บทที่ 5

การทำซ้ำ : ข้อความสั่ง while, for และ repeat

(Repetition : while, for and repeat Statements)

โปรแกรมซึ่งเราได้ศึกษามาแล้วจนถึงขณะนี้ ข้อความสั่งต่างๆ ในตัวโปรแกรมกระทำการเพียงหนึ่งครั้งเท่านั้น อย่างไรก็ตาม ซอฟต์แวร์เชิงพาณิชย์ส่วนใหญ่ กระบวนการอาจทำซ้ำหลายครั้ง ตัวอย่างเช่น ใน editor ของ Turbo Pascal ผู้ใช้สามารถเคลื่อนย้ายตัวชี้ตำแหน่ง (cnrsor) ไปยังบรรทัดโปรแกรมและกระทำการดำเนินการปรับแต่งหรือแก้ไข (edit) ได้บ่อยครั้งตามความจำเป็น

การทำซ้ำ หมายถึง โครงสร้างควบคุมโปรแกรมชนิดที่สาม (seqnence, selection, repetition) และการทำซ้ำของขั้นตอนต่างๆ ในโปรแกรมเรียกว่า **ส่วนวนซ้ำ หรือลูบ** (loop) ในบทนี้จะอธิบายข้อความสั่งควบคุมส่วนวนซ้ำของ Pascal สามชนิด ได้แก่ while, for และ repeat รวมทั้งอธิบายข้อดีของข้อความสั่งแต่ละชนิด สิ่งที่คล้ายกับข้อความสั่ง if คือส่วนวนซ้ำ มีการซ้อนในได้ และในบทนี้จะแสดงให้เห็นว่าจะเขียนและใช้ส่วนวนซ้ำซ้อนใน (nested loops) ในโปรแกรมอย่างไร

5.1 ข้อความสั่ง while (The while Statement)

การศึกษาเรื่องการวนซ้ำ เริ่มต้นด้วยข้อความสั่งควบคุมการวนซ้ำ ซึ่งคล่องตัวมากที่สุด คือ ข้อความสั่ง while ส่วนของโปรแกรมในรูป 5.1 คำนวณและแสดงผลค่าจ้างรายสัปดาห์ของพนักงานแต่ละคนมี 7 คน สมมติว่าไม่มีค่าจ้างล่วงเวลา คำสั่งซึ่งทำซ้ำกันเรียกว่า ตัวส่วนวนซ้ำ (loop body) ตามหลังคำว่า do

ตัวส่วนวนซ้ำ

หมายถึง ข้อความสั่งประกอบ ปิดล้อมด้วย begin และ end {while} ประกอบด้วยคำสั่งต่างๆ ซึ่งอ่านข้อมูลเงินเดือนของพนักงานหนึ่งคนและคำนวณ แสดงผลค่าจ้างรวมของพนักงานหลังจากแสดงผล ปริมาณเงินค่าจ้างรายสัปดาห์ ข้อความสั่งตามหลัง ตัวส่วนวนซ้ำ กระทำการ และแสดงผลข้อความ All employee processed

ตัวส่วนวนซ้ำ หมายถึง คำสั่งต่างๆ ซึ่งทำซ้ำๆ กันในการวนซ้ำ (Loop body is the instructions that are repeated in the loop.)

```
CountEmp := 10 ; {no employees processed yet}
While CountEmp < 7 do {test value of CountEmp}
  begin
    Write (' Hours > ') ;
    ReasLn (Hours) ;
    Write (' Rate > $ ') ;
    ReadLn (Rate) ;
    Pay := Hours * Rate ;
    WriteLn ('Weekly pay is $ ' , Pay : 4 : 2) ;
    CountEmp := CountEmp + 1 {increment CountEmp}
  end {while}
WriteLn ('All employees processed') ;
```

Loop body

รูป 5.1 การวนซ้ำเพื่อประมวลผลพนักงาน 7 คน

ในรูป 5.1 มีสามคำสั่ง ซึ่งควบคุมการประมวลผลการวนซ้ำ ข้อความสั่งแรกได้แก่

CountEmp := 0 ; {no employees processed yet}

เก็บค่าเริ่มต้น 0 ในตัวแปร CountEmp ซึ่งแทนการนับจำนวนพนักงานซึ่งได้มีการประมวลผลบรรทัดถัดไป ประเมินผลนิพจน์แบบบูล CountEmp < 7 ถ้าเป็นจริง ตัวส่วนวนซ้ำ ถูกกระทำการ ทำให้โปรแกรม อ่านคู่ใหม่ของค่าข้อมูล คำนวณ และแสดงผลปริมาณเงินเดือนของคนใหม่ คำสั่งสุดท้ายในตัวส่วนวนซ้ำ

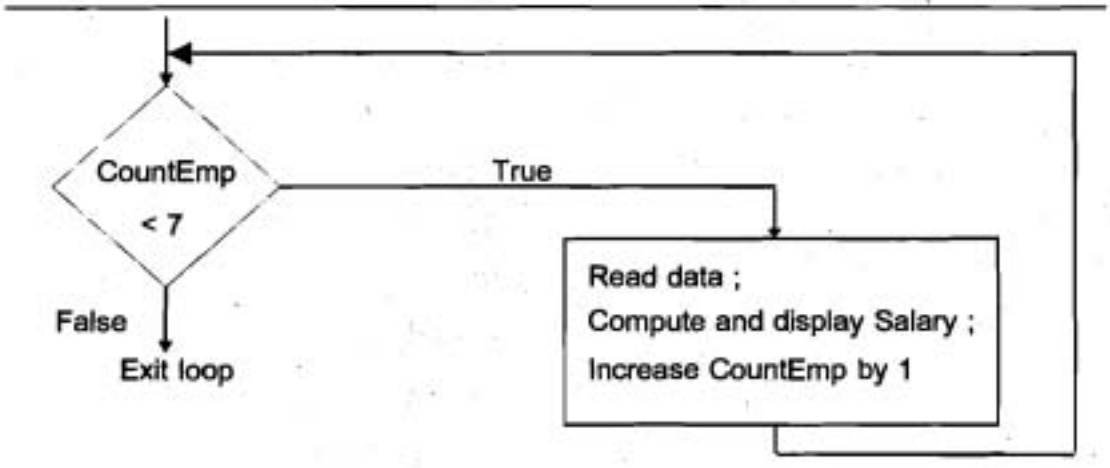
CountEmp := CountEmp + 1 {increment CountEmp}

บวก 1 กับค่าของ CountEmp หลังจากคำสั่งสุดท้ายในตัวส่วนวนซ้ำถูกกระทำการแล้ว การควบคุมกลับคืนไปยังบรรทัดเริ่มต้นที่มี while และนิพจน์แบบบูล CountEmp < 7 ถูกประเมินผลอีกครั้งหนึ่ง สำหรับค่าถัดไปของ CountEmp

ตัวส่วนวนซ้ำถูกกระทำอีกครั้งสำหรับค่าแต่ละตัวของ CountEmp จาก 0 ถึง 6 สุดท้ายเมื่อ CountEmp เท่ากับ 7 ซึ่งครั้งนี้นิพจน์แบบบูล ประเมินผลเป็นเท็จ ที่จุดนี้ออกจากส่วนวนซ้ำและการควบคุมส่งไปข้อความสั่ง WriteLn ตามหลังตัวส่วนวนซ้ำ

นิพจน์แบบบูล หลังคำสงวน while เรียกว่า **เงื่อนไขทำซ้ำของการวนซ้ำ** (loop – repetition condition) ซึ่งจะถูกประเมินผลก่อนการทำซ้ำแต่ละครั้งของตัวส่วนวนซ้ำ

ตัวส่วนวนซ้ำถูกทำซ้ำ เมื่อเงื่อนไขนี้เป็นจริง และออกไป (exit) เมื่อเงื่อนไขนี้เป็นเท็จ ผังงานของ while loop ในรูป 5.2 แสดงให้เห็นการกระทำการนี้ เงื่อนไขในกล่องรูปข้าวหลามตัด ถูกประเมินผลเป็นอันดับแรก ถ้าเป็นจริง ตัวส่วนวนซ้ำ ถูกกระทำการ



รูป 5.2 ผังงานของ while Loop

และประมวลผลจนกระทั่งเงื่อนไขเป็นเท็จ ณ จุดนี้จึงออกจาก while loop

เพื่อดูว่าเราเข้าใจ ความแตกต่างระหว่าง ข้อความสั่ง while ในรูป 5.1 กับข้อความสั่ง if

```
if  CountEmp < 7 then
  begin
    ...
  end ; {if}
```

ข้อความสั่งประกอบ หลังคำสงวน then กระทำการอย่างมากที่สุดหนึ่งครั้ง ส่วนในข้อความสั่ง while ข้อความสั่งประกอบหลังคำสงวน do อาจกระทำการมากกว่าหนึ่งครั้ง

ตัวแปร CountEmp ในรูป 5.1 เรียกว่าตัวแปรควบคุมการวนซ้ำ (loop – control variable) เพราะว่าค่าของมันกำหนดว่า ตัวส่วนวนซ้ำ จะทำซ้ำหรือไม่

ตัวแปรควบคุมการวนซ้ำ CountEmp ต้องถูกกำหนดค่าเริ่มต้น ทดสอบ และปรับให้เป็นปัจจุบันสำหรับส่วนวนซ้ำที่จะกระทำการอย่างถูกต้อง

Initialization CountEmp กำหนดค่าเริ่มต้นเป็น 0 (กำหนดค่าเริ่มต้นให้เป็น 0) ก่อนถึงข้อความสั่ง while

Testing CountEmp ถูกทดสอบก่อนเริ่มต้นกระทำซ้ำของแต่ละ loop (เรียกว่า การวนซ้ำ (iteration) หรือ pass)

Updating CountEmp ถูกปรับ (ค่าของมันเพิ่มขึ้น 1) ระหว่างการวนซ้ำแต่ละครั้ง

เงื่อนไขทำซ้ำลูป หมายถึงเงื่อนไขตามหลัง while ซึ่งควบคุมการทำซ้ำของลูป (Loop – repetition condition is the condition following while that controls loop repetition.)

ตัวแปรควบคุมลูป หมายถึงตัวแปรซึ่งค่าของมันควบคุมการทำซ้ำของลูป (Loop – control variable is the variable whose value controls loop repetition.)

ขั้นตอนที่คล้ายกันต้องถูกกระทำสำหรับทุกครั้งของ while ลูปถ้าไม่มีการเริ่มต้น การทดสอบเริ่มต้นของ CountEmp จะไม่มีความหมายใดๆ ขั้นการปรับต้องมั่นใจว่าโปรแกรมจะก้าวหน้าไปยังเป้าหมายสุดท้าย (CountEmp > = 7) ระหว่างการทำซ้ำของลูป ถ้าตัวแปรควบคุมลูปไม่มีการปรับค่า ลูปจะกระทำ "ตลอดไป" ลูปเช่นนี้เรียกว่า **ลูปอนันต์** (infinite loop) while ลูป ในรูป 5.1 เรียกว่า ลูปการนับ (coumting loop) หรือ **ลูปควบคุมโดยตัวนับ**

(Counter – controlled loop) เพราะว่าการทำซ้ำของมัน ถูกควบคุมโดยตัวแปร ซึ่งค่าของมันแทนจำนวนของการทำซ้ำลูป ซึ่งจะกระทำต่อไป เราใช้ลูปการนับเมื่อเราสามารถกำหนดจำนวนครั้งก่อนการกระทำลูป อย่างชัดเจน การทำซ้ำลูปจะมีจำนวนที่ครั้งเป็นสิ่งสำคัญในการแก้ปัญหา เงื่อนไข while ต่อไปจะเปรียบเทียบกับตัวนับ (counter) ของจำนวนการทำซ้ำของลูปต่อไป เราอธิบายวากยสัมพันธ์ของข้อความสั่ง while และแผนภาพแสดงข้อความสั่งในรูป 5.3

Syntax Display

The while Statement

Form :

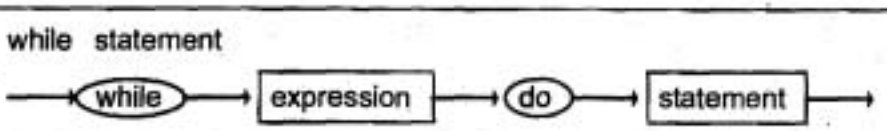
```
while expression do
   statement
```

ตัวอย่าง

```
{Display N asterisks.}
CountStar := 0 ;
While CounStar < N do
   begin
      Write (' * ') ;
      CountStar := CounStar + 1
   end {while}
```

มีความหมายดังนี้ นิพจน์ (เงื่อนไขการทำซ้ำลูป) ถูกทดสอบถ้าเป็นจริง statement (ตัวส่วนวนซ้ำ) ถูกกระทำการและนิพจน์ถูกทดสอบใหม่ statement ถูกทำซ้ำดราบใดที่นิพจน์เป็นจริง เมื่อ expression ถูกทดสอบ และพบว่าเป็นเท็จออกจากลูป และข้อความสั่งโปรแกรมถัดไป หลังข้อความสั่ง while จะถูกกระทำการ

ข้อสังเกต ถ้า expression ประเมินผลเป็นเท็จในครั้งแรกที่ทดสอบ statement จะไม่ถูกกระทำการ



รูป 5.3 แผนภาพวากยสัมพันธ์ สำหรับข้อความสั่ง while

ตำแหน่ง semicolon ในข้อความสั่ง while ให้ระมัดระวัง ไม่ใส่ semicolon หลังคำว่า do เพราะคอมไพเลอร์ Pascal จะเข้าใจว่าข้อความสั่งว่าง (empty statement) เป็นตัวส่วนวนซ้ำ ซึ่ง "statement" จะไม่ทำสิ่งใดตลอดไป เป็น infinite loop

สไตล์ของโปรแกรม (Program Style)

การจัดรูปแบบข้อความสั่ง while (Formatting the while Statement) เพื่อทำให้ชัดเจน ให้ย่อหน้า body ของ while ลูปถ้าตัวส่วนวนซ้ำ (body ของ while ลูป) เป็นข้อความสั่งประกอบให้ปิดล้อมด้วยคู่ begin – end {while}

แบบฝึกหัด 5.1 Self –Check

1. ตัวส่วนวนซ้ำ (loop body) ข้างล่างนี้ทำซ้ำกี่ครั้ง แต่ละครั้งของการทำซ้ำของตัวส่วนวนซ้ำ พิมพ์อะไร และหลังออกจากลูปแล้ว พิมพ์อะไร

```
program Assingment5 ;
  var X, Count : Integer ;
    X := 3 ;
    Count := 0 ;
    while Count < 3 do
      begin
        X := X * X ;
        WriteLn (X) ;
        Count := Count + 1
      end ; {while}
  WriteLn {Count}
end.
```

2. จงตอบคำถามแบบฝึกหัดข้อ 1 สมมติว่าข้อความสั่งสุดท้ายในตัวส่วนวนซ้ำ คือ

   Count := Count + 2

3. จงตอบคำถามแบบฝึกหัดข้อ 1 โดยเอาข้อความสั่งสุดท้ายในตัวส่วนวนซ้ำออกไป

เขียนโปรแกรม

1. จงเขียน while ลูป ซึ่งแสดงผลเลขจำนวนเต็มแต่ละตัวจาก 1 ถึง 10 บนบรรทัดแยกจากกัน กับกำลังสองของมัน

2. จงเขียน while ลูป ซึ่งแสดงผล เลขจำนวนเต็ม แต่ละตัวจาก 4 ลดลงไปจนถึง –6 บนบรรทัด แยกจากกัน แสดงผลค่าในลำดับ 4, 2, 0 เรื่อยไป

5.2 การใช้ลูปเพื่อสะสมผลบวก (Using Loops to Accumulate a Sum)

ลูปบ่อยครั้งสะสมผลบวกโดยการทำซ้ำการดำเนินการบวกดังแสดงให้เห็นในตัวอย่าง 5.1

ตัวอย่าง 5.1

โปรแกรมในรูป 5.4 มี while ลูปคล้ายกับลูปในรูป 5.1 ยกเว้นการแสดงผล ค่าจ้างรายสัปดาห์ของพนักงานแต่ละคนมีการคำนวณ และแสดงผล เงินเดือนรวมของบริษัท (Total Pay) ก่อนการกระทำลูป ข้อความสั่ง

```
TotalPay := 0. 0 ;
CountEmp := 0 ; {Start with first employee}
```

กำหนดค่าเริ่มต้นให้ TotalPay และ CountEmp ทั้งคู่ให้เท่ากับ 0 เมื่อ CountEmp หมายถึงตัวแปรนับจำนวนและ TotalPay หมายถึงตัวแปรสะสม (ตัวสะสมค่าเงินเดือนทั้งหมด) การเริ่มต้นให้ TotalPay มีค่าเท่ากับ 0 มีความสำคัญ ถ้าไม่มีขั้นตอนนี้ จะทำให้ผลรวมทั้งหมดสุดท้ายไม่มีค่าอะไร เก็บใน TotalPay เมื่อโปรแกรมเริ่มการกระทำการ

ตัวสะสม หมายถึง ตัวแปร ซึ่งใช้เก็บค่าที่กำลังคำนวณในการเพิ่มค่าระหว่างการทำการลูป (Accumulator is a variable used to store a value being computed in increments during the execution of a loop.)

