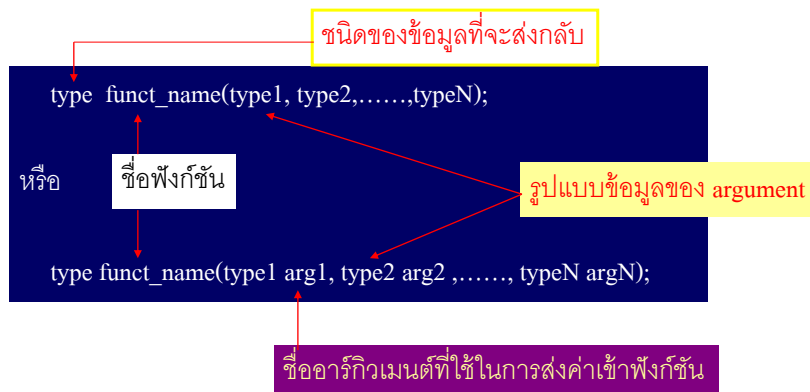
ประเภทเดียวกัน

```
type func_name(type1 arg1, type2 arg2, ..., typeN argN)
{
    local variable declaration;
    statement();
    return (value);
}
```

## การประกาศรูปแบบฟังก์ชัน (prototype)

คล้ายกับการประกาศตัวแปร เพื่อบอกว่าในโปรแกรมของเรามีฟังก์ชันอะไรที่สร้างขึ้นเอง ฟังก์ชันนั้น ๆ มีการเรียกใช้อย่างไร ใช้ตัวแปรอย่างไร ตำแหน่งที่ประกาศ ควรจะอยู่ก่อน main()

รูปแบบการประกาศฟังก์ชัน





## ตัวอย่างการประกาศรูปแบบฟังก์ชัน

**ตัวอย่าง** รูปแบบฟังก์ชัน (prototype)

1. float add(int, float);
2. int sum(int p, int q); หรือ int sum(int, int);
3. void move(int, float, int);
4. float calculate(float, float);
5. void point(int\*, int\*, float);

ฟังก์ชันนี้ใช้อาร์กิวเมนต์ที่เป็นตัวแปรแบบพอยน์เตอร์

ฟังก์ชันนี้ไม่มีการคืนค่า จะรับอาร์กิวเมนต์เป็น int, float และ int เช่น move(5, 2.70, 12);

## การประกาศตัวแปรของฟังก์ชัน

### ตัวแปรโกลบอล ( Global Variable )

เป็นตัวแปรที่ทุกส่วนของโปรแกรมสามารถเรียกใช้ได้ บางครั้งจะเรียกว่าตัวแปรเอ็กเทอร์นัล (External variables)

### ตัวแปรโลคอล ( Local Variable )

เป็นตัวแปรที่สร้างขึ้นภายในฟังก์ชัน ใช้ในฟังก์ชันนั้น ๆ เมื่อออกจากฟังก์ชันค่าตัวแปรจะหายไป การใช้ตัวแปรประเภทนี้จะทำให้ฟังก์ชันต่าง ๆ สามารถใช้ชื่อตัวแปรชื่อเดียวกันได้ ตัวแปรประเภทนี้บางครั้งเรียกว่า ออโตเมติก (automatic variable)

## ตัวแปรโกลบอล ( Global Variable )

- สามารถประกาศตัวแปรไว้ที่ภายนอกฟังก์ชันบริเวณส่วนเริ่มของโปรแกรมจะเรียกว่า ตัวแปรโกลบอล ( Global Variable ) ซึ่งเป็นตัวแปรที่สามารถเรียกใช้ที่ตำแหน่งใด ๆ ในโปรแกรมก็ได้
- ตัวแปรโกลบอล ( Global Variable ) จะใช้พื้นที่หน่วยความจำจนกว่าจะจบโปรแกรม ทำให้สิ้นเปลืองหน่วยความจำเป็นอย่างมาก

### ตัวอย่างการใช้ตัวแปรแบบทั่วไประวมกับฟังก์ชัน (global variables)

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

```
int a;
```

```
void Ex()
```

```
{
```

```
    a = 5;
```

```
    printf(“%d\n”,a);
```

```
}
```

```
main()
```

```
{
```

```
    a = 3;
```

```
    printf(“%d\n”,a);
```

```
    Ex();
```

```
    printf(“%d\n”,a);
```

```
    return 0;
```

```
}
```

