บทที่ 9

ฟังก์ชัน (Functions)

ฟังก์ชัน (Function) จะคล้ายกับโปรแกรมย่อยหรือโพซีเยอร์ แต่ต่างกันตรงที่ฟังก์ชันสามารถคืนค่าผลลัพธ์ออกมาหนึ่งค่าให้กับชื่อฟังก์ชันได้ เราสามารถให้ฟังก์ชันประมวลผลอย่างใดอย่างหนึ่งในโปรแกรมได้และส่วนอื่น ๆ ของโปรแกรมสามารถเรียกใช้ฟังก์ชันได้ ตำแหน่งของฟังก์ชันจะเป็นตำแหน่งเดียวกับโพซีเยอร์ตามรูปแบบโครงสร้างโปรแกรมที่ได้ศึกษามา นอกจากนี้ในเทอร์โบปาสคาลได้สร้างฟังก์ชันมาตรฐานเอาไว้ซึ่งเราสามารถเรียกใช้ได้เช่นเดียวกับโพซีเยอร์

9.1 โครงสร้างของฟังก์ชัน

ในการสร้างฟังก์ชันจะต้องกำหนดรูปแบบของฟังก์ชันดังต่อไปนี้

```
FUNCTION     ชื่อฟังก์ชัน (พารามิเตอร์;;) : ประเภทข้อมูลของผลลัพธ์;
BEGIN
        สเตตเมนต์;
END;
```

จะเห็นว่าโครงสร้างของฟังก์ชันจะคล้ายกับโพซีเยอร์ แต่จะเริ่มต้นด้วยคำว่าฟังก์ชัน ตามด้วยชื่อฟังก์ชัน และมีพารามิเตอร์ต่าง ๆ ที่ต้องใช้ในฟังก์ชัน จากนั้นการทำงานต่าง ๆ ภายในฟังก์ชันจะเริ่มต้นด้วย BEGIN และจบท้ายด้วย END; และก่อนที่จะจบฟังก์ชันด้วย END; จะต้องมีการคืนค่าให้กับชื่อฟังก์ชันเสียก่อน นอกจากนี้ในฟังก์ชันสามารถประกาศตัวแปรได้เช่นเดียวโพซีเยอร์ ตัวอย่างเช่น

```
FUNCTION  ADD(x,y : integer) : integer;
BEGIN
        ADD  := X + Y;                    { คืนค่าให้ชื่อฟังก์ชัน }
END;
```

ฟังก์ชันนี้ชื่อว่า ADD โดยจะทำหน้าที่บวกเลขที่รับเข้าไปในพารามิเตอร์ x และ y โดยตัวเลขที่รับมาจะเป็นเลขจำนวนเต็ม และค่าของผลลัพธ์จากการทำฟังก์ชันจะเป็นจำนวนเต็มเช่นกัน สำหรับการเรียกใช้ฟังก์ชันสามารถทำได้เช่นเดียวกับโพซีเยอร์ โดยเรียกชื่อฟังก์ชันขึ้นมาเลย แต่ชื่อของฟังก์ชันจะมีค่าอยู่ภายในตัวมันเอง ตัวอย่างต่อไปเป็นการสร้างฟังก์ชันสำหรับหาด้านตรงข้ามมุมฉากของสามเหลี่ยม โดยพารามิเตอร์จะรับค่าความยาวของด้านประกอบมุมฉาก x และ y จากนั้นจะให้ค่าความยาวด้านตรงข้ามมุมฉากออกมา

```
FUNCTION HYPOT(x,y : real) : real;
BEGIN
        HYPOT  :=  sqrt (sqr(x) + sqr (y) );
END;
```

ตัวอย่างที่ 9.1 โปรแกรมนี้จะสร้างฟังก์ชันสำหรับบวกเลขที่เป็นจำนวนเต็มสองจำนวน โดยรับเลขทางแป้นพิมพ์เข้าไปสองจำนวนและแสดงผลบวกออกมา

```
PROGRAM TEST;
USES CRT;
VAR    A,B  :  INTEGER;

FUNCTION ADD(X,Y : INTEGER) : INTEGER;
BEGIN
        ADD  :=  X + Y;                          { คือค่าให้ชื่อฟังก์ชัน }
END;

BEGIN
        CLRSCR;
        WRITE('INPUT A ');
        READLN(A);
        WRITE('INPUT B ');
        READLN(B);
        WRITELN(A,' + ',B,' = ',ADD(A,B));       { เรียกใช้ฟังก์ชัน ADD }
END.
```

จากโปรแกรมเมื่อรันเครื่องจะให้ใส่ค่าตัวเลขสองตัว โดยจะเก็บอยู่ในตัวแปร A และ B สำหรับในบรรทัดที่เรียกใช้ฟังก์ชัน ตัวแปรพารามิเตอร์ X และ Y ที่ประกาศอยู่ในฟังก์ชันจะรับค่าจากตัวแปร A และ B ไปคำนวณ สำหรับผลลัพธ์ที่ได้จากการคำนวณจะเป็นจำนวนเต็มเช่นกัน ตามที่ได้ประกาศไว้หลังชื่อฟังก์ชัน

