บทที่ 6
การเขียนโปรแกรมส่วนจำเพาะ
(Modular Programming)

6.1 แนะนำรายการพารามิเตอร์
6.2 การกลับคืนสารสนเทศจากกระบวนงาน
6.3 กฎวากยสัมพันธ์สำหรับกระบวนงานที่มีรายการพารามิเตอร์
6.4 สโคปของไอเดนติไฟเออร์
6.5 ฟังก์ชัน : ส่วนจำเพาะ ซึ่งกลับคืนผลลัพธ์หนึ่งค่า
6.6 การออกแบบการแบ่งละเอียดที่มีฟังก์ชันและกระบวนงาน
กรณีศึกษา : ปัญหาผลบวกและค่าเฉลี่ยทั่วไป
6.7 การแก้จุดบกพร่องและการทดสอบโปรแกรมที่มีส่วนจำเพาะ
6.8 ฟังก์ชันเรียกซ้ำ
6.9 ข้อผิดพลาดร่วมของการเขียนโปรแกรม

การออกแบบตัวสร้างโปรแกรมอย่างรอบคอบโดยใช้กระบวนงาน และฟังก์ชันใช้คุณสมบัติร่วมบางอย่างของระบบเครื่องเสียง (stereo system) อุปกรณ์ประกอบเครื่องเสียงแต่ละตัวเป็นอุปกรณ์อิสระ (independent device) ซึ่งกระทำการดำเนินการเฉพาะหนึ่งอย่าง เราอาจทราบวัตถุประสงค์ของอุปกรณ์แต่ละตัว แต่เราไม่จำเป็นต้องทราบว่าส่วนประกอบแต่ละตัวประกอบด้วยส่วนอิเล็กทรอนิกส์อะไร หรือแต่ละตัวมันทำหน้าที่เล่นหรือบันทึกเพลงอย่างไร

สัญญาณเสียงอิเล็กทรอนิกส์เคลื่อนย้ายกลับไปกลับมาผ่านสายไฟ (fire) ซึ่งเชื่อมต่อส่วนประกอบเครื่องเสียงผ่านทางเต้าเสียบ (plugs) ด้านหลังของเครื่องรับเครื่องเสียงที่ทำเครื่องหมายว่าเป็นอินพุตหรือเป็นเอาต์พุต สายไฟซึ่งติดกับเต้าเสียบอินพุต นำสัญญาณอิเล็กทรอนิกส์ไปยังตัวรับที่ซึ่งมันจะถูกประมวลผล สัญญาณเหล่านี้อาจมาจากเครื่องเล่นเทป เครื่องรับวิทยุ (tuner) หรือเครื่องเล่น CD

เครื่องรับส่งสัญญาณอิเล็กทรอนิกส์ไหม่ผ่านทางเต้าเสียบเอาต์พุตไปยังลำโพง (speaker) หรือกลับไปยังเครื่องเล่นเทปสำหรับการบันทึก

การแนะนำกระบวนงานในบทที่ 3 แสดงให้เห็นแล้วว่าจะเขียนส่วนประกอบโปรแกรมแยกต่างหากอย่างไร เป็นกระบวนงานของโปรแกรม กระบวนงาน (procedure) เหล่านี้ (หรือฟังก์ชัน) สมนัยกับแต่ละขั้นตอนในการแก้ปัญหา เนื้อหาในบทนี้เราจะเรียนรู้ว่าการเชื่อมต่อส่วนจำเพาะ (modules) เพื่อสร้างหนึ่งระบบโปรแกรมทำอย่างไร การจัดการส่วนต่างๆ ซึ่งทำให้โปรแกรมของเราทำหน้าที่คล้ายระบบเครื่องเสียง เช่น มันส่งสารสนเทศจากส่วนจำเพาะหนึ่งชุด ไปยังส่วนจำเพาะอีกหนึ่งชุด

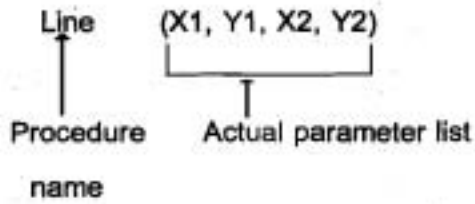
6.1 แนะนำรายการพารามิเตอร์ (Introduction to Parameter Lists)

รายการพารามิเตอร์เตรียมการเชื่อมโยงการสื่อสารระหว่างโปรแกรมหลักกับส่วนจำเพาะของมัน พารามิเตอร์ทำให้กระบวนงานและฟังก์ชันมีความคล่องตัวมากขึ้น เพราะว่ามันสามารถทำให้ส่วนจำเพาะหนึ่งชุด จัดดำเนินการกับข้อมูลที่แตกต่างกันแต่ละครั้งที่มันถูกเรียก หัวข้อนี้อธิบายว่า โปรแกรมเมอร์ใช้พารามิเตอร์เพื่อส่งสารสนเทศระหว่างโปรแกรมและส่วนจำเพาะของมันอย่างไร หรือระหว่างส่วนจำเพาะสองชุดอย่างไร

พารามิเตอร์จริงและพารามิเตอร์รูปนัย (Actual and Formal Parameters)

ข้อความสั่งเรียกกระบวนงานแต่ละชุด มีสองส่วน : ชื่อกระบวนงาน (procedure name) และรายการพารามิเตอร์ (actual parameter list)

ตัวอย่างเช่น กระบวนงานเชิงกราฟิก ชื่อ Line ลากเส้นจากจุดหนึ่งไปยังอีกจุดหนึ่ง และข้อความสั่งเรียกกระบวนงานของมัน



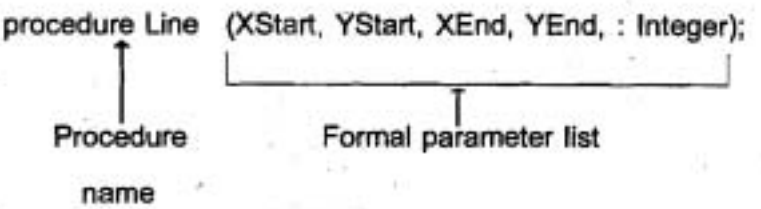
ประกอบด้วยชื่อกระบวนงาน Line และรายการพารามิเตอร์จริง (X1, Y1, X2, Y2) ค่าของพารามิเตอร์จริง 4 ตัวนี้ถูกส่งไปยังกระบวนงาน Line ซึ่งลากเส้นจากจุด (X1, Y1) ไปยังจุด (X2, Y2) จงลากเส้นต่อไปจากจุด (X2, Y2) ไปยังจุด (X3, Y3) เราใช้ข้อความสั่งเรียกกระบวนงานดังนี้

Line (X2, Y2, X3, Y3)

ในการเรียกกระบวนงานแต่ละครั้ง โปรแกรมเมอร์เตรียมการกระบวนงาน Line ด้วยตัวแปรสี่ตัว หรือค่าซึ่งแทนพิกัด (coordinates) X, Y ของสองจุดบนจอภาพ

เนื่องจากพิกัดสี่ตัวนี้ สามารถเปลี่ยนแปลงได้ทุกครั้งที่ Line ถูกเรียก เราจึงต้องแทนมันด้วยวิธีใดก็ตามในการประกาศกระบวนงาน

การทำสิ่งนี้ เราใช้ชื่อดัมมี (dummy names) เรียกว่า พารามิเตอร์รูปนัย (formal parameter)



รายการพารามิเตอร์รูปนัย แสดงให้เห็นพารามิเตอร์รูปนัยสี่ตัว ซึ่งใช้ภายในกระบวนงานเพื่อแทนพิกัด (coordinates) X, Y ของจุดปลาย (end points) ของเส้นที่จะวาดคู่ XStart, YSart แทนจุดปลายด้านหนึ่ง และคู่ XEnd, YEnd แทนจุดปลายอีกด้านหนึ่ง รายการพารามิเตอร์ รูปนัยนั้นระบุแบบชนิดข้อมูลของพารามิเตอร์เหล่านี้เป็น Integer

รายการพารามิเตอร์ หมายถึง รายการของตัวแปร (หรือค่า) อยู่ภายในวงเล็บเพื่อส่งไปยังกระบวนงาน เขียนตามหลังชื่อกระบวนงานในการเรียกกระบวนงาน (Actual parameter list is a parenthesized list of variables (or values) passed to the procedure; follows the procedure name in the procedure call.)

รายการพารามิเตอร์รูปนัย หมายถึง รายการของชื่อดัมมี (พร้อมกับแบบชนิดข้อมูล) ซึ่งใช้ในกระบวนงานเพื่อแทนพารามิเตอร์จริง เขียนตามหลังชื่อกระบวนงานในหัวเรื่องกระบวนงาน (Formal parameter list is a parenthesized list of dummy names (with data types) used in the procedure to represent actual parameters; follows the procedure name in the procedure heading.)

