บทที่ 8
ตัวประมวลผลก่อน C
(The C Preprocessor)

ภาษา C ได้จัดเตรียมสภาพแวดล้อมในการเขียนโปรแกรมที่สำคัญอย่างหนึ่ง คือ มีการจัดเตรียมโปรแกรมชื่อ ตัวประมวลผลก่อน C (C preprocessor) โดยตัวประมวลก่อน C จะทำการประมวลประโยคต่าง ๆ ที่ขึ้นต้นด้วยสัญลักษณ์ '#'(hash character) และนำหมายเหตุ (comments) ออกจากโปรแกรมต้นฉบับก่อนที่จะส่งผลลัพธ์จากการทำงานนี้ให้แก่คอมไพเลอร์ การใช้งานตัวประมวลผลก่อน C ที่นิยมใช้มากนั้นมีอยู่ 2 ลักษณะ คือ

1. การรวมแฟ้มเข้าไปในโปรแกรมต้นฉบับก่อนที่จะแปลโปรแกรม ซึ่งในกรณีนี้จะใช้คำสั่ง #include แล้วตามด้วยชื่อแฟ้มที่ต้องการรวมเข้าไปในโปรแกรมต้นฉบับ
2. การกำหนดค่าให้กับตัวระบุ (identifiers) ตัวระบุในที่นี้คือชื่อแมคโคร (macro name) และชื่อค่าคงตัว(constant name) คำสั่งที่ใช้ในการกำหนดค่า คือ #define

8.1 การรวมแฟ้มโดยใช้คำสั่ง #include

รูปแบบคำสั่งที่ใช้สำหรับรวมข้อมูลในแฟ้มที่กำหนดเข้าไปในโปรแกรมต้นฉบับมี 2 แบบ คือ

```
แบบที่1 #include <filename>
แบบที่2 #include "filename"
```

ตัวอย่างการใช้ เช่น

```
(1)    #include    <stdio.h>
(2)    #include    "stdio.h"
(3)    #include    <ctype.h>
(4)    #include    "bool.h"
```

เมื่อตัวประมวลผลก่อน C พบคำสั่งที่ขึ้นต้นด้วย #include ตัวประมวลผลก่อน C จะทำการอ่านแฟ้มข้อมูลที่กำหนดไว้หลังคำสั่ง #include ที่อยู่ในจานบันทึกโดยถ้ากำหนดชื่อแฟ้มไว้ภายในเครื่องหมาย '< >' ดังตัวอย่าง (1) และ (3) ตัวประมวลผลก่อน C จะทำการค้นหาแฟ้ม 'stdio.h' กับแฟ้ม

'ctype.h' ในคลังมาตรฐานแล้วนำข้อความทั้งหมดในแฟ้มดังกล่าวมาแทนที่บรรทัดที่บรรจุคำสั่ง #include

แต่ถ้ากำหนดชื่อแฟ้มไว้ในเครื่องหมาย" " ดังในตัวอย่าง (2) กับ (4) ตัวประมวลผลก่อน C จะทำการค้นหาแฟ้ม 'stdio.h' กับ 'bool.h' ในสารบบ (directory) ที่ทำงานอยู่ ณ ขณะนั้น แล้วนำข้อความทั้งหมดในแฟ้มดังกล่าวมาแทนที่บรรทัดที่บรรจุคำสั่ง #include เช่นกัน

8.2 คำสั่งกำหนดค่า #define

คำสั่ง #define จะใช้เพื่อกำหนดค่าให้กับตัวระบุ ซึ่งตัวระบุเหล่านี้จะถูกแทนด้วยค่าคงที่เหล่านี้ในแฟ้มต้นฉบับทุกครั้งที่ตัวประมวลผลก่อน C ตรวจพบ จากตัวอย่างในบทก่อนจะพบว่า การทำงานของประโยคคำสั่ง #define นั้นมีลักษณะการใช้งานคล้ายกันกับคำสั่งประกาศค่าคงตัว ในภาษาปาสคาล แต่แท้จริงแล้วการใช้คำสั่ง #define จะเป็นการสร้างแมคโครให้กับตัวระบุนั้นๆ เพื่อที่จะได้นำนิยามของแมคโครที่สร้างไว้ไปแทนที่ เมื่อมีการอ้างอิงถึงชื่อตัวระบุนั้น