ตัวอย่างที่ 9.2 เป็นตัวอย่างการสร้างฟังก์ชันหาค่ายกกำลังสามของตัวเลขที่รับเข้าไป โดยสร้างฟังก์ชันชื่อ FINDCUBE

```
PROGRAM TEST;
USES  CRT;
VAR   NUM,CUBE  :  LONGINT;

FUNCTION FINDCUBE(n : LONGINT) : LONGINT;
BEGIN
        FINDCUBE  :=  n*n*n;
END;

BEGIN  {MAIN}
        CLRSCR;
        WRITE('Enter an integer  ');
        READLN(NUM);
        CUBE  :=  FINDCUBE(NUM);
        WRITELN(NUM,'  CUBED  IS  ',CUBE)
END.
```

จากโปรแกรมจะเห็นว่าในฟังก์ชันที่สร้างขึ้นจะนำค่าที่รับเข้ามาผ่านทางตัวแปร n มาคูณกันสามครั้ง จากนั้นคืนค่าให้กับชื่อฟังก์ชัน ส่วนในโปรแกรมหลักจะให้ใส่ตัวเลขเข้าไปเก็บไว้ในตัวแปร NUM จากนั้นเรียกชื่อฟังก์ชันและนำค่าที่ได้จากการทำฟังก์ชันมาเก็บไว้ในตัวแปร CUBE และพิมพ์ออกเป็นคำตอบ

ตัวอย่างที่ 9.3 เป็นตัวอย่างการสร้างโปรแกรมหาค่าผลบวกของตัวเลขจำนวนเต็มที่รับเข้าไปเช่น ถ้าใส่ค่า 5 เข้าไป ฟังก์ชันจะทำการบวกเลขตั้งแต่ 1 ถึง 5

```
PROGRAM TEST;
USES  CRT;
VAR   n  :  INTEGER;

FUNCTION FINDSUM(numb : INTEGER) : INTEGER;
VAR   i,sum  : INTEGER;
BEGIN
          sum  :=  0;
          FOR i := 1 TO numb DO
                    sum  :=  sum + i;
          FINDSUM  :=  sum
END;

BEGIN  {MAIN}
          CLRSCR;
          WRITELN('Will find the sum of the first n integer');
          WRITE('Enter a value for n  ');
          READLN(n);
          WRITELN('  Sum of first  ', n,' integer:  ',FINDSUM(n))
END.
```

จากโปรแกรมจะเห็นว่าฟังก์ชันที่หาค่าผลบวกที่สร้างขึ้นจะมีการประกาศตัวแปรแบบเฉพาะที่ชื่อ sum และ i สำหรับใช้ภายในฟังก์ชัน โดยการหาค่าผลบวกจำทำโดยใช้คำสั่งทำซ้ำให้บวกเลขตั้งแต่ค่า 1 ถึงค่าที่ต้องการ

ตัวอย่างที่ 9.4 ตัวอย่างนี้จะสร้างฟังก์ชั่นหาค่า x ยกกำลัง y โดยสร้างฟังก์ชั่นชื่อ XtoTheY และผ่านค่าตัวแปรเข้าไป 2 ตัว และให้โปรแกรมหลักวนลูปเรียกใช้ฟังก์ชั่น เมื่อโปรแกรมทำงานจะแสดงค่า 2 ยกกำลัง 1 ถึง 16 ออกทางจอภาพ

```
PROGRAM Powers;
USES CRT;
FUNCTION XToTheY(X: Integer; Y: Integer): Real;   { หาค่า X ยกกำลัง Y }
BEGIN
```

```
        XToTheY := Exp(Y * Ln(X));
END;

VAR     i       :  Integer;
        Result  :  Real;
BEGIN {main}
        Clrscr;
        FOR i :=  1   TO   16   DO
        BEGIN
                Result  :=  XToTheY(2,i);          { คำนวณค่า 2 ยกกำลัง i }
                WriteLn(Result:5:0);               { พิมพ์ผลลัพธ์ }
        END;
        WriteLn;
        WriteLn('Press ENTER to end this program');
        ReadLn;
END.
```

หลังจากที่ได้ศึกษามาถึงตรงนี้เราสามารถสรุปได้ว่าการสร้างฟังก์ชันจะมีรูปแบบดังนี้

```
FUNCTION FNAME (parameter  list) : ftype;  { ส่วนหัวฟังก์ชัน }
Local declaration section
BEGIN
        Calculate result                   { ส่วนตัวฟังก์ชัน }
        And assign it to fname
END;
```

ตัวอย่างที่ 9.5 ถ้าหากเราต้องการสร้างฟังก์ชันที่คำนวณค่าในลักษณะดังนี้

$$3^4 = 3 * 3 * 3 * 3 = 81$$

โดยสร้างเป็นฟังก์ชันชื่อ POWER ให้รับค่าเข้าไปสองค่า โดยค่าแรกเป็นค่าฐาน ค่าที่สองเป็นค่ายกกำลัง เช่น POWER(3,4) สามารถเขียนฟังก์ชันได้ดังนี้

```
FUNCTION   POWER (x  : real;n : integer) : real;
VAR      I          :  integer;
         Prod       :  real;
BEGIN
         Prod  :=  1;
         For  I := 1 to n do
                  Prod  :=  Prod * x;
         POWER  :=  prod
END;
```