ในตัวส่วนวนซ้ำ ข้อความสั่ง

```
Total Pay := TotalPay + Pay ; {Add next pay.}
```

บวกค่าปัจจุบันของ Pay กับผลบวกสะสมใน TotalPay ด้วยเหตุนี้ จึงเป็นการเพิ่มค่าของ TotalPay ด้วยการวนซ้ำลูปแต่ละครั้ง รูป 5.5 ตามรอยผลของการทำซ้ำข้อความสั่งนี้ สำหรับค่าสามค่าของ Pay ซึ่งแสดงให้เห็นในการวิ่งตัวอย่าง โปรดจำไว้ว่า การวนซ้ำหมายถึง การผ่านตลอดลูป (Recall that iteration means a pass through the loop.)

Edit Window

```
program payroll ;
{Computes the payroll for a company}
  var
    NumberEmp,          {number of employees}
    CountEmp : Integer, {current cmployee}
    Hours,              {hours worked}
    Rate,               {hourly rate}
    Pay,                {weekly pay}
    TotalPay : Real ;   {company payroll}
begin {Payroll}
  {Enter number of employees.}
  Write ('Enter number of employus > ') ;
  ReadLn (NumberEmp) ;
  {Compute each employee ' s pay and add it to the payroll.}
  TotalPay := 0.0 ;
  CountEmp := 0 ;    {Start with first employee.}
  while CountEmp < NumberEmp do
    begin
      Write (' Hours > ') ;
      ReadLn (Hours) ;
      Write ('Rate > $ ') ;
      ReadLn (Rate) ;
```

```
            Pay := Hours * Rate ;
            WriteLn ('Pay is $ ', Pay : 4 :2) ;
            WriteLn;
            TotalPay := TotalPay + Pay ; {Add next pay.}
            CountEmp := CountEmp + 1
        end ; {while}
    WriteLn (' All employees processed') ;
    WriteLn ('Total payroll is $ ' , Total Pay : 4 : 2)
end . {Payroll}
```

Out put Window

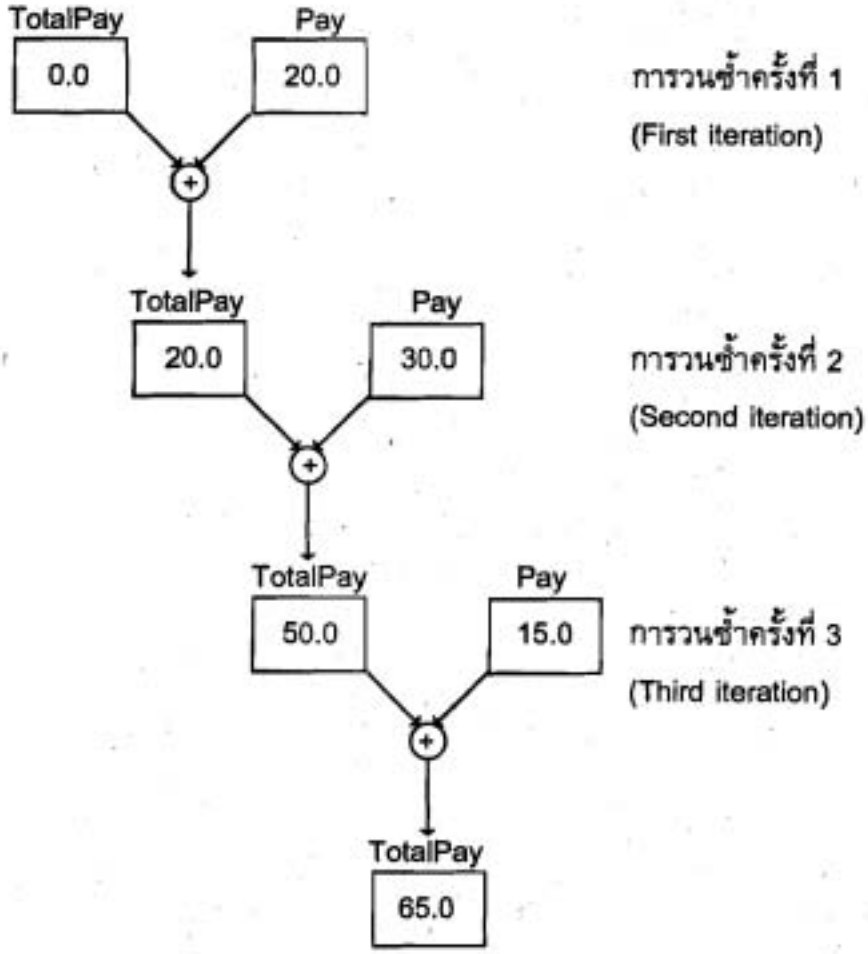
```
Enter number of employees > 3
Hours > 5
Rate > $ 4.00
Pay is $ 20.00

Hours > 6
Rate > $ 5.00
Pay is $ 30.00

Hours > 1.5
Rate > $ 10.00
Pay is $ 15.00

All employees processed
Total payroll is $ 65.00
```

รูป 5.4 โปรแกรมคำนวณเงินเดือน



รูป 5.5 การสะสมผลบวก

สไตล์ของโปรแกรม (Program Style)

การเขียนลูปทั่วไป (Writing General Loops)

เนื่องจากลูปในรูป 5.1 ใช้เงื่อนไขการทำซ้ำลูป CountEmp < 7 มันจึงประมวลผลพนักงาน 7 คนเท่านั้น ลูปทั่วไปมากกว่าในรูป 5.4 ใช้เงื่อนไขการทำซ้ำลูป CountEmp < NumberEmp มันจึงประมวลผลเท่ากับจำนวนพนักงาน จำนวนพนักงานซึ่งจะถูกประมวลผลในลูปชุดหลังนี้ จึงต้องอ่านไว้ในตัวแปร NumberEmp ก่อนกระทำการข้อความสั่ง while

แบบฝึกหัด 5.2 Self – Check

1. จงบอกค่าเอาต์พุต ซึ่งจะถูกแสดงผลโดย while ลูปข้างล่างนี้ เมื่อข้อมูลมีค่าเท่ากับ 5

```
WriteLn ('Enter an integer > ') ;
ReadLn (x) ;
Product := 1 ;
Count := 0 ;
while Count < 4 do
   begin
     WriteLn (Product) ;
     Product := Product * X ;
     Count := Count + 1
   end ; {while}
```

2. ถ้าการเรียก WriteLn ย้ายมาตอนล่างสุดของลูปแทนที่จะเป็นตอนเริ่มต้น loop จงบอกค่าต่าง ๆ ที่แสดงผล

3. การดำเนินการคณิตศาสตร์อะไร ซึ่งกระทำส่วนของการคำนวณข้างล่างนี้

```
Write ('Enter X > ') ;
ReadLn (X) ;
Write ('Enter y > ') ;
ReadLn (y) ;
Product := 1 ;
while y > 0 do
   begin
      Product := Product * X ;
      Y := y – 1
   end ; {while}
WriteLn ('Result = ' , Product)
```

4. จงดัดแปร (modify) โปรแกรมในรูป 5.4 เพื่อให้แสดงผลค่าเฉลี่ยเงินเดือนของพนักงาน นอกเหนือจากปริมาณเงินเดือนทั้งหมด

เขียนโปรแกรม

1. เมื่อลูกคนใหม่ของ Robin เกิดเธอเปิดบัญชีเงินฝากออมทรัพย์ด้วยเงิน $1000.00 และทุกๆ วันเกิดของลูก เริ่มต้นจากปีแรกธนาคาร บวกดอกเบี้ยเพิ่ม 4.5% ของเงินในบัญชี และ Robin เพิ่มอีก $ 500.00 ในบัญชี จงเขียนลูปเพื่อคำนวณว่าเมื่อลูกของเธออายุครบ 18 ปี จะมีเงินในบัญชีเท่าใด

5.3 ลูปชนิดควบคุมโดยเหตุการณ์ (Event – Controlled Loops)

โปรแกรมเมอร์ใช้ลูปสองชนิด : ลูปควบคุมโดยตัวนับ (counter – controlled loops) และลูปควบคุมโดยเหตุการณ์ (event – controlled loop (หรือ conditional loop)) ในชนิดแรกนั้น ตัวส่วนวนซ้ำทำซ้ำๆ กัน ตามจำนวนครั้งที่กำหนด ส่วนชนิดที่สอง การทำซ้ำลูปหยุดเมื่อมีเหตุการณ์ (event) อย่างหนึ่งเกิดขึ้น

ตัวอย่างเช่น มีใบเสร็จเก็บเงิน (bill) อยู่กองหนึ่งซึ่งเราจะต้องเขียนเช็คเงินสดรายเดือน เราไม่ทราบว่าจะต้องเขียนเช็คจำนวนกี่ใบ แต่อย่างไรก็ตาม ถ้าเราต้องการจ่ายเงินให้มากเท่ากับใบเสร็จเก็บเงิน เท่าที่เป็นไปได้ เราอาจเขียนเรียงตามอันดับจำนวนเงิน (ขั้นแรกคือใบเก็บเงินที่มีจำนวนเงินน้อยที่สุด) และหยุดเขียนเมื่อมีเหตุการณ์ เงินเกินตัวเลขในบัญชี พูดอีกอย่างหนึ่งคือ เราต้องเขียนเช็คไปเรื่อยๆ ตราบใดที่เงินยังไม่เกินเงินในบัญชี ซึ่งแสดงด้วยรหัสเทียบข้างล่างนี้

```
while account is not overdrawn do
   begin
      Read the next bill
      Pay the bill if there are sufficient funds
      Update the balance
   end
```

จำนวนที่แท้จริงของการทำซ้ำลูปขึ้นอยู่กับปริมาณของเงินในบัญชีตอนเริ่มต้น และจำนวนเงินที่เป็นหนี้บนใบเสร็จเก็บเงินแต่ละใบ

ตัวอย่าง 5.2

โปรแกรมในรูป 5.6 ใช้อัลกอริทึม ข้างต้นเพื่อจ่ายเงินให้ใบเสร็จเก็บเงิน สมมติว่าจำนวนเงินที่เป็นหนี้ทั้งหมดมากกว่าเงินในบัญชีตอนเริ่มต้น while ลูปอ่านใบเสร็จเก็บเงิน

แต่ละใบ ประมวลผลและข้อความสั่ง if ซ้อนใน ซึ่งอยู่ภายในลูป ตรวจดูว่าจะจ่ายเงินให้แก่ใบเสร็จได้หรือไม่ โดยการเปรียบเทียบเงินในบัญชีปัจจุบันกับจำนวนเงินในใบเสร็จถ้า Balance > Bill เป็นจริง ใบเสร็จนั้นจ่ายเงินได้กรณีอื่นๆ ไม่สามารถจ่ายเงินได้และค่าบวกสุดท้ายของเงินในบัญชีจะถูกแสดงผล

Edit Window

```
program PayBills ;
{
Authorizes payment of each bill as long as there are suffiuent funds in the
checking account. Assumes bills are entered in order starting with the
smallest and the total amount owed exceeds the initial balance.
}
var
   Creditor : String ;    {input – name of creditor}
   Bill,                  {input – amount of bill}
   InitBal,               {input – starting balance}
   Balance : Real ;       {current account balance}
begin {PayBills}
   Write ('Enter initial account balance > $ ') ;
   ReadLn (IniBal) ;
   {
   Pay each bill as long as the account is not overdrawn. Decrease the
   balance by the bill amount after each bill is processed.
   }
   Balance := InitBal ;
   While Balance > = 0.0 do
      begin
         WriteLn;          {Skip a line}
```

```
            Write ('Enter next creditor > ') ;
            ReadLn (Creditor) ;
            Write ('Enter amount owed > $ ') ;
            ReadLn (Bill) ;
            if Balance > = Bill then
               WriteLn ('Issue cheek for $' , Bill : 3 : 2 "to ' , Creditor)
            else
            WriteLn ('No checked issued - -' , 'Account balance is only $' ,
                                                Balance : 3 :2);
            Balance := Balance – Bill {Update balance.}
        end ; {while}
    WriteLn ('Insufficient funds to pay any more bills !)'
end.  {PayBills}
```

Output Window

```
Enter initial account balance > $ 120.00

Enter next creditor > Sam Jones
Enter amount owed > $ 65.00
Issue check for $ 65.00 to Sam Jones

Enter next creditor > Shirley Valentine
Enter amount owed > $ 70.00
No Check issued - - Account balance is only $ 55.00
Insuflicient funds to pay any more bills !
```

รูป 5.6 การจ่ายเงินใบเสร็จเก็บเงินรายเดือน

ตาราง 5.1 ตามรอยการกระทำการของโปรแกรมสำหรับข้อมูล ซึ่งแสดงให้เห็นในการวิ่งตัวอย่าง เพื่อให้รวบรัด เราจึงไม่ตามรอยตัวแปรชนิด String ชื่อ Creditor ระหว่างการวนซ้ำลูป ครั้งที่หนึ่ง ข้อความสั่ง if แสดงผลคำสั่งสำหรับการเขียน เช็คระหว่างการวนซ้ำครั้งที่สอง ข้อความสั่ง if แสดงผล ตัวเลขในบัญชีเป็นบวกครั้งสุดท้าย การกระทำการถัดไปของข้อความสั่ง

Balance := Balance – Bill {Update balance.}

กำหนดค่าลบให้ Balance ดังนั้นลูปจะต้องออกเมื่อเงื่อนไขทำซ้ำของมันถูกทดสอบใหม่

ตาราง 5.1 ตามรอยโปรแกรมในรูป 5.1

Statement	InitBal	Balance	Bill	Effect
ReadLn (Init Bal)	120.00	?	?	Enter Start balance
Balance := Init Bal		120.00		Initialize Balance
while Balance > = 0.0 do				120.00> = 0.0 Is true
ReadLn (Bill);			65.0	Enter first bill
If Balance > = Bill				120.00> = 65.0
then				is true
WriteLn (' Issue Check				Pay the bill
...				
Balance := Balance – Bill		55.00		Reduce Balance
while Balance > = 0.0 do				55.00 > = 0.0 is true
ReadLn (Bill)			70.00	Enter second bill
If Balance > = Bill				55.00 > = 70.00
then				is false
WriteLn ('No Check...				Display Balance
Balance := Balance – Bill		-15.00		Reduce Balance
while Balance > = 0.0 do				-15.00 >= 0.0 is false, exit loop
WriteLn ('Insufficient...				Display final Message