รูปแบบคำสั่ง #define

```
#define macro-name string
```

ในที่นี้ macro-name คือ ชื่อตัวระบุ และ string คือ ค่าคงที่หรือสายอักขระที่จะนำไปแทนที่ตัวระบุในโปรแกรมต้นฉบับนั้นเอง การตั้งชื่อแมคโครหรือตัวระบุนั้น ใช้หลักการเดียวกันกับการตั้งชื่อตัวแปรในภาษา C แต่นิยมตั้งชื่อแมคโครด้วยตัวอักษรพิมพ์ใหญ่ทั้งหมด เพื่อบอกให้ผู้ใช้รู้ว่าเป็นชื่อแมคโครไม่ใช่ตัวแปรทั่ว ๆ ไป การใช้คำสั่ง #define กำหนดค่าให้กับตัวระบุสามารถกำหนดได้หลายแบบดังนี้

(1) กำหนดค่าคงที่ที่เป็นจำนวนให้กับตัวระบุ

ตัวอย่างเช่น ถ้าเราต้องการใช้ตัวระบุ “TRUE” สำหรับค่าคงที่ ‘1’ และ “FALSE” สำหรับค่าคงที่ ‘0’ สามารถกำหนดนิยามแมคโคร ได้เป็น

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

จากคำสั่ง #define ข้างต้นจะทำให้ ตัวประมวลผลก่อน C แทนที่ TRUE และ FALSE ด้วย '1' และ '0' ทุก ๆ ครั้งที่พบ

ดังนั้นจากคำสั่ง

```
Printf ("%d %d %d", FALSE, TRUE, TRUE +1);
```

ตัวประมวลก่อน C จะแทนค่า TRUE กับ FALSE ในคำสั่งดังกล่าวได้เป็น

```
Printf("%d %d %d" 0, 1, 1+1);
```

ซึ่งมีผลทำให้คอมพิวเตอร์พิมพ์ค่า '0 1 2' บนจอภาพ

(2) กำหนดค่าคงที่สายอักขระให้กับตัวระบุ เช่น

```
#define DIGITS "0123456789";
```

ในกรณีนี้ตัวประมวลผลก่อน C จะแทน DIGITS ด้วยสายอักขระ "0123456789" ทุก ๆ ครั้งที่ตรวจพบ

(3) การกำหนดค่าในรูปของนิพจน์ให้กับตัวระบุ เช่น

```
#define TWOPT (3.1415926*2.0);
```

ในกรณีนี้ตัวประมวลผลก่อน C จะแทนที่ TWOPI ด้วยนิพจน์ (3.1415926*2.0) โดยไม่มีการคำนวณค่า

ดังนั้นตัวประมวลผลก่อน C จะแทน

```
Circumf = radius * TWOPI;
ด้วย
Circumf = radius * (3.1415926 * 2.0);
```

(4) การกำหนดชื่อแมคโครในรูปของชื่อแมคโครอื่น ๆ ที่ถูกกำหนดมาก่อนหน้า เช่น

```
#define  MIN  1                          /*first array element*/

#define  MAX  99                         /*last array element*/

#define  COUNT (MAX-MIN+1)               /*total number of items */
```

เมื่อตัวประมวลผลก่อน C ตรวจพบคำสั่ง

```
mid     =  COUNT/2 ;
```

จะแทนที่ COUNT ดังนี้

```
mid     =  (MAX – MIN + 1)/2;
```

แล้วจึงแทนที่ MAX และ MIN ตามลำดับได้

```
mid     = (99-1+1)/2;
```

8.3 การใช้พารามิเตอร์กับคำสั่ง #define

ในหัวข้อ 8.2 จะเห็นว่าชื่อแมคโครหนึ่ง ๆ จะถูกแทนที่ด้วยค่าคงตัวตัวเดียวตลอดโปรแกรม ถ้าเราต้องการแทนชื่อแมคโครด้วยค่าที่เปลี่ยนได้ จะต้องใช้ **'พารามิเตอร์'** ซึ่งมีรูปแบบการกำหนดค่าดังนี้

```
#define  macro-name (param-list)  replacement-text
```

macro-name เป็นชื่อตัวระบุ

param-list เป็นรายชื่อของพารามิเตอร์ และคั้นระหว่างพารามิเตอร์แต่ละตัวด้วยเครื่องหมายจุลภาค (,)

replacement-text เป็นนิพจน์ที่กำหนดการทำงานของ macro-name

ตัวอย่างที่ 8.1 โปรแกรมแสดง macro การหาค่าที่น้อยที่สุดระหว่างจำนวน 2 จำนวน

```
#define     MIN(a,b)      ((a<b)? a : b)
main ( )
{
        int   x, y ;
        x     = 10;
        y     = 20;
      printf (" the minimum is : %d", MIN (x, y) );
      return 0;
}
```