การทำงานภายในฟังก์ชันอาจให้มีการเลือกทำหลายทางได้อีกด้วย โดยใช้คำสั่ง case หรือคำสั่ง IF ในตัวอย่างโปรแกรมที่ 9.6 เป็นการสร้างฟังก์ชันเพื่อคำนวณต้นทุนสินค้า ซึ่งขึ้นกับจำนวนผลผลิตสามช่วงด้วยกัน ถ้าเกิน 10 ชิ้น จะราคาชิ้นละ 40 บาท ถ้าเกิน 5 ชิ้นราคาชิ้นละ 42 บาท แต่ถ้าไม่เกิน 5 ชิ้นราคาชิ้นละ 44 บาท โดยจะสร้างฟังก์ชันชื่อ FINDCOSE เพื่อคำนวณราคาสินค้าดังกล่าว

ตัวอย่างที่ 9.6 คำนวณราคาเสื้อแจ๊กเก็ต โดยโปรแกรมจะให้ใส่จำนวนเสื้อเข้าไปและจะแจ้งราคาออกมา

```
PROGRAM  TEST;
USES   CRT
VAR      n,cost  :  integer;
FUNCTION       FINDCOST(n : integer) : integer;
BEGIN
         If n >= 10  Then
                  FINDCOST  :=  n * 40
         Else     if n >= 5 Then
                  FINDCOST  :=  n * 42
         Else     FINDCOST  :=  n * 44
END;
```

```
BEGIN {MAIN}
        WRITE('Enter number of jackets  ');
        READLN(n);
        Cost  :=  FINDCOST(n);                  { เรียกใช้ฟังก์ชันและคืนค่าให้ Cost }
        WRITELN (n,'  jackets  cost $ ',Cost)
END.
```

9.2 ฟังก์ชันมาตรฐานของเทอร์โบปาสคาล

การเขียนโปรแกรมที่มีการสร้างฟังก์ชันหรือโพซีเยอร์ถ้าหากเราต้องการเรียกใช้ฟังก์ชันหรือโพซีเยอร์เหล่านั้นเราจะต้องทราบว่าตัวแปรที่จะส่งเข้าไปเป็นตัวแปรประเภทใด และหลังจากการทำงานจะได้ข้อมูลประเภทใดออกมา ในเทอร์โบปาสคาลได้มีการสร้างฟังก์ชันมาตรฐานเอาไว้ให้เราเรียกใช้ได้ โดยเราจะต้องทราบว่าการใช้ฟังก์ชันต่าง ๆ จะต้องส่งข้อมูลเข้าไปในฟังก์ชันอย่างไร ตัวอย่างต่อไปเป็นการใช้ฟังก์ชันมาตรฐานทางคณิตศาสตร์ของเทอร์โบปาสคาลที่น่าสนใจบางส่วน สำหรับฟังก์ชันอื่น ๆ ให้ศึกษาจากคู่มือของโปรแกรมโดยตรง

1. ฟังก์ชันหาค่าสัมบูรณ์ (Absolute Value Function)

ฟังก์ชันนี้จะให้ค่าที่เป็นบวกเสมอ รูปแบบของฟังก์ชันเป็นดังนี้

```
ABS (R : real) : real
ABS (I  : Integer) : Integer
```

โดยถ้ารับค่าเข้าไปเป็นจำนวนจริงจะให้ผลลัพธ์เป็นจำนวนจริงบวก ถ้ารับค่าเข้าไปเป็นจำนวนเต็มจะให้ค่าออกมาเป็นจำนวนเต็มบวก ตัวอย่างเช่น

```
x       :=      -6.31;
y       :=      ABS(x) * 2;
```

ผลลัพธ์ที่ได้จะเป็นการนำค่า 6.31 ไปคูณกับ 2 และเก็บไว้ในตัวแปร y

2. ฟังก์ชันหาค่ายกกำลังสอง (Square Function)

ใช้สำหรับหาค่ายกกำลังสองของจำนวนที่ใส่เข้าไป อย่างเช่นค่ากำลังสองของ 1.2 จะมีค่าเป็น 1.2 * 1.2 หรือ 1.44 ค่ากำลังสองของ –25 มีค่าเป็น –25 * -25 หรือ 625 รูปแบบเป็นดังนี้

```
SQR(R : real) : real
```

ตัวอย่างเช่น

```
x       :=      2;
```

```
y       :=      3 * SQR(x) + x – 4;
```

จะเป็นการคำนวณค่า $3*x^2 + x - 4$

3. ฟังก์ชันหาค่ารากที่สอง (Square Root)

จะให้ค่ารากที่สองของจำนวนบวกที่ใส่เข้าไป โดยผลลัพธ์ที่ได้จะเป็นเลขจำนวนจริง อย่างเช่นถ้าหารากของ 25 จะได้เป็น 5 รูปแบบของการใช้ฟังก์ชันเป็นดังต่อไปนี้

