บทที่ 6-2

ฟังก์ชันในภาษาซี

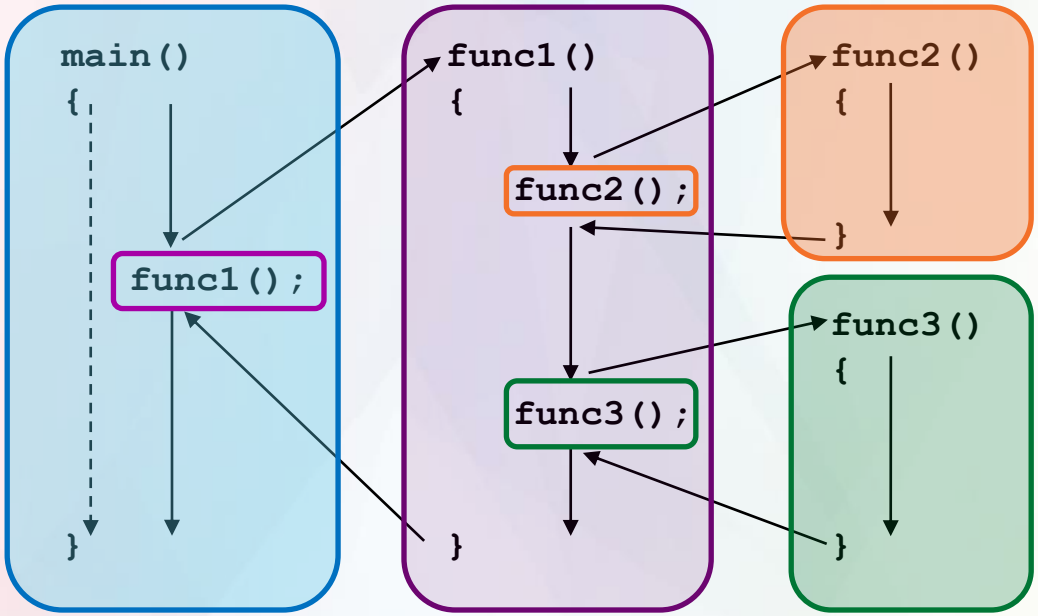
Function in C Language

Outline

- โครงสร้างการเรียกใช้ Function
- ฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้น (User-Defined Function)
- โครงสร้างของฟังก์ชัน (Structure of Function)
- รูปแบบการรับ-ส่งค่าฟังก์ชัน
- Pass by Value
- Pass by Reference
- Function Prototype
- Local and Global Variables
- การส่งอาร์เรย์เป็นพารามิเตอร์เข้าสู่ฟังก์ชัน

โครงสร้างการเรียกใช้ฟังก์ชัน

ในการเขียนโปรแกรมด้วยภาษาซีต้องมีฟังก์ชัน main( ) เสมอ และหากต้องการให้โปรแกรมทำงานฟังก์ชันใด ฟังก์ชัน main( ) จะเป็นผู้เรียกฟังก์ชันนั้นขึ้นมาทำงาน ซึ่งในฟังก์ชันนั้นเองก็ยังสามารถเรียกใช้ฟังก์ชันอื่น ๆ ขึ้นมาทำงานได้อีกด้วย ดังตัวอย่าง



ฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้น (User-Defined Function)

ฟังก์ชันมาตรฐานที่ภาษาซีเตรียมไว้ให้แล้ว ในบางครั้งอาจไม่ตรงกับความต้องการของผู้เขียนโปรแกรม ภาษาซีจึงเปิดช่องทางให้ผู้เขียนโปรแกรมสามารถสร้างฟังก์ชันขึ้นเองได้ตามต้องการ เราเรียกฟังก์ชันลักษณะนี้ว่า **User-Defined Function**

โครงสร้างของฟังก์ชัน

โครงสร้างของฟังก์ชัน มีรูปแบบดังนี้

```
return_type function-name(type para-1, type para-2, ... , type para-n)
{
   local variable-declarations ;
   statement-1 ;
   statement-2 ;
   ...
   statement-n ;
   return(value) ;
}
```

- return_type เป็นชนิดของผลลัพธ์ที่ได้จากการทำงานของฟังก์ชัน ในกรณีที่ไม่ต้องการส่งค่ากลับให้ใส่ void
- function-name คือ ชื่อของฟังก์ชันที่สร้างขึ้น ควรตั้งให้สื่อความหมายและมีหลักการตั้งชื่อเช่นเดียวกับตัวแปร