ผลลัพธ์ที่ได้คือ

```
The minimum is : 10
```

เมื่อโปรแกรมถูกแปล นิพจน์ MIN (x,y) จะถูกแทนที่ นั่นคือ คำสั่ง printf จะถูกแทนที่ด้วยนิยามของ MIN (x,y) แล้วได้ผลดังนี้

```
printf ("the minimum is : %d", ((x<y)? x : y) );
```

ข้อควรระวังในการใช้พารามิเตอร์ คือ ควรจะใช้วงเล็บปิดหัวท้ายของพารามิเตอร์ทุกตัว เพื่อป้องกันการทำงานผิดพลาดจากขั้นตอนที่ควรจะเป็น ดังโปรแกรมในตัวอย่างที่ 8.2 ต่อไปนี้

ตัวอย่างที่ 8.2 โปรแกรมแสดงการทำงานที่ผิดพลาดจากที่ควรจะเป็นของแมคโคร

```
/* This program will give the wrong answer */

#define  EVEN(a)  a%2 = = 0 ? 1 : 0

main ( )
{
        if      ( EVEN (9+ 1) )
printf ("is even");
        else
printf ("is odd");
}
```

เมื่อโปรแกรมถูก นิพจน์ EVEN (9+1) จะถูกแทนที่เป็น

```
9 + 1 % 2 = = 0 ? 1 : 0
```

เนื่องจากตัวดำเนินการ % (modulus) มีลำดับความสำคัญมากกว่าการบวก จะทำให้มีการคำนวณค่า '1%2' ก่อนได้ผลลัพธ์เป็น '1' แล้วจึงนำ '1' ไปบวกกับ '9' ได้ผลรวมเป็น '10' ซึ่งทำให้การตรวจสอบ 10 = = 0 เป็นเท็จ และโปรแกรมจะรายงานผลว่า a + 1 เป็นจำนวนคี่ ซึ่งเป็นผลลัพธ์ที่ไม่ถูกต้อง

ดังนั้น เพื่อให้การดำเนินการถูกต้องตามที่ต้องการให้ใส่วงเล็บปิดหัวท้ายตัวพารามิเตอร์ทุกตัว โปรแกรมในตัวอย่างที่ 8.2 สามารถปรับเปลี่ยนให้ถูกต้องได้ดังนี้

```
#define EVEN(a) (a) % 2 = = 0 ? 1 : 0
main ( )
{
    if      ( EVEN (9+1))
            printf ("is even");
    else
            printf ("is odd");
}
```

การใช้งานแมคโครที่มีพารามิเตอร์จะทำงานคล้ายกับเป็นฟังก์ชัน แต่จริง ๆ แล้วแมคโครจะเป็นเพียงการแทนที่ด้วยสายอักขระต่าง ๆ เท่านั้น และเรานิยมใช้แมคโครแทนฟังก์ชัน เฉพาะกรณีที่เป็นการทำงานสั้น ๆ ที่สามารถบรรจุคำสั่งไว้ในบรรทัดเดียว ถ้าเป็นการทำงานที่ซับซ้อน เราจะนิยมกำหนดในรูปของฟังก์ชัน ดังที่กล่าวมาแล้วในบทก่อน แต่อย่างไรก็ตามข้อดีของการใช้แมคโคร คือ เราไม่จำเป็นจะต้องกำหนดประเภทข้อมูลของพารามิเตอร์ ตัวอย่างเช่น

```
#define  SQUARE(X)  ( (X)*(X))

#define ODD(X) (x) % 2 = = 1 ? 1: 0
```

8.4 คำสั่ง #Undef

เมื่อมีการกำหนดนิยามของแมคโครด้วยคำสั่ง #define นั้น เราจะสามารถใช้งานแมคโครนี้ได้ตลอดโปรแกรม แต่ถ้าผู้ใช้ต้องการกำหนดขอบเขตการแทนค่าแมคโครในบางส่วนของโปรแกรมเท่านั้น จะสามารถทำได้โดยกำหนดจุดสิ้นสุดของการแทนค่าด้วยคำสั่ง #undef ซึ่งมีรูปแบบดังนี้

```
#undef   macro-name
```