## ตัวแปรโลคอล ( Local Variable)

- ตัวแปรที่ประกาศภายในแต่ละฟังก์ชันจะทำงานอยู่ภายในฟังก์ชันที่มีการประกาศค่าเท่านั้น และใช้พื้นที่ในการเก็บข้อมูลคนละส่วนกันกับตัวแปรที่ประกาศภายนอกฟังก์ชันนั้น ๆ แม้ว่าจะใช้ชื่อตัวแปรเหมือนกันก็ตาม
- ขอบเขตการทำงานของตัวแปรแต่ละตัวจะกำหนดอยู่ภายในบล็อกของคำสั่งภายในเครื่องหมายปีกกา { } หรือการประกาศในช่วงของการประกาศฟังก์ชันนั้น ๆ ฟังก์ชันอื่นจะไปเรียกใช้ไม่ได้
- ในกรณีที่มีการประกาศตัวแปรโลคอล ( Local Variable) ที่มีชื่อเดียวกันกับตัวแปรโกลบอล ( Global Variable ) ภายในบล็อกหรือฟังก์ชัน จะใช้ตัวแปรโกลบอล ( Global Variable ) ไม่ได้

### ตัวอย่างการใช้ตัวแปรเฉพาะที่ (local variables)

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

```
int a;
```

```
void Ex()
```

```
{
```

```
    int a;
```

```
    a = 5;
```

```
    printf(“%d\n”,a);
```

```
}
```

```
main()
```

```
{
```

```
    a = 3;
```

```
    printf(“%d\n”,a);
```

```
    Ex();
```

```
    printf(“%d\n”,a);
```

```
    return 0;
```

```
}
```

## ตัวอย่างฟังก์ชันที่คืนค่าเป็นเลขจำนวนเต็ม

```
#include <stdio.h>

int x,y;

int add_num(int a, int b)
{
    int m;
    m = a + b;
    return m;
}

void main(){
    printf("INPUT  X ");
    scanf("%d",&x);
    printf("INPUT  Y ");
    scanf("%d",&y);
    printf("%d + %d = %d\n",
    x,y,add_num(x,y) );
}
```

ส่งตัวแปร x,y ไปให้ a กับ b

ยศธชนก เขานแก้ว CIS KMUTNB

## ตัวอย่างการฟังก์ชันที่ส่งผ่านตัวแปรหลายประเภท

```
#include <stdio.h>

void test_f(int a, int b, double f, char c, char s[20]) {
    printf("Integer %d, %d\n",a, b);
    printf("Float %f\n", f);
    printf("Char %c\n", c);
    printf("String %s\n",s);
}

void main() {
    test_f(23, 34, 3.564, 'p', "Yellow River");
}
```

## ผลการทำโปรแกรม

Integer 23, 34

Float 3.564000

Char p

String Yellow River

ยศธชนก เขานแก้ว CIS KMUTNB

## การรับค่าที่คืนมาจากฟังก์ชัน

การเรียกใช้ฟังก์ชันที่มีการคืนค่า จะใช้รูปแบบดังต่อไปนี้

ค่าที่รับ = ฟังก์ชัน (อาร์กิวเมนต์)

## ตัวอย่าง โปรแกรมการบวกเลขจำนวนจริง 2 จำนวน

```
#include <stdio.h>

float InputFloat();
float SumFloat(float,float);
void PrintOut(float);
void main(){
    float  a1, a2, sumVal;
    a1 = InputFloat();
    a2 = InputFloat();
    sumVal = SumFloat(a1,a2);
    PrintOut(sumVal);
}

float InputFloat(){
    float x;
    printf ("\nInput real value : ");
    scanf("%f",&x);
    return x;
}

float SumFloat(float x,float y){
    return (x+y);
}

void PrintOut(float x){
    printf ("\nx is : %.2f",x);
}
```

### ตัวอย่างโปรแกรมการบวกเลขจำนวนจริง 2 จำนวน

a1 ต้องมีชนิดเป็น float เนื่องจากค่าที่จะส่งคืนกลับมาจากฟังก์ชันมีชนิดเป็น float

```
a1 = InputFloat ( );  
ใช้คู่กับโปรโตไทป์  
float InputFloat ( );
```