คล้ายกับตัวนับ (counter) ในลูปการนับ ตัวแปรควบคุมลูป Balance สำหรับลูปชนิดควบคุมโดยเหตุการณ์ต้อง initialized, tested และ updated สำหรับลูปให้กระทำการอย่างถูกต้อง

การเริ่มต้น Balance มีค่าเริ่มต้นเป็น InitBal ก่อนถึงลูป

การทดสอบ Balance ถูกทดสอบก่อนการกระทำการแต่ละครั้งของตัวส่วนวนซ้ำ (loop body)

การปรับ Balance ถูกปรับให้เป็นปัจจุบัน (ลดลงด้วยค่าของ Bill) ระหว่างการวนซ้ำแต่ละครั้ง

โปรดจำไว้ว่าเราต้องใส่ขั้นตอนต่างๆ คล้ายกับที่กล่าวข้างต้นในทุกลูปที่เราเขียน

การตรวจสอบลูปการวนซ้ำศูนย์ครั้ง (Checking Zero – Iteration Loops)

ถึงแม้เราจะคาดว่า ตัวส่วนซ้ำจะทำซ้ำในบางกรณี ตัวส่วนวนซ้ำไม่กระทำการเลย (เรียกว่า ลูปการวนซ้ำศูนย์ครั้ง) ตัวส่วนวนซ้ำจะไม่กระทำการถ้าเงื่อนไขการทำซ้ำลูปประเมินผลเป็นเท็จตั้งแต่ทดสอบครั้งแรก เพราะว่าขั้นตอนการเริ่มต้นลูป ต้องกระทำเสมอ ถึงแม้จะเป็นลูปการวนซ้ำศูนย์ครั้งก็ตาม ต้องเขียนเสมอเพื่อให้มั่นใจว่า โปรแกรมที่เป็นลูปการวนซ้ำศูนย์ครั้งสร้างผลลัพธ์ที่มีความหมาย

โปรแกรมจ่ายเงินใบเสร็จเก็บเงิน ของรูป 5.6 มีลูปชนิดการวนซ้ำศูนย์ครั้ง เมื่อเงินในบัญชีเริ่มต้นเป็นค่าลบ ถ้า InitBal คือ -120.0 ขั้นการเริ่มต้นลูปจะกำหนดค่า -120.00 ให้ Balance ลูปจะถูกข้าม (Skipped) และข้อความสิ่งต่างๆ ซึ่งอยู่ตามหลังลูปจะถูกกระทำการ เอาต์พุตโปรแกรมเป็นดังนี้

```
Enter initial account balance > $ - 120.00
Insufficient funds to pay any more bills !
```

แบบฝึกหัด 5.3 Self – Check

1. จำนวนครั้งน้อยที่สุดซึ่งตัว body ของ while ลูปจะถูกกระทำการเท่ากับเท่าไหร่
2. เอาต์พุตของ segment ข้างล่างนี้จะเกิดข้อผิดพลาดเมื่อใด และจะแก้ไขอย่างไร

```
Total := 0 ;
Write (' Enter number of items > ') ;
ReadLn (Num) ;
```

```
Count := 0 ;
while Count < Num do
   begin
      Write ('Enter a value > ') ;
      ReadLn (Value) ;
      Last := Value
   end; {while}
   WriteLn ('The last value entered was ', Last)
```

3. จงตามรอยโปรแกรมในรูป 5.6 สำหรับข้อมูลต่อไปนี้ :

   150.0, 75.00, 50.00, 25.00, 30.00

4. a) เราจะดัดแปร ลูปในรูป 5.6 อย่างไรเพื่อให้มันนับจำนวนใบเสร็จเก็บเงิน (CountBills)

   b) เราจะดัดแปรลูป ในรูป 5.6 อย่างไร ถ้ามันเป็นไปได้ ที่มีการจ่ายเงินสำหรับใบเสร็จเก็บเงินทั้งหมด ก่อนที่เงินในบัญชีจะไม่พอจ่าย สมมติว่าจำนวนใบเสร็จเก็บเงิน (NumberBills) เช่นเดียวกับเงินในบัญชีเริ่มต้นถูกจัดเป็นข้อมูล

   ข้อแนะนำ ใช้เงื่อนไขการทำซ้ำลูป ดังนี้

   (Balance > = 0.0) and (Count Bills < NumberBills)

เขียนโปรแกรม

1. ในเมืองหนึ่งซึ่งมีประชากร 9,870 คน ประชากรเพิ่มขึ้นปีละ 10% จงเขียนลูปเพื่อคำนวณว่าจะเป็นเวลานานกี่ปี (CountYears) ถ้าจะให้มีจำนวนประชากรเกินกว่า 30,000 คน

5.4 การออกแบบส่วนวนซ้ำ (Loop Design)

สิ่งหนึ่งเพื่อวิเคราะห์การดำเนินการของลูปและอีกสิ่งหนึ่งเพื่อออกแบบ ส่วนวนซ้ำของเราเอง การออกแบบส่วนวนซ้ำมีสองวิธี วิธีที่หนึ่ง วิเคราะห์ความต้องการของลูปใหม่เพื่อกำหนดการเริ่มต้นความต้องการ การทดสอบ การปรับของตัวแปรควบคุมลูป วิธีที่สอง พัฒนาแผ่นแบบ (templates) สำหรับการเวียนเกิดของรูปแบบลูป ซึ่งใช้บ่อยๆ และใช้แผ่นแบบเป็นฐานหลัก (basis) สำหรับลูปใหม่

การวิเคราะห์ความต้องการลูป (Analyzing Loop Requirements)

เพื่อให้มีความเข้าใจเรื่องการออกแบบของลูปที่จำเป็นโปรแกรมการจ่ายเงินใบเสร็จให้ศึกษาคอมเมนต์ ในรูป 5.6 ซึ่งเป็นข้อสรุปเป้าหมายของลูป

{

Pay each bill as long as the account is not overdrawn. Decrease the balance by the bill amount after each bill is processed.

{

เพื่อให้เป้าหมายเหล่านี้ประสบผลสำเร็จ เราต้องเน้นที่การควบคุมลูป และการประมวลผลลูป เน้นที่การควบคุมลูป หมายถึง ทำให้มั่นใจว่า ออกจากลูปเกิดขึ้นเมื่อควรต้องออก เน้นการประมวลผลลูป หมายความว่าทำให้มั่นใจว่าส่วนการวนซ้ำ กระทำการดำเนินการที่ต้องการ

การวางเกณฑ์ขั้นตอนที่จำเป็นการควบคุมลูป และการประมวลผลลูป เราเริ่มต้นด้วยรายการที่เราทราบเกี่ยวกับลูป ในตัวอย่างนี้ ถ้า Balance คือ ตัวแปรควบคุมลูป เรามีข้อสังเกต 4 ข้อ ดังนี้

1. Balance ต้องเท่ากับ InitBill ก่อนเริ่มต้นลูป
2. เราจ่ายเงินให้ใบเสร็จปัจจุบัน ถ้าเงินในบัญชีมีเพียงพอ
3. Balance ระหว่าง next pass ต้องน้อยกว่า Balance ระหว่าง current pass ด้วยปริมาณเงินของ current bill
4. เราหยุดการจ่ายเงินให้ใบเสร็จ เมื่อ Balance มีค่าเป็นลบ

ความต้องการข้อ 1 บอกเราว่า การเริ่มต้นชุดใด ต้องกระทำ

ความต้องการข้อ 2 บอกว่า เมื่อใดให้จ่ายเงินแก่ใบเสร็จปัจจุบัน (ถ้า Balance > = Bill เป็นจริง)

ความต้องการข้อ 3 บอกว่าการปรับ Balance ภายในตัวส่วนวนซ้ำ (Balance := Balance – Bill) ทำอย่างไร

ความต้องการข้อ 4 บอกว่าลูปควรต้องออกไป เมื่อใบเสร็จใบแรกทำให้ Balance ค่าลบถูกประมวลผล ความต้องการเหล่านี้ ประกอบกันเป็นฐานหลัก สำหรับการออกแบบอัลกอริทึม ต่อไปนี้

เงื่อนไขการทำซ้ำลูป Balance > = 0.0 คือสิ่งตรงกันข้ามกับเงื่อนไขการออกจากลูป Balance < 0.0

1. Initialize Balance to InitBal

2. while Balance > = 0.0 do

    begin

    3. Read the data for the currrent bill

    4. Display the check –writing information If the bill can be paid

    5. Balance := Balance – Bill

  end {while}

วิธีร่วมอีกหนึ่งอย่างเพื่อสร้างเอาต์พุตจากลูป คือ การจัดเรียงมันในตาราง ในตัวอย่างถัดไป เราวิเคราะห์ความต้องการลูป เพื่อให้ได้ตารางของค่างต่างๆ

ตัวอย่าง 5.3 การออกแบบลูป เพื่อแสดงผลตารางของค่าต่างๆ

จงเขียนโปรแกรมแสดงผลของแรงดึงดูดของโลกบนวัตถุ ซึ่งกำลังตกอย่างอิสระ (free – falling object) ผลลัพธ์เป็นรายการในตาราง ซึ่งแสดงความสูงของวัตถุตกจากหอคอยสำหรับทุกๆ วินาทีที่มันกำลังจะตก

สมมติว่า t คือเวลาของการตกอย่างอิสระ เราทำข้อสังเกตเกี่ยวกับความสูงของวัตถุซึ่งตกจากหอคอย ดังนี้

1. ที่ t = 0.0 ความสูงของวัตถุเท่ากับความสูงของหอคอย

2. ขณะที่วัตถุกำลังตก ความสูงของมัน คือ ความสูงของหอคอย ลบระยะทางที่วัตถุเดินทาง

3. การตกอย่างอิสระ จบเมื่อความสูงของวัตถุน้อยกว่า หรือเท่ากับ 0.0

ความต้องการเหล่านี้ กำหนดการออกแบบของ while ลูป ซึ่งแสดงให้เห็นในรูป 5.7 ความสูงของวัตถุ ใช้ชื่อ Height เป็นค่าเริ่มต้นให้กับความสูงของหอคอย ชื่อ Tower (จากความต้องการข้อ 1) เงื่อนไข while

  Height > 0.0

ทำให้มั่นใจว่า การออกจากลูป เกิดขึ้นเมื่อวัตถุกระทบพื้น (จากความต้องการข้อ 3) ภายในตัวส่วนวนซ้ำ ข้อความสั่ง กำหนดค่า

  Height := Tower – 0.5 * G * Sqr (T)

คำนวณ ความสูงของวัตถุ (จากความต้องการ ข้อ 2)

เมื่อระยะทางเดินทาง ถูกแทนด้วยสูตร

$$\text{distance} = {}^{1}/_{2}\, gt^2$$

.เมื่อ g เป็นค่าคงตัวเชิงแรงดึงดูดของโลก (gravitional constant) จำนวนของการวนซ้ำลูป (loop iterations) ขึ้นอยู่กับช่วงเวลาระหว่างการวนซ้ำ (DeltaT) และความสูงของหอคอย (Tower) ทั้งคู่เป็นค่าของข้อมูล ระหว่างการวนซ้ำแต่ละครั้ง ช่วงผ่านปัจจุบัน (current elapsed time), T, และความสูงวัตถุปัจจุบัน (current height), Height จะแสดงผลบนบรรทัดใหม่ ในตารางและค่าใหม่ถูกกำหนดให้ตัวแปรเหล่านี้ ข้อความตามหลังตารางแสดงเมื่อวัตถุกระทบพื้น

Edit Window

```
program FreeFall ;
{
   Displays the height of an object dropped from a tower until it hits the
   ground
{
   const
      G = 9.80665 ;        {gravitaltional constant for metric units}

   Var
      Height, {height of object}
      Tower,               {height of tower}
      T,                   {elapsed time}
      Detat : Real ;       {time interval}

   begin {FreeFall}
      {Enter tower height and time interval.}
```

```
    Write ('Tower height in meters > ') ;
    ReadLn (Tower) ;
    Write ('Time in seconds between table lines >') ;
    ReadLn (DeltaT) ;

    {Diplay object height until it hits the ground.}
    WriteLn ;
    WriteLn ('Time' : 10, ' Height ' : 10) ;
    T := 0.0 ;
    Height := Tower ;
    while Height > 0.0 do
      begin
        WriteLn (T : 10 :2, Height : 10 :2);
        T := T + DeltaT ;
        Height := Tower – 0.5 * G * Sqr (T)
      end; {while}

    {Object hits the ground.}
    WriteLn;
    WriteLn ('SPLATT !!!')
  end. {FreeFall}
```

Output Window

```
Tower height in meter > 100.0
Time in seconds between table lines > 1.0
   Time     Height
   0.00     100.00
   1.00     95.10
   2.00     80.39
```

```
   3.00     55.87
   4.00     21.55
SPLATT ! ! !
```

รูป 5.7 การตกของวัตถุจากหอคอย

สไตล์ของโปรแกรม (Program Style)

โปรแกรมแสดงผลตาราง (Displaying a Table) ในรูป 5.7 แสดงตารางของค่าเอาต์พุต ก่อนถึงลูป

ข้อความสั่ง

```
WriteLn ('Time' : 10 , ' Height' : 10) ;
```

แสดงสายอักขระสองชุด ซึ่งปรากฏเป็นหัวตาราง เนื่องจากสายอักขระพิมพ์แบบจัดขวา (right – jnstified) ในเขตข้อมูลของมัน (ดูตาราง 2.13) อักขระตัวขวามือสุดของสายอักขระชุดแรกปรากฏที่สดมภ์ที่ 10 และอักษรตัวขวามือสุดของสายอักขระชุดที่สองปรากฏที่สดมภ์ที่ 20 (10 + 10)

ภายในตัวส่วนวนซ้ำ ข้อความสั่ง

```
WriteLn (T : 10 2 , Height : 10 : 2)
```

แสดงคู่ของค่าเอาต์พุตทุกครั้งที่มันถูกกระทำการ เลขโดดตัวขวามือสุด ของเลขจำนวนแรกปรากฏในสดมภ์ที่ 10 และเลขโดดตัวขวามือสุดของเลขจำนวนที่สอง ปรากฏในสดมภ์ที่ 20 เพราะฉะนั้น ตารางถ้ามีสองสดมภ์ของเลขที่แสดงผล แต่ละจำนวนพิมพ์แบบจัดขวาที่มีหัวเรื่องของมัน ให้มั่นใจว่า ความกว้างของเขตข้อมูล (ในกรณีนี้เท่ากับ 10) ใหญ่พอที่จะเก็บค่าใหญ่สุดที่จะให้พิมพ์เทคนิคอีกอย่างหนึ่ง สำหรับการใช้ความต้องการลูปเพื่อออกแบบลูป คือ การทำงานแบบย้อนหลัง (back ward) จากผลลัพธ์ที่ต้องการไปจนถึงค่าแรกที่จะให้ผลลัพธ์เหล่านี้ ในตัวอย่างถัดไปเราจะใช้เทคนิคนี้ เพื่อหาขั้นตอนการเริ่มต้นสำหรับลูป

ตัวอย่าง 5.4 การทำงานแบบย้อนหลัง เพื่อหาการเริ่มต้นของลูป (Working Backward to Determine Loop Initialization)