ตัวอย่างที่ 8.3 ส่วนของโปรแกรมแสดงการใช้คำสั่ง undef

```
#define    LEN 100
#define    WIDTH  50
⋮
char       array[LEN][WIDTH];
⋮
#undef     LEN
#undef     WIDTH

/*at this point both LEN and WIDTH are undefind */
```

ตัวระบุ LEN และ WIDTH จะถูกกำหนดค่าเป็น '100' และ '50' ตามลำดับ จนกระทั่งพบคำสั่ง #undef ดังข้างต้น

ตัวอย่างที่ 8.4 โปรแกรม countem.c ต่อไปนี้ใช้คำสั่ง #undef เพื่อยกเลิกการใช้นิยามของฟังก์ชัน isalpha ซึ่งได้ถูกกำหนดไว้ในแฟ้ม ctype.h ซึ่งจัดเก็บอยู่ในคลังมาตรฐาน แล้วกำหนดฟังก์ชัน isalpha ใหม่ในรูปแบบที่ผู้ใช้ต้องการ

```
/*    countem.c -- Print counts of various types of characters (spaces,
 *  digits, letters, punctuation, and so on
 */
#include <stdio.h>
#include <ctype.h>

#define PRINT_TOTALS(x,total) printf(#x "\t%7ld %4.1f%%\n", x, (x*100.0/total))

#undef isalpha   /*want our own definition */

int isalpha(int c)
{
        return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');
}

main()
{
        int c;
        long spaces, letters, digits, puncts, others, tot;
        spaces = letters = digits = puncts = others = tot = 0;
        clrscr();
        while (( c = getchar()) != EOF)
                if (isspace(c))
                        spaces++;
                else if (isalpha(c))
                        letters++;
                else if (isdigit(c))
                        digits++;
                else if (ispunct(c))
                        puncts++;
                else
                        others++;

        tot= spaces + letters + digits + puncts + others;
        printf("Total     %71d\n\n", tot);

        if (tot != 0)
        {
                PRINT_TOTALS(spaces,tot);
                PRINT_TOTALS(letters,tot);
                PRINT_TOTALS(digits,tot);
                PRINT_TOTALS(puncts,tot);
                PRINT_TOTALS(others,tot);
        }
        return 0;
}
```

ผลลัพธ์ที่ได้

```
12
jflsjf
45
%#*
^Z
Total
7

spaces          4 23.5%
letters         6 35.3%
digits          4 23.5%
puncts          3 17.6%
others          0  0.0%
```

8.5 การแปลโปรแกรมแบบมีเงื่อนไข

เราสามารถบอกให้ตัวประมวลผลก่อน C เลือกบางคำสั่งของโปรแกรมต้นฉบับก่อนจะส่งผลลัพธ์ให้กับคอมไพเลอร์ได้ เราเรียกขั้นตอนการประมวลผลนี้ว่า “การแปลโปรแกรมแบบมีเงื่อนไข” ซึ่งคำสั่งต่าง ๆ ในกลุ่มนี้ได้แก่ #if, #endif, #else, #elif, #ifdef และ #ifndef

1. คำสั่ง #if และ #endif

รูปแบบของคำสั่ง #if และ #endif คือ

```
#if     <constant_expression>
        <statement sequence>
#endif
```

ในที่นี้ถ้าค่าของ <constant_expresion> ที่อยู่หลังคำสั่ง #if เป็นจริงแล้ว คำสั่งทุกคำสั่งหลัง #if (statement sequence) จะถูกแปล แต่ถ้าเป็นเท็จคำสั่งหลัง #if จะไม่ถูกแปล

ตัวอย่างที่ 8.5 โปรแกรมแสดงการใช้ ‘#if’ อย่างง่าย

```
#define MAX 100
main ( )
{
   #if MAX > 99
        printf (“Compiled for array greater than 99\n”);
  #endif
return 0;
}
```

ผลลัพธ์ที่ได้

```
Compiled for array greater than 99
```

โปรแกรมนี้จะแสดงข้อความ "Compiled for array greater then 99" เพราะว่า MAX ถูกกำหนดให้มีค่าเป็น 100 ซึ่งมากกว่า 99

ข้อควรระวัง

เนื่องจาก <constant-expression> ในตัวอย่างที่ 8.5 คือ (MAX > 99) จะถูกคำนวณค่าขณะที่โปรแกรมถูกแปล ดังนั้น constant-expression จะต้องประกอบด้วยตัวระบุที่ถูกกำหนดค่ามาก่อนกับค่าคงที่เท่านั้นจะเป็นตัวแปรไม่ได้