โครงสร้างของฟังก์ชัน

โครงสร้างของฟังก์ชัน มีรูปแบบดังนี้

```
return_type function-name(type para-1, type para-2, ... , type para-n)
{
   local variable-declarations ;
   statement-1 ;
   statement-2 ;
   ...
   statement-n ;
   return(value) ;
}
```

- type para-1, type para-2, ... , type para-n คือ formal parameter(s) เป็นตัวแปรที่ใช้สำหรับรับข้อมูลที่ส่งเข้ามาเพื่อนำมาประมวลผลเมื่อมีการเรียกใช้ฟังก์ชัน ซึ่งต้องมีการกำหนดชนิดของตัวแปรด้วยคำว่า “type” ไว้ข้างหน้า หากมีตัวแปรมากกว่าหนึ่งให้คั่นแต่ละตัวด้วยเครื่องหมาย "," หากไม่มีตัวแปรส่งเข้ามาให้ใส่ (void)

โครงสร้างของฟังก์ชัน

โครงสร้างของฟังก์ชัน มีรูปแบบดังนี้

```
return_type function-name(type para-1, type para-2, ... , type para-n)
{
    local variable-declarations ;
    statement-1 ;
    statement-2 ;
    ...
    statement-n ;
    return(value) ;
}
```

- ชุดคำสั่งทั้งหมด จะต้องรวมอยู่ในวงเล็บปีกกา { }
- local variable-declarations เป็นการสร้างตัวแปรเพื่อใช้งานภายในฟังก์ชัน
- statement-1, statement-2, ... , statement-n ชุดคำสั่งที่รวมอยู่ภายในฟังก์ชัน

โครงสร้างของฟังก์ชัน

โครงสร้างของฟังก์ชัน มีรูปแบบดังนี้

```
return_type function-name(type para-1, type para-2, ... , type para-n)
{
   local variable-declarations ;
   statement-1 ;
   statement-2 ;
   ...
   statement-n ;
   return(value) ;
}
```

- return(value) เป็นคำสั่งที่ใช้สำหรับคืนค่าออกจากฟังก์ชัน โดยชนิดของค่าที่คืนออกไปต้องเป็นชนิดเดียวกันกับ return_type ที่ประกาศไว้ก่อนหน้าชื่อฟังก์ชัน หากฟังก์ชันไม่มีการคืนค่า ไม่ต้องใช้คำสั่งนี้ และ return_type ต้องถูกกำหนดให้เป็น void

ตัวอย่างโปรแกรมที่ยังไม่ได้สร้างฟังก์ชัน

โปรแกรม

```
1  #include <stdio.h>
2  int main()
3  {
4    printf("********************\n");
5    printf("StarLine Printing\n");
6    printf("********************\n");
7    return(0);
8  }
```

ผลการทำงาน

```
********************
StarLine Printing
********************
```

ตัวอย่างการสร้างและเรียกใช้ฟังก์ชัน

โปรแกรม

```
 1  #include <stdio.h>
 2
 3
 4
 5
 6
 7  int main()
 8  {
 9    printf("********************\n");
10    printf("StarLine Printing\n");
11    printf("********************\n");
12    return(0);
13  }
```

แทรกช่องว่าง

ผลการทำงาน

```
********************
StarLine Printing
********************
```

ตัวอย่างการสร้างและเรียกใช้ฟังก์ชัน (ต่อ)

โปรแกรม

```
1  #include <stdio.h>
2
3       StarLine(     )
4
5
6
7  int main()
8  {
9    StarLine();
10   printf("StarLine Printing\n");
11   StarLine();
12   return(0);
13 }
```

ตั้งชื่อฟังก์ชันขึ้นมาเอง

บรรจุคำสั่งเข้าไปใน function

ใส่ชื่อฟังก์ชันเพื่อเรียกใช้งาน

ใส่ชื่อฟังก์ชันเพื่อเรียกใช้งาน

ตัวอย่าง การเรียกใช้ฟังก์ชันของโปรแกรม

โปรแกรม

```
1  #include <stdio.h>
2
3  void StarLine(void) // ไม่มีการรับ-ส่งค่า จึงใส่ void ทั้งสองตำแหน่ง
4  {
5    printf("**********\n");
6  }
7  int main()
8  {
9    StarLine();                  // เรียกฟังก์ชัน StarLine (โปรแกรมจะกระโดดเข้าไปทำงานใน StarLine)
10   printf("StarLine Printing\n");
11   StarLine();
12   return(0);
13 }
```