### ตัวอย่างโปรแกรมการบวกเลขจำนวนจริง 2 จำนวน

a1 และ a2 ต้องมีชนิดเป็น float เพื่อให้ตรงกับชนิดตัวแปรของอาร์กิวเมนต์ที่ประกาศในโปรโตไทป์

```
sumVal = SumFloat (a1,a2 );  
ใช้คู่กับโปรโตไทป์  
float SumFloat(float,float);
```

## ตัวอย่างโปรแกรมการบวกเลขจำนวนจริง 2 จำนวน

PrintOut( sumVal );  
ใช้คู่กับโปรโตไทป์  
void PrintOut(float);

ประกาศให้รู้ว่าฟังก์ชันนี้ไม่มีการคืนค่า

## ตัวอย่างการหาค่าของ $f(x) = x^2 + 3x + 1$

```
#include <stdio.h>

int f_x(int x);

main(){
    int i;
    printf(" x      x2 + 3x + 1  \n");
    for(i = 1; i <= 10; i++){
        printf("%d      %d \n",i,f_x(i) );
    }
    getch();
}
```

```
int f_x(int x)
{
    int y;
    y = x*x + 3*x + 1;
    return y;
}
```

มีการเรียกฟังก์ชันซ้ำหลาย ๆ ครั้ง ใน Loop

## การใช้ฟังก์ชันกับแถวลำดับ

ลักษณะการคืนค่ากลับของฟังก์ชันที่สร้างขึ้นเองนั้น เราสามารถสร้างฟังก์ชันให้คืนค่ากลับได้ดังนี้

1. ฟังก์ชันที่ไม่คืนค่ากลับ
2. ฟังก์ชันที่คืนค่ากลับ 1 ค่า
3. ฟังก์ชันที่คืนค่ากลับมากกว่า 1 ค่า

## การใช้ฟังก์ชันกับแถวลำดับ

- การส่งแถวลำดับไปฟังก์ชันสามารถทำได้โดยถ้าการประกาศรูปแบบฟังก์ชัน (prototype) มีแถวลำดับอยู่แล้ว เช่น

```
void function2( double b[ ], int num_elem );
```

เมื่อเรียกใช้งานก็ใช้คำสั่งว่า `function2( c, 5 );`

โดยที่ `c` คือ แถวลำดับขนาด 5 ตัว คือ `double c[5];` และนิยมนำขนาดของแถวลำดับไปด้วย คือ ส่ง 5 ให้ตัวแปร `num_elem`



## ตัวอย่างการส่งแฉวลำดับไปฟังก์ชัน

```
void function1(int *d, int e);
```

```
void function2( double b[ ], int num_elem);
```

```
function1(&a[5], a[8]) ;
```

```
function2(c, 5) ;
```

```
void function1(int *d, int e) {  
    ...  
}
```

```
void function2(double b[ ], int  
num_elem) {  
    ...  
}
```

ยณต์ชนก เขานแก้ว CIS KMUTNB

## ฟังก์ชันที่คืนค่ากลับมากกว่า 1 ค่า

- หากต้องการค่าที่ได้จากการประมวลผลของตัวแปรแฉวลำดับแต่ละตัว เราต้องให้ฟังก์ชันที่คืนค่ากลับมาเป็นที่อยู่ของตัวแปรแฉวลำดับของฟังก์ชันนั้น เช่น

```
int * getArray(int x[ ],int n) {  
    ...  
    return x;  
}
```

ค่าที่ได้จากคำสั่ง return x; จะเป็นที่อยู่ของ x ต้องใช้ตัวแปรสำหรับเก็บที่อยู่มารับค่า เช่น

```
int *A;  
A = getArray(a,10);
```

## ตัวอย่างการคืนค่ากลับมากกว่า 1 ค่า

```
int * getArray(int[],int);
```



```
A = getArray(a,10);
```



```
int * getArray(int x[],int n)
{
    ...
    ...
    return x;
}
```

```
printf( "%d,%d",*A ,*(A + 1)); /* พิมพ์ค่า x[0] , x[1] */
```