2. คำสั่ง #else

คำสั่ง '#else' จะทำงานคล้ายกับคำสั่ง 'else' ในภาษา C นั่นเอง

ตัวอย่างที่ 8.6 แสดงการใช้คำสั่ง #if ที่มีคำสั่ง #else เพื่อกำหนดทางเลือกอื่นให้ เมื่อ constant-expression หลัง #f เป็นเท็จ

```
#define         MAX    10
main ( )
{
   #if          MAX > 99
                printf ("compiled for array greater than 99\n");
   #else
                printf ("Compiled for small array\n");
   #endif
 }
```

ผลลัพธ์ที่ได้

```
Compiled for small array
```

ในกรณีนี้ค่าของ MAX ถูกกำหนดให้น้อยกว่า 99 ดังนั้น คำสั่งหลัง #if จะไม่ถูกแปล แต่จะแปลคำสั่งหลัง #else

ดังนั้นโปรแกรมนี้จะแสดงข้อความว่า

"Compiled for small array"

3. คำสั่ง #elif

คำสั่ง #elif จะมีความหมายเช่นเดียวกันกับ "else if" และจะใช้ในกรณีที่เงื่อนไขในการแปลมีหลาย ๆ เงื่อนไข

รูปแบบของคำสั่ง #elif เป็นดังนี้

```
# if     expression_0
        statement_0
#elif   expression_1
        statement_1
#elif   expression_2
        statement_2
#elif   expression_n-1
        statement n-1
#else   statement_N

#endif
```

ตัวอย่างที่ 8.7 แสดงการใช้ค่าของ ACTIVE_COUNTRY เพื่อกำหนดหน่วยของเงินตรา

```
        0
ENGLAND     1
FRANCE      2
ACTIVE_COUNTRY  US

#if     ACTIVE_COUNTRY    == US
        char currency[] ="dollar";
#elif   ACTIVE_COUNTRY == ENGLAND
        char currency[] ="pound";
#else
        char currency[] ="franc";
#endif

int i=0;
printf("\n");

while (currency[i]!='\0'){
        printf("%c",currency[i]);
        i++;
}

return 0;
```

ผลลัพธ์ที่ได้คือ

```
dollar
```

4. คำสั่ง #ifdef และ #ifndef

รูปแบบของคำสั่ง #ifdef คือ

```
#ifdef    macro-name
          statement sequence
#endif
```

ถ้าชื่อแมคโครได้ถูกกำหนดมาก่อนด้วยคำสั่ง #define แล้วคำสั่งที่อยู่ระหว่าง #ifdef กับ #endif จะถูกแปล

ในทำนองเดียวกันรูปแบบของคำสั่ง #ifndef คือ

```
#ifndef       <macro-name>
              <statement sequence >
#endif
```

ในกรณีนี้ถ้าชื่อแมคโครไม่ได้ถูกกำหนดมาก่อนด้วยคำสั่ง #define แล้วคำสั่งที่อยู่ระหว่าง #ifndef กับคำสั่ง #endif จะถูกแปล

ตัวอย่างที่ 8.8 โปรแกรมแสดงการใช้คำสั่ง #ifdef กับ #ifndef

```
#define       TED             10
main ( )
{
#ifdef  TED
        printf ("Hi Ted\n");
#else
        printf ("Hi anyone\n");
#endif
#ifndef     RALPH
        printf ("RALPH not defind");
#endif
return 0;
}
```

ผลลัพธ์ที่ได้ คือ

```
Hi anyone
RALPH not defind
```

8.6 การใช้งาน #define และ #include

คำสั่ง #include ช่วยให้เราสามารถรวมแฟ้มที่บรรจุคำสั่งต่าง ๆ ที่กำหนดไว้ มารวมในแฟ้มต้นฉบับ และคำสั่ง #define ช่วยให้เราสามารถกำหนดแมคโครต่างๆ แล้วนำมาใช้ได้ ดังนั้น ในการใช้งานโดยทั่ว ๆ ไป จะนิยมกำหนดแมคโครไว้ในแฟ้มที่มีนามสกุลเป็น .h เพื่อบอกว่าคุณสมบัติของแฟ้มนี้เป็น header แล้วผู้ใช้สามารถนำแมคโครในแฟ้มนี้มาใช้ในโปรแกรมต่าง ๆ โดยใช้คำสั่ง #include