Sqrt (<nonnegative real>)

ตัวอย่าง

```
x       :=      -1.44;
y       :=      Sqrt(Abs(x));
```

จะเห็นว่าเป็นการเรียกใช้ฟังก์ชันซ้อนฟังก์ชันโดยค่าที่ให้กับฟังก์ชัน Sqrt ต้องเป็นบวกเท่านั้น ค่า x จะเป็นเลขลบและหลังจากทำฟังก์ชัน Abs จะทำให้ได้ค่าเป็นบวกและส่งให้ฟังก์ชัน Sqrt ทำงานต่อไป ผลลัพธ์ที่ได้จะได้ค่า y เป็น 1.2

ตัวอย่าง

```
a       :=      5;
b       :=      12;
c       :=      Sqrt( Sqr(a) + Sqr(b) )
```

การเขียนฟังก์ชั่นแบบนี้คล้ายกับการหาด้านตรงข้ามมุมฉากของสามเหลี่ยมที่มีด้านประกอบมุมฉากเป็น a และ b

4. ฟังก์ชั่นหาค่า Pi

การคำนวณต่าง ๆ โดยเฉพาะการคำนวณเกี่ยวกับวงกลมที่ต้องใช้ค่า Pi เราสามารถใช้ฟังก์ชันนี้ได้ โดยฟังก์ชัน Pi จะคือค่า 3.1415926535897932385 ซึ่งเป็นค่าอัตราส่วนระหว่างเส้นรอบวงกลมกับเส้นผ่าศูนย์กลาง ฟังก์ชันนี้จะไม่มีการรับค่าอาร์กิวเมนต์

ตัวอย่าง

```
rad     :=      5;
area    :=      Pi * Sqr(rad);
WRITELN ('Area :  ',area:6:2);
```

เป็นการคำนวณหาค่าพื้นที่ ผลลัพธ์ที่ได้จะเป็น Ares : 78.54

5. ฟังก์ชั่น Round

ฟังก์ชั่นนี้จะรับค่าเลขจำนวนจริงเข้าไป และจะคืนค่าเลขจำนวนเต็มที่ใกล้ค่าเลขจำนวนจริงนั้นมากที่สุด ตัวอย่างเช่นถ้าหาค่าของ –3.34 จะได้ค่า –3 กลับมา ถ้าหาค่าของ 3.54 จะได้ค่า 4 กลับมา รูปแบบของฟังก์ชั่นเป็นดังนี้

```
Round (R : Real) : Longint
```

ตัวอย่าง

```
ans     :=      Round(Sqrt(74.35));
```

ค่ารากของ 74.35 จะมีค่าเป็น 8.6 ถ้าเขียนประโยคตามข้างบนนี้จะได้ค่าเป็นจำนวนเต็มกลับมา ซึ่งจะทำให้ ans มีค่าเป็น 9

ตัวอย่าง

```
r       :=      34.567;
s       :=      Round(r * 100 + 0.5)/100;
```

จากประโยคด้านบน ค่า r คูณกับ 100 จะได้เป็น 3456.7 และบวกด้วย 0.5 จะได้เป็น 3457.2 ฟังก์ชัน Round จะปรับค่าให้เป็น 3457 จากนั้นหารด้วย 100 จะทำให้ s มีค่าเป็น 34.57

6. ฟังก์ชัน Trunc

ฟังก์ชันนี้จะนำค่าที่รับเข้าไปเป็นจำนวนจริงและตัดทดนิยมทิ้ง รูปแบบเป็นดังนี้

```
Trunc ( <real number> )
```

ตัวอย่าง

```
Trunc (3.34)            จะได้ 3
Trunc (-3.54)           จะได้ –3
```

ตัวอย่าง

```
ans     :=      Trunc(Sqrt(74.35) );
```

เป็นการหาค่ารากที่สองของ 74.35 โดยทำให้เป็นเลขจำนวนเต็ม จะได้ค่า ans เป็น 8

7. ฟังก์ชัน Random

ใช้สำหรับสุ่มตัวเลข รูปแบบการใช้มีสองลักษณะดังนี้

```
Random                  : real
Random(<Integer>)       : Integer
```

ฟังก์ชั่นนี้จะให้ค่าตัวเลขสุ่มที่สร้างขึ้นโดยเทอร์โบปาสคาล ถ้าไม่มีพารามิเตอร์จะสุ่มตัวเลขที่เป็นจำนวนจริงมีค่าตั้งแต่ 0 ถึง 1 แต่ถ้ามีการระบุพารามิเตอร์จะสุ่มตัวเลขตั้งแต่ 0 ถึงค่าพารามิเตอร์ที่ระบุโดยจะคืนค่าเป็นจำนวนเต็ม

ตัวอย่าง

```
I       :=      Random(10)
```

จะเป็นการสุ่มค่าตั้งแต่ 0 ถึง 10

การใช้ฟังก์ชันสุ่มตัวเลขนี้เมื่อรันโปรแกรมครั้งต่อ ๆ ไปโปรแกรมจะสุ่มได้ค่าเดิม ถ้าต้องการให้การสุ่มค่าต่าง ๆ มีค่าเปลี่ยนไปจะต้องเรียกโพซีเยอร์ Randomize ที่ต้นโปรแกรม โดยโพซีเยอร์นี้จะไม่มีพารามิเตอร์