กำหนดขอบเขตฟังก์ชันด้วย { }

ผลการทำงาน

```
**********
StarLine Printing
**********
```

รูปแบบการรับ-ส่งค่าฟังก์ชัน

ฟังก์ชันสามารถรับ-ส่งค่าได้ 4 รูปแบบดังนี้

- **ไม่มีการรับ-ส่งค่าใด ๆ**

  ตัวอย่างเช่น `void function_name (void)`

  หมายเหตุ: ฟังก์ชัน StarLine ที่ผ่านมา ไม่มีการรับ-ส่งค่า (void) StarLine (void)

- **รับค่าเข้าอย่างเดียว**

  ตัวอย่างเช่น `void function_name (int n)`

- **ส่งค่าออกอย่างเดียว**

  ตัวอย่างเช่น `float function_name (void)`

- **รับและส่งค่า**

  ตัวอย่างเช่น `float function_name (int n)`

ตัวอย่าง ฟังก์ชันที่มีการรับค่าเข้าอย่างเดียว

โปรแกรม

```
1  #include <stdio.h>
2
3  void StarLine(int n)
4  {
5    int i;
6    for(i=1; i<=n; i++)
7      printf("*");
8    printf("\n");
9  }
10 int main()
11 {
12   StarLine(8);
13   printf("StarLine Printing\n");
14   StarLine(17);
15   return(0);
16 }
```

n เป็น Formal Parameter เป็น local variable ที่มีวัตถุประสงค์เพื่อใช้รับค่าที่ส่งเข้ามาจาก Actual Parameter

ค่า 8 และ 17 เป็น Actual Parameter เป็นค่าที่ต้องการส่งเข้าไปทำงานใน function

ผลการทำงาน

```
********
StarLine Printing
*****************
```

ตัวอย่าง ฟังก์ชันที่มีการส่งค่าออกอย่างเดียว

โปรแกรม

```
 1  #include <stdio.h>
 2  int get100(void) {
 3    int num, flag = 0;
 4    do {
 5      printf("Please enter score (0-100): ");
 6      scanf("%d",&num);
 7      if(num >= 0 && num <= 100)
 8        flag = 1;
 9    } while(flag == 0);
10    return num;
11  }
12  int main(void) {
13    int score;
14    score = get100();
15    printf("The score entered is %d",score);
16    return 0;
17  }
```

เข้าไปทำงานใน get100
โดยไม่ส่งค่าเข้าไป

get100 คืนค่า num กลับมา ซึ่งเป็นชนิด integer ตามที่ระบุไว้ข้างหน้า fucntion

นำค่าที่ได้รับกลับมา ไปกำหนดให้กับ score

ผลการทำงาน

```
Please enter score (0-100):-5
Please enter score (0-100):99
The score entered is 99
```

ตัวอย่าง ฟังก์ชันที่มีการรับและส่งค่ากลับ

โปรแกรม

```
 1  #include <stdio.h>
 2  float FindArea(float radius)
 3  {
 4    return(3.14*radius*radius);
 5  }
 6  int main(void)
 7  {
 8    float radius, area;
 9    printf("Enter Radius: ");
10    scanf("%f", &radius);
11    area = FindArea(radius);
12    printf("Area is %0.2f\n", area);
13    return(0);
14  }
```

เข้าไปทำงานใน FindAreaโดยส่งค่า Actual Parameter ไปเก็บไว้ใน Formal Parameter (ชื่อ 2 Parameters สามารถตั้งเหมือนกันได้ และไม่ผิดเพราะต่าง function กัน)

FindArea คืนค่า ที่คำนวณได้กลับมา

นำค่าที่ได้รับกลับมา ไปกำหนดให้กับ area

ผลการทำงาน

```
Enter Radius: 5
Area is 78.50
```

การส่งข้อมูลเข้าสู่ฟังก์ชัน

ฟังก์ชันที่มีการรับค่า สามารถแบ่งย่อยได้ตามลักษณะของพารามิเตอร์ที่ส่งให้ฟังก์ชัน ดังนี้