ตัวอย่างที่ 8.9 ตัวอย่างนี้แสดงแฟ้ม 'bool.h' ซึ่งใช้เก็บนิยามของแมคโคร BOOLEAN, TRUE, FALSE, NOT, AND, OR และ BOOL VAL ตามลำดับ

```
/*
*     bool.h   - - Header file defining BOOLEAN data type
*/

#define       BOOLEAN       short
#define       TRUE          1
#define       FALSE         0
#define       NOT           !
#define       AND           &&
#define       OR            ||
#define       BOOLVAL(X)  (x)? "TRUE" : "FALSE"
```

แบบฝึกหัด

1. จงตอบคำถามต่อไปนี้

   1.1 คำสั่งของตัวประมวลผลก่อน C ทุกคำสั่งจะต้องขึ้นต้นด้วยสัญลักษณ์ใด

   1.2 คำสั่งสำหรับการแปลโปรแกรมอย่างมีเงื่อนไข เมื่อมีหลาย ๆ เงื่อนไขสามารถใช้คำสั่งใดได้บ้าง

   1.3 คำสั่งของตัวประมวลผลก่อน C ที่ใช้กำหนดค่าคงตัวหรือนิยามแมคโครคือคำสั่งใด

   1.4 คำสั่งของตัวประมวลผลก่อน C คำสั่งใดที่ใช้เพื่อหมายถึง

   (1) #if define(name)

   (2) #if !define(macro_name)

   1.5 คำสั่งของตัวประมวลผลก่อน C คำสั่งใดใช้เพื่อรวมแฟ้มข้อมูลเข้าไปในแฟ้มอื่น

2. จงเขียนคำสั่งตัวประมวลผลก่อน C เพื่อดำเนินการต่อไปนี้

   (1) กำหนดค่าคงตัว YES ให้มีค่า 1

   (2) กำหนดค่าคงตัว NO ให้มีค่า 0

   (3) ให้รวมแฟ้ม common.h เข้ามาในแฟ้มต้นฉบับเพื่อแฟ้ม common.h อยู่ในสารบบที่ทำงานอยู่ ณ ขณะนั้น

   (4) ถ้า TRUE ถูกกำหนดค่ามาก่อน ให้ยกเลิกนิยามของ TRUE แล้วกำหนดค่าใหม่เป็น 1

   (5) ถ้า TRUE ไม่เท่ากับ 0 ให้กำหนด FALSE เป็น 0 และกำหนดให้ FALSE เป็น 1 ในกรณีอื่น ๆ

   (6) กำหนดแมคโคร

   1) ADD (x, y) เพื่อหาผลบวกระหว่าง x กับ y
   2) MUL (x, y) เพื่อหาผลคูณระหว่าง x กับ y
   3) MAX (x, y) เพื่อหาค่าที่มากกว่าระหว่าง x กับ y
   4) DIV (x, y) ซึ่งคืนค่า x/y เมื่อ y != 0 และคืนค่า 0 ในกรณีอื่นๆ
   5) PRINTINT (x) เพื่อพิมพ์จำนวนเต็ม x ออกทางอุปกรณ์ส่งออกมาตรฐาน
   6) SQUARE_VOLUME เพื่อหาค่าปริมาตรของรูปลูกบาศก์

3. จงกำหนดแฟ้ม pascal.h ให้เป็นคำสั่งในภาษา C สำหรับคำสั่งภาษา Pascal ต่อไปนี้ BEGIN, END, IF, THEN, ELSE

4. จงเขียนโปรแกรมที่มีการกำหนดแมคโครที่มีอาร์กิวเมนต์ 1 ตัว เพื่อคำนวณปริมาตรของรูปทรงกลม โดยให้แสดงผลลัพธ์ปริมาตรของรูปทรงกลมที่มีรัศมี 1-10 หน่วยในรูปของตาราง เมื่อสูตรที่ใช้คือ (4/3)*PI*r3(PI = 3.1415926)

5. จงเขียนโปรแกรมที่ใช้แมคโคร ‘MINIMUM2’ ในการหาค่าที่น้อยที่สุดระหว่างจำนวนเต็ม 2 จำนวนที่รับมาจากแผงแป้นอักขระ
6. จงเขียนโปรแกรมหาค่าน้อยที่สุดระหว่างจำนวนเต็ม 3 จำนวนที่รับค่ามาจากแผงแป้นอักขระ โดยใช้แมคโคร ‘MINIMUM3’ และแมคโคร ‘MINIMUM3’ เรียกใช้แมคโคร ‘MINIMUM 2’