# บทที่ 7 C Preprocessor

Preprocessor เป็นการสั่งงานให้ Compiler ทำก่อนที่จะเริ่มทำการแปล Source Code โดยขั้นตอนการทำงานของโปรแกรม C Compiler จะเป็นดังต่อไปนี้

*C Source Code* ➔ *Preprocessor* ➔ *Compiler*

ซึ่งโดยทั่วไป ก็จะเป็นการรวมไฟล์ Header (ในไฟล์ Header จะมีการประกาศตัวแปร การกำหนดชนิดของตัวแปร การประกาศฟังก์ชัน และอื่น ๆ) การกำหนดค่าคงที่และ Macro ต่าง ๆ

คำสั่งสำหรับ Preprocessor จะขึ้นต้นด้วย # (hash mark) แม้ว่า C จะไม่ต้องการรูปแบบที่แน่นอนอย่าง FORTRAN กล่าวคือ บรรทัดในภาษา C จะไม่มีความหมายมากนัก เพราะประโยคจะจบที่เครื่องหมาย ; แต่ใน Preprocessor คำสั่งจะจบตามบรรทัด (แต่ในกรณีที่เป็นประโยคที่ยาวเกิน 1 บรรทัด สามารถจะเขียนระบุให้เป็นประโยคคำสั่งเดียวกันได้ โดยใส่เครื่องหมาย \ (Back Slash) ไว้ท้ายบรรทัด) และ Preprecessor จะไม่รู้จักคำสั่งในภาษา C ทั่วไป จะต้องใช้คำสั่งสำหรับ Preprocessor โดยเฉพาะ ซึ่งจะมีไวยากรณ์ที่ต่างไปจากภาษา C ปกติ

รูปแบบโดยทั่วไปของประโยคคำสั่งของ Preprocessor จะเป็นดังต่อไปนี้

```
#+instruction  parameters
```

## 7.1 File Inclusion (#include Statement)

คำสั่ง **#include** มีไว้สำหรับสั่งให้โปรแกรมรวม Source Code ที่อยู่ในไฟล์อื่นเข้ามาอยู่ในโปรแกรม เสมือนว่าเป็นไฟล์เดียวกัน

```
Usage : #include filename
```

ตัวอย่างการใช้ที่เห็นกันทั่วไป ได้แก่

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

ในโปรแกรมส่วนใหญ่ ประโยคคำสั่งอันนี้ จะสั่งให้โปรแกรมหาไฟล์ชื่อ *stdio.h* และทำการรวมเข้ามาอยู่ในโปรแกรม ไฟล์ลักษณะนี้ เราเรียกว่า Header File สัญลักษณ์ < > จะหมายถึง Header File มาตรฐานของ C ได้แก่ *stdio.h stdlib.h math.h* เป็นต้น แต่ถ้าเราจะรวม Header File ที่เราสร้างขึ้นมาเอง ก็สามารถทำได้ เพียงแต่จะใช้สัญลักษณ์ที่ต่างกัน คือ จะใช้เครื่องหมาย double quote (") เช่น

```
#include "my_header.h"
```

## 7.2 #define statement

คำสั่ง #define มีไว้สำหรับกำหนดค่าคงที่ หรือสัญลักษณ์พิเศษบางอย่างที่ต้องการใช้ในโปรแกรม เช่น ค่า PI = 3.141592654, MAX(x,y) = (x > y) ? x : y เป็นต้น เพื่อเป็นการเพิ่มความสะดวก และความง่ายในการเขียนโปรแกรม ยกตัวอย่างค่า PI ถ้าหากว่า เราต้องเขียนค่า 3.141592654 ตลอดทุกครั้งที่ต้องการใช้ PI ในการคำนวณ ก็จะสร้างความลำบากแก่ผู้เขียน ( และเป็นการทำให้ Source Code ยาวขึ้นโดยไม่จำเป็น ) ในกรณีนี้ เราสามารถทำได้โดยกำหนดให้สัญลักษณ์ PI เท่ากับ 3.141592654 เมื่อ C Compiler พบสัญลักษณ์ PI นี้ ก็จะทำการแปลงเป็นค่า 3.141592654 โดยอัตโนมัติ เสมือนกับว่า เราได้เขียนค่า 3.141592654 ที่คำสั่งนั้น เราเรียกค่าคงที่ หรือสัญลักษณ์พิเศษเหล่านี้ว่า *Macro*

```
Usage :      #define macro_name value
```