- **Pass by Value** เป็นการสำเนาค่าของ Actual parameter ไปใส่ไว้ใน Formal Parameter ดังนั้น หากค่าของ Formal Parameter เปลี่ยนไป จะไม่มีผลกับค่าใน Actual Parameter
- **Pass by Reference** เป็นการส่งตำแหน่งหน่วยความจำของ Actual parameter ให้กับ Formal Parameter ทำให้ Formal Parameter เสมือนเป็นตัวแปรเดียวกันกับ Actual Parameter

ตัวอย่าง Pass by Value

โปรแกรม

```
 1  #include <stdio.h>
 2  float FindArea(float radius)
 3  {
 4    return(3.14*radius*radius);
 5  }
 6  int main(void)
 7  {
 8    float radius, area;
 9    printf("Enter Radius: ");
10    scanf("%f", &radius);
11    area = FindArea(radius);
12    printf("Area is %0.2f\n", area);
13    return(0);
14  }
```

เข้าไปทำงานใน FindAreaโดยส่งค่า Actual Parameter ไปเก็บไว้ใน Formal Parameter (ชื่อ 2 Parameters สามารถตั้งเหมือนกันได้ และไม่ผิดเพราะต่าง function กัน)

FindArea คืนค่า ที่คำนวณได้กลับมา

นำค่าที่ได้รับกลับมา ไปกำหนดให้กับ area

ผลการทำงาน

```
Enter Radius: 5
Area is 78.50
```

Pass by Reference

Pass by Reference เป็นการส่งตำแหน่งหน่วยความจำของ Actual Parameter ให้กับ Formal Parameter ทำให้ Formal Parameter เสมือนเป็นตัวแปรเดียวกันกับ Actual Parameter

องค์ความรู้พื้นฐานที่จำเป็นต่อการทำความเข้าใจ Pass by Reference

- ตำแหน่งหน่วยความจำ (Memory Address)
- ตัวแปร Pointer

Pass by Reference (ต่อ)

ตำแหน่งหน่วยความจำ (Memory Address)

เราสามารถสร้างตัวแปรได้หลายตัวในภาษาซี แต่ต้องอยู่ภายใต้ข้อจำกัดบางประการ เช่น ระบบปฏิบัติการที่ใช้, ขนาดของ RAM โดยตัวแปรแต่ละตัวจะถูกสร้างขึ้นใน RAM และมีหมายเลขตำแหน่งใน RAM เรียกว่า “Address” เมื่อเรากำหนดค่าให้กับตัวแปร ค่านั้นจะถูกเก็บไว้ใน RAM ที่ Address ของตัวแปรนั้น ดังตัวอย่าง

```
int a;
```

เปรียบเสมือนการสร้างกล่องชื่อว่า a เพื่อเก็บข้อมูลชนิด integer และกล่องนี้อยู่ ณ Address #2020

```
a = 19;
```

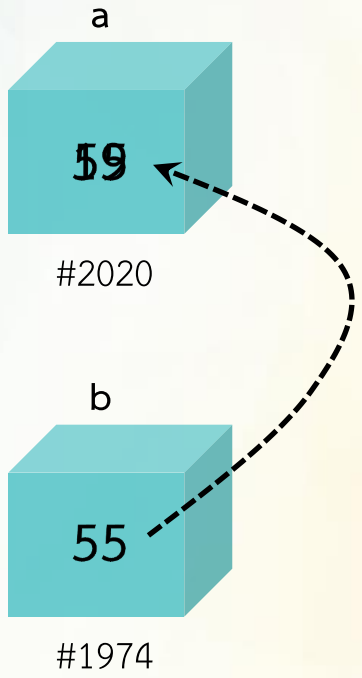
นำค่า 19 ไปเก็บไว้ในกล่องชื่อว่า a
ซึ่งก็คือ การนำค่า 19 ไปเก็บไว้ใน Address #2020 ของ RAM

```
int b;

b = 55;

a = b;
```

สำเนาค่าในกล่อง b ไปเก็บไว้ในกล่อง a ซึ่งก็คือการสำเนาค่าใน RAM Address #1974 ไปเก็บไว้ใน RAM Address #2020



Pass by Reference (ต่อ)