ตัวอย่าง

```
PROGRAM     TEST
VAR    J   :  Integer;
BEGIN
        Randomize;
        J  :=  Random(100);
        WRITELN(J);
END.
```

โปรแกรมข้างต้นถ้าไม่เรียกโพซีเยอร์ Randomize รันโปรแกรมกี่ครั้งจะได้ค่าเดิม

ตัวอย่างที่ 9.7 เป็นตัวอย่างการใช้ฟังก์ชั่นของเทอร์โบปาสคาล โดยการใช้งานเราจะทราบจากคู่มือว่าการใช้ฟังก์ชั่นนั้น ๆ ต้องใส่อินพุตข้อมูลประเภทใดเข้าไปและข้อมูลที่ได้ออกมาเป็นข้อมูลประเภทใด

```
PROGRAM TEST;
USES  CRT;
BEGIN
        CLRSCR;
        WRITELN('ABS(-32.56)     =  ',ABS(-32.56));
        WRITELN('Exp(5)          =  ',Exp(5));
        WRITELN('Ln(100)         =  ',Ln(100));
        WRITELN('Sqr(5)          =  ',SQR(5));
        WRITELN('Sqrt(25.63)     =  ',SQRT(25.63));
        WRITELN('Round(32.567) =  ',Round(32.567));
        READLN;
END.
```

ผลลัพธ์ที่ได้จากการรันโปรแกรมจะเป็นดังนี้

ตัวอย่างที่ 9.8 เป็นตัวอย่างการสร้างฟังก์ชั่นให้คำนวนค่าฟังก์ชั่นทางคณิตศาสตร์ดังนี้

$$f(x) = X^3 + 3*X^2 + C$$

โดยโปรแกรมจะให้อินพุตค่า X และค่า C ในโปรแกรมจะประกาศตัวแปร X,Y เป็นตัวแปรแบบทั่วไป โดยให้ X แทนค่า X ในฟังก์ชั่น f(x) และ Y แทนค่า C ในฟังก์ชั่น f(x) ในโปรแกรมได้สร้างฟังก์ชันชื่อ CAL_1 ให้ทำการคำนวณค่าตามที่กำหนด

```
PROGRAM TEST;
USES  CRT;
VAR   X,Y  :  INTEGER;

FUNCTION CAL_1(A,B  :  INTEGER) : INTEGER;
VAR   OUT  :  INTEGER;
BEGIN
        OUT   :=  A*A*A + 3*A*A + B;
        CAL_1 := OUT;
END;

BEGIN  { MAIN }
        CLRSCR;
        WRITELN('CAL  X^3 + 3*X^2 + C ');
        WRITE('Input X :=  ');
        READLN(X);
        WRITE('Input C :=  ');
        READLN(Y);
        WRITELN('X^3 + 3*X^2 + C  = ',CAL_1(X,Y));
        READLN;
END.
```

ตัวอย่างที่ 9.9 จะนำโปรแกรมตัวอย่างที่ 9.8 มาเขียนขึ้นใหม่ โดยให้โปรแกรมวนทำงานซ้ำเมื่อต้องการคำนวณค่าฟังก์ชั่นอีก โดยหลังจากคำนวณค่าฟังก์ชั่นแล้วเครื่องจะถามว่าต้องการทำอีกหรือไม่ ถ้ากดคีย์ N ทางแป้นพิมพ์เครื่องจะออกจากโปรแกรม

```
PROGRAM TEST;
USES  CRT;
VAR   X,Y  :  INTEGER;
Ch    :  Char;

FUNCTION CAL_1(A,B  :  INTEGER) : INTEGER;
VAR   OUT  :  INTEGER;
BEGIN
        OUT   :=  A*A*A + 3*A*A + B;
        CAL_1 := OUT;
END;

BEGIN
        REPEAT
                CLRSCR;
                WRITELN('CAL  X^3 + 3*X^2 + C ');
                WRITE('Input X :=  ');
                READLN(X);
                WRITE('Input C :=  ');
                READLN(Y);
                WRITELN('X^3 + 3*X^2 + C  = ',CAL_1(X,Y));
                WRITE('Do you want to continue ? (Y/N) ');
                Ch := readkey;
        UNTIL (Ch = 'n') or (Ch = 'N');
END.
```

การทำงานของโปรแกรมเมื่อจบการคำนวณฟังก์ชั่น โปรแกรมจะเรียกโพซีเยอร์ชื่อ readkey ของเทอร์โบปาสคาล โดยโพซีเยอร์นี้จะรับอักขระหนึ่งตัวโดยไม่ต้องกดคีย์ Enter จากนั้นส่งค่าให้ตัวแปร Ch ส่วนในการทำงานซ้ำนั้นจะใช้ลูปแบบ REPEAT... UNTIL โดยหลังจากทำลูปหนึ่งครั้งเครื่องจะมาตรวจสอบเงื่อนไขว่าค่าที่กดเป็นตัว N หรือ n จริงหรือไม่ ถ้าจริงจะจบ