หรือ

```
             #define macro_name substitute-text
```

ยกตัวอย่าง เช่น

```
#define SIZE 20     /* Define SIZE = 20 */
#define FOO bar     /* Replace the word "FOO" with the word "bar" */
#define EQUAL ==    /* Replace the word "EQUAL" with "==" */
#define integer int /* Replace the word "integer" with "int" */
```

```
#define FOR_ALL  for(i = 0 ; i < ARRAY_SIZE ; i++ )
```

วิธีการใช้ จะใช้ในลักษณะดังนี้

```
if (x EQUAL 3) { ... }
```

จะมีความหมายเหมือนกับ if (x == 3) { ... }

```
integer a, b, c ;
/* Clear the array */
FOR_ALL
      data[i] = 0 ;
```

จะเป็นการสั่งให้โปรแกรมกำหนดค่าของตัวประกอบทุกตัวใน Array *data* ให้เท่ากับ 0

สังเกตว่า คำสั่งในลักษณะ #define SIZE=20 จะไม่ถูกต้อง

#define มักจะนิยมใช้ในการกำหนดนิยามของ **symbolic constant** (ค่าคงที่ที่อยู่ในรูปสัญลักษณ์) เช่น

```
#define TRUE 1
#define FALSE 0
```

และวิธีการใช้จะเป็นในลักษณะดังนี้

```
if ( fig == FALSE ) ...
```

จะมีค่าเท่ากับ

```
if ( fig == 0 ) ...
```

**ตัวอย่างที่ 7.1** Usage of #define statement

```
#include <stdio.h>
#define FIRST_PART 7
#define LAST_PART   5
#define ALL_PARTS   FIRST_PART + LAST_PART
int main()
{
      printf("The square of all the parts is %d\n",
                                 ALL_PARTS * ALL_PARTS) ;
      return 0;
}
```

คำถาม: จงหาว่า โปรแกรมข้างต้นให้ผลลัพธ์เท่าใด และเพราะเหตุใด

***Defined Macro (Macro ที่ได้รับการกำหนดนิยามแล้ว)***

```
__FILE__    The name of source file that is on processing
__TIME__    Compile time
__DATE__    Compile date
__LINE__    The number of line of source file that is on processing
__STDC__    1 if ANSI C (STDC means STanDard C.)
```

มีเพียงตัวสุดท้ายเท่านั้นที่เป็น int นอกนั้นเป็น string ทั้งหมด

## 7.3 #undef statement

คำสั่ง #undef มีไว้สำหรับยกเลิกการกำหนดนิยามของ Preprocessor

Usage : #undef *macro_name*

ยกตัวอย่าง เช่น ถ้าหากจะยกเลิกนิยาม #define SIZE 20 สามารถทำได้โดยใช้คำสั่ง

```
#undef SIZE
```

คำสั่งนี้ มักจะใช้ในกรณีต้องการเปลี่ยนค่าของ Macro เช่น ถ้าหากว่า เราต้องการเปลี่ยนค่า Macro *SIZE* จาก 20 เป็น 200 เราสามารถทำได้โดย

```
#define SIZE 20      /* first definition */
...
#undef SIZE          /* Cancel first definition */
#define SIZE 200     /* second definition */
```

หมายเหตุ หากไม่ใช้ #undef ตอน Compile อาจจะเกิด Warning ขึ้น แต่จะสามารถ Compile ได้

## 7.4 Parameterized Macros

เราสามารถสั่งให้ Macro รับค่า Parameter ได้ ตัวอย่างต่อไปนี้ เป็น Macro สำหรับคำนวณเลขยกกำลังสอง

```
#define SQR(x)      ((x) * (x))   /* square a number */
```

เวลาเราใช้ Macro อันนี้ สมมุติให้ x มีค่าเท่ากับ 5 Compiler จะทำการเปลี่ยนข้อความ SQR(x) ในโปรแกรมให้กลายเป็นข้อความ ((5) * (5))

หมายเหตุ การเขียนวงเล็บล้อมรอบ Parameter ของ Macro เป็นการดี เพราะจะเป็นการป้องการข้อผิดพลาดอันอาจเกิดจาก Macro ได้