ตัวแปร Pointer

ตัวแปร Pointer เป็นตัวแปรที่ใช้สำหรับเก็บ Address (ตำแหน่งในหน่วยความจำ) ของตัวแปรใด ๆ ซึ่งแตกต่างกับตัวแปรชนิดพื้นฐานที่สร้างขึ้นมาเพื่อเก็บข้อมูล เช่น จำนวนเต็ม จำนวนจริง

```
int a = 20;
```

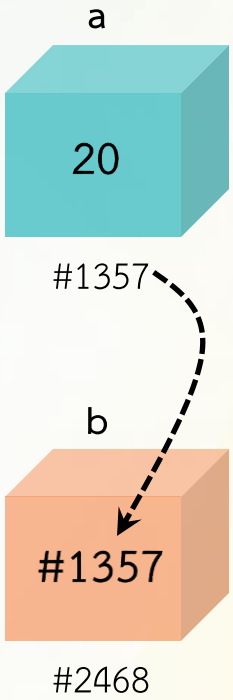
สร้างตัวแปร a เพื่อเก็บข้อมูล integer พร้อมกับกำหนดค่า 20 เข้าไป และตัวแปรนี้อยู่ใน RAM Address #1357

```
int *b;
```

เป็นการสร้าง pointer ชื่อว่า b อยู่ใน Address #2468

```
b = &a;
```

กำหนดค่า Address ของ a (#1357) ให้กับ pointer b



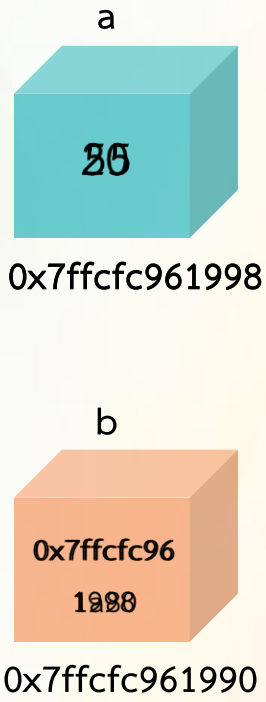
ดังนั้น เมื่อเข้าถึงข้อมูลใน b จะพบ Address ของ a (#1357) และด้วย address นี้เอง ทำให้เราสามารถเข้าถึงข้อมูลใน a (20) ได้

ตัวอย่าง Pass by Reference (ต่อ)

รู้จักการทำงานตัวแปร Pointer ให้มากขึ้น

```
1  #include <stdio.h>
2  int main(void) {
3    int a=20;
4    int *b;    //การสร้างตัวแปร pointer ชื่อ b
5    printf("a ถูกจองอยู่ใน RAM Address ที่ %p และเก็บค่า %d ไว้แล้ว\n",&a,a);
6    printf("pointer b ถูกจองใน RAM Address ที่ %p และเก็บค่า %p\n",&b,b);
7    b = &a;
8    printf("Pointer b ได้เก็บ Address ตำแหน่งที่ %p ซึ่งเป็น Address ของ a\n",b);
9    *b = 55;   // *b หมายถึง ข้อมูลของตัวแปร ณ Address ที่ b เก็บไว้
10   printf("ข้อมูลในตัวแปร a ถูกเปลี่ยนเป็น %d (ผ่านทาง pointer b)\n",a);
11   return 0;
12 }
```

%p คือ format string ที่ใช้แสดงค่า Address ของตัวแปร



ผลการทำงาน

a ถูกจองอยู่ใน RAM Address ที่ 0x7ffcfc961998 และเก็บค่า 20 ไว้แล้ว

pointer b ถูกจองใน RAM Address ที่ 0x7ffcfc961990 และเก็บค่า 0x7ffcfc961a80

Pointer b ได้เก็บ Address ตำแหน่งที่ 0x7ffcfc961998 ซึ่งเป็น Address ของ a

ข้อมูลในตัวแปร a ถูกเปลี่ยนเป็น 55 (ผ่านทาง pointer b)