โปรแกรม สาเหตุที่ต้องตรวจสอบว่าเป็น N หรือ n นั้นเนื่องจากว่าโปรแกรมไม่ทราบว่าแป้นพิมพ์ที่ใช้อยู่ในคีย์ Caps Lock ถูกกดอยู่หรือไม่

ในโปรแกรมที่ 9.9 เมื่อรันโปรแกรมหลังจากเครื่องถามว่าต้องการทำงานต่อหรือไม่ ถ้ากด N จะออกจากโปรแกรม แต่ถ้ากดคีย์อื่น ๆ ไม่จำเป็นต้องเป็นคีย์ Y โปรแกรมก็จะทำงานต่อ ซึ่งถือได้ว่าโปรแกรมนี้ไม่สมบูรณ์ ในโปรแกรมตัวอย่างที่ 9.10 จะสร้างฟังก์ชันสำหรับตรวจสอบการกดคีย์ขึ้นมา โดยจะรับค่าคีย์ N หรือ Y สองคีย์เท่านั้น ถ้ากดคีย์อื่น ๆ โปรแกรมจะทำงานรับคีย์ซ้ำ ฟังก์ชั่นนี้ให้ชื่อว่า Get_Yes_No ภายในฟังก์ชั่นจะใช้ลูปแบบ REPEAT…UNTIL วนซ้ำ ถ้าไม่ได้กด N หรือ Y จะวนซ้ำ

ตัวอย่างที่ 9.10 เป็นโปรแกรมทำงานคำนวณตามฟังก์ชันที่กำหนด หลังจากทำฟังก์ชั่นถ้ากด Y จะให้โปรแกรมทำการคำนวณอีกครั้ง ถ้ากด N จะออกจากโปรแกรม

```
PROGRAM TEST;
USES  CRT;
VAR   X,Y          :  INTEGER;
          Ch       :  Char;

FUNCTION CAL_1(A,B  :  INTEGER) : INTEGER;
VAR   OUT  :  INTEGER;
BEGIN
          OUT   :=  A*A*A + 3*A*A + B;
          CAL_1 := OUT;
END;
FUNCTION Get_Yes_No  :  char;
VAR   Ch  :  Char;
BEGIN
          REPEAT
                    Ch  :=  upcase(readkey);
          UNTIL ch in ['N','Y'];
          Get_Yes_No  :=  Ch;
END;
BEGIN  { MAIN }
          REPEAT
                    CLRSCR;
```

```
                WRITELN('CAL  X^3 + 3*X^2 + C ');
                WRITE('Input X :=  ');
                READLN(X);
                WRITE('Input C :=  ');
                READLN(Y);
                WRITELN('X^3 + 3*X^2 + C  = ',CAL_1(X,Y));
                WRITE('Do you want to continue ? (Y/N) ');
                Ch  :=  Get_Yes_No;
        UNTIL ch = 'N';
END.
```

ตัวอย่างที่ 9.11 เป็นโปรแกรมคำนวณตามฟังก์ชัน

$$f(X) = X^2 + X$$

โดยให้ X มีค่าตั้งแต่ 1 ถึง 10 และแสดงผลทางหน้าจอคอมพิวเตอร์

```
PROGRAM TEST;
USES  CRT;
VAR   I  :  INTEGER;

FUNCTION CAL_1(A :  INTEGER) : Real;
BEGIN
        CAL_1   :=  SQR(A) + A;
END;

BEGIN
        CLRSCR;
        WRITELN('X          X^2 + X');
        WRITELN('_____________________');
        FOR I := 1 TO 10 DO
                WRITELN(I,CAL_1(I):15:2);
        READLN;
END.
```

ในโปรแกรมจะสร้างฟังก์ชันชื่อ CAL_1 โดยรับค่าเข้าไปเป็นจำนวนเต็มและคือค่าออกมาเป็นจำนวนจริง หลังจากรันโปรแกรมจะได้ผลลัพธ์ดังนี้

ตัวอย่างต่อไปเป็นการเขียนโปรแกรมในงานประยุกต์ทางวิทยาศาสตร์ในการพื้นที่ใต้กราฟของฟังก์ชั่น y = f(x) โดยให้ x มีช่วงระหว่าง a ถึง b การหาค่าพื้นที่ใต้กราฟนี้จะมีค่าเท่ากับการอินทริกัลของฟังก์ชั่น f(x) จาก a ไป b โดยหลักการหาพื้นที่ใต้กราฟจะทำได้โดยแบ่งช่วง a ถึง b ออกเป็นสี่เหลี่ยมเล็ก ๆ n รูป สี่เหลี่ยมแต่ละรูปมีความกว้างเท่ากับ Δx และสูงเท่ากับ f(x) เมื่อนำพื้นที่ของสี่เหลี่ยมแต่ละรูปมารวมกันก็จะได้พื้นที่ใต้กราฟที่ต้องการหา ถ้า n มีค่ามากจะทำให้ค่าที่ได้ถูกต้องมากขึ้น แต่เวลาที่ใช้ในการคำนวณจะนานขึ้นตามไปด้วย