หลานของเราอายุ 10 ปี กำลังเรียนระบบเลขฐานสอง (binary number system) และเธอขอให้เราเขียนโปรแกรมแสดงผลเลข 2 ยกกำลัง ทั้งหมดซึ่งมีค่าน้อยกว่า 10,000

สมมติว่าเลข 2 ยกกำลัง แต่ละครั้งเก็บในตัวแปร Power เราสามารถทำข้อสังเกต สองข้อเกี่ยวกับลูป ดังนี้

1. Power ระหว่างการวนซ้ำถัดไป (next iteration) คือ 2 คูณ กับ Power ระหว่างการวนซ้ำปัจจุบัน (current iteration)

2. Power ต้อง >= 10,000 หลักจากออกจากลูป

ความต้องการข้อ 1 ได้มาจากความจริงที่ว่า กำลังของ 2 คือ ทั้งหมดคูณด้วย 2 และความต้องการข้อ 2 มากจากความจริงว่าเฉพาะ powers ที่น้อยกว่า 10,000 เท่านั้นที่ให้แสดงผลจากความต้องการข้อ 1 เราทราบว่า Power ต้องคูณด้วย 2 ภายในตัวส่วนวนซ้ำ จากความต้องการข้อ 2 เราทราบว่าการออกจากลูป เกิดขึ้น ถ้า Power > = 10000 เป็นจริง ดังนั้น เงื่อนไขทำซ้ำของลูป คือ Power < 10000

ข้อพิจารณาเหล่านี้ นำเราไปสู่เค้าร่างข้างล่างนี้

```
1. Initialize Power to _____
2. while Power < 10000 do
       begin
          3. Display Power
          4. Multiply Power by 2
       end
```

วิธีหนึ่งเพื่อทำให้ขั้นที่ 1 เสร็จสมบูรณ์ คือ ค่าอะไรควรจะแสดงผลระหว่าง การวนซ้ำลูปครั้งแรก เนื่องจากค่าของเลขใด ๆ ก็ตามเมื่อยกกำลัง 0 เท่ากับ 1, การเริ่มต้น Power คือ 1

```
1. Initialize Power to 1
```

จะทำให้ขั้นที่ 3 แสดงผลค่าถูกต้องสำหรับ Power ระหว่างการวนซ้ำ ครั้งที่ 1 การวนซ้ำครั้งที่ 2 จะแสดงผล 1 x 2 หรือ 2 (2^1) ; การวนซ้ำครั้งที่ 3 จะแสดง 2 x 2 หรือ 4 (2^2) การวนซ้ำครั้งที่ 4 จะแสดงผล 4 x 2 หรือ 8 (2^3) เช่นนี้เรื่อยไป

การใช้แผ่นแบบเพื่อออกแบบลูป (Using Templates to Design Loops)

เนื่องจากลูปจำนวนมากมีโครงสร้างคล้ายกัน เราสามารถสร้างโครงร่าง (frameworks) ทั่วไป หรือ แผ่นแบบ (templates) เพื่อใช้ในการออกแบบ ลูปร่วมเหล่านี้

ลูปควบคุมโดยตัวนับ (Counter – Controlled Loops)

เราใช้ตัวนับเพื่อควบคุมลูป ในรูป 5.1 และรูป 5.4 ถึงแม้ว่ามันสามารถใช้ประโยชน์ในวัตถุประสงค์ที่แตกต่างกัน จำนวนมาก เราทำให้ขั้นตอนต่างๆ ของ ลูปควบคุมโดยตัวนับให้เป็นมาตรฐานในแผ่นแบบต่อไปนี้

แผ่นแบบสำหรับลูปควบคุมโดยตัวนับ (Template for Counter – Controlled Loop)

```
1. Set counter variable to 0
2. while counter variable < final value do
   begin
      ...
      3. Increase counter variable by 1
   end
```

ลูปร่วมอีกสองชนิดคือ sentinel loops และลูปควบคุม โดยตัวบ่งชี้แบบบูล (Boolean flags) ทั้งสองชนิดนี้ สามารถทำให้เป็นมาตรฐาน เพื่อเป็นแผ่นแบบช่วยในการออกแบบ

ลูปควบคุมโดย Sentinel (Sentinel – Controlled Loops)

โปรแกรมจำนวนมากที่มีลูป อ่านหน่วยข้อมูล (data item) หนึ่งตัว หรือมากกว่าหนึ่งตัว ทุกครั้งที่มันทำซ้ำ ตัวส่วนวนซ้ำ บ่อยครั้งเราไม่ทราบจำนวนหน่วยข้อมูลว่ามีมากเท่าใด ซึ่งลูปควรประมวลผลเมื่อมันเริ่มดันกระทำการ ดังนั้นเราจึงต้องหาวิธีให้สัญญาน (Signal) โปรแกรม เพื่อหยุดการอ่านและการประมวลผลข้อมูลตัวใหม่

วิธีหนึ่ง คือสั่งผู้ใช้ให้ใส่ค่าข้อมูล ซึ่งเป็นได้หนึ่งเดียว (a unique data value) เรียกว่า sentinel value หลังหน่วยข้อมูลตัวสุดท้าย เงื่อนไขการทำซ้ำลูป ทดสอบหน่วยข้อมูลแต่ละตัวและเหตุให้ออกจากลูป คือเมื่ออ่านพบ sentinel value การเลือกค่า sentinel ต้องทำอย่างระมัดระวัง มันต้องเป็นค่าซึ่งไม่ใช่ข้อมูลเกิดอย่างปกติ

ค่า sentinel หมายถึง ตัวทำเครื่องหมายจบ ซึ่งตามหลัง หน่วยข้อมูลตัวสุดท้าย (Sentinel value is an end marker that follows the last data item.)

โปรแกรม ซึ่ง คำนวณผลบวกของกลุ่มคะแนนสอบ คือ การเสนอตัวสำหรับการใช้ sentincl .value ถ้าชั้นเรียนมีขนาดใหญ่ ผู้สอนอาจจะไม่ทราบจำนวนแน่นอนของนักศึกษา

ที่เข้าสอบโปรแกรมควรทำงานได้โดยไม่สนใจขนาดของชั้นเรียน ลูปข้างล่างนี้ใช้ Sum เป็นตัวแปรสะสม (accumulator variable) และ Score เป็นตัวแปรอินพุต (input variable)

```
1. Initialize Sum to 0
2. Read the first score into Score
3. While Score is not the sentinel do
   begin
      4. Add Score to Sum
      5. Read the next score into Score
   end
```

ขั้นที่ 2 อ่านค่าข้อมูลตัวแรกไว้ใน Score ก่อนไปถึงลูป ในตัวส่วนวนซ้ำ ขั้นที่ 4 บวกคะแนนแต่ละตัวไว้ใน Sum (ค่าแรก คือ 0) ขั้นที่ 5 อ่านค่าข้อมูลแต่ละตัว ครั้งละหนึ่งตัวรวมทั้งค่า sentinel ลูปทำซ้ำกันตามใดที่ค่าข้อมูลก่อนหน้า ซึ่งอ่านเข้ามาไม่ใช่ sentinel การวนซ้ำแต่ละครั้งทำให้ค่าข้อมูลตัวก่อนหน้าบวกกัน Sum และค่าข้อมูลตัวถัดไปอ่านไว้ใน Score เนื่องจากการออกจากลูปเกิดขึ้นทันทีหลังจากอ่านค่า sentinel ดังนั้นค่า sentincl จึงไม่บวกกับ Sum

ขั้นตอนเหล่านี้ มีสองขั้นตอน คือ ขั้นที่ 2 และขั้นที่ 5 ซึ่งอ่านข้อมูลไปไว้ใน Score การอ่านเริ่มแรกก่อนเข้าลูป (ขั้นที่ 2) อ่านเฉพาะคะแนนแรกเท่านั้น เรียกว่า priming read

รูป 5.8 แสดงให้เห็นโปรแกรม Pascal ซึ่งเขียนลูปนี้ค่า sentinel คือ –1 เพราะว่าคะแนนสอบทั้งหมดจะไม่ใช่ค่าลบ การประกาศค่าคงตัว

```
const
   Sentinel = -1 ; {sentinel value}
```

เกี่ยวข้องกับค่าคงตัว Sntinel ด้วยค่า sentinel ถึงแม้ว่าอาจจะดูแปลกที่ครั้งแรกเห็นข้อความสั่ง

```
ReadLn (Score) ;
```

มีอยู่สองแห่งในโปรแกรม นี่คือ การฝึกปฏิบัติเขียนโปรแกรมที่ดีอย่างสมบูรณ์และไม่มีปัญหาใดๆ การวิ่งตัวอย่างในรูป 5.8 ประมวลผลคะแนน 55, 33 และ 77 ค่าเซนทิเนล (-1) คือค่าข้อมูลตัวสุดท้าย ซึ่งอ่านเข้ามา แต่ไม่รวมในผลบวก

Edit Window

```
program SumScores ;
{Accumulates the sum of exam scores}

  const
    Sentinel = -1 ;    {sentinel value}

  var
    Score,             {input – each exam score}
    Sum : Integer ;    {output – sum of scores}

begin    {SumScores}
  {Accumulate the sum.}
  Sum := 0 ;
  WriteLn ('When done, enter -1 to stop'.) ;
  Write (' Enter the first score > ') ;
 ReadLn (Score) ;
 while Score < > Sentinel do
  begin
      Sum := Sum + Score ;
      Write ('Enter the next score >') ;
      ReadLn (Score)
    end ; {while}

  {Display the sum.}
  WriteLn ;
  WriteLn ('Sum of exam score is ', Sum :1)
end. {Sum Score}
```

Output Window

```
When done, enter – 1 to stop.
Enter the first score > 55
Enter the next score > 33
Enter the next score > 77
Enter the next score > -1

Sum of exam score is 165
```

รูป 5.8 การใช้ Sentinel – Controlled Loop

เราควรทวนสอบ (verify) ว่าโปรแกรมถูกต้อง เมื่อไม่มีหน่วยข้อมูลเหลือให้ประมวลผล ในกรณีนี้ ใส่ค่า sentinel เป็นคะแนนแรกลูปควรออกทันที หลังจากการทดสอบเงื่อนไขการทำซ้ำลูปครั้งแรก เท่านั้น ดังนั้น ตัวส่วนวนซ้ำ (loop body) จะไม่ถูกกระทำการ (zero – iteration loop) Sum ยังคงมีค่าเริ่มต้นของมันเป็น 0 ซึ่งแสดงว่าถูกต้อง

แผ่นแบบสำหรับลูปควบคุมโดยค่าเซนทิเนล (Template for a Sentinel – Controlled Loop)

```
1. Read the first value of input vairable
2. while input variable is not equal to the sentinel do
   begin
      . . .
      3. Read the next value of input variable
   end
```

ค่า sentinel ต้องเป็นค่าซึ่งไม่ใช่หน่วยข้อมูลปกติที่จะใส่เข้าไปเป็นอินพุต และจะต้องไม่ถูกประมวลผลในตัวส่วนวนซ้ำ เพื่อสนับสนุนให้อ่านโปรแกรมง่าย ปกติเราเก็บค่าเซนทิเนล ในค่าคงตัว

การควบคุมลูปโดยตัวบ่งชี้แบบบูล (Loops Controlled by Boolean Flags)

ลูปซึ่งถูกควบคุมโดยเหตุการณ์ร่วมอีกชนิดหนึ่งคือ ลูปซึ่งการกระทำการถูกควบคุมด้วยตัวแปรแบบบูล

ตัวบ่งชี้โปรแกรม หรือ **ตัวบ่งชี้** หมายถึงตัวแปรแบบบูล ซึ่งค่าของมัน (จริงหรือเท็จ) ส่งสัญญานว่ามีเหตุการณ์อย่างหนึ่งเกิดขึ้นหรือไม่ (A program flag, or flag, is a Boolean variable whose value (True or False) signals whether a particular cvent occurs.)

ค่าตัวบ่งชี้ ควรจัดให้เป็น False ในตอนเริ่มต้นและจัดใหม่ให้เป็น True เมื่อเกิดเหตุการณ์ขึ้น ลูปควบคุมโดยตัวบ่งชี้ กระทำการจนกระทั่งเหตุการณ์ซึ่งกำลังเฝ้าสังเกตเกิดขึ้น

ตัวอย่างเช่น สมมติโปรแกรมอ่านตัวอักขระหลากหลาย ซึ่งคีย์ที่คีย์บอร์ด แต่เราสนใจเฉพาะตัวอักขระเลข (digit characters) เท่านั้น

(0, 1, 2, . . ., 9)

ดังนั้น โปรแกรมของเราจึงไม่สนใจ อักขระอื่นๆ เช่น ตัวว่าง ตัวอักษร สัญลักษณ์ เป็นต้น และเก็บตัวอักขระเลขตัวแรกที่อ่านเข้ามา ตัวแปรแบบบูลชื่อ DigitRead ควรใช้เป็นตัวบ่งชี้ เพื่อเฝ้าระวังว่าตัวอักขระ ซึ่งอ่านเข้ามาเป็นอักขระเลขหรือไม่

ตัวแปรโปรแกรม (Program variable)

DigitRead : Boolean {program flag – value is False until a digit character is read ; after a digit character is read, value is True}

เนื่องจากยังไม่มีตัวอักขระใดๆ อ่านเข้ามาก่อนที่ลูปจะกระทำการ เราจึงกำหนดค่าเริ่มต้นของ DigitRead ให้เป็น False ใน while ลูป ต้องกระทำการต่อเนื่องตราบใดที่ Digit Read เป็น False เพราะว่าสิ่งนี้มีความหมายว่า เหตุการณ์ "ข้อมูลที่อ่านเข้ามาเป็นอักขระเลข" ยัง ไม่เกิดขึ้น เพราะฉะนั้นเงื่อนไขการทำซ้ำลูปควรเป็น not DigitRead เพราะว่าเงื่อนไขนี้เป็นจริง เมื่อ DigitRead เป็นเท็จภายในตัวส่วนวนซ้ำ เราจะอ่านตัวอักขระข้อมูลทีละตัว และจัดค่าของ DigitRead ให้เป็น True ถ้าข้อมูลตัวอักขระเป็น เลขโดด while ลูปเขียนดังนี้

```
DigitRead := False ; {Assume no digit character was read.}
while not DigitRead do
   begin
      Write ('Enter another data character > ') ;
```

```
        ReadLn (NextChar) ;
        DigitRead := (' 0 ' < = NextChar) and (NextChar < = '9')
    end {while}
```

ภายในตัวส่วนวนซ้ำ ข้อความสั่งกำหนดค่า

```
DigitRead := ('0' < = NextChar) and (NextChar < = '9')
```

กำหนดค่า True ให้กับ DigitRead ถ้า NextChar เป็นตัวอักขระเลข กรณีอื่นๆ DigitRead ยังคงเป็น False ถ้า DigtRead เป็นจริงออกจากลูป ถ้า DigitRead ยังคงเป็นเท็จ ลูปกระทำการอย่างต่อเนื่อง จนกระทั่งในที่สุดมีการอ่านตัวอักขระเลข

แผ่นแบบสำหรับการวนซ้ำซึ่งควบคุมโดยตัวบ่งชี้ (Template for a Flag – Controlled Loop)

```
1. Initialize flag to False
2. while not flag do
      begin
         . . .
         3. Reset flag to True if the event being monitored occurs.
      end
```

ขั้นตอนสุดท้ายในตัวส่วนวนซ้ำปรับค่าของ flag จัดให้เป็น True หลังการเกิดครั้งแรกของเหตุการณ์ซึ่งกำลังเฝ้าระวัง