สรุป: *ชื่อ pointer ใช้อ้างถึง ข้อมูลตัวแปร ณ Address ที่เก็บไว้ใน pointer

ชื่อ pointer ใช้อ้างถึง Address ที่เก็บอยู่ใน pointer

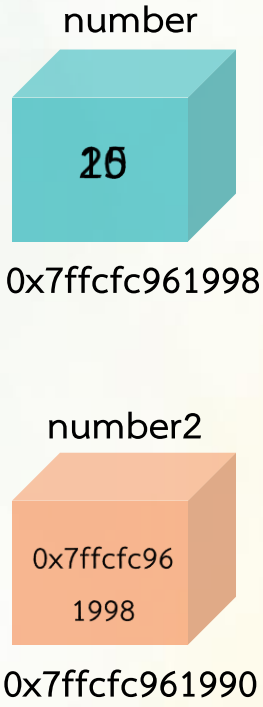
&ชื่อตัวแปร ใช้อ้างถึง Address ของตัวมันเอง

var-type *var-name เป็นการสร้าง pointer

ตัวอย่าง Pass by Reference (ต่อ)

โปรแกรม

```
1  #include <stdio.h>
2  void NumChange(int *number2)
3  {
4    printf("Before change : %d\n",*number2);
5    *number2 += 5;
6    printf("After change  : %d\n",*number2);
7  }
8  int main(void)
9  {
10   int number = 15;
11   NumChange(&number);
12   printf("After entering the function : %d\n",number);
13   return(0);
14 }
```



คำถามชวนคิด:

การทำ Pass by Reference นั้น Actual Parameter เป็นค่าคงที่ ได้หรือไม่?

คำตอบ:

ไม่ได้ เพราะค่าคงที่นั้นเป็นเพียงค่าที่กำหนดขึ้น ไม่มีการจองหน่วยความจำให้กับค่าคงที่ จึงไม่มี Address เพื่อส่งให้ Formal Parameter

ผลการทำงาน

```
Before change : 15
After change  : 20
After entering the function : 20
```

Function Prototype

ในกรณีที่สร้างฟังก์ชันไว้หลายฟังก์ชัน ฟังก์ชันใด ๆ สามารถมองเห็นและเรียกใช้งานฟังก์ชันที่อยู่บรรทัดบนกว่าได้ ทั้งนี้เนื่องจากฟังก์ชันส่วนบนนั้นได้ผ่านการแปลโดย Compiler เรียบร้อยแล้ว แต่มีข้อจำกัดคือฟังก์ชันบนกว่าไม่สามารถมองเห็นและเรียกใช้งานฟังก์ชันที่อยู่ล่างกว่าได้

```
#include <stdio.h>

void StarLine(void)
{...}  <---------- ยังไม่รู้จัก NumChange and FindArea จึงเรียกใช้สองฟังก์ชันนี้ไม่ได้
void NumChange(int number)
{...}  <---------- รู้จัก StarLine แล้วจึงเรียกใช้งานได้ แต่ยังไม่รู้จัก FindArea
float FindArea(float radius)
{...}  <---------- รู้จัก StarLine และ NumChange แล้วจึงเรียกใช้สองฟังก์ชันนี้ได้
int main()
{...}  <---------- รู้จักทั้ง 3 ฟังก์ชันแล้วจึงเรียกใช้ทั้ง 3 ฟังก์ชันได้
```

Function Prototype (ต่อ)

เพื่อแก้ไขข้อจำกัดดังกล่าว ภาษาซีจึงเสนอให้มีการสร้าง Function Prototype ขึ้น ซึ่ง Prototype นี้เปรียบเสมือนโครงสร้างโดยคร่าวของ Function ซึ่งประกอบไปด้วย var_type, Function Name และ Parameter Set

ตำแหน่งของ Prototype ควรอยู่บริเวณต้นโปรแกรม (ถัดจาก Preprocessor Directives) หรือบริเวณต้นของ main() เพื่อที่ Compiler จะได้รู้จักชื่อและโครงสร้าง (return_type and parameter(s)) ของฟังก์ชัน และสามารถแปลคำสั่งที่มีการเรียกใช้ฟังก์ชันที่สร้างไว้ในบรรทัดล่างกว่าได้ (เพราะได้รู้จักโครงสร้างโดยคร่าวที่ประกาศไว้ก่อนหน้านี้แล้ว)

Function Prototype (ต่อ)

ตัวอย่างโครงสร้างโปรแกรมที่มีการสร้าง Prototype

```
#include <stdio.h>

void StarLine(void);
void NumChange(int number);
float FindArea(float radius);

int main(void)
{...}
void StarLine(void)
{...}
void NumChange(int number)
{...}
float FindArea(float radius)
{...}
```