สมมุติว่าต้องการหาพื้นที่ใต้กราฟของฟังก์ชั่น $f(x) = x^2 + 1$ โดย x อยู่ในช่วง 0 ถึง 1 และแบ่งออกเป็น 3 ช่วงจะได้ดังรูปต่อไปนี้

โปรแกรมในตัวอย่างที่ 9.12 เป็นโปรแกรมชื่อ ApproxArea ใช้สำหรับหาพื้นที่ใต้กราฟของฟังก์ชั่น f(x) ที่กำหนดในฟังก์ชั่น F โดยฟังก์ชั่น f(x) นี้ผู้ใช้สามารถเปลี่ยนแปลงตามต้องการได้ สำหรับในโปรแกรมจะกำหนดให้มีค่าเท่ากับ $x^2 + 1$ สำหรับความกว้างของสี่เหลี่ยมเล็ก ๆ หาได้โดย

width := (b – a) / n;

เมื่อรันโปรแกรมเครื่องจะให้ป้อนค่าช่วง a ถึง b และใส่ค่า n ที่ต้องการแบ่ง จากนั้นจะแสดงค่าพื้นที่ใต้กราฟออกมา

ตัวอย่างที่ 9.12 โปรแกรมหาพื้นที่ใต้กราฟ หรือโปรแกรมหาค่าอินทริกัลของฟังก์ชั่น f(x) อย่างง่าย ผู้ใช้โปรแกรมสามารถเปลี่ยนแปลงค่าของฟังก์ชั่น f(x) ได้

```
PROGRAM ApproxArea;
USES CRT;
VAR     area     : Real;
        a,b,n    : Integer;
FUNCTION F(x:Real) : Real;                  { กำหนดฟังก์ชั่น }
BEGIN
        F := (x*x) + 1;
END;
FUNCTION FINDAREA(a,b,n : Integer) : Real;
VAR     I : Integer;
        width,midpoint,height,Asum : Real;
BEGIN
        width := (b - a)/n;                 { ความกว้างของสี่เหลี่ยมเล็ก ๆ }
        Asum  := 0;
        midpoint := a + width / 2;
        FOR i := 1 TO n DO                  { หาผลรวมของพื้นที่ทั้งหมด }
        BEGIN
                height := F(midpoint);
                Asum := Asum + height*width;
                midpoint := midpoint + width;
        END;
        FINDAREA := Asum
END;
```

```
BEGIN
        CLRSCR;
        WRITE('Enter endpts for interval and number of rectangles ');
        READLN(a,b,n);
        area  :=  FINDAREA(a,b,n);
        WRITELN('Approximate area is  ',area:7:4);
        READLN;
END.
```

ในการสร้างฟังก์ชั่นขึ้นมาเราสามารถให้ฟังก์ชั่นคืนค่าที่เป็นค่าทางบูลีนออกมาได้เช่นกัน ในตัวอย่างโปรแกรมต่อไปจะเป็นโปรแกรมให้ใส่ค่าเลข ID number จำนวนสามหลัก ถ้าใส่ค่าไม่ถูกต้องเครื่องจะแจ้งออกมาว่าไม่ถูกต้อง โดยโปรแกรมจะรับเลข ID เข้ามาสามหลักที่มีคุณสมบัติดังนี้

หลักร้อย = หลักสิบ + หลักหน่วย

จากคุณสมบัติที่กำหนดถ้าป้อนเป็น 123 จะไม่ถูกต้อง แต่ถ้าป้อนเป็น 321 หรือ 532 จะเป็นค่าที่ถูกต้อง

การทำงานของโปรแกรมเริ่มแรกเครื่องจะให้ป้อนค่ากลุ่มเลข ID เข้าไป จากนั้นจะตรวจสอบว่ากลุ่มเลขนั้นเป็นไปตามคุณสมบัติที่กำหนดหรือไม่ ถ้าเป็นเครื่องจะแจ้งว่า You may enter ถ้าใส่ไม่ถูกต้องจะแจ้งว่า You are under arrest พร้อมกับส่งเสียงเตือน โดยการออกแบบโปรแกรมจะสร้างฟังก์ชั่นชื่อ TESTS_OK ซึ่งเป็นฟังก์ชั่นที่มีค่าเป็นบูลีน โดยออกแบบโปรแกรมแบบ Top down design ดังนี้

```
        get  Idnumber
        if  TESTS_OK  (IDnumber)
                then   writeln('You may enter ');
                else    writeln(chr(7),'You are under arrest')
```

ตัวอย่างที่ 9.13 โปรแกรมรับเลข ID

```
PROGRAM IDtest;
USES CRT;
VAR     IDnumber  :  Integer;

FUNCTION TESTS_OK(number : Integer) : Boolean;
VAR     ones,tens,hundreds  :  Integer;
BEGIN
        ones  :=  number mod 10;
        tens  :=  (number div 10) mod 10;
        hundreds  :=  number div 100;
        IF hundreds  =  ones + tens
                THEN  TESTS_OK  :=  True
                ELSE  TESTS_OK  :=  False
END;

BEGIN
        CLRSCR;
        WRITE('Enter your 3 digit ID number  ');
        READLN(IDnumber);
        IF TESTS_OK(IDnumber)
                THEN  WRITELN('You may enter')
                ELSE  WRITELN(Chr(7),'You are under arrest');
        READLN;
END.
```