แบบฝึกหัด 5.4 Self – Check

1. เมื่อสร้างตารางในลูป ทำไมจึงต้องเพิ่ม format specifiers ให้กับเอาต์พุต list item ทุกตัวรวมทั้ง integers (ซึ่งไม่มีจุดทศนิยม)

2. ทำไมการย้ายข้อความสั่งกำหนดค่าใน sentinel – controlled loop ของรูป 5.8 ไปไว้ตอนจบของตัวส่วนวนซ้ำ จึงไม่ถูกต้อง

3. ให้เขียนใหม่ sentinel – controlled loop ในรูป 5.8 เป็น flag – controlled loop ในกรณีนี้เหตุการณ์ ซึ่งกำลังเฝ้าระวังควรเป็น "sentinel value was read"

4. เมื่อใด flag – controlled loop จึงเป็นทางเลือกที่ดีกว่า sentinel – controlled loop

เขียนโปรแกรม

1. จงเขียนโปรแกรม เพื่อบำรุงรักษา (maintain) สมุดฝากถอน เช็ค (check book) ขั้นแรกควรถามยอดบัญชีคงเหลือ (balance) จากนั้นถามรายการเปลี่ยนแปลง (transactions) และใส่เงินฝาก (deposits) เป็นค่าบวก และเช็ค (checks) เป็นค่าลบ หลังจากเช็คหรือฝากเงินแต่ละครั้งแล้ว ควรพิมพ์ตัวเลขใหม่ของยอดบัญชีคงเหลือให้ใช้ 0 เป็น ค่าเซนทิเนล (sentinel valve)

2. จงเขียนส่วนของโปรแกรม (program segment) ซึ่งให้ผู้ใช้ใส่ค่าต่างๆ และพิมพ์จำนวนของค่าบวกที่ใส่ และจำนวนของค่าลบที่ใส่ ให้ใช้ 0 เป็นค่าเซนทิเนล

3. จงเขียน while ลูปซึ่งแสดงผลกำลังทั้งหมดของเลขจำนวนเต็ม n (all powers of an integer, n) ซึ่งมีค่าน้อยกว่าค่าที่กำหนด ชื่อ MaxPower บนแต่ละบรรทัดของตารางแสดงกำลัง (0, 1, 2, . . .) และค่าของเลขจำนวนเต็ม n ยกกำลัง power ตัวนั้น

4. จงเขียนลูปซึ่งพิมพ์ตารางของมุม (angle) การวัดค่า sine และ cosine สมมติว่าการวัดมุมแรก และมุมสุดท้าย (มีหน่วยเป็นองศา) มีให้ใช้ใน InitDeg และ FinalDege (type Real) ตามลำดับและการเปลี่ยนแปลงในหน่วยวัดมุมระหว่างข้อมูลในตารางกำหนดโดย StepDeg (ข้อแนะนำอย่าลืมเปลี่ยนมุมจากองศาเป็นเรเดียน)

5. จงเขียน flag-controlled loop ซึ่งกระทำต่อเนื่องอ่านคู่ของเลขจำนวนเต็มจนกระทั่งคู่ที่อ่านนั้นมีคุณสมบัติว่าเลขตัวแรกในคู่ หารด้วยเลขตัวที่สองลงตัว

5.5 ข้อความสั่ง for (The for Statement)

นอกจากข้อความสั่ง while แล้ว Pascal ยังมีข้อความสั่งอีกชนิดหนึ่ง สำหรับเขียนลูปควบคุมโดยตัวนับ (counter-controlled loops) ข้อความสั่ง for ย่อ (condenses) รหัสสำหรับลูปควบคุมโดยตัวนับ (counter – contrallce Loop) เช่นที่จะเห็นโดยการเปรียบเทียบรหัสเทียมข้างล่างนี้

รหัสเทียมสำหรับ for ลูป

```
for counter := initial to final do
   begin
      . . .
   eng {for}
```

รหัสเทียบสำหรับ while ลูป

```
counter := initial ;
while counter < = final do
   begin
      . . .
      counter := counter + 1
   end {while}
```

บรรทัดแรก คือ หัวเรื่อง (header) ของ for ลูป กำหนดการจัดดำเนินการทั้งหมดของตัวแปร counter การดำเนินการสามเรื่อง ได้แก่

(1) กำหนดค่าเริ่มต้นของ counter ให้เท่ากับ initial

(2) ทดสอบ if counter <= final

(3) เพิ่มค่า counter ให้เป็นค่าถัดไปของมันก่อนการทดสอบแต่ละครั้ง

ตัวอย่าง 5.5

สองลูป ข้างล่างนี้ให้ผลลัพธ์เหมือนกัน

for ลูป

```
{Print N blank lines.}
for Line := 1 to N do
   WriteLn
```

while ลูป

```
{Print N blank lines.}
   Line := 1 ;
   while Line <= N do
      begin
         WriteLn :
         Line := Line + 1
      end {while}
```

ถ้า Line ประกาศเป็นตัวแปรชนิด Integer ข้อความสั่ง for กระทำการดำเนินการ WriteLn N ครั้ง การ implement ของ for ลูปสั้นกว่า while ลูป เพราะว่าไม่ต้องมีข้อความสั่งกำหนดค่า

```
Line := 1 ;
```

```
Line := Line + 1
```

ซึ่งกำหนดค่าเริ่มต้นและปรับค่าตัวนับ Line

Syntax Display

ข้อความสั่ง for

Form :

```
for counter := initial to final do statement
```

```
for counter := initial downto downto final do statement
```

ตัวอย่าง

```
for I := 1 to 5 do
   begin
      ReadLn (InData, NextNum) ;
      Sum := Sum + NextNum
   end ; {for}
for CountDown := 10 downto 0 do
   WriteLn (CountDown : 2)
```

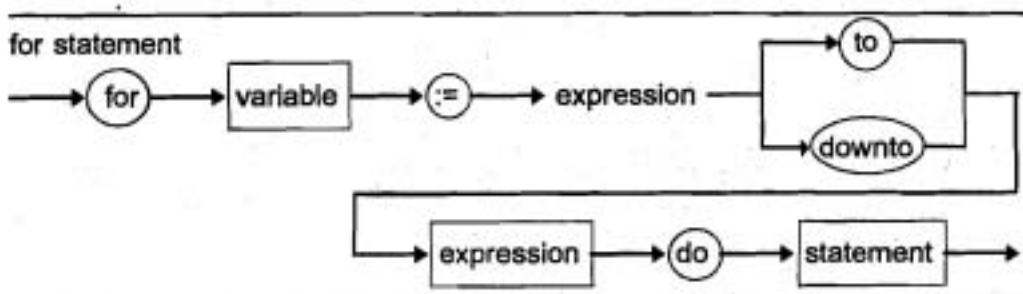
มีความหมายดังนี้ ข้อความสั่ง ซึ่งประกอบด้วยตัวส่วนวนซ้ำถูกกระทำการหนึ่งครั้งสำหรับแต่ละค่าของ counter ระหว่าง initial และ final นับทั้งต้นและท้าย (inclusive) เมื่อ initial และ final อาจจะเป็นค่าคงตัว (constant) ตัวแปร หรือ นิพจน์เป็นข้อมูล ชนิดเชิงอันดับที่ (ordinal type) เหมือนกับชนิดของ counter ใช้ downto แทนที่จะเป็น to ถ้า initial มีค่ามากกว่า final ในกรณีนี้ หลังจากการทำซ้ำลูปแต่ละครั้ง initial มีค่าลดลง 1

ข้อสังเกต 1 : ค่าของ counter ต้องไม่มีการดัดแปร (modified) ใน statement

ข้อสังเกต 2 : ค่าของ final คำนวณครั้งเดียวก่อนเข้าถึงลูป การเปลี่ยนแปลงค่าของ final ในตัวส่วนวนซ้ำ (loop body) จะไม่มีผล (no effect) แต่อย่างใดกับจำนวนครั้งการกระทำ

ข้อสังเกต 3 : หลังจากออกจากลูป ตัวแปร counter ยังคงเก็บค่าของมันระหว่างการวนซ้ำครั้งสุดท้ายใน Turbo Pascal แต่ไม่เก็บค่าครั้งสุดท้ายของมันใน standard Pascal

ข้อสังเกต 4 : statement จะไม่ถูกกระทำการ ถ้า initial มีค่ามากกว่า final (เมื่อใช้รูปแบบ to) ในรูปแบบ downto, statement จะไม่ถูกกระทำการถ้า initial มีค่าน้อยกว่า final



รูป 5.9 แผนภาพวากยสัมพันธ์ของข้อความสั่ง for

ตัวอย่าง 5.6

ข้อความสั่ง for ในรูป 5.10 อ่านข้อมูลเงินเดือน สำหรับพนักงาน 7 คน และคำนวณแสดงผลเงินค่าจ้างรายสัปดาห์ของพนักงานแต่ละคน จงเปรียบเทียบตัวอย่างนี้กับตัวอย่างที่มีข้อความสั่ง while ซึ่งแสดงในรูป 5.1

```
for    EmpNumber   := 1 to 7 do
   begin
      Write ('Hours > ') ;
      ReadLn (Hours) ;
      Write ('Rate > $') ;
      ReadLn (Rate) ;
      Pay := Hours * Rate ;
      WriteLn ('Weekly Pay is $ ' , Pay : 4 : 2)
   end ; {for}
WriteLn ('All employees proussed')
```

รูป 5.10 for ลูป เพื่อประมวลผลพนักงานเจ็ดคน

อ่านบรรทัดแรกของรูป 5.10 เป็น

"for each value of EmpNumber from 1 to 7 do"

เราไม่ต้องจัด ข้อความสั่ง Pascal เพิ่มเติมเพื่อกำหนด ค่าเริ่มต้นให้ EmpNumber หรือปรับค่าของ EmpNumber การดำเนินการสองอย่างนี้ ถูกกระทำอัตโนมัติใน for ลูป

คล้ายกับข้อความสั่ง while ตัวแปร counter ในข้อความสั่ง for นำไปใช้ในการคำนวณได้ แต่สิ่งที่ไม่เหมือนกับข้อความสั่ง while คือ ค่าของ counter เปลี่ยนแปลงไม่ได้ในตัวส่วนวนซ้ำ (loop body) ในตัวอย่างถัดไป เราจะเห็นข้อความสั่ง for ซึ่งใช้ตัวแปร counter ของมันเป็นอาร์กิวเมนต์ของฟังก์ชัน

ตัวอย่าง 5.7

for ลูปในรูป 5.11 พิมพ์รายการของค่าจำนวนเต็ม และกำลังสองและรากที่สองของมัน ระหว่างการทำซ้ำแต่ละครั้งของตัวส่วนวนซ้ำ ข้อความสั่ง

Square := Sqr (I) ;

Root := Sqr t (I) ;

คำนวณกำลังสองและรากที่สองของตัวแปร counter I จากนั้นแสดงผลค่าของ I, Square และ Root ตาราง 5.2 ตามรอยการกระทำการของ for ลูป

Edit Window

```
program Squares ;
{Displays a table of integers end their squares and square roots}
  const
    MaxI = 4 ;          {largest integer in table}
 var
  I ,                   {counter variable}
  Square : Inteqer ;      {output – square of I}
  Root : Real ;         {output – square root of I}

begin {Squares}
  {Prints a list of inteqers , their squares, and their square roots}
  WriteLn ('I' : 10 , 'I * I' : 10 , ' Square root' : 15) ;
  for I := 1 to MaxI do
    begin
      Square := Sqr (I) ;
      Root := Sqrt (I) ;
      WriteLn (I : 10, Square :10, Root : 15 : 1)
    end {for}
end. {Squares}
```

Output Window

I	I * I	Square Root
1	1	1.0
2	4	1.4
3	9	1.7
4	16	2.0

รูป 5.11 ตารางของจำนวนเต็ม กำลังสองและรากที่สอง

การตามรอยในตาราง 5.2 แสดงให้เห็นว่าตัวแปร ตัวนับ I ค่าเริ่มต้น คือ 1 เมื่อมาถึง for ลูป หลังจากการทำซ้ำลูป แต่ละครั้ง I มีค่าเพิ่มอีก 1 และทดสอบว่าค่าของ I ยังคงน้อยกว่า หรือเท่ากับ MaxI หรือไม่ ถ้าผลลัพธ์ของการทดสอบเป็นจริง ตัวส่วนวนซ้ำจะถูกกระทำการอีกครั้ง และค่าถัดไป ของ I, Square และ Root จะถูกพิมพ์ ถ้าผลลัพธ์ของการทดสอบเป็นเท็จ ให้ออกจากลูป

ตาราง 5.2 ตามรอยโปรแกรมในรูป 5.11

Statement	I	Square	Root	Effect
	?	?	?	
for I := 1 to MaxI	1			Initialize I to 1
Square := Sqr (I) ;		1		Assign 1 to Sauare
Root := Sqrt (I) ;			1.0	Assign 1.0 to Root
WriteLn . . .				Print 1, 1, 1.0
Increment and test I	2			2 <= 4 is true
Square := Sqr (I) ;		4		Assign 4 to Square
Root := Sqrt (I) ;			1.4	Assign 1.4 to Root
WriteLn . . .				Print 2, 4, 1.4
Increment and test I	3			3 <= 4 is true
Square := Sqr (I) ;		9		Assign 9 to Square

Statement	I	Square	Root	Effect
Root := Sqrt (I);			1.7	Assign 1.7 to Root
WriteLn . . .				Print 3, 9, 1.7
Increment and test I	4			4 <= 4 is true
Squar := Sgr (I),		16		Assingn 16 to Square
Root := Sqrt (I) ;			2.0	Assingn 2.0 to Root
WriteLn . . .				Print 4, 16, 2.0
Inerment and test I	5			5 < = 4 is false
	4			Exit loop. Reset I to last value

I มีค่าเท่ากับ MaxI ระหว่างการทำซ้ำลูปครั้งสุดท้าย ใน Turbo Pascal นั้น I ยังคงเก็บค่าสุดท้ายของมันเมื่อออกจากลูป แต่ใน Standard Pascal ค่าของ I จะไม่ถูกนิยาม (undefined) เมื่อออกจากลูป ดังนั้นใน Standard Pascal เราจึงไม่ควรอ้างถึงตัวแปร I อีกต่อไป จนกว่าจะมีการกำหนดค่าใหม่ให้ I

ตัวแปร counter ชนิด Char (Type Char Counter Variables)

for ลูปสามารถใช้กับข้อมูลเชิงอันดับที่ชนิดอื่นนอกเหนือจากชนิด Integer ได้ ตัวอย่างถัดไปใช้ตัวแปร counter ชนิด Char

ตัวอย่าง 5.8

for ลูป ข้างล่างนี้ แสดงผลตัวอักษรตัวพิมพ์ใหญ่ (uppercase letters) บนหนึ่งบรรทัด ตัวแปร counter ชื่อ NextCh จึงต้องเป็นชนิด Char

```
for NextCh := 'A' to 'Z' do
   Write (NextCh);
.WriteLn
```

การนับลง (Counting Down) ตัวอย่างต่างๆ ที่กำหนดให้มาจนถึงบัดนี้ ใช้คำสงวน to และเพิ่มค่าของตัวแปร counter หลังจากการทำซ้ำลูปแต่ละครั้ง ถ้าเราใช้คำสงวน downto แทนที่คำสงวน to ค่าของตัวแปร counter ลดลง หลังจากการทำซ้ำลูปแต่ละครั้ง

ตัวอย่าง 5.9

นักศึกษาต้องการตารางแสดงอุณหภูมิที่เท่ากันของหน่วยเซลเซียส (Celsius) จาก 5 องศาถึง – 10 องศา กับอุณหภูมิหน่วยฟาเรนไฮท์ (Fahrenheit) โดยใช้ for ลูปข้างล่างนี้

```
for Celsius    := 5 downto –10 do
  begin
    Fahrenhcit := 1.8 * Celsius + 32 ;
    WriteLn (Celsius : 10, Fahrenheit : 15 : 1)
  end {for}
```