Function Prototypes

Function

ในฟังก์ชัน StarLine() สามารถ สามารถเรียกใช้ฟังก์ชัน NumChange และ FindArea ได้แล้ว เนื่องจากรู้จักชื่อและ input output ของฟังก์ชันเหล่านี้ตั้งแต่ตอนสร้าง Prototype ที่ต้นโปรแกรม

Function Prototype (ต่อ)

ตัวอย่าง โปรแกรมที่มีการสร้าง Prototype

```
 1  #include <stdio.h>
 2  void StarLine(void);                  // Prototype of StarLine Function
 3  int main()
 4  {
 5    StarLine();                         // Function Call
 6    printf("StarLine Printing\n");
 7    StarLine();                         // Function Call
 8    return(0);
 9  }
10  void StarLine(void)                   // StarLine Function
11  {
12    printf("********************\n");
13  }
```

ผลการทำงาน

```
********************
StarLine Printing
********************
```

Function Prototype (ต่อ)

ตัวอย่าง โปรแกรมที่มีการสร้าง Prototype

```
 1  #include <stdio.h>
 2  int ADD_AB(int A, int B);          // Prototype
 3  int main()
 4  {
 5    int A=2, B=7, C;
 6    C = ADD_AB(A,B);                 // Function Call
 7    printf("Answer is %d",C);
 8    return(0);
 9  }
10  int ADD_AB(int A, int B)           // Add_AB Function
11  {
12    int x;
13    x = A+B;
14    return(x);
15  }
```

ผลการทำงาน

```
Answer is 9
```

ชนิดของตัวแปรในฟังก์ชัน

ชนิดของตัวแปรแยกตามขอบเขตการทำงาน

- **ตัวแปรชนิดโกลบอล (Global Variable)** เป็นตัวแปรที่ประกาศไว้นอกฟังก์ชัน อยู่ส่วนหัวโปรแกรม สามารถใช้ได้ทุกที่ ทุกฟังก์ชันในโปรแกรม
- **ตัวแปรชนิดโลคอล (Local Variable)** ถูกสร้างขึ้นภายในฟังก์ชัน การเปลี่ยนแปลงจะมีผลภายในฟังก์ชันเท่านั้น หากมีชื่อซ้ำกับ Global variable จะถือว่าเป็นคนละตัวแปรกัน ตัวแปรประเภทนี้ เมื่อโปรแกรมออกจากฟังก์ชันจะถูกทำลาย

Global Variable

ตัวอย่าง โปรแกรมที่มีการสร้าง Global Variable

```
 1  #include <stdio.h>
 2
 3  int num = 19;                  // num is Global Variable
 4  void increase()
 5  {
 6    num = num + 5;
 7  }
 8  int main()
 9  {
10    printf("num = %d\n", num);
11    increase();
12    printf("num = %d\n", num);
13    return(0);
14  }
```

ผลการทำงาน

```
num = 19
num = 24
```

Local Variable

ตัวอย่าง โปรแกรมที่มีการสร้าง Local Variable

```
 1  #include <stdio.h>
 2  void new_value()
 3  {
 4    int x;                    // x is Local Variable of new_value()
 5    x = 100;
 6  }
 7  int main()
 8  {
 9    int x = 59;               // x is Local Variable of main()
10    printf("x = %d\n", x);
11    new_value();
12    printf("x = %d\n", x);
13    return(0);
14  }
```

ผลการทำงาน

```
x = 59
x = 59
```

การส่งอาร์เรย์เป็นพารามิเตอร์ให้กับฟังก์ชัน

Array มีขนาดใหญ่เล็กแค่ไหนนั้น ขึ้นอยู่กับผู้เขียนโปรแกรมเป็นผู้กำหนด ดังนั้นในหลายภาษาคอมพิวเตอร์ (รวมถึงภาษา C) กำหนดให้กลไกการส่ง Array เข้าไปทำงานในฟังก์ชันอยู่ในรูปแบบ Pass by Reference ทั้งนี้เนื่องจาก

- ประหยัดพื้นที่หน่วยความจำ (ไม่ต้องจองหน่วยความจำขึ้นอีก 1 ชุดที่มีขนาดเท่ากับ Array ที่เป็น Actual Parameter)
- ลดการประมวลผล CPU ในการสำเนาข้อมูลจาก Actual สู่ Formal Parameter
- ลดการประมวลผล CPU ในการส่งค่า Array ทั้งชุดออกจากฟังก์ชัน (กรณีที่ต้องการส่ง Array ที่ส่งเข้าไปกลับมาประมวลผลต่อ)