## ตัวอย่าง โปรแกรมรับค่าคืนค่ากลับเป็น Array

```
int * getArray(int[],int);
```

```
int main () {
```

```
    int *A;
```

```
    int a[10]= {0,1,2,3,4,5,6,7,8,9};
```

```
    int i;
```

```
    A = getArray(a,10);
```

```
    for ( i = 0; i < 10; i++ ) {
```

```
        printf( "a[%d]: %d\n", i, *(A + i));
```

```
    }
```

```
    return 0;
```

```
}
```

```
int * getArray(int x[],int n) {
```

```
    int i;
```

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

```
        x[i]*=10;
```

```
    }
```

```
    return x;
```

```
}
```

## สรุป

- การทำงานของโปรแกรมภาษาซีจะทำงานที่ฟังก์ชัน main ( ) ก่อนเสมอ เมื่อฟังก์ชัน main ( ) เรียกใช้งานฟังก์ชันอื่น ก็จะมีการส่งคอนโทรล (Control) ที่ควบคุมการทำงานไปยังฟังก์ชันนั้น ๆ จนกว่าจะจบฟังก์ชัน หรือพบคำสั่ง return
- ลักษณะการคืนค่ากลับของฟังก์ชัน เราสามารถสร้างฟังก์ชันให้คืนค่ากลับได้หลายแบบ คือ ฟังก์ชันที่ไม่คืนค่ากลับ (ไม่มี คำสั่ง return) หรือ ฟังก์ชันที่คืนค่ากลับ 1 ค่า (return 0; หรือ return อื่น ๆ ) หรือ ฟังก์ชันที่คืนค่ากลับมากกว่า 1 ค่า เช่น การส่งคืนสมาชิกของแถวลำดับไปยัง main()
- เมื่อมีการเรียกใช้งานฟังก์ชันจะมีการจองพื้นที่หน่วยความจำสำหรับตัวแปรที่ต้องใช้ภายในฟังก์ชันนั้น และเมื่อสิ้นสุดการทำงานของฟังก์ชันก็จะมีการคืนพื้นที่หน่วยความจำส่วนนั้นกลับสู่ระบบ การใช้งานตัวแปรแต่ละตัวจะมีขอบเขตของการใช้งานขึ้นอยู่กับตำแหน่งที่ประกาศตัวแปรนั้น

## การส่งค่าให้กับ main()

- ในภาษาซีกำหนดให้ main() เป็นจุดเริ่มต้นการทำงานเท่านั้น
- ในภาษาซีเราสามารถนำข้อมูลส่งผ่านให้ main() ได้
- ในภาษาซี ไม่มีข้อมูลที่ส่งผ่านจาก main() ไปยัง Operating System
- ในภาษาซี ไม่มีข้อมูลที่ส่งผ่านจาก Operating System ไปยัง main()

รูปแบบ main()

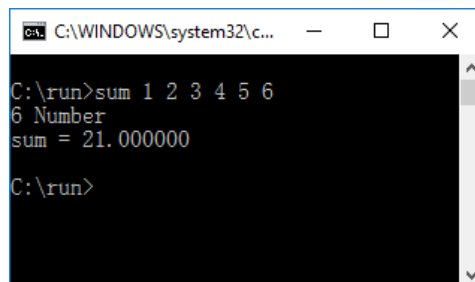
void main(void)

int main(void)

int main(argv[], argc \*)

## การส่งค่าให้กับ main()

```
int main(int x,char* y[]){
    int i;
    float sum = 0;
    printf("%d Number\n",x-1);
    if(x>1){
        for(i=1;i<x;i++){
            sum +=atof(y[i]);
        }
        printf("sum = %f\n",sum);
    }
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\run>sum 1 2 3 4 5 6
6 Number
sum = 21.000000
C:\run>
```

## แหล่งข้อมูล

- Function, อาจารย์ ยนต์ชนก เขาแก้ว ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ
- ฟังก์ชันและการประยุกต์, ศูนย์ สอวน. สาขาวิชาคอมพิวเตอร์ โรงเรียนสามเสนวิทยาลัย กรุงเทพมหานคร
- ภาษาซีฉบับภาษาชาวบ้าน , กวินวิชญ์ พุ่มสาขา ศูนย์เทคโนโลยีเพื่อการเรียนการสอน โรงเรียนสตรีอ่างทอง