แบบฝึกหัด 5.5 Self – Check

1. จงตามรอย (trace) ส่วนของโปรแกรมข้างล่างนี้

```
    J := 10 ;
  for I := 1 to 5 do
    begin
      WriteLn (I, J) ;
      J := J – 2
    end ; {for}
```

2. จำนวนครั้งน้อยที่สุดของ statement ซึ่งประกอบเป็นตัวส่วนวนซ้ำ (loop body) ของ for จะถูกกระทำการกี่ครั้ง จงยกตัวอย่างส่วนของโปรแกรม ซึ่งจะกระทำการจำนวนครั้งน้อยที่สุด

3. จงเขียนหัวเรื่อง (header) ของ for ลูป ซึ่งประมวลค่าทั้งหมดของ Celsius (ชนิด Integer) ในพิสัยข้างล่างนี้

   a) - 10 ถึง + 10

   b) 100 ถึง 1

   c) 15 ถึง 50

   d) 50 ถึง –75

4. มีตัวแปรชนิดใดบ้างซึ่งสามารถประกาศเป็นตัวแปร counter ของ for ลูป ได้

เขียนโปรแกรม

1. จงเขียนข้อความสั่ง for คำนวณผลบวกของเลขจำนวนเต็มคี่ (odd integers) ในพิสัย 0 ถึง 100 นับทั้งต้นและท้าย (inclusive)

2. จงเขียนส่วนของโปรแกรมที่มีข้อความสั่ง for ซึ่งสะสมจำนวนวันทั้งหมด ตั้งแต่ปี ค.ศ. 1950 ถึง ค.ศ. 2000

ข้อควรจำ ปี ค.ศ.ใดๆ ก็ตามซึ่งหารด้วย 4 ลงตัวเป็นปีอธิกวาร (leap year) และมี 366 วัน

5.6 ข้อความสั่ง repeat (The repeat Statement)

·ข้อความสั่ง repeat กำหนดลูปมีเงื่อนไข ซึ่งทำซ้ำจนกระทั่งเงื่อนไขเป็นจริง ลูปเช่นนี้เรียกว่า repeat – until ลูปใช้ตัวอย่างอธิบาย จงเปรียบเทียบส่วนของโปรแกรมสองชุด ซึ่งพิมพ์ กำลัง 2 ของเลขระหว่าง 1 ถึง 1000

ข้อความสั่ง repeat

```
Power := 1 ;
Repeat
   Write (Power : 5) ;
Power := Power * 2
until Power >= 1000
```

ข้อความสั่ง while

```
Power := 1 ;
while Power < 1000 do
   begin
      Write (Power : 5) ;
      Power := Power * 2
   end {while}
```

·การทดสอบใน repeat – until ลูป (Power > = 1000)

เรียกว่าส่วนเติมเต็มเชิงตรรกะ (logical complement) หรือ สิ่งตรงข้ามของการทดสอบใน while ลูป ในข้อความสั่ง repeat ตัวส่วนวนซ้ำ (loop body) จะทำซ้ำจนกระทั่งค่าของ Power มากกว่าหรือเท่ากับ 1000 เนื่องจากการทำซ้ำลูปหยุด (stop) เมื่อเงื่อนไขเป็นจริง การทดสอบนี้ เรียกว่าการทดสอบการเลิกลูป (loop – termination test) ไม่ใช่การทดสอบการทำซ้ำลูป (loop – repetition test)

ส่วนเติมเต็มเชิงตรรกะ หมายถึง นิพจน์แบบบูลที่มีค่าตรงกันข้าม (Logical complements are Boolean expressions with opposite values.)

การทดสอบการเลิกลูป หมายถึง เงื่อนไขตามหลัง until ซึ่งทำให้ออกจากลูปเมื่อมันเป็นจริง (Loop – termination test is the condition following until that canses loop exit when it becomes True.) เราอธิบายวากยสัมพันธ์ของข้อความสั่ง repeat และแผนภาพของมันในรูป 5.13 โปรดสังเกตว่าไม่มีคู่ begin – end ปิดล้อมตัวส่วนวนซ้ำ เพราะว่าคำสงวน repeat และ until กระทำหน้าที่นี้

Syntax Display

ข้อความสั่ง repeat (repeat – until Loop)

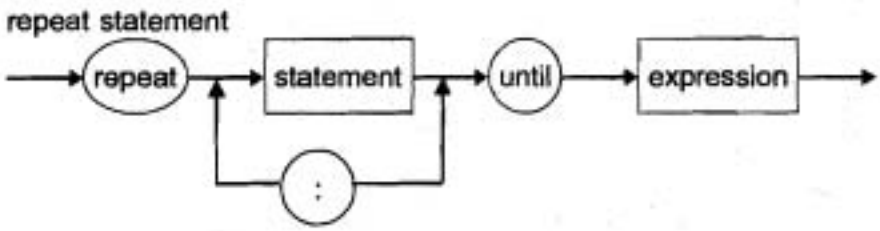
Form :

```
repeat
    Loop – body
until termination – condition
```

ตัวอย่าง

```
repeat
   Write (' Enter a digit > ') ;
   Read (Ch)
until ('0' < = Ch) and (Ch < = '9')
```

มีความหมายดังนี้ หลังจากการกระทำการแต่ละครั้งของตัวส่วนวนซ้ำ เงื่อนไขการเลิกถูกประเมินผล ถ้าเงื่อนไขการเลิกเป็นจริงให้ออกจากลูป และข้อความสั่งโปรแกรมถัดไป จะถูกกระทำการ ถ้าเงื่อนไขการเลิกเป็นเท็จ ตัวส่วนวนซ้ำ ถูกทำซ้ำ ปกติตัวส่วนวนซ้ำจะถูกกระทำการอย่างน้อยที่สุดหนึ่งครั้งเสมอ



รูป 5.13 แผนภาพวากยสัมพันธ์ของข้อความสั่ง repeat

ตัวอย่าง 5.11

โปรแกรมในรูป 5.14 ใช้ repeat – until ลูป เพื่อหาค่าใหญ่ที่สุดในลำดับของหน่วยข้อมูล (data items)

ตัวแปร Item เก็บหน่วยข้อมูลแต่ละตัว และตัวแปร LargestSoFar เก็บ ค่าข้อมูลตัวใหญ่ที่สุดที่พบ ภายในลูป

ข้อความสั่ง if

```
if  Item > LargestSoFar then
    LargestSoFar := Item {Save the new largest number.}
```

ให้นิยามใหม่ค่าของ LargestSoFar ถ้าหน่วยข้อมูลปัจจุบันมีค่าใหญ่กว่า ค่าข้อมูลก่อนหน้าทั้งหมด

Edit Window

```
program Largest ;
{Finds the largest number in a sequence of integer values}
  const
    MinValue = - MaxInt ;      {a very small integer}
  var
    Item,                      {each data valve}
    LargestSoFar : Integer;    {largest value so far}

begin {Largest}
  {Initialize LargestSoFar to a very small integer}
  LargestSoFar := MinValue ;

  {Save the largest number encontered so far}
  WriteLn ('Finding the largest value in a sequence :') ;
  repeat
    Write ('Enter an integer or ' , MinValue : 1, ' to stop > ') ;
```

```
        ReadLn (Item) ;
        if Item > LargestSoFar then
            LargestSoFar := Item {Save the new largest number}
    until Item = MinValue ;

    WriteLn ('The largest value entered was' : LargestSoFar : 1)
end. {Largest}
```

Output Window

```
.Finding the largest value in a sequence :
Enter an integer or –32767 to stop > - 999
Enter an integer or –32767 to stop > 500
Enter an integer or –32767 to stop > -32767
The largest value entered was 500
```

รูป 5.14 การหาค่าใหญ่ที่สุด

ค่าคงตัว MinValue ซึ่งแทนค่าจำนวนเต็มน้อยมากใช้ในสองวัตถุประสงค์ คือ กำหนดค่าเริ่มต้นของ LargestSoFar ให้เป็น MinValue ก่อนเข้าถึงลูป ทำให้เงื่อนไข Item > LargestSoFar เป็นจริงระหว่างการวนซ้ำครั้งแรก ดังนั้น หน่วยข้อมูลตัวแรกจึงเป็นค่าใหญ่ที่สุด ในขณะนี้ จากนั้น MinValue ถูกนำมาใช้เป็นค่า sentinel ด้วยเพราะว่ามันไม่เหมือนกับหน่วยข้อมูลปกติที่ใส่เข้ามาสำหรับโปรแกรมการหาเลขที่มีค่าใหญ่ที่สุดในลำดับ

ตัวอย่าง 5.12

เมื่อเราใช้ Turbo Pascal จะมีแถบรายการเลือก (menu bar) ปรากฏที่ตอนบนของจอภาพแสดงการกระทำที่เป็นไปได้ เช่น file, edit run, compile, debug, tools, options, window เป็นต้น

โปรแกรมซึ่งใช้รายการเลือก (menu) เพื่อกำหนดการดำเนินการต่อไปของมัน เรียกว่า**โปรแกรมทำงานด้วยรายการเลือก** (menu – driven program) ตัวอย่างเช่น โปรแกรมสถิติจะมีรายการที่เป็นไปได้ของการดำเนินการตามรูปแบบรายการเลือกข้างล่างนี้ และผู้ใช้โปรแกรมจะใส่เลข ระหว่าง 1 ถึง 6 เพื่อให้เลือกการดำเนินการโปรแกรมถัดไป

1. Compute an average.
2. Compute a standard deviation.
3. Find the median.
4. Find the smallest and largest values.
5. Plot the data.
6. Exit the program.

ข้อความสั่ง repeat สามารถใช้เป็นข้อความสั่งควบคุมหลักสำหรับโปรแกรมทำงานด้วยรายการเลือก ดังแสดงให้เห็นในรหัสเทียมข้างล่างนี้

```
repeat
   Display the menu
   Read the user's choice
   Perform the user's choice
.until user's choice is exit program
```

ส่วนของโปรแกรมในรูป 5.15 implements ลูปนี้ สำหรับการวนซ้ำแต่ละครั้ง กระบวนงาน DisplayMenu แสดงรายการเลือกจากนั้น อ่านทางเลือกของผู้ใช้ไว้ใน Choice และส่งไปยังกระบวนงาน DoChoice ลูปทำซ้ำจนกระทั่งทางเลือกของผู้ใช้ คือ ExitChoice (ค่าเท่ากับ 6)

```
repeat
   DisplayMenu ;          {Display the menu choices}
   WriteLn ('Enter a number between 1 and ', ExitChoice : 1) ;
   ReadLn (Choice) ;
   DoChoice (Choice)      {Perform the user's choice}
until Choice = ExitChoice
```

รูป 5.15 ลูปควบคุมหลักสำหรับโปรแกรมทำงานด้วยรายการเลือก

การทำส่วนเติมเต็มของนิพจน์แบบบูล (Complementing a Boolean Expression)

การแทนที่ข้อความสั่ง while ด้วยข้อความสั่ง repeat เราจำเป็นต้องทราบ ส่วนเติมเต็มของเงื่อนไขทำซ้ำของ while ลูป การทำส่วนเติมเต็ม การเปรียบเทียบอย่างง่าย คือ เปลี่ยนตัวดำเนินการของมัน ดังแสดงให้เห็นในตาราง (ตัวอย่างเช่น Power > = 1000 มีส่วนเติมเต็มเป็น Power < 1000)

ตัวดำเนินการ (Operator)	ตัวดำเนินการในส่วนเติมเต็ม (Operator in Complement)
<	>=
<=	>
<	<=
>=	<
=	< >
< >	=

เราสามารถประกอบส่วนเติมเต็ม ของนิพจน์แบบบูล โดยใส่ not ไว้ข้างหน้านิพจน์ทั้งหมด ตัวอย่างเช่น นิพจน์

(Ch >= 'a') and (Ch <= 'Z')

เป็นจริงเมื่อ Ch เป็นอักษรตัวพิมพ์เล็ก ส่วนเติมเต็มของนิพจน์คือ

not ((Ch >= 'a') and (Ch <= 'Z'))

ทฤษฎีบทของ DeMorgan

not (expression$_1$ and expression$_2$) = (not expression$_1$) or (not expression$_2$)

not (expression$_1$ or expression$_2$) = (not expression$_1$) and (not expression$_2$)

เป็นอีกวิธีหนึ่งในการหาส่วนเติมเต็มของนิพจน์แบบบูล ซึ่งประกอบด้วยตัวดำเนินการ and และ ตัวดำเนินการ or เขียนส่วนเติมเต็มของนิพจน์แบบบูล แยกเป็นคนละชุด และเปลี่ยน and แต่ละตัวให้เป็น or และเปลี่ยน or แต่ละตัวให้เป็น and การใช้ทฤษฎีบทของ DeMorgan ส่วนเติมเต็มของนิพจน์เดิม จะเขียนดังนี้

(Ch < ' a') or (Ch > 'Z')

ซึ่งเป็นจริง ถ้า Ch ไม่ใช่อักษรตัวพิมพ์เล็ก

```
forCount := StartValue to StopValue do {for loop}
   begin
      . . .
   end {for}
```

```
Count := StartValue ;                {while loop}
while Count < = StopValue do
   begin
      . . .
      Count := Count + 1
   end {while}
```

```
Count := StartValue;                 {repeat – until loop}
if  StartValue <= StopValue then
   repeat
      . . .
      Count := Count + 1
   until Count > StopValue
```

รูป 5.16 การเปรียบเทียบลูป 3 รูปแบบ

ในรูป 5.16 Count, StartValue และ StopValue ทั้งหมดนี้ต้องเป็นข้อมูลชนิดเชิงอันดับที่ (ordinal type) ข้อความสั่งกำหนดค่า

Count := Count + 1

ใช้ทั้งใน while ลูป และ repeat ลูป เพื่อปรับตัวแปรควบคุมลูป Count แต่ใน for ลูป ไม่ต้องใช้

แบบฝึกหัด 5.6 Self-Check

1. จงใช้ทฤษฎีบทของ DeMorgan เพื่อหาส่วนเติมเต็มของเงื่อนไข ต่อไปนี้

   a) (X <= Y) and (X < > 15)

   b) (X <= Y) or (Z = 7.5)

   c) (X < > 15) or (Z = 7.5) and (X <= Y)

   d) Flag or not (X < > 15.7)

   e) Not Flag and (X <= 8)

2. ข้อความสั่ง repeat ข้างล่างนี้แสดงผลอะไร จงอธิบายความแตกต่างระหว่าง repeat . . . until False และ while False do

```
repeat
   WriteLn ('False conditional example.')
until False
```

*3. เมื่อใดเราจึงควรใช้ repeat – until ลูป ไม่ใช้ while ลูป

เขียนโปรแกรม

1. จงเขียนส่วนหนึ่งของโปรแกรมซึ่งอ่านค่าข้อมูลตัวเนื่องตราบใดที่ค่าข้อมูลไม่ลดลง ส่วนของโปรแกรมหยุดอ่านเมื่อใดก็ตามที่มีเลขค่าน้อยกว่าเลขตัวซึ่งอ่านก่อนหน้านั้น

   จงเขียนสองเวอร์ชัน (versions) เวอร์ชันแรกใช้ข้อความสั่ง repeat และอีกเวอร์ชันหนึ่งใช้ข้อความสั่ง while

2. จงเขียนส่วนหนึ่งของโปรแกรม สำหรับลูปควบคุมหลักในโปรแกรมทำงานด้วยรายการเลือก (menu – driven program) ซึ่งปรับ (update) เงินคงเหลือในบัญชี (W = withdrawal, D = deposite, Q = quit) สมมติว่ากระบวนงาน ProcessWithdrawal และ ProcessDeposit มีให้ใช้อยู่แล้ว และถูกเรียกด้วย พารามิเตอร์จริง Balance แจ้งให้ผู้ใช้ทราบรหัส รายการเปลี่ยนแปลง (W, D หรือ Q) และเรียกกระบวนงานที่ถูกต้อง

5.7 ลูปซ้อนใน (Nested Loops)

ลูปสามารถเขียนซ้อนในได้เหมือนกับข้อความสั่ง if ลูปซ้อนในประกอบด้วย ลูปชั้นนอก (outer loop) หนึ่งชุด ซึ่งมีลูปชั้นใน (inner loop) หนึ่งชุด หรือมากกว่าหนึ่งชุด แต่ละครั้งที่ลูปชั้นนอกทำซ้ำ ลูปชั้นในต้องเริ่มต้นใหม่ นิพจน์ควบคุมลูปถูกประเมินผลใหม่ และการดำเนินการที่ต้องการทั้งหมดถูกกระทำ

ตัวอย่าง 5.13

รูป 5.17 แสดงการวิ่งของโปรแกรมตัวอย่าง ที่มีการซ้อนในสองชุด ของลูป for ลูปชั้นนอกทำซ้ำ สามครั้ง (for I เท่ากับ 1, 2, 3) แต่ละครั้งที่ลูปชั้นนอกทำซ้ำ ลูปชั้นใน

ข้อความสั่ง

```
WriteLn ('Outer' : 5, I : 7) ;
```

จะแสดงผลสายอักขระ 'Outer' และค่าของ I (ตัวแปรควบคุมลูปชั้นนอก ต่อไปเข้ามาในลูปชั้นใน ตัวแปรควบคุมลูปชั้นใน คือ J ตั้งใหม่เป็น 1 จำนวนครั้งการทำซ้ำของลูปชั้นในขึ้นอยู่กับค่าปัจจุบันของ I ทุกครั้งที่ลูปชั้นในทำซ้ำ

ข้อความสั่ง

```
WriteLn ('Inner' : 7, I : 5, J : 5)
```

จะแสดงผลสายอักขระ 'Inner' และค่าของ I และ J

ตัวแปรควบคุมลูปชั้นนอก I เป็นนิพจน์เช่นกัน ซึ่งค่าของมันกำหนดจำนวนการทำซ้ำของลูปชั้นใน ถึงแม้ว่า สิ่งนี้ถูกต้องบริบูรณ์แต่เราไม่ควรใช้ตัวแปรชื่อเหมือนกันเป็นตัวแปรควบคุมลูปของทั้ง for ลูปชั้นนอกและชั้นในทั้งคู่ในการซ้อนในชุดเดียว

Edit Window

```
program NestLoop ;
{Illustrates a pair of nested for loops}

  var
    I, J : Integer ;        {loop – control variables}

begin {NestLoop}
  WriteLn ('I' : 12, 'J' : 5) ; {Print heading}
  for I := 1 to 3 do
    begin {outer loop}
      WriteLn ('Outer ' : 5, I : 7) ;
        for J := 1 to I do
```

```
            WriteLn ('Inner' : 7 , I : 5, J : 5)
        end. {outer Loop}
end.  {NestLoop}
```

Output Window

```
            I     J
Outer       1
    Inner   1     1
Outer       2
    Inner   2     1
    Inner   2     2
Outer       3
    Inner   3     1
    Inner   3     2
    Inner   3     3
```

รูป 5.17 โปรแกรมที่มี for ลูปซ้อนใน

ตัวอย่าง 5.14

โปรแกรม Triangle (รูป 5.18) ใช้ลูปชั้นนอก (ตัวแปรควบคุมลูป Row) และลูปชั้นในสองชุด เมื่อวาดรูปสามเหลี่ยมหน้าจั่ว (isosceles triangle) ทุกครั้งที่ทำซ้ำลูปชั้นนอก ลูปชั้นในสองชุดจะถูกกระทำการ ลูปชั้นในชุดแรกพิมพ์อักขระว่างนำหน้า ลูปชั้นในชุดที่สองพิมพ์ดาว (asterisk) หนึ่งตัวหรือมากกว่าหนึ่งตัว

ลูปชั้นนอกทำซ้ำห้าครั้ง ค่าของ Row กำหนดจำนวนการทำซ้ำซึ่งกระทำโดยลูปชั้นใน ตาราง 5.3 แสดงรายการพารามิเตอร์ ซึ่งควบคุมลูปชั้นในสำหรับแต่ละค่าของ Row, เมื่อ Row มีค่าเท่ากับ 1 พิมพ์อักขระว่างสี่ตัว และดาวหนึ่งตัวเมื่อ Row มีค่าเท่ากับ 2 พิมพ์อักขระว่างสามตัว และดาวสามตัวเมื่อ Row มีค่าเท่ากับ 3 พิมพ์อักขระว่างสองตัวและดาวห้าตัว เมื่อ Row มีค่าเท่ากับ 4 พิมพ์อักขระว่างหนึ่งตัวและดาวเจ็ดตัว เมื่อ Row มีค่าเท่ากับ 5 ข้ามลูปชั้นในชุดแรก และพิมพ์ดาวเก้าตัว (2 * 5 -1)

ตาราง 5.3 พารามิเตอร์ควบคุมลูปชั้นใน (Inner Loop – Control Parameters)

Row	Lead Blanks	CountStars	Effect
1	4 downto 1	1 to 1	Displays 4 blanks and 1 star
2	3 downto 1	1 to 3	Displays 3 blanks and 3 stars
3	2 downto 1	1 to 5	Displays 2 blanks and 5 stars
4	1 downto 1	1 to 7	Displays 1 blank and 7 stars
5	0	1 to 9	Displays 9 stars

Edit Window

```
program Triangle ;
{Draws an isosceles Triangle}
   const
      NumLincs = 5 ;          {number of rows in triangle}
      Blank = ' ' ;           {output characters}
      Star = '*' ;
   Var
      Row,                    {loop control for outer loop}
      Lead Blanks,            {loop control for first inner loop}
      CountStars : Integer ;  {loop control for second inner loop}
begin   {Triangle}
   for   Row := 1 to NumLines do
      begin                   {Draw each row}
         for   LeadBlanks := NumLines – Row downto 1 do
               Write (Blank) ;   {Print leading blanks}
         for   CountStars := 1 to 2 * Row –1 do
               Write (Star);     {Print asterisks}
```

```
        WriteLn                {Terminate line}
    end. {for Row}
end. {Triangle}
```

Output Window

```
          *
        * * *
      * * * * *
    * * * * * * *
  * * * * * * * * *
```

รูป 5.18 โปรแกรมสามเหลี่ยมหน้าจั่ว (Isosceles Triangle Program)

แบบฝึกหัด 5.7 Self-check

1. จงแสดงเอาต์พุตของโปรแกรมเซกเมนต์ข้างล่างนี้ สมมติว่า M มีค่าเท่ากับ 3 และ N มีค่าเท่ากับ 5

a)
```
for I := 1 to N do
    begin
        for J := 1 to I do
            Write (' * ') ;
        WriteLn
    end {for I}
```

b)
```
for I := N downto 1 do
    begin
        for J := M downto 1 do
            Write (' * ') ;
        WriteLn
    end {for I}
```

2. จงแสดงเอาต์พุตของลูปซ้อนใน (nested loops) ข้างล่างนี้

```
for I := 1 to 2 do
   begin
      WriteLn ('Outer' : 5, I : 5) ;
      for J := 1 to 3 do
         WriteLn ('Inner' : 7, I : 3, J : 3)
      for K := 2 downto J do
         WriteLn ('Inner' : 7, I :3, K : 3)
end {for I}
```

เขียนโปรแกรม

1. จงเขียนโปรแกรมเซกเมนต์ กำหนดค่าของอินพุต N แสดงผล N แถวของรูปแบบ 1 2, . . . N, 2 3 . . . N + 1 เช่นนี้ เรื่อยไป

ตัวอย่างเช่น สำหรับอินพุต N มีค่าเท่ากับ 5 แสดงผลดังนี้

```
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
```

5.8 การแก้จุดบกพร่อง และการทดสอบโปรแกรม (Debugging and Testing Programs)

ในหัวข้อ 2.9 เราได้อธิบายข้อผิดพลาดทั่วไปแบ่งออกเป็นสามชนิด ได้แก่ syntax errors, run –time errors และ logic errors บางครั้งเหตุของ run – time error หรือ แหล่งของ logic error ปรากฏให้เห็นและข้อผิดพลาดนั้นแก้ไขง่าย แต่บ่อยครั้งข้อผิดพลาดนั้นไม่เห็นชัดเจนและอาจต้องใช้เวลาในการหาตำแหน่งที่ผิด

ขั้นแรกในการหาข้อผิดพลาดซ่อน (hidden error) คือการตรวจสอบเอาต์พุตโปรแกรม เพื่อหาว่าส่วนใดของโปรแกรม ซึ่งทำให้เกิดผลลัพธ์ไม่ถูกต้อง หลังจากนั้นเราสามารถเน้นที่ข้อความสั่ง ในส่วนนั้นเพื่อหาว่า อะไรที่ผิด การหาพื้นที่ของปัญหาระหว่าง

การแก้จุดบกพร่อง ให้ใส่ข้อความสั่ง WriteLn เพิ่มเพื่อแสดงผลลัพธ์ทันทีที่จุดแตกต่างกันในโปรแกรม ตัวอย่าง เช่น ถ้าลูป ในรูป 5.8 คำนวณผลบวก (sum) ไม่ถูกต้องข้อความสั่ง WriteLn วินิจฉัยจะแสดงผลแต่ละค่าของ Score และ Sum เครื่องหมาย * (asterisks) แสดงเอาต์พุตวินิจฉัยให้เด่นขึ้นในการวิ่งแก้จุดบกพร่อง และข้อความสั่ง WriteLn วินิจฉัยในโปรแกรมต้นฉบับ

```
ReadLn (Score) ;
While Score < > Sentinel do
  begin
    Sum := Sum + Score ;
    WriteLn (' * * * * * score is ' , Score, ' Sum is ' Sum);
    Write (' Enter the next score > ');
    ReadLn (Score)
  end {while}
```

ให้ระวังเมื่อใส่ข้อความสั่ง WriteLn วินิจฉัย บางครั้งเราต้องใส่คู่ begin-end ถ้าภายในข้อความสั่ง if หรือ while มีเพียงหนึ่งข้อความสั่ง เพื่อทำให้เป็นข้อความสั่งประกอบ เมื่อเราใส่ diagnostic WriteLn ถ้าเราใส่ข้อความสั่ง WriteLn หลังข้อความสั่งสุดท้ายในตัวส่วนวนซ้ำ อย่างลืมข้างหน้า WriteLn ใส่ semicolon ด้วย

เมื่อพบตำแหน่งที่ทำให้เกิดข้อผิดพลาดแล้ว แทนที่จะเอาข้อความสั่ง WriteLn วินิจฉัยออกจากโปรแกรม เราทำให้ข้อความสั่งนี้เป็นคอมเมนต์ โดยให้อยู่ภายในวงเล็บปีกกา วิธีนี้ถ้าข้อผิดพลาดเหมือนเดิมเกิดขึ้นอีก ในการทดสอบครั้งต่อมา การลบวงเล็บปีกกา จะง่ายกว่า การพิมพ์ข้อความสั่งวินิจฉัยขึ้นใหม่

ข้อผิดพลาดชนิดวนซ้ำขาดไปหนึ่งรอบ (off – by – one Loop Errors)

ข้อผิดพลาดเชิงตรรกะร่วมตรงที่สุด คือ ส่วนวนซ้ำ (loop) ซึ่งกระทำการเกินไปหนึ่งครั้ง หรือกระทำการน้อยไปหนึ่งครั้ง ถ้า while ลูปควบคุมโดย sentinel กระทำซ้ำเกินไปหนึ่งครั้งอาจเป็นข้อผิดพลาด ซึ่งประมวลผลค่า sentinel รวมไปกับข้อมูลปกติ

ถ้า while ลูปกระทำการดำเนินการนับจำนวนเราต้องมั่นใจว่าค่าเริ่มต้น (initial value) และค่าสุดท้าย (final value) ของตัวแปรควบคุมลูป (loop-control variable) กำหนดถูกต้อง ตัวอย่างเช่น ตัวส่วนวนซ้ำข้างล่างนี้ กระทำการ N+1 ครั้ง แทนที่จะเป็น N ครั้ง ถ้าเราต้องการให้ตัวส่วนวนซ้ำกระทำการ N ครั้งเท่านั้น ต้องเปลี่ยนเงื่อนไข while เป็น Count < N

```
Count := 0 ;
while Count <= N do
  begin
    Sum := Sum + Count ;
    Count : = Count + 1
  end {while}
```

การตรวจสอบเขตแดนของลูป (Checking Loop Boundaries)

เราสามารถตรวจว่าลูปถูกต้องหรือไม่ โดยการตรวจสอบเขตแดนของลูป นั่นคือ ค่าแรกและค่าสุดท้ายของตัวแปรควบคุมลูป สำหรับ for ลูป การประเมินผลอย่างรอบคอบนิพจน์เริ่มต้นและนิพจน์สุดท้าย เพื่อดูว่าค่าของมันถูกต้อง หลังจากนั้นแทนค่าเหล่านี้ทุกแห่งที่ตัวแปร counter ปรากฏในตัวส่วนวนซ้ำ และทวนสอบว่า เราได้ผลลัพธ์ตามที่คาดไว้ที่เขตแดน ตัวอย่างเช่น ใน for ลูป

```
Sum := 0 ;
.for I := K to N-K do
   Sum := Sum + Sqr (I)
```

ตรวจดูว่าค่าแรกของตัวแปร counter I ต้องเป็น K และค่าสุดท้ายต้องเป็น N-K ต่อไปตรวจว่าข้อความสั่งกำหนดค่า

```
Sum := Sum + Sqr (I)
```

ถูกต้องที่ค่าเขตแดน เมื่อ I มีค่าเท่ากับ K ค่าของ Sum คือ ค่ากำลังสองของ K และเมื่อ I เท่ากับ N-K, ค่าของ Sum คือค่า Sum ก่อนหน้านั้นบวกกับ $(N - K)^2$ การตรวจสอบสุดท้ายให้ N และ K มีค่าเล็ก (สมมติว่าเป็น 3 และ 1) และตามรอยการกระทำการส่วนวนซ้ำว่าคำนวณ Sum ถูกต้อง

การใช้โปรแกรมตรวจแก้จุดบกพร่อง (Using the Debugger)

โปรแกรม debugger หาตำแหน่งข้อผิดพลาดเชิงตรรกะสิ่งแวดล้อมรวมของ Turbo Pascal มีโปรแกรม debugger ของมันเอง ซึ่งทำให้เราโต้ตอบการกระทำการโปรแกรมได้ครั้งละหนึ่งข้อความสั่ง ขณะสังเกตการเปลี่ยนแปลงของตัวแปรหรือนิพจน์ซึ่งเลือกไว้ เราสามารถหยุดการกระทำการของโปรแกรมที่ข้อความสั่งหนึ่งได้ (breakpoints) เพื่อตรวจดูหรือเปลี่ยนค่าของตัวแปรเลือกก่อนกระทำการต่อไป

การใช้ Turbo Pascal debugger ให้เลือก Trace into option (ฟังก์ชันคีย์ F7) จากเมนู Run แถบการกระทำการ (execntion bar) จะปรากฏเหนือ บรรทัด begin ของโปรแกรมหลัก (main program) จากนั้นจะเลื่อนไปยังข้อความสั่งแรกของโปรแกรม

ถ้าเรากดฟังก์ชันคีย์ F7 ทุกครั้งที่เรากด F7 ข้อความสั่งใต้แถบกระทำการจะถูกกระทำการ และแถบกระทำการจะเลื่อนไปยังข้อความสั่งถัดไป การกดฟังก์ชันคีย์ F7 ซ้ำๆ กัน จะทำให้ debugger กระทำการโปรแกรมครั้งละหนึ่งข้อความสั่ง

·การทดสอบ (Testing)

หลังจากเราแก้ไขข้อผิดพลาดทั้งหมดแล้ว และโปรแกรมกระทำการ เช่นที่คาดไว้ ทดสอบโปรแกรมตลอดเพื่อให้มั่นใจว่ามันทำงานได้ ในหัวข้อ 4.6 เราได้อภิปรายการตามรอยอัลกอริทึม และแนะนำว่าเราควรจัดเซตของข้อมูลใช้ทดสอบ เพื่อให้มั่นใจว่าทุกวิธีที่เป็นไปได้ทั้งหมดถูกตามรอย สิ่งที่เหมือนกันเป็นจริงสำหรับโปรแกรมที่เสร็จสมบูรณ์ วิ่งทดสอบให้เพียงพอ เพื่อทวนสอบว่าโปรแกรมทำงานถูกต้อง สำหรับตัวอย่างที่เป็นตัวแทนของการจัดกลุ่มข้อมูลที่เป็นไปได้ทั้งหมด

แบบฝึกหัด 5.8 Self- Check

1. ทำไมส่วนของรหัสข้างล่างนี้จึงใช้ไม่ได้ (fail) และรหัสข้างล่างนี้ มีเขตแดนส่วนวนซ้ำ (loop boundaries) เป็นค่าอะไร

```
X := 10 ;
repeat
   X := X - 1 ;
   WriteLn (X, Sqrt (X))
until X < 0
```

5.9 ข้อผิดพลาดร่วมของการเขียนโปรแกรม (Commo Programming Errors)

โปรแกรมเมอร์มือใหม่บางครั้งอาจสับสน ข้อความสั่ง if และข้อความสั่ง while เพราะว่าทั้งคู่ประกอบด้วยเงื่อนไขให้ใช้ข้อความสั่ง if ทุกครั้ง เพื่อ implement ขั้นการตัดสินใจ และใช้ข้อความสั่ง while ทุกครั้ง เพื่อ implement ลูปแบบมีเงื่อนไขจงระมัดระวังเมื่อใช้ทดสอบความไม่เท่ากันเพื่อควบคุมการทำซ้ำของ while ลูป ตัวอย่างเช่น ลูปข้างล่างนี้ตั้งใจ

ประมวลผลรายการเปลี่ยนแปลงทั้งหมดของบัญชีธนาคาร เมื่อยอดเงินคงเหลือ (balance) มีค่าเป็นบวก

```
while Balance < > 0.0 do
    Update (Balance)
```

ถ้ายอดเงินคงเหลือ เริ่มจากจำนวนบวกไปเป็นจำนวนโดยไม่มีค่า 0.0 พอดี ลูปจะไม่จบ (เรียกว่าลูปอนันต์ (infinite loop)) ดังนั้นตัวอย่างนี้ ลูปควรเขียนดังนี้

```
while Balance > 0 do
   Update (Balance)
```

จงทวนสอบให้แน่ใจว่าเงื่อนไขทำซ้ำสำหรับ while ลูปในที่สุดจะเป็นเท็จ ถ้าเราใช้ลูปควบคุมโดยค่า sentinel อย่าลืมแจ้งผู้ใช้โปรแกรมว่าค่าอะไรที่ใช้เป็น sentinel ทำให้มั่นใจว่าค่า sentincl ไม่สับสนกับหน่วยข้อมูลปกติ

ถ้าตัวส่วนวนซ้ำ (loop body) มีมากกว่าหนึ่งข้อความสั่ง จำไว้ว่าต้องใส่คู่ begin - end เสมอ (ยกเว้น repeat - until ลูป) กรณีอื่นๆ จะมีเฉพาะข้อความสั่งแรกเท่านั้นที่จะถูกทำซ้ำ และข้อความสั่งที่เหลือ จะไม่ถูกกระทำการ เมื่อออกจากลูป

ลูปข้างล่างนี้จะไม่จบ เพราะว่าขั้นตอนซึ่งปรับ (updata) ตัวแปรควบคุมลูป ไม่ได้เป็นส่วนหนึ่งของตัวส่วนวนซ้ำ (loop body) โปรแกรมจะทำงานต่อเนื่องไป เพื่อพิมพ์ค่าแรกของPower จนกระทั่งเราสั่งคอมพิวเตอร์ให้จบการกระทำการของมัน การจบโปรแกรมให้กด Ctrl-C

```
while Power <= 10000 do
   WriteLn ('Next power of N is ', Power : 6) ;
   Power := Power * N
```

จงแน่ใจว่าได้กำหนดค่าเริ่มต้น 0 ให้ตัวแปรสะสมใช้สำหรับการสะสม ผลบวกโดยการบวกซ้ำๆ กัน และกำหนดค่าเริ่มต้น 1 ให้กับตัวแปรซึ่งใช้สำหรับสะสมผลคูณ โดยการคูณซ้ำๆ กัน การไม่มีขั้นตอนนี้ นำไปสู่ผลลัพธ์ไม่ถูกต้อง

ค่าของตัวแปร counter ในข้อความสั่ง for อาจจะเพิ่มขึ้นครั้งละ 1 (รูปแบบ to) หรือลดลงครั้งละ 1 (รูปแบบ downto) หลังจากการทำซ้ำแต่ละครั้ง ถ้า Larger มีค่ามากกว่า Smaller ข้อความสั่ง WriteLn ข้างล่างนี้จะไม่กระทำการ เพราะว่าค่าเริ่มต้น ซึ่งกำหนดให้กับ I มีค่ามากกว่า ค่าสุดท้ายของมัน

```
for    I := Larger to Smaller do
       WriteLn (I)
```

ในทำนองเดียวกัน ข้อความสั่ง WriteLn ถัดไปจะไม่กระทำการ เพราะว่า ค่าเริ่มต้นซึ่งกำหนดให้ I มีค่าน้อยกว่าค่าสุดท้ายของมัน

```
for    I := Smaller downto Larger do
       WriteLn (I)
```

ลูป repeat –until จะกระทำการเสมออย่างน้อยที่สุดหนึ่งครั้ง ให้ใช้ข้อความสั่ง repeat เฉพาะเมื่อแน่ใจว่าไม่มีทางเป็นไปได้ที่การทำซ้ำลูป จะเป็นศูนย์ครั้ง กรณีอื่นๆ ใช้ while ลูป

จงแน่ใจว่าเพื่อให้ตามรอย การซ้อนในแต่ละชุดของลูปทำอย่างระมัดระวัง ตรวจสอบตัวแปรควบคุมลูปทั้งชั้นนอกและชั้นใน ตัวแปรควบคุมลูป ของข้อความสั่ง for ชั้นนอกเปลี่ยนแปลงไม่ได้ภายในข้อความสั่ง for ของชั้นในและเราไม่สามารถใช้ตัวแปรควบคุมลูปชื่อเหมือนกันในข้อความสั่ง for สองชุด ภายในการซ้อนในชุดเดียวกัน

ข้อสรุปตัวสร้างใหม่ของ Pascal

Construct	Effect
ข้อความสั่ง while	

Sum := 0 ;
while Sum <= MaxSum do
  begin
  Write ('Next integer > ') ;
  ReadLn (Next) :
  Sum := Sum + Next
end {while}
	อ่านกลุ่มของหน่วยข้อมูล อินพุต และผลบวกของมันสะสมใน Sum การประมวลนี้หยุดเมื่อผลบวกสะสมมีค่ามากกว่า MaxSum

ข้อความสั่ง for	

for CurMonth := 3 to 9 do
  begin
  ReadLn (MonthSales) ;
	ตัวส่วนวนซ้ำ ถูกทำซ้ำสำหรับแต่ละค่าของ CurMonth จาก 3 to 9 นับทั้งต้นและท้าย

Construct	Effect
YearSales := YearSales + MonthSales end {for}	แต่ละเดือน ค่าของ MonthSales ถูกอ่านและบวกกับ YearSales
ข้อความสั่ง repeat Sum : = 0 ; repeat Sum := Sum + NextInt WriteLn (NextInt); until NextInt = 0	อ่านค่า Integer และผลบวกของมันสะสมใน Sum การประมวลผล จบหลังจากค่าศูนย์ ตัวแรกถูกอ่าน

แบบฝึกหัด Quick – Check

1. ถ้าเงื่อนไขลูป เป็นตัวแปรแบบบูล ลูปชนิดนี้มีชื่อเรียกว่าอะไร

2. มันเป็นข้อผิดพลาดถ้าตัวส่วนวนซ้ำของ while ลูปไม่เคยกระทำการเลย ข้อความที่กล่าวข้างต้นนี้จริง (true) หรือเท็จ (false).

3. ขั้นตอน priming สำหรับลูปควบคุมโดยค่า sentinel เป็นข้อความสั่งชนิดใด และข้อความสั่งนี้ จะวางตำแหน่งที่ใด อะไรคือวัตถุประสงค์ของข้อความสั่งที่คล้ายกันในตัวส่วนวนซ้ำ

4. ค่า sentincl จะเป็นค่าสุดท้ายเสมอ ที่บวกกับผลบวกซึ่งกำลังสะสมในลูปควบคุมด้วยค่า sentinel ข้อความที่กล่าวข้างต้นนี้จริง (true) หรือเท็จ (false)

5. คุณสมบัติต่อไปนี้เป็นลูปชนิดใด (for, repeat, while)
   a) กระทำการอย่างน้อยที่สุด หนึ่งครั้ง
   b) เป็นรูปแบบทั่วไปมากที่สุด
   c) ยุบรวมรหัส สำหรับลูปการนับจำนวน
   d) ควรใช้ในโปรแกรม ซึ่งใช้งานด้วยรายการเลือก (menu – driven program)

6. ส่วนของโปรแกรมต่อไปนี้แสดงผลอะไร

```
Product := 1 ;
Counter := 2 ;
```

```
While Counter <= 5 do
   begin
      Product := Product * Counter ;
      Counter := Counter + 1
   end {while}
WriteLn (Product)
```

7. ลูปซึ่งกระทำการคำนวณไม่ถูกต้อง ระหว่างการวนซ้ำครั้งสุดท้ายของมัน มีปัญหาที่ใด

8. ระหว่างการกระทำการของโปรแกรมเซกเมนต์ ต่อไปนี้

```
for I : = 1 to 10 do
   begin
      for J := 1 to I do
         Write (I * J) ;
      WriteLn
   end
```

a) ข้อความสั่ง Write ถูกกระทำการกี่ครั้ง

b) ข้อความสั่ง WriteLn ถูกกระทำการกี่ครั้ง

c) ค่าสุดท้ายซึ่งแสดงผลคืออะไร

9. จงบอกเอาต์พุตของเซกเมนต์ ข้างล่างนี้

```
N := 10 ;
for I := , to N do
   begin
      Write (I : 3) ;
      N : = N-I
   end {for}
WriteLn (N : 4)
```

คำถามทบทวน (Review Questions)

1. ค่า sentinel แตกต่างจาก program flag ในความหมายของการควบคุมลูปอย่างไร

2. การใช้ที่ถูกต้องของค่า sentinel เมื่ออ่านข้อมูล ข้อความสั่งอินพุต ควรปรากฏที่ใด

3. เราสามารถใช้ while ลูป แทนที่ for หรือ repeat ลูปได้เสมอหรือไม่ ให้อธิบาย

4. จงพิจารณา โปรแกรมเซกเมนต์ ต่อไปนี้

```
Count := 0 ;
for I := 1 to N do
   begin
      Read (X) ;
      If x = I then
         Count := Count + 1
   end {for}
```

a) จงเขียน while ลูปที่มีความหมายเหมือนกับ for ลูป

b) จงเขียน repeat – until ลูป ที่มีความหมายเหมือนกับ for ลูป

โครงการเขียนโปรแกรม (Programming Projects)

1. จงเขียนโปรแกรมหาค่าเล็กที่สุด ใหญ่ที่สุด และค่าเฉลี่ยในกลุ่มของ N จำนวน ให้อ่านค่าของ N ก่อนการอ่านค่าแต่ละตัวในกลุ่มของเลข N จำนวน

2. จงดัดแปร ข้อ 1 ให้คำนวณและแสดงผลทั้งพิสัย และส่วนเบี่ยนเบนมาตรฐานของกลุ่มข้อมูลการคำนวณส่วนเบี่ยงเบนมาตรฐาน ให้สะสมผลบวกของข้อมูล (Sum) และผลบวกของกำลังสองของค่าข้อมูล (Sum Squares) ใน main ลูป หลังจากออกจากลูป ใช้สูตร

$$\text{Standard deviation} = \sqrt{\frac{\text{SumSquares} - \text{Sum}^2}{N}}$$

3. จงเขียนโปรแกรม เพื่อทำปฏิทินรายปี โปรแกรมรับปีและวันของสัปดาห์สำหรับวันที่ 1 มกราคมของปีนั้น (1 = วันอาทิตย์, 7 = วันเสาร์) ข้อควรจำ เดือนกุมภาพันธ์มี 29 วัน ถ้า ปี ค.ศ.หารด้วย 4 ลงตัว

ปฏิทินพิมพ์ในรูปแบบข้างล่างนี้ (สำหรับแต่ละเดือน)

January

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

4. a) จงเขียนโปรแกรมอ่านกลุ่มของคะแนนสอบ ซึ่งมีค่าระหว่าง 0 ถึง 100 โปรแกรมนี้นับและพิมพ์จำนวนของคะแนนดีเยี่ยม (90-100), จำนวนของคะแนนที่สอบผ่าน (60-89) และจำนวนของคะแนนสอบไม่ผ่าน (0-59) จากนั้นแสดงผลประเภทแต่ละกลุ่มคะแนน ให้ทดสอบด้วยข้อมูลต่อไปนี้

63 75 72 72 78 67 80 63 75
90 89 43 59 99 82 12 100

b) จงดัดแปรโปรแกรมในข้อ (a) เพื่อให้แสดงผลค่าเฉลี่ยของคะแนนสอบ (เลขจำนวนจริง) ที่ตอนจบของการวิ่งโปรแกรม

5. จงเขียนโปรแกรม เพื่อประมวลผลบัตรเวลาทำงานของพนักงานรายสัปดาห์ สำหรับพนักงานทุกคนในบริษัทแห่งหนึ่ง พนักงานแต่ละคนมีหน่วยข้อมูลสามชุด : เลขประจำตัวพนักงาน, อัตราค่าจ้างรายชั่วโมง และ จำนวนชั่วโมงทำงานระหว่างสัปดาห์ที่กำหนด พนักงานแต่ละคนทำงานเกิน 40 ชั่วโมง เฉพาะส่วนที่เกิน 40 ชั่วโมง จะได้ค่าตอบแทนในอัตราหนึ่งเท่าครึ่งของค่าตอบแทนปกติ ภาษี 3.625% ของเงินได้รวม จะถูกหักออกเอาต์พุตของโปรแกรมให้แสดงหมายเลขประจำตัวของพนักงานและเงินได้สุทธิ แสดงผลเงินเดือนรวมทั้งหมด และค่าเฉลี่ยซึ่งได้รับที่ตอนจบของการวิ่งโปรแกรม