ตัวอย่างต่อไปจะเป็นการเขียนโปรแกรมเพื่อหาตัวเลขจำนวนเฉพาะ (Prime Number) ซึ่งเป็นตัวเลขจำนวนเต็มตั้งแต่ 1 ขึ้นไปที่มีเฉพาะเลข 1 และตัวมันเองเท่านั้นที่หารได้ลงตัว เช่นเลข 2,3,5,7 และ 11 เป็นต้น สำหรับเลข 4 จะไม่เป็นจำนวนเฉพาะเนื่องจากว่าสามารถนำเลข 2 หารได้ลงตัว ตามทฤษฎีแล้วจะกล่าวว่าถ้าเลขจำนวนเต็ม n ไม่มีค่าที่อยู่ระหว่าง 2 ถึงค่ารากที่สองของ n หารได้ลงตัว จะเรียกเลข n ว่าจำนวนเฉพาะ ตัวอย่างเช่นเลข 67 จะเป็นจำนวนเฉพาะเนื่องจากว่าค่าระหว่าง 2 ถึง round(sqrt(67)) หรือค่าระหว่าง 2 ถึง 8 ไม่มีค่าใดหารได้ลงตัว

โปรแกรมในตัวอย่างที่ 9.14 เป็นโปรแกรมหาเลขจำนวนเฉพาะที่มีค่าระหว่าง 100 ถึง 200 ในการออกแบบโปรแกรมเราอาจพิจารณาง่าย ๆ ว่าจำนวนเฉพาะที่ได้จะต้องเป็นตัวเลขคี่แน่นอน ดังนั้นจะสร้างฟังก์ชั่นทางบูลีนชื่อ IS_ODDPRIME ถ้าตัวเลขที่ตรวจสอบเป็นจำนวนเฉพาะจะคืนค่าจริงออกมา ซูโดโคดของฟังก์ชั่น IS_ODDPRIME ทำได้โดยนำเลขคี่ d ตั้งแต่ 3 ถึงรากที่สองของ n ไปหารโดยใช้ลูป repeat_until ในการทำซ้ำ

```
FoundDivisor := false          { กำหนดค่าเริ่มต้น }
d  := 1;
Repeat
        d := d + 2;            { เริ่มใช้ค่า 3 เป็นค่าแรกในการหาร }
        if d divides n then FoundDivisor := true
until (FoundDivisor) or (d > √ n );
if-then-else test to assign value to IS_ODDPRIME
```

ตัวอย่างที่ 9.14 โปรแกรมหาค่าเลขจำนวนเฉพาะที่อยู่ระหว่าง 100 ถึง 200

```
PROGRAM PRIMES;
USES CRT;
VAR n : Integer;
FUNCTION IS_ODDPRIME(n : Integer) : Boolean;
VAR     d,upper         : Integer;
        FoundDivisor    : boolean;
BEGIN
        upper := round(sqrt(n));
        FoundDivisor := false;
        d := 1;
        REPEAT
                d := d + 2;
                IF n mod d = 0 THEN FoundDivisor := true
        UNTIL (FoundDivisor) OR (d >= upper);
        IF FoundDivisor
                THEN IS_ODDPRIME := false
                ELSE IS_ODDPRIME := true
END;
```

```
BEGIN
        CLRSCR;
        WRITELN('The primes between 100 and 200 are  ');
        n  :=  101;
        WHILE  n <= 200  do
        BEGIN
                IF  IS_ODDPRIME(n)  THEN  WRITE(n,' ');
                n  :=  n+2
        END;
        READLN;
END.
```

เมื่อรันโปรแกรมผลลัพธ์ที่ได้จะเป็น

```
The primes between 100 and 200 are
101  103  107  113  127  ……..
```

แบบฝึกหัดท้ายบท

1. จงบอกเอาต์พุตจากการทำโปรแกรมต่อไปนี้

```
PROGRAM  TEST_1;
VAR      I  :  Integer;
FUNCTION  Test (n : integer) : Integer;
VAR      A,B  :  integer;
BEGIN
          A  :=  n;
          B  :=  n * n – 1;
          IF (a > b) and (a <= 2)
                    Then  Test := 1
                    Else    Test := 2
END;
BEGIN   {MAIN}
          FOR  I := 1  to 3 do
          BEGIN
                    WRITELN('I  equals  ',I);
                    If TEST (I) = 1
                              Then  Writeln('one')
                              Else  Writeln('Two');
                    Writeln ('three')
          END;
          Writeln('four')
END.
```

2. จากตัวอย่างที่ 9.1 จงแก้ไขฟังก์ชัน POWER ให้สามารถรับค่าได้ทั้งบวกและลบ

3. จงเขียนโปรแกรมให้คอมพิวเตอร์คำนวณหาค่าต่อไปนี้

1/101 + 1/102 + 1/103 + ……..+ 1/500