**ตัวอย่างที่ 7.2** : Usage and comparison of 2 SQR(x)'s

```
#include <stdio.h>
#define SQR1(x) (x * x)
#define SQR2(x) ((x) * (x))
/* Notice the difference between 2 above macros */
int main()
{
       int    a, b ;
       a = 3 ;       b = 4 ;
       printf("result1 = %d result2 = %d \n", SQR1(a + b), SQR2(a + b)) ;
       return 0;
}
```

**คำถาม** : จงหาผลที่ได้จากโปรแกรมตัวอย่างที่ 7.2 แล้วจงวิเคราะห์หาเหตุผล

## 7.5 Conditional Compilation

ปัญหาสำคัญปัญหาหนึ่งในการเขียนโปรแกรมก็คือ จะเขียน Source Code อย่างไรให้สามารถ Compile และใช้ได้กับเครื่องคอมพิวเตอร์ในหลาย Platform (กล่าวคือ เครื่องคอมพิวเตอร์ที่ใช้ CPU หรือระบบปฏิบัติการที่ต่างกัน) Preprocessor ช่วยผู้เขียนโปรแกรมให้สามารถเขียนโปรแกรมที่มีความยืดหยุ่นมากขึ้น กล่าวคือ สามารถกำหนดเงื่อนไขในการ Compile ได้ ซึ่งความสามารถนี้ทำให้ภาษา C เป็นภาษาที่มี Portability สูง

โดยทั่วไป เราสามารถกำหนดค่า Macro โดยใช้ -D command-line option (ยกตัวอย่าง เช่น gcc -DDEBUG ... หรือ gcc -DSIZE=20 ... ) เราสามารถใช้ option ที่ว่านี้ในการกำหนดเงื่อนไขในการ Compile ได้ หรือไม่ก็ใช้คำสั่ง #if, #elif, #else, #endif etc. ใน Preprocessor

***#if, #elif, #else, #endif Statement***

```
Usage:                                  Example :
#if definition                          #if MEMSIZE==640
       (Compile this statement)                 char  bfr[32000] ;
#elif definition                        #elif MEMSIZE>=512
       (Compile this statement)                 char  bfr[30000] ;
#else                                   #else
       (Compile this statement)                 char  bfr[25000] ;
#endif                                  #endif
```

***#if defined, #if !defined Statement***

Usage:

```
#if defined macro_name /* if macro_name is defined */
#if defined (macro_name) /* same as above */
#if !defined macro_name    /* if macro_name is not defined */
#if !defined (macro_name) /* same as above */
```

ตัวอย่าง เช่น

```
       #if defined (DEBUG)
       printf("debug : a = %d b = %d\n", a, b) ;
       #endif
```

***#ifdef, #ifndef Statement***

```
Usage :  #ifdef macro_name          /* same as if defined */
         #ifndef macro_name  /* same as if !defined */
```

## แบบฝึกหัด

1. ใช้ #define สร้าง macro สำหรับแสดงค่าคงที่ต่าง ๆ ที่ใช้ในการคำนวณทางวิศวกรรมศาสตร์ เช่น ค่า $\pi$, ค่า $e$ (Exponential ), ค่า k ( Boltzmann's constant, $1.3803 \times 10^{-23}$ J/ °K) , mile ( เท่ากับ 1609.344 m ), ค่า $\varepsilon_0$ ( ค่า Electric Constant = $8.8541878 \times 10^{-12}$ F/m ) ค่า $g_n$ ( Standard acceleration of free fall = 9.80665 $ms^{-2}$ ) ค่า $c$ (Velocity of Light = $2.99792458 \times 10^8$ $ms^{-1}$ เป็นต้น
2. สร้าง macro สำหรับให้คืนค่า TRUE เมื่อ parameter สามารถหารด้วย 10 ลงตัว
3. สร้าง macro สำหรับคำนวณหาค่าสูงสุดของเลข 2 จำนวน
4. สร้าง macro สำหรับคำนวณหาค่า ABSOLUTE
5. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
#include <stdio.h>
#define min( x, y ) ( (x) < (y) ) ? (x) : (y) )
void main()
{
      printf("%d\n", min( 3, 6 ) + 1 ) ;
}
```

6. จงบอกข้อผิดพลาดที่เกิดจากการใช้ Macro ต่อไปนี้

```
#define DBL(x) 2 * x
```