การสมนัยระหว่างพารามิเตอร์จริงกับพารามิเตอร์รูปนัย (Correspondence Between Actual and Formal Parameters)

กระบวนงาน Line ไม่รู้ว่าค่าที่มันจะรับมาคืออะไร จนกระทั่งมันถูกเรียก โปรแกรมเรียก (calling program) ส่งสารสนเทศที่ต้องการของ Line ผ่านทางรายการพารามิเตอร์จริง ซึ่งจับคู่ (matching) พารามิเตอร์จริงแต่ละตัวกับพารามิเตอร์รูปนัย ซึ่งสมนัยกับตัวมัน

รูป 6.1 แสดงให้เห็นส่วนของโปรแกรมซึ่งประกอบด้วย กระบวนงาน Line และการเรียกกระบวนงาน ลูกศร แสดงการไหล (flow) ของสารสนเทศระหว่างพารามิเตอร์จริงและพารามิเตอร์รูปนัยสี่ตัว

การสมนัยของพารามิเตอร์สรุปดังนี้

Actual Parameter	corresponds to	**Fomal Parameter**
X1		XStart
Y1		YStart
X2		XEnd
Y2		YEnd

```
program Main;
  var
    X1, X2, Y1, Y2, X3, Y3 : Integer;     {Main program variables}

                    {Formal parameters}
  procedure Line (XStart, YStart, XEnd, YEnd : Integer);
  . . .
  end; {Line}
begin {Main}
  Line  (X1, Y1, X2, Y2);        {Actual parameters}
  Line  (X2, Y2, X3, Y3);
  . . .
end.  {Main}
```

รูป 6.1 โปรแกรมหลักที่มีการเรียกกระบวนงาน Line

รูป 6.1 แสดงให้เห็นว่าค่าของตัวแปร X1 ในโปรแกรมหลักส่งไปยังพารามิเตอร์รูปนัย XStart, ค่าของตัวแปร Y1 ส่งไปยังพารามิเตอร์รูปนัย YStart เช่นนี้เรื่อยไป ถึงแม้ว่าจะไม่ได้แสดงเป็นตัวเลข พารามิเตอร์จริงทั้งสี่ตัว ต้องมีค่าซึ่งกำหนดไว้แล้วก่อนที่จะกระทำการเรียกกระบวนงาน

โปรดสังเกตว่า การสมนัยของพารามิเตอร์รูปนัยและพารามิเตอร์จริงมีชื่อแตกต่างกัน ในรูป 6.1 สิ่งนี้ไม่มีปัญหาแต่อย่างใด เพราะว่า การสมนัยของพารามิเตอร์ถูกกำหนดโดยตำแหน่ง (by position) ในรายการพารามิเตอร์แต่ละตัว ไม่ใช่กำหนดโดยชื่อ

ตัวแปรเฉพาะที่ หมายถึง ตัวแปรซึ่งประกาศในกระบวนงาน ไม่ใช่ตัวแปรซึ่งประกาศในโปรแกรมหลัก (Local variables are variables declared in a procedure, not in the main program.)

ตัวอย่าง 6.1

กระบวนงาน ReportSumAve ในรูป 6.2 คำนวณและแสดงผลบวกและค่าเฉลี่ยของค่าชนิด Real สองตัว ซึ่งส่งมายังกระบวนงานในคอมเมนต์ {input} ระบุว่า พารามิเตอร์รูปนัย Num1 และ Num2 เป็นอินพุตของกระบวนงาน

ตัวแปรเฉพาะที่สองตัวชื่อ Sum และ Average ถูกประกาศในกระบวนงาน ซึ่งเป็นที่ซึ่งตัวแปรเฉพาะที่ของกระบวนงานเท่านั้น สามารถเข้าถึงได้ ข้อความสั่ง

```
Sum     := Num1 + Num2;
Average := Sum / 2.0;
```

กำหนดค่าให้ตัวแปรเฉพาะที่เป็นดังนี้ ผลบวกของค่าซึ่งส่งไปยังพารามิเตอร์ Num1 และ Num2 เก็บใน Sum และค่าเฉลี่ยของมันเก็บใน Average

สำหรับข้อความสั่งเรียกกระบวนงาน

```
ReportSumAve (6.5, 3.5)
```

ค่า 6.5 ส่งไปยังพารามิเตอร์รูปนัย Num1 และค่า 35 ส่งไปยังพารามิเตอร์รูปนัย Num2 ค่าซึ่งกำหนดให้ Sum คือ 10.0 และค่าซึ่งกำหนดให้ Average คือ 5.0 กระบวนงานนี้แสดงผลสองค่า การสมนัยระหว่างพารามิเตอร์จริงกับพารามิเตอร์รูปนัยเป็นดังนี้

Actual Parameter	corresponds to	**Fomal Parameter**
6.5		Num1
3.5		Num2

สำหรับข้อความสั่งเรียกกระบวนงาน

```
ReportSumAve (X, Y)
```

ค่าของ X ส่งไปยังพารามิเตอร์รูปนัย Num1 และค่าของ Y ส่งไปยังพารามิเตอร์รูปนัย Num2

Actual Parameter	corresponds to	Fomal Parameter
X		Num1
Y		Num2

```
procedure ReportSumAve (Num1, Num2    {input} : Real);
{
   Computes and displays the sum and average of Num1 and Num2.
   Pre : Num1 and Num2 are assigned values.
   Post :   The sum and average value of Num1 and Num2 are computed
            and displayed.
}
   var
      Sum,              {sum of Num1, Num2}
      Average : Real;   {average of Num1, Num2}

begin {ReportSumAve}
   Sum := Num1 + Num2;
   Average := Sum/2.0;
   WriteLn ('The sum is ' , Sum ; 4 : 2);
   WriteLn ('The average is ' , Average : 4 : 2}
end; {ReportSumAve}
```

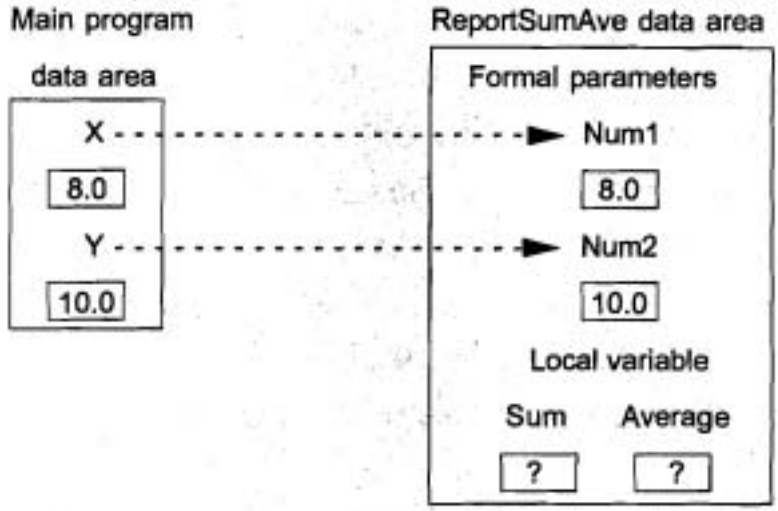
รูป 6.2 กระบวนงานแสดงผล Sum และ Average

รูป 6.3 แสดงให้เห็นพื้นที่ข้อมูล (data area) ของโปรแกรมหลักและพื้นที่ข้อมูลสำหรับกระบวนงาน ReportSumAve หลังจากข้อความสั่งเรียกกระทำการ ค่า 8.0 และ 10.0 ส่งไปยังพารามิเตอร์รูปนัย Num1 และ Num2 ตามลำดับ

ตัวแปรเฉพาะที่ Sum และ Average ตอนเริ่มต้นยังไม่ถูกนิยาม การกระทำการของตัวกระบวนงาน (procedure body) เปลี่ยนค่าของตัวแปรสองตัวนี้เป็น 18.0 และ 9.0 ตามลำดับ

พื้นที่ข้อมูลของกระบวนงาน (The Procedure Data Area)

ทุกครั้งที่ข้อความสั่งเรียกกระบวนงานถูกกระทำการ พื้นที่ของหน่วยความจำจะจัดสรรสำหรับหน่วยเก็บของข้อมูลของกระบวนงานนั้น



รูป 6.3 พื้นที่ข้อมูลหลังจากเรียกกระบวนงาน ReportSumAve

พื้นที่ข้อมูลของกระบวนงาน ได้แก่ เซลล์หน่วยเก็บ (storage cells) สำหรับพารามิเตอร์รูปนัยของมัน และตัวแปรเฉพาะที่หรือค่าคงตัวใดๆ ก็ตามซึ่งอาจถูกประกาศในกระบวนงาน เมื่อใดก็ตามที่จบกระบวนงานพื้นที่ข้อมูลของกระบวนงานจะถูกลบทิ้ง เมื่อกระบวนงานถูกเรียกอีกครั้งหนึ่ง พื้นที่ข้อมูลถูกสร้างใหม่เป็นพื้นที่ว่าง (ค่าทั้งหมดไม่ถูกนิยาม)