ตัวอย่าง ฟังก์ชันที่รับค่าเป็นอาร์เรย์

โปรแกรม

```
1  #include <stdio.h>
2  void Dollar(float baht[], int num)
3  {
4    int i;
5    printf("Before changing : %.2f\n", baht[0]);
6    for(i=0; i<num; i++)
7      baht[i] *= 40.0;
8    printf("After changing  : %.2f\n", baht[0]);
9  }
10 int main()
11 {
12   float money[3] = {200.0, 400.0, 600.0};
13   Dollar(money, 3);
14   printf("After entering function: %.2f\n", money[0]);
15   return(0);
16 }
```

เหมือนการประกาศ Array แต่ไม่ต้องระบุขนาด (เพราะเปิดกว้างให้ Array ที่ส่งเข้ามามีขนาดเท่าใดก็ได้)

อ้างแค่ชื่อ Array ในตำแหน่ง Actual Parameter

ผลการทำงาน

```
Before changing : 200.00
After changing  : 8000.00
After entering function: 8000.00
```

ตัวอย่าง ฟังก์ชันที่รับค่าเป็นอาร์เรย์

โปรแกรม

```
 1  #include <stdio.h>
 2  void printArray(int [][3]);
 3  int main(void) {
 4    int array1[2][3] = {{1,2,3},{4,5,6}},
 5        array2[2][3] = {1,2,3,4,5},
 6        array3[2][3] = {{1,2},{4}};
 7    printf("Value in array1 by row are:\n");
 8    printArray(array1);
 9    printf("Value in array2 by row are:\n");
10    printArray(array2);
11    printf("Value in array3 by row are:\n");
12    printArray(array3);
13    return 0;
14  }
15  void printArray(int a[][3]) {
16    int i,j;
17    for (i=0; i<2; i++) {
18      for (j=0; j<3; j++)
19        printf("%d ",a[i][j]);
20      printf("\n");
21    }
22  }
```

ผลการทำงาน

```
Value in array1 by row are:
1 2 3
4 5 6
Value in array2 by row are:
1 2 3
4 5 0
Value in array3 by row are:
1 2 0
4 0 0
```

ตัวอย่าง การส่งผ่านค่าสตริงระหว่างฟังก์ชัน

โปรแกรม

```
 1  #include <stdio.h>
 2  void greeting(char []);
 3  void main (void)
 4  {    char string[20];
 5       printf("What is your name => ");
 6       gets(string);
 7       greeting(string);
 8  }
 9  void greeting(char name[])
10  {
11       printf("Hello there %s!" , name );
12  }
```

มีหลักการเหมือนกับการส่ง Array เข้าสู่ฟังก์ชัน เนื่องจาก String มีโครงสร้างเป็น Array of Character

ผลการทำงาน

```
What is your name => Harry Potter
Hello there Harry Potter!
```

ตัวอย่าง การส่งผ่านค่าสตริงระหว่างฟังก์ชัน

โปรแกรม

```
1  #include <stdio.h>
2  void greeting(char [] );
3  void main (void)
4  {    char string[20];
5       greeting(string);
6       printf("You are scaring me: %s!\n" , string);
7  }
8  void greeting(char name[])
9  {    printf("Enter your first name => ");
10      gets(name);
11 }
```

ข้อมูลใน string เปลี่ยนตาม name ในฟังก์ชัน greeting

เปลี่ยนแปลงข้อมูลในฟังก์ชัน greeting

ผลการทำงาน

```
Enter your first name => Voldemort
You are scaring me: Voldemort!
```

สรุปการส่งผ่านค่าอาร์เรย์และสตริงระหว่างฟังก์ชัน

การส่งผ่านค่าอาร์เรย์และสตริงระหว่างฟังก์ชัน Formal Parameter ที่รับค่าอาร์เรย์หรือสตริงจะไม่ระบุขนาดของอาร์เรย์มิติแรก ดังตัวอย่างต่อไปนี้

```
void Dollar(float baht[], int num) {...}
void modifyArray(int b[], int size) {...}
void printArray(int a[][3]) {...}
void greeting(char name[]) {...}
```

จบบทที่ 6-2

ฟังก์ชันในภาษาซี

Function in C Language