ข้อผิดพลาดของการแทนที่พารามิเตอร์ไม่ถูกต้อง (Illegal Parameters Substitution Errors)

แบบชนิดข้อมูลของพารามิเตอร์จริงแต่ละตัวต้องกำหนดค่าที่เข้ากันได้ (assignment compatible) กับแบบชนิดข้อมูลของพารามิเตอร์รูปนัยซึ่งสมนัยกันของมัน (ดูหัวข้อ 2.5)

มิฉะนั้นจะเกิดข้อผิดพลาดวากยสัมพันธ์ชนิด mismatch คอมไพเลอร์ Pascal รู้ได้อย่างไรว่าพารามิเตอร์จริงมีแบบชนิดข้อมูลถูกต้องหรือไม่

คอมไพเลอร์ Pascal รู้ว่าพารามิเตอร์จริงแต่ละตัวจะต้องมีแบบชนิดข้อมูลเป็นอย่างไร เพราะว่า การประกาศของกระบวนงานต้องอยู่ก่อนการเรียกครั้งแรกของมัน รายการพารามิเตอร์รูปนัย (ในส่วนการประกาศกระบวนงาน) กำหนดแบบชนิดข้อมูลของพารามิเตอร์แต่ละตัวของกระบวนงาน

สไตล์ของโปรแกรม (Program Style)

การเลือกชื่อของพารามิเตอร์รูปนัย (Choosing Formal Parameter Names)

ถึงแม้ว่าเราสามารถเลือกไอเดนติไฟเออร์ที่ถูกต้องใดๆ เป็นชื่อพารามิเตอร์รูปนัย เราควรทำตามข้อตกลงของการเลือกชื่อซึ่งช่วยให้เป็นเอกสารใช้พารามิเตอร์รูปนัย ข้อควรจำการสมนัยระหว่างพารามิเตอร์จริงและพารามิเตอร์รูปนัย กำหนดโดยตำแหน่งในรายการพารามิเตอร์อย่างเดียว โดยไม่สนใจว่าใช้ชื่ออะไร

วัตถุประสงค์อย่างหนึ่งของกระบวนงานและฟังก์ชันคือความสะดวกในการนำส่วนจำเพาะ (modules) ที่เขียนและทดสอบก่อนหน้าแล้วกลับมาใช้ใหม่ (reuse) พยายามเลือกชื่อพารามิเตอร์รูปนัยที่มีความหมายและใช้ทั่วไป ไม่ใช่ชื่อเฉพาะซึ่งใช้กับโปรแกรมประยุกต์หนึ่งงาน

เงื่อนไขก่อนและเงื่อนไขหลัง (Preconditions and Postconditions)

คอมเมนต์หลายบรรทัดที่ตอนต้นของกระบวนงาน ReportSumAve อธิบายการดำเนินการของกระบวนงาน สำหรับคอมเมนต์ซึ่งมีหลายบรรทัด ใช้รูปแบบดังนี้

```
{
   . . . comments . . .
}
```

บรรทัดคอมเมนต์

Pre : Num1 and Num2 are assigned values.

อธิบายเงื่อนไขซึ่งเป็นจริงก่อนกระบวนงานถูกเรียก เงื่อนไขนี้เรียกว่า **เงื่อนไขก่อน** (precondition) บรรทัดคอมเมนต์

Post : The sum and average value of Num1 and Num2 are computed and displayed.

อธิบายเงื่อนไขซึ่งเป็นจริงหลังจากการกระทำการกระบวนงานเสร็จสิ้น เงื่อนไขนี้เรียกว่า **เงื่อนไขหลัง** (postcondition)

เงื่อนไขก่อนและเงื่อนไขหลังเป็นเอกสารการดำเนินการของกระบวนงานให้กับโปรแกรมเมอร์อื่นๆ ซึ่งจะใช้กระบวนงานนี้

ตัวอย่างเช่น เงื่อนไขก่อนบอกโปรแกรมเมอร์ว่ามีอะไรต้องทำก่อนกระบวนงานถูกเรียก ในกรณีนี้ ค่าข้อมูลสองตัวต้องถูกกำหนด หรืออ่านเข้าไปยังพารามิเตอร์จริงก่อนการเรียก ReportSumAve เงื่อนไขหลัง บอกโปรแกรมเมอร์ถึงผลลัพธ์ของการกระทำการของกระบวนงานบนพารามิเตอร์ของมัน ในกรณีนี้คือ ผลบวกและค่าเฉลี่ยถูกคำนวณและแสดงผล

เงื่อนไขก่อน หมายถึง เงื่อนไขซึ่งสมมติว่าเป็นจริงก่อนเรียกกระบวนงานหรือฟังก์ชัน (Precondition is a condition assumed to be true before a procedure or function call.)

เงื่อนไขหลัง หมายถึง เงื่อนไขซึ่งสมมติว่าเป็นจริง หลังจากกระทำการกระบวนงานหรือฟังก์ชัน (Postcondition is a condition assumed to be true after a procedure or function executes.)

แบบฝึกหัด 6.1 Self-Check

1. จงพิจารณากระบวนงาน Down

```
procedure Down (N : Integer);
begin {Down}
   while N > o do
      begin
         Write (N);
         N := N - 1
      end {while}
```

a) เมื่อกระทำการข้อความสั่งเรียกกระบวนงาน ดังนี้

Down (3) เอาต์พุตคืออะไร

b) ถ้า M มีค่าเท่ากับ 5 เกิดอะไรขึ้นเมื่อกระทำการข้อความสั่ง เรียกกระบวนงาน Down (M)

c) พารามิเตอร์จริง M มีค่าเท่ากับอะไร หลังจากการกระทำการกระบวนงาน

d) ตัวแปร M ควรประกาศไว้ที่ใด และควรจะมีแบบชนิดข้อมูลเป็นอะไร

2. จงเขียนเงื่อนไขก่อนและเงื่อนไขหลังสำหรับกระบวนงาน Down ในแบบฝึกหัดข้อ 1

เขียนโปรแกรม

1. จงเขียนกระบวนงานคำนวณและแสดงผล รากที่สองของพารามิเตอร์รูปนัย X ถ้า X มีค่ามากกว่าหรือเท่ากับศูนย์ กรณีอื่นๆ กระบวนงานควรแสดงผลข้อความระบุความผิดพลาด (error message)

6.2 การกลับคืนสารสนเทศจากกระบวนงาน (Returning Information from Procedure)

ในหัวข้อ 6.1 เราส่งสารสนเทศจากโปรแกรมหลักไปยังกระบวนงานที่ซึ่งสารสนเทศจะถูกประมวลผล กระบวนงานไม่เพียงแต่ประมวลผลสารสนเทศเท่านั้น แต่กระบวนงานยังกลับคืน (return) สารสนเทศไปยังโปรแกรมหลักหรือกระบวนงานอีกชุดหนึ่งได้เพื่อให้ประมวลผลเพิ่มเติม หัวข้อนี้จะอธิบายว่า กระบวนงานกลับคืนสารสนเทศไปยังส่วนจำเพาะเรียก (calling modules) ได้อย่างไร

พารามิเตอร์ตัวแปรและพารามิเตอร์ค่า (Variable and Values Parameters)

อีกครั้งหนึ่ง ตัวเชื่อมโยงการสื่อสารระหว่างกระบวนงานและส่วนจำเพาะเรียกคือรายการพารามิเตอร์

พารามิเตอร์รูปนัยแบ่งออกเป็นสองชนิด คือ พารามิเตอร์จริง และพารามิเตอร์ตัวแปร

พารามิเตอร์จริง รับ (receives) สารสนเทศซึ่งส่งมายังกระบวนงาน

พารามิเตอร์ตัวแปร กลับคืน (returns) ผลลัพธ์ของกระบวนงาน โดยการเปลี่ยนแปลงค่าพารามิเตอร์จริง ซึ่งสมนัยกับมัน จากหัวข้อที่ผ่านมา เราใช้แต่พารามิเตอร์จริงเท่านั้น

พารามิเตอร์จริง หมายถึง พารามิเตอร์รูปนัยชนิดหนึ่งซึ่งรับสารสนเทศส่งเข้ามายังกระบวนงาน (Value parameter is a type of formal parameter that receives information passed into procedure.)

พารามิเตอร์ตัวแปร หมายถึง พารามิเตอร์รูปนัยชนิดหนึ่งซึ่งกลับคืนผลลัพธ์ของกระบวนงาน (Variable parameter is a type of formal parameter that returns a procedure result.)

ตัวอย่าง 6.2 แสดงให้เห็นความแตกต่างระหว่างพารามิเตอร์ค่า กับ พารามิเตอร์ตัวแปร เช่นที่เราติดตามตัวอย่าง เมื่อกระทำการเรียกกระบวนงาน คอมพิวเตอร์จัดสรรเนื้อที่หน่วยความจำในพื้นที่ข้อมูลของกระบวนงานสำหรับพารามิเตอร์รูปนัยแต่ละตัว สำหรับอินพุตพารามิเตอร์ ทำสำเนาค่าของอินพุตพารามิเตอร์จริงแต่ละตัวเก็บในเซลล์หน่วยความจำซึ่งจัดสรรให้กับพารามิเตอร์รูปนัยซึ่งสมนัยของมัน ตัวกระบวนงาน (procedure body) จัดดำเนินการค่านี้

ตัวอย่าง 6.2

กระบวนงาน ComputeSumAve ในรูป 6.4 คล้ายกับกระบวนงาน ReportSumAve ในรูป 6.2 ข้อแตกต่างที่สำคัญคือ ชุดแรก (รูป 6.4) มีพารามิเตอร์รูปนัยสี่ตัว ได้แก่ สองตัวสำหรับอินพุต (Num1 และ Num2) และอีกสองตัวสำหรับเอาต์พุต (Sum และ Average)

กระบวนงาน ComputesumAve คำนวณผลบวกและค่าเฉลี่ยของอินพุตของมัน แต่ไม่แสดงผล ค่าเหล่านี้ถูกกำหนดให้กับพารามิเตอร์รูปนัย Sum และ Average และกลับคืน (return) เป็นผลลัพธ์ของกระบวนงานไปยังส่วนจำเพาะเรียก (calling module) เพื่อดูว่ากระบวนงานนี้ทำงานอย่างไร สมมติว่าโปรแกรมหลักประกาศ X, Y, Total และ Mean เป็นตัวแปรชนิด Real ข้อความสั่งเรียกกระบวนงาน

ComputesumAve (X, Y, Total, Mean)

จัดตัวการสมนัยดังนี้

Actual Parameter	corresponds to	**Formal Parameter**
X		Num1
Y		Num2
Total		Sum
Mean		Average

```
procedure ComputeSumAve (Num1, Num2 {input} : Real;
                         var Sum, Average {output} : Real);
```

```
{
   Computes the sum and average of Num1 and Num2.
   Pre  :  Num1 and Num2 are assigned values.
   Post :  The sum and average of Num1 and Num2 are computed and
           returned.
}
begin   {ComputeSumAve}
   Sum := Num1 + Num2;
   Average := Sum / 2.0
end;  {ComputeSumAve}
```

รูป 6.4 โปรซีเดอร์คำนวณหา Sum และ Average

ค่าของ X และ Y เกี่ยวข้องกับพารามิเตอร์รูปนัย Num1 และ Num2 ซึ่งส่งไปยังกระบวนงานเมื่อมันถูกเรียกครั้งแรก ข้อความสั่ง

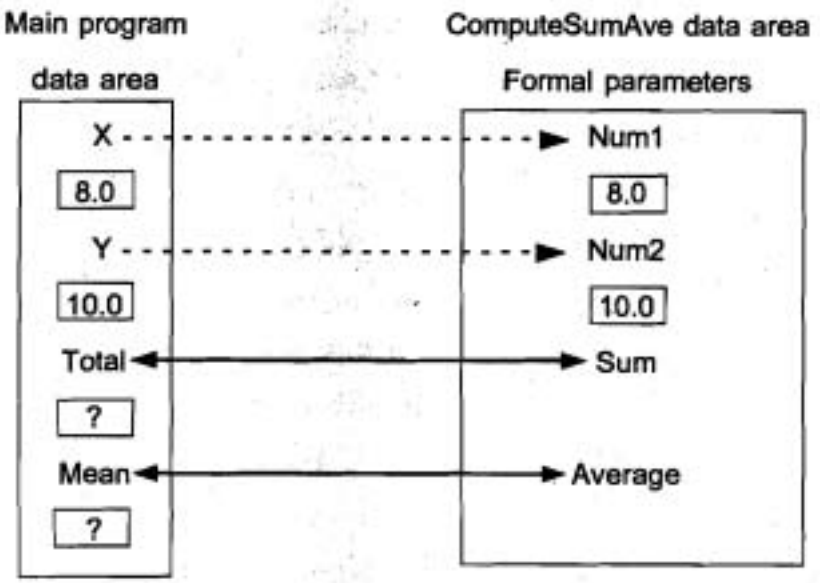
Sum := Num1 + Num2;

เก็บผลบวกของกระบวนงานอินพุต ในตัวแปร Total (พารามิเตอร์จริงตัวที่สาม) ของโปรแกรมหลัก

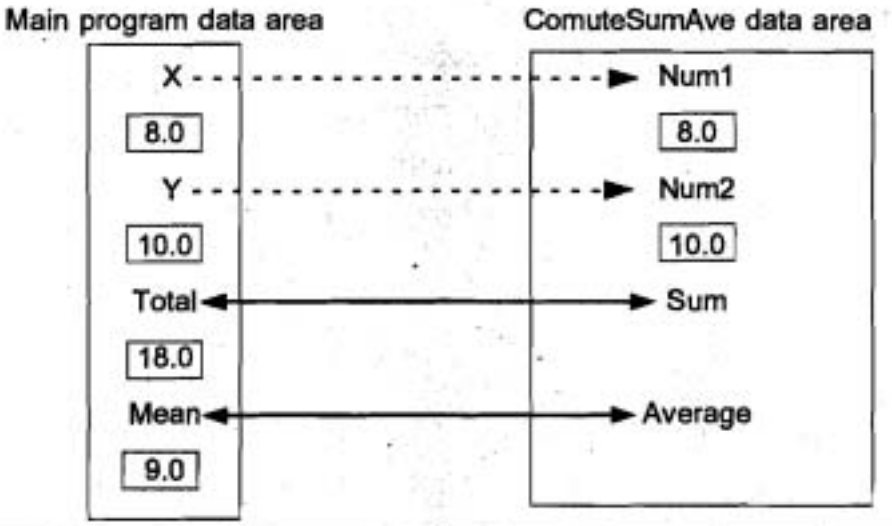
ข้อความสั่ง

Average := Sum / 2.0

หารค่าที่เก็บในตัวแปร Total ของโปรแกรมหลักด้วย 2.0 และเก็บผลงานในตัวแปร Mean (พารามิเตอร์จริงตัวที่สี่) ของโปรแกรมหลัก รูป 6.5 แสดงพื้นที่ข้อมูลของโปรแกรมหลักและพื้นที่ข้อมูลของกระบวนงานหลังจากเรียกกระบวนงาน แต่ก่อนที่ตัวกระบวนงานจะเริ่มต้นกระทำการ รูป 6.6 แสดงพื้นที่ข้อมูลเหล่านี้หลังจากตัวกระบวนงานกระทำการแล้ว การกระทำของกระบวนงานตั้งค่าของตัวแปร Total และ Mean ของโปรแกรมหลัก ให้เป็น 18.0 และ 9.0 ตามลำดับ



รูป 6.5 พื้นที่ข้อมูลหลังจากเรียกกระบวนงาน



รูป 6.6 พื้นที่ข้อมูลหลังจากกระทำการกระบวนงาน

รูป 6.5 ชี้สิ่งสำคัญหนึ่งอย่างของความแตกต่างระหว่างพารามิเตอร์รูปนัยซึ่งเป็นกระบวนงานอินพุต และพารามิเตอร์รูปนัยซึ่งใช้เป็นกระบวนงานเอาต์พุต เพราะว่า พารามิ-

เดอร์รูปนัย Num1 และ Num2 เป็นพารามิเตอร์ค่า แต่ละตัวรับสำเนาค่าพารามิเตอร์จริงที่สมนัยของมัน ซึ่งกระบวนงานใช้ในการคำนวณค่าซึ่งถูกส่งไปยังพารามิเตอร์รูปนัย Num1 เก็บในพื้นที่ข้อมูลของกระบวนงาน ณ เวลาที่เรียกกระบวนงาน และไม่มีการเชื่อมต่ออีกต่อไประหว่างพารามิเตอร์รูปนัย Num1 กับ พารามิเตอร์จริงซึ่งสมนัยของมัน ลูกศรเส้นประ (broken arrow) ในรูป 6.5 แสดงถึงเงื่อนไขนี้ และหัวลูกศรชี้การไหลของข้อมูลเป็นหนึ่งทิศทางเท่านั้นคือจากส่วนจำเพาะเรียก (calling module) ไปยังกระบวนงาน

พารามิเตอร์รูปนัย Sum และ Average ในทางตรงกันข้ามเป็นพารามิเตอร์ตัวแปร คอมไพเลอร์เก็บเลขที่อยู่ (address) หน่วยความจำของตัวแปรจริงในพื้นที่ข้อมูลของกระบวนงาน ซึ่งสมนัยกับพารามิเตอร์ตัวแปรแต่ละตัว

ผ่านทางเลขที่อยู่นี้ กระบวนงานเข้าถึงพารามิเตอร์จริงในส่วนจำเพาะเรียก ดังนั้นกระบวนงานสามารถเปลี่ยนแปลงค่าของพารามิเตอร์จริง หรือใช้ค่าของพารามิเตอร์จริง ในการคำนวณในรูป 6.5 ความสัมพันธ์นี้แสดงด้วยลูกศรสองหัว (double-headed arrow) เชื่อมต่อพารามิเตอร์ตัวแปรแต่ละตัวที่สมนัยกับพารามิเตอร์จริงของมัน

ลูกศรสองหัว แสดงว่าข้อมูลสามารถไหลเข้าไปยังกระบวนงานและไหลออกจากกระบวนงานผ่านทางพารามิเตอร์ตัวแปร

เนื่องจากคอมไพเลอร์ของ Pascal ต้องรู้ว่าพารามิเตอร์รูปนัยเป็นพารามิเตอร์ค่าหรือเป็นพารามิเตอร์ตัวแปร เมื่อมันแปลการประกาศในกระบวนงาน **เราระบุว่าเป็นพารามิเตอร์ตัวแปรโดยการใส่คำสงวน var ไว้หน้าพารามิเตอร์** (ดูบรรทัดที่สองในรูป 6.4) โปรดสังเกตว่า var ปรากฏเฉพาะในรายการพารามิเตอร์รูปนัยเท่านั้น ไม่ปรากฏในรายการพารามิเตอร์จริง

พารามิเตอร์ตัวแปรมีความคล่องตัวมากกว่าพารามิเตอร์ค่า เพราะว่าค่าของมันสามารถใช้การคำนวณเช่นเดียวกับ เปลี่ยนแปลงค่าได้โดยการกระทำการของกระบวนงาน ทำไมจึงไม่ทำให้พารามิเตอร์ทั้งหมด แม้กระทั่งพารามิเตอร์อินพุต เป็นพารามิเตอร์ตัวแปร เหตุผลคือพารามิเตอร์ค่าให้การป้องกันบางอย่างให้กับบูรณาภาพของข้อมูล (data integrity)

เพราะว่าการทำสำเนาพารามิเตอร์ค่าซึ่งเก็บเฉพาะที่ในพื้นที่ข้อมูลของกระบวนงาน Pascal ปกป้องค่าของพารามิเตอร์จริง และป้องกันมันจากข้อผิดพลาด ซึ่งเปลี่ยนแปลงโดยการกระทำการของกระบวนงาน

ตัวอย่างเช่น ถ้าเราเพิ่มข้อความสั่ง

```
Num1 := -5.0
```

ที่ตอนท้ายของกระบวนงาน ComputesumAve ค่าของตัวแปรรูปนัย Num1 จะเปลี่ยนเป็น -5.0 แต่ค่าซึ่งเก็บใน X (พารามิเตอร์จริงซึ่งสมนัยกัน) จะยังคงเป็น 8.0

ถ้าโปรแกรมเมอร์ไม่ใส่ใจที่จะประกาศพารามิเตอร์รูปนัยเอาต์พุต เป็นพารามิเตอร์ตัวแปร หลังจากนั้นค่าของมัน (ไม่ใช่เลขที่อยู่ของมัน) จะเก็บในกระบวนงาน และการเปลี่ยนแปลงใดๆ ก็ตามกับค่าของมัน จะไม่กลับคืนไปยังโปรแกรมเรียก (calling program) นี่คือข้อผิดพลาดร่วมอย่างมากในการใช้พารามิเตอร์

สไตล์ของโปรแกรม (Program Style)

การเขียนรายการพารามิเตอร์รูปนัย

ในรูป 6.4 รายการพารามิเตอร์รูปนัย

```
(Num1, Num2, {input} : Real;
var Sum, Average {output} : Real);
```

เขียนบนสองบรรทัดเพื่อทำให้อ่านโปรแกรมง่าย พารามิเตอร์ค่าอยู่บนบรรทัดแรกพร้อมกับคอมเมนต์ {input} เพื่อเป็นเอกสารว่าใช้เป็นอินพุตของกระบวนงาน ส่วนพารามิเตอร์ตัวแปรบนบรรทัดที่สองที่มีคอมเมนต์ {output}

โดยทั่วไปเราทำตามการฝึกปฏิบัติซึ่งแสดงในรูป 6.4 ในการเขียนรายการพารามิเตอร์รูปนัย กล่าวคือ รายการของอินพุตพารามิเตอร์ เขียนเป็นอันดับแรก และรายการของเอาต์พุตพารามิเตอร์ใดๆ ก็ตามเขียนเป็นอันดับสุดท้าย

เมื่อใดใช้พารามิเตอร์ตัวแปรหรือใช้พารามิเตอร์ค่า (When to Use a Variable Parameter or a Value Parameter)

เราตัดสินใจได้อย่างไรว่าจะใช้พารามิเตอร์ตัวแปรหรือจะใช้พารามิเตอร์ค่า ในที่นี้คือกฎอย่างคร่าวๆ

- ถ้าสารสนเทศถูกส่งไปยังกระบวนงาน แต่ไม่กลับคืนหรือไม่ส่งออกจากกระบวนงาน ดังนั้น พารามิเตอร์รูปนัยซึ่งแทนสารสนเทศนั้นควรเป็นพารามิเตอร์ค่า (ตัวอย่างเช่น Num1 และ Num2 ในรูป 6.2 และรูป 6.4) พารามิเตอร์ซึ่งใช้ในวิธีนี้เรียกว่า **อินพุตพารามิเตอร์** (input parameter)
- ถ้าสารสนเทศต้องกลับคืนมายังโปรแกรมเรียกจากกระบวนงาน ดังนั้น พารามิเตอร์ตัวแปร (ตัวอย่างเช่น Sum และ Average ในรูป 6.4) พารามิเตอร์ซึ่งใช้ในวิธีนี้ เรียกว่า **เอาต์พุตพารามิเตอร์** (output parameter)

• ถ้าสารสนเทศถูกส่งไปยังกระบวนงาน และมีการดัดแปร จากนั้นค่าใหม่ส่งกลับคืนพารามิเตอร์รูปนัยซึ่งแทนสารสนเทศต้องเป็นพารามิเตอร์ตัวแปร พารามิเตอร์ซึ่งใช้ในวิธีนี้ เรียกว่า **อินพุต/เอาต์พุต พารามิเตอร์** (input/output parameter)

ถึงแม้ว่าเราจะแยกเอาต์พุตพารามิเตอร์ออกจากอินพุต/เอาต์พุต พารามิเตอร์ แต่ Pascal ไม่เป็นเช่นนั้น พารามิเตอร์ทั้งสองชนิด ต้องประกาศเป็นพารามิเตอร์ตัวแปร ดังนั้น เลขที่อยู่ (address) ของพารามิเตอร์จริง ซึ่งสมนัยกันถูกเก็บในพื้นที่ข้อมูลของกระบวนงาน เมื่อกระบวนงานถูกเรียก สำหรับอินพุต/เอาต์พุตพารามิเตอร์ เราสมมติว่ามีข้อมูลซึ่งมีความหมายอยู่ในพารามิเตอร์จริงก่อนกระทำการกระบวนงาน สำหรับเอาต์พุตพารามิเตอร์เราไม่มีข้อสมมติเช่นนี้

การส่งนิพจน์ไปยังพารามิเตอร์ค่า (Passing Expressions to Value Parameter)

นิพจน์ หรือ ตัวแปร หรือ ค่าคงตัว แบบเดียวกัน (Assignment-compatible expressions or variables or constants) สามารถนำมาใช้เป็นพารามิเตอร์จริงสมนัยกับพารามิเตอร์ค่า

ตัวอย่างเช่น ข้อความสั่งเรียกกระบวนงาน

ComputeSumAve (X + Y, 10.5 MySum, MyAve);

เรียก ComputeSumAve เพื่อคำนวณผลบวก (กลับคืนใน MySum) และค่าเฉลี่ย (กลับคืนใน MyAve) ของนิพจน์ X + Y และค่าจำนวนจริง 10.5 อย่างไรก็ตาม มีเฉพาะตัวแปรเท่านั้น ซึ่งสมนัยกับพารามิเตอร์ตัวแปร ดังนั้น MySum และ MyAve ต้องประกาศเป็นตัวแปรชนิด Real ในส่วนจำเพาะเรียก

ข้อจำกัดนี้เป็นการบังคับให้รับ เพราะว่า พารามิเตอร์จริงซึ่งสมนัยกับพารามิเตอร์ตัวแปร อาจถูกดัดแปร (modified) เมื่อกระทำการกระบวนงานจึงเป็นตรรกะที่ไม่ถูกต้องถ้ายอมให้กระบวนงานเปลี่ยนแปลงค่าของค่าคงตัวหรือนิพจน์

การเรียกกระบวนงานมากกว่าหนึ่งครั้ง (Multiple Calls to a Procedure)

ข้อควรจำ พารามิเตอร์ทำให้หนึ่งกระบวนงานกระทำการหนึ่งเซตของคำสั่งบนหลายเซตที่แตกต่างกันของข้อมูลในแต่ละครั้งที่กระบวนงานถูกเรียก ในตัวอย่าง 6.3 จะแสดงให้เห็นว่าหนึ่งกระบวนงานถูกเรียกมากกว่าหนึ่งครั้ง ในโปรแกรมซึ่งกำหนดให้ และประมวลผลข้อมูลแตกต่างกันในการเรียกแต่ละครั้ง

ตัวอย่าง 6.3 การเรียงลำดับเลขสามตัว (Sorting Three Numbers)

โปรแกรม Sort3Number ในรูป 6.7 อ่านค่าข้อมูลสามตัวไว้ใน Num1, Num2 และ Num3 และจัดเรียงใหม่ข้อมูล เพื่อให้มันอยู่ในลำดับเรียงจากน้อยไปหามาก (increasing sequence) ค่าน้อยที่สุดให้อยู่ใน Num1 การเรียกกระบวนงาน Order สามครั้งกระทำการดำเนินการเรียงลำดับนี้

เรียงลำดับ หมายถึง การจัดเรียงใหม่ข้อมูล เพื่อให้ข้อมูลอยู่ในลำดับจากน้อยไปหามาก (**Sort** is a rearrangement of data such that the data are in increasing sequence.)

ตัวของกระบวนงาน Order ประกอบด้วยข้อความสั่ง if จากรูป 4.10 หัวเรื่องของกระบวนงานประกอบด้วยรายการพารามิเตอร์รูปนัย

(var X, Y {input/output} : Real)

ซึ่งแสดงว่า X และ Y เป็นพารามิเตอร์รูปนัย X และ Y เป็นอินพุต/เอาต์พุตพารา-มิเตอร์ เพราะว่า กระบวนงานใช้ค่าพารามิเตอร์จริง ปัจจุบันเป็นอินพุตและอาจจะกลับคืนค่าใหม่

หลังจากกระทำการกระบวนงาน Order แล้วค่าที่น้อยกว่าของพารามิเตอร์สองตัวนี้จะเก็บในพารามิเตอร์จริงตัวแรก และค่าที่ใหญ่กว่าจะเก็บในพารามิเตอร์จริงตัวที่สอง เพราะฉะนั้น

Order (Num1, Num2); {Order the data in Num1 and Num2.}

เก็บค่าตัวที่น้อยกว่าของ Num1 และ Num2 ใน Num1 และเก็บค่าตัวที่ใหญ่กว่าใน Num2 ในการวิ่งตัวอย่าง Num1 มีค่าเท่ากับ 8.0 และ Num2 มีค่าเท่ากับ 10.0 ดังนั้นค่าเหล่านี้จึงไม่มีการเปลี่ยนแปลง โดยการกระทำของกระบวนงาน แต่ข้อความสั่งเรียกกระบวนงาน

Order (Num1, Num3); {Order the data in Num1 and Num3}

สลับค่าของ Num1 (ค่าเริ่มต้นคือ 8.0) และ Num3 (ค่าเริ่มต้นคือ 6.0) ตาราง 6.1 ตามรอยการกระทำการของโปรแกรม

Edit Window

program Sort3Numbers ;

```
{
   Reads three numbers and sorts them so that they are in increasing order
}
   var
      Num1, Num2, Num3 : Real ; {three variables being sorted}
   procedure order (var X, Y} input/output} : Real);
{
   Order a pair of numbers represented by X and Y so that the smaller
   number is in X and the larger number is in Y.
   Pre : X and Y are assigned values.
   Post : X is the smaller of the pair and Y is the larger.
}
   var
      Temp : Real ; {copy of number originally in X}
begin {Order}
   if X > Y then
      begin {Switch the values of X and Y}
         Temp:= X;        {Store old X in Temp}
            X := Y;       {Store old Y in X}
            Y := Temp;    {Store old X in Y}
      end {if}
   end. {Order}

begin {Sort3Number}
   WriteLn ('Enter 3 numbers separated by spaces > ');
   ReadLn (Num1, Num2, Num3);

   {Sort the numbers}
```

```
    Order (Num1, Num2); {Order the data in Num1 and Num2}
    Order (Num1, Num3); {Order the data in Num1 and Num3}
    Order (Num2, Num3); {Order the data in Num2 and Num3}

    {Print the results}
    WriteLn ('The three Numbers in order are : ');
    WriteLn (Num1 : 8 : 2, Num2 : 8 : 2, Num3 : 8 : 2)
end. {Sort3Numbers}
```

Output Window

```
Enter 3 numbers sepqrated by spaces>
8.0 10.0 6.0
The three numbers in order are :
   6.00 8.00 10.00
```

รูป 6.7 โปรแกรมเรียงอันดับเลขสามตัว

ตาราง 6.1 ตามรอยโปรแกรม Sort3Numbers

Statement	Num1	Num2	Num3	Effect
ReadLn (Num1, Num2, Num3);	8.0	10.0	6.0	Enters data
Order (Num1, Num2);				No change
Order (Num1, Num3);	6.0		8.0	Switches Num1 and Num3
Order (Num2, Num3);		8.0	10.0	Switches Num2 and Num3
WriteLn (Num1, Num2, Num3)				Displays 6.0, 8.0, 10.0

แบบฝึกหัด 6.2 Self - Check

1. สมมติว่า X, Y และ Z เป็นตัวแปรชนิด Integer และ X มีค่าเท่ากับ 5, Y มีค่าเท่ากับ 7 และ Z มีค่าเท่ากับ 2 จงตามรอยการกระทำการของการเรียก Shuffle (X,Y,Z)

```
procedure Shuffle (X, Y : Integer;
                   var Z : Integer);
    var
      Temp : Integer ;
begin
  Temp := X ;
    X := Y ;
    Y := Z ;
    Z := Temp
end ; {Shuffle}
```

2. a) จงแสดงเอาต์พุตของโปรแกรมข้างล่างนี้ ในรูปตารางของค่า X, Y, Z และ W

```
program Show ;
  var
    W, X, Y, Z : Integer ;

  procedure SumDiff (Num1, Num2 : Integer ;
                     var Num3, Num4 : Integer);
  begin {SumDiff}
    Num3 := Num1 + Num2;
    Num4 := Num1 - Num2;
end ; {SumDiff}

begin {Show}
  X := 5 ; Y := 3 ; Z := 7 ; W := 9 ;
```

```
        WriteLn ('  X   Y   Z   W  ');
        SumDiff (X, Y, Z, W);
        WriteLn (X : 4, Y : 4, Z : 4, W : 4);
        SumDiff (Y, X, Z, W);
        WriteLn (X : 4, Y : 4, Z : 4, W : 4);
        SumDiff (Z, W, Y, X);
        WriteLn (X : 4, Y : 4, Z : 4, W : 4);
        SumDiff (Z, Z, X, Y);
        WriteLn (X : 4, Y : 4, Z : 4, W : 4);
        SumDiff (Y, Y, Y, W);
        WriteLn (X : 4, Y : 4, Z : 4, W : 4);
    end. {Show}
```

b) จงเขียนเงื่อนไขก่อน (preconditions) และเงื่อนไขหลัง (postconditions) ของกระบวนงาน Sumdiff

c) จงวาดรูปพื้นที่ข้อมูลของโปรแกรม Show และพื้นที่ข้อมูลของกระบวนงาน SumDiff เพื่อแสดงค่าปัจจุบันของอาร์กิวเมนต์และพารามิเตอร์ทุกตัว

3. กระบวนหนึ่งชุดมีพารามิเตอร์รูปนัยสี่ตัวคือ W, X, Y และ Z (ทุกตัวมีแบบชนิดข้อมูลเป็น Real) ระหว่างการกระทำการกระบวนงานเก็บผลบวกของ W และ X ใน Y และเก็บผลคูณของ W และ X ใน Z จงบอกว่าพารามิเตอร์ตัวใดบ้างเป็นอินพุต และตัวใดบ้างเป็นเอาต์พุต

เขียนโปรแกรม

1. จงเขียนกระบวนงานสำหรับแบบฝึกหัดข้อ 3 Self-Check

2. จงเขียนกระบวนงานซึ่งรับอินพุตเป็นความสูง h และรัศมี r ของรูปกรวยวงกลม (a right circular cone) และเอาต์พุตเป็นปริมาตรของรูปกรวย (volume of the cone) สูตรปริมาตรของรูปกรวยคือ volume = $\pi r^2 h$

เฉลย ข้อ 2. a

	X	Y	Z	W
Line1 :	5	3	7	9
Line2 :	5	3	8	2
Line3 :	5	3	8	-2
Line4 :	10	6	8	-2
Line5 :	16	0	8	-2
Line6 :	16	0	8	0

6.3 กฎวากยสัมพันธ์สำหรับกระบวนงานที่มีรายการพารามิเตอร์ (Syntax Rules for Procedures with Parameter Lists)

หัวข้อนี้รวมการอภิปรายของพารามิเตอร์ผ่านการแสดงวากยสัมพันธ์สำหรับการประกาศกระบวนงาน และการเรียกกระบวนงานแผนภาพวากยสัมพันธ์สำหรับรายการพารา-มิเตอร์รูปนัย และกฎสำหรับการสมนัยของรายการพารามิเตอร์

Syntax Display

การประกาศกระบวนงาน (กระบวนงานที่มีพารามิเตอร์) (Procedure Declaration (Procedure with Parameters)

Form :

```
procedure pname (formal parameters);
declaration section
begin
   procedure body
end;
```

ตัวอย่าง

```
procedure Highlight (Ch {input} : Char; var NumStars {output} : Integer);
{
   Displays Ch between two asterisks and returns the numbers of asterisks
```

```
    printed.
    Pre  :  Ch is defined.
    Post :  Returns 3 in NumStars if Ch = Border; otherwise, returns 2 in
            NumStars
}
  const
    Border = '*';

begin {Highlight}
  Write (Border); Write (Ch); Write (Border);
  if  Ch = Border then
    NumStars := 3
  else
    NumStars := 2
end; {Highlight}
```

มีความหมายดังนี้ กระบวนงาน pname ถูกประกาศตามด้วย formal parameters อยู่ภายในเครื่องหมายวงเล็บ ไอเดนติไฟเออร์ (identifiers) ซึ่งประกาศใน declaration section ถูกจำกัดเฉพาะที่ในกระบวนงาน และถูกนิยามระหว่างการกระทำการของกระบวนงานเท่านั้น procedure body อธิบายการจัดดำเนินการข้อมูลที่จะกระทำโดยกระบวนงานโดยใช้พารามิเตอร์รูปนัย (formal parameters) เป็นชื่อดัมมี (dummy names) สำหรับพารามิเตอร์จริง ค่าของพารามิเตอร์รูปนัยรับจากข้อมูลซึ่งส่งเข้ามายังกระบวนงาน และพารามิเตอร์รูปนัยตัวแปร (นำหน้าด้วยคำว่า var) กลับคืนผลลัพธ์ของกระบวนงานไปยังส่วนจำเพาะเรียก (calling module)

Syntax Display

ข้อความสั่งเรียกกระบวนงาน (กระบวนงานที่มีพารามิเตอร์) (Procedure Call Statement (Procedure with Parameters)

Form : | pname (actual parameters) |

ตัวอย่าง : Highlight ('A', NumAsterisks)

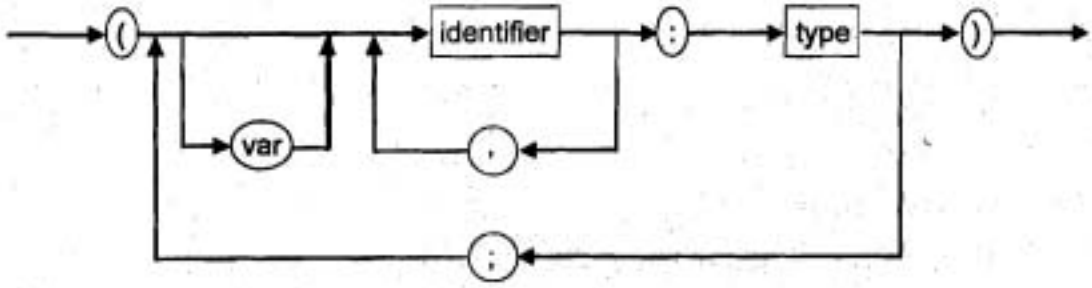
มีความหมายดังนี้ actual parameters อยู่ภายในเครื่องหมายวงเล็บ เมื่อกระบวนงาน pname ถูกเรียกให้กระทำการ พารามิเตอร์จริงตัวแรกเกี่ยวข้องกับพารามิเตอร์รูปนัยตัวแรก พารามิเตอร์จริงตัวที่สองเกี่ยวข้องกับพารามิเตอร์รูปนัยตัวที่สอง เช่นนี้เรื่อยไป สำหรับพารามิเตอร์ค่า (value parameter) ค่าของพารามิเตอร์จริง ถูกเก็บในพื้นที่ข้อมูลของกระบวนงาน และกระบวนงานประมวลผลเป็นค่าเฉพาะที่ (local value) สำหรับพารามิเตอร์ตัวแปร (variable parameter) เลขที่อยู่ (address) ของพารามิเตอร์จริงถูกเก็บในพื้นที่ข้อมูลของกระบวนงาน และกระบวนงานประมวลผลพารามิเตอร์จริง

โปรดสังเกต พารามิเตอร์จริงต้องเป็นไปตามกฎสำหรับการสมนัยของรายการพารามิเตอร์ซึ่งจะได้อภิปรายต่อไปในหัวข้อนี้

แผนภาพวากยสัมพันธ์ข้างล่างนี้ แสดงให้เห็นว่ารายการของ formal parameter ต้องอยู่ภายในวงเล็บเสมอ และประกอบด้วยรายการของไอเดนติไฟเออร์หนึ่งตัวหรือมากกว่าหนึ่งตัว รายการแต่ละชุดอาจนำหน้าด้วย var (ละเว้นได้) และต้องจบด้วย colon ตามด้วยชื่อของแบบชนิดข้อมูล (ตัวอย่างเช่น Real หรือ Char เป็นต้น)

ไอเดนติไฟเออร์ในรายการแต่ละตัวให้คั่นด้วยเครื่องหมาย comma และรายการของไอเดนติไฟเออร์ (list of identifiers) ให้คั่นด้วยเครื่องหมาย semicolon

Formal Parameter List



ตัวอย่าง 6.4

รายการพารามิเตอร์รูปนัยสองชุดข้างล่างนี้เขียนหลายบรรทัดเพื่อทำให้อ่านง่ายขึ้น

ชุดที่หนึ่ง

```
(Ch3 : Char;
var X, Y, Z : Real)
```

ชุดที่สอง

```
(M, N, O : Integer;
var   X, Y, Z : Real;
      A, B, C : Real)
```

ทั้งสองชุด X, Y, Z ประกาศเป็นพารามิเตอร์ตัวแปรชนิด Real

Ch3 เป็นพารามิเตอร์ค่า ชนิด Char

A, B, C เป็นพารามิเตอร์ค่า ชนิด Real และ

M, N, O เป็นพารามิเตอร์ค่า ชนิด Integer

ตัวอย่างนี้ชี้ให้เห็นข้อผิดพลาดร่วมในการจัดรูปแบบของรายการพารามิเตอร์รูปนัย ดังนี้ รายการในชุดที่สองการย่อหน้าของ A, B และ C อาจทำให้นักศึกษาบางคนคิดว่า คำว่า var แสดงว่าโดยนัย A, B และ C เป็นพารามิเตอร์ตัวแปรด้วย **ที่จริงไม่ใช่** คำว่า var จะต้องปรากฏก่อนรายการแต่ละชุดของพารามิเตอร์ตัวแปรเสมอ

รายการพารามิเตอร์รูปนัยกำหนดรูปแบบของรายการพารามิเตอร์จริงชุดใดก็ตามในการเรียกกระบวนงาน รูปแบบนี้ถูกกำหนดระหว่างการแปลของโปรแกรม เมื่อคอมไพเลอร์ประมวลผลการประกาศกระบวนงาน ต่อมาเมื่อมันพบข้อความเรียกกระบวนงานคอมไพเลอร์ตรวจสอบรายการพารามิเตอร์จริง ถึงความต้องกัน (consistency) กับรายการพารามิเตอร์รูปนัย

รายการพารามิเตอร์จริงอาจจะเป็นรายการของนิพจน์ ตัวแปร หรือค่าคงตัวแต่ละตัวคั่นด้วยเครื่องหมาย comma รายการพารามิเตอร์จริงกับรายการพารามิเตอร์รูปนัยที่สมนัยของมันจะต้องตรงกันทั้งจำนวน (number) การเรียงอันดับ (order) และชนิด (type) ซึ่งจะได้อธิบายในกฎต่อไป

กฎสำหรับการสมนัยของรายการพารามิเตอร์ (Rules for Parameter List Correspondence)

1. การสมนัยระหว่างพารามิเตอร์จริงและพารามิเตอร์รูปนัย ถูกกำหนดโดยตำแหน่งในรายการพารามิเตอร์ตามลำดับของมัน รายการเหล่านี้ต้องมีขนาดเหมือนกัน (same size) ถึงแม้ว่าชื่อของพารามิเตอร์จริงและพารามิเตอร์รูปนัยซึ่งสมนัยกันอาจแตกต่างกัน

2. สำหรับพารามิเตอร์ตัวแปร ชนิดของพารามิเตอร์จริงและพารามิเตอร์ซึ่งสมนัยกันต้องเหมือนกัน สำหรับพารามิเตอร์ค่า พารามิเตอร์จริงต้องกำหนดค่าเข้ากันได้ (assignment compatible) กับพารามิเตอร์รูปนัยซึ่งสมนัยกับมัน (ดูหัวข้อ 2.5 และหัวข้อ 7.6)

3. สำหรับพารามิเตอร์ตัวแปร พารามิเตอร์จริงต้องเป็นตัวแปรสำหรับพารามิเตอร์ค่า พารามิเตอร์จริงอาจจะเป็นตัวแปร ค่าคงตัว หรือนิพจน์

ตัวอย่าง 6.5

โปรแกรมหลัก ประกอบด้วยการประกาศดังนี้

```
var
   X, Y :  Real;
      M :  Integer;
   Next :  Char;
procedure Test  (A, B : Integer;
                 var C, D : Real;
                 var E : Char);
```

จากหัวเรื่องของกระบวนงาน Test อธิบายดังนี้

กระบวนงาน Test มีพารามิเตอร์ค่าสองตัว (คือ A และ B) และมีพารามิเตอร์ตัวแปรสามตัว (C, D และ E)

ข้อความสั่งเรียกกระบวนงานต่อไปนี้ ถูกต้องเชิงวากยสัมพันธ์อยู่ในโปรแกรมหลัก

```
Test (M+3, 10, X, Y, Next);
Test (M, MaxInt, Y, X, Next);
Test (35, M*10, Y, X, Next)
```

รายการพารามิเตอร์จริงชุดแรกแสดงให้เห็นว่านิพจน์ (เช่น M+3) หรือค่าคงตัว (เช่น 10) อาจเกี่ยวข้องกับพารามิเตอร์รูปนัยค่าการสมนัยกำหนดโดยรายการพารามิเตอร์ชุดนี้แสดงในตาราง 6.2

ตาราง 6.2 การสมนัยของพารามิเตอร์สำหรับข้อความสั่งเรียกกระบวนงาน Text (M+3, 10, X, Y, Next)

Actual parameter	Formal Parameter	Parameter Kind
M+3	A	Integer, value
10	B	Integer, value
X	C	Real, variable
Y	D	Real, variable
Next	E	Cahr, variable

ข้อความสั่งเรียกกระบวนงานทั้งหมดในตาราง 6.3 มีข้อผิดพลาดวากยสัมพันธ์ ข้อความสั่งเรียกกระบวนงานชุดสุดท้าย ชี้ให้เห็นข้อผิดพลาดซึ่งบ่อยครั้งเกิดขึ้นจากการใช้กระบวนงานพารามิเตอร์จริงสามตัวหลัง (C, D, E) มีชื่อเหมือนกับพารามิเตอร์รูปนัย ซึ่งสมนัยกับมัน แต่ทั้งสามชื่อนี้ไม่ได้ประกาศเป็นตัวแปรในโปรแกรมหลัก ดังนั้น จึงใช้เป็นพารามิเตอร์จริงไม่ได้

ตาราง 6.3 ข้อความสั่งเรียกกระบวนงานซึ่งไม่ถูกต้อง

Procedure Call Statement	Error
Test (30, 10, M, X, Next)	แบบชนิดข้อมูลของ M ไม่ใช่ Real
Test (M, 19, X, Y)	จำนวนพารามิเตอร์จริง ไม่เท่ากับจำนวนพารามิเตอร์รูปนัย
Test (M, 10, 35, Y, 'E')	ค่าคงตัว 35 และ 'E' ไม่สมนัยกับพารามิเตอร์ตัวแปร
Test (M, 3.0, X, Y, Next)	3.0 ไม่ใช่ข้อมูลชนิด Integer
Test (30, 10, X, X+Y, Next)	นิพจน์ X+Y ไม่สมนัยกับพารามิเตอร์ตัวแปร
Test (30, 10, C, D, E)	C, D และ E ไม่ได้ประกาศในโปรแกรมหลัก

เมื่อเขียนรายการพารามิเตอร์ค่อนข้างยาวเช่นตัวอย่างนี้ ให้รอบคอบไม่สลับเปลี่ยน (not to transpose) พารามิเตอร์จริงสองตัว หรืออาจมีข้อผิดพลาดวากยสัมพันธ์ เป็นผลลัพธ์ถ้าไม่ฝ่าฝืนวากยสัมพันธ์แต่อย่างใด การกระทำการของกระบวนงานน่าจะให้ผลลัพธ์ถูกต้อง

แบบฝึกหัด 6.3 Self-Check

1. จงจัดเรียง (arrange) รายการพารามิเตอร์ที่ถูกต้องที่เหลืออีกสองชุดในตัวอย่าง 6.5 ในรูปแบบที่คล้ายกับตาราง 6.2
2. จงแก้ไขข้อผิดพลาดวากยสัมพันธ์ในรายการพารามิเตอร์รูปนัยข้างล่างนี้

   (var A, B : Integer, C : Real)

   (value M : Integer; var Next : Char)

   (var Account, Real; X+Y, Real)
3. สมมติให้มีการประกาศดังนี้

```
var
   X, Y, Z  : Real;
      M, N : Integer;
procedure Message (var A, B : Real;
                          X : Integer);
```

ข้อความสั่งเรียกกระบวนงานต่อไปนี้มีชุดใดถูกต้องหรือไม่ ถ้าชุดใดไม่ถูกต้องให้อธิบายเหตุผล

a) Message (X, Y, Z)

b) Message (X, Y, 8)

c) Message (Y, X, N)

d) Message (M, Y, N)

e) Message (25.0, 15, X)

f) Message (X, Y, M+N)

g) Message (A, B, X)

h) Message (Y, Z, M)

i) Message (Y+Z, Y-Z, M)

j) Message (Z, Y, X)

k) Message (X, Y, M, 10)

l) Message (Z, Y, MaxInt)

เขียนโปรแกรม

1. จงเขียนกระบวนงานแสดงผลเป็นตารางให้เห็นกำลังทั้งหมดของอาร์กิวเมนต์ตัวแรกของมันจากกำลังศูนย์จนถึงกำลังซึ่งระบุโดยอาร์กิวเมนต์ตัวที่สอง (เลขจำนวนเต็มบวก) กระบวนงานกลับคืน (return) ผลบวกของค่าทั้งหมดที่แสดง ตัวอย่างเช่น ถ้าอาร์กิวเมนต์ตัวที่หนึ่งคือ 10 และอาร์กิวเมนต์ตัวที่สองคือ 3 กระบวนงานนี้จะแสดงผลเป็น 1, 10, 100 และ 1000 ผลลัพธ์ที่กลับคืน คือ 1111

6.4 สโคปของไอเดนติไฟเออร์ (Scope of Identifiers)

ไอเดนติไฟเออร์ทุกตัวในโปรแกรม Pascal มีโดเมน หมายถึง ส่วนของโปรแกรมซึ่งมันถูกนิยาม เรียกว่า สโคปของไอเดนติไฟเออร์ ไอเดนติไฟเออร์ถูกอ้างได้โดยข้อความสั่งภายในสโคปของมันเท่านั้น ซึ่งอาจจะเป็นสโคปเฉพาะที่หรือสโคปส่วนกลาง

สโคปของไอเดนติไฟเออร์ หมายถึง ส่วนของโปรแกรม ซึ่งสามารถอ้างถึงไอเดนติไฟเออร์ที่ประกาศในโปรแกรม (Indentifier scope is the part of a program that can reference an identifier declared in the program.)

กฎสโคปข้อที่หนึ่ง : สโคบเฉพาะที่และสโคปส่วนกลาง (Scope Rule 1 : Local and Global Scopes)

สโคปของไอเดนติไฟเออร์ หมายถึง บล็อก (โปรแกรมหรือกระบวนงาน) ซึ่งมันถูกประกาศ เพราะฉะนั้นไอเดนติไฟเออร์ซึ่งประกาศภายในโปรแกรมหลัก (main program) มีสโคปส่วนกลางและถูกอ้างถึงที่ใดก็ได้ในโปรแกรมไอเดนติไฟเออร์ซึ่งประกาศภายในส่วนจำเพาะ (module) มีสโคปเฉพาะที่ ดังนั้นจึงถูกอ้างได้เฉพาะภายในส่วนจำเพาะนั้นเท่านั้นและส่วนจำเพาะอื่นๆ ซึ่งมันประกาศ

กฎสโคปประยุกต์ใช้กับไอเดนติไฟเออร์ทั้งหมดที่ใช้ในโปรแกรม รวมทั้งชื่อของค่าคงตัว ตัวแปร และพารามิเตอร์รูปนัย ดังนั้น ขณะที่กระบวนงานที่ประกาศในส่วนประกาศของโปรแกรมหลักมีสโคปส่วนกลาง รายการพารามิเตอร์รูปนัยของมันหมายถึงส่วนของบล็อก กระบวนงานจึงมีสโคปเฉพาะที่

สโคปส่วนกลาง หมายถึง สโคปของไอเดนติไฟเออร์ที่ประกาศในโปรแกรมหลัก (Global scope is the scope of an identifier declared in the main program.)

สโคปเฉพาะที่ หมายถึง สโคปของไอเดนติไฟเออร์ที่ประกาศในกระบวนงานหรือในฟังก์ชัน (Local scope is the scope of an identifier declared in a procedure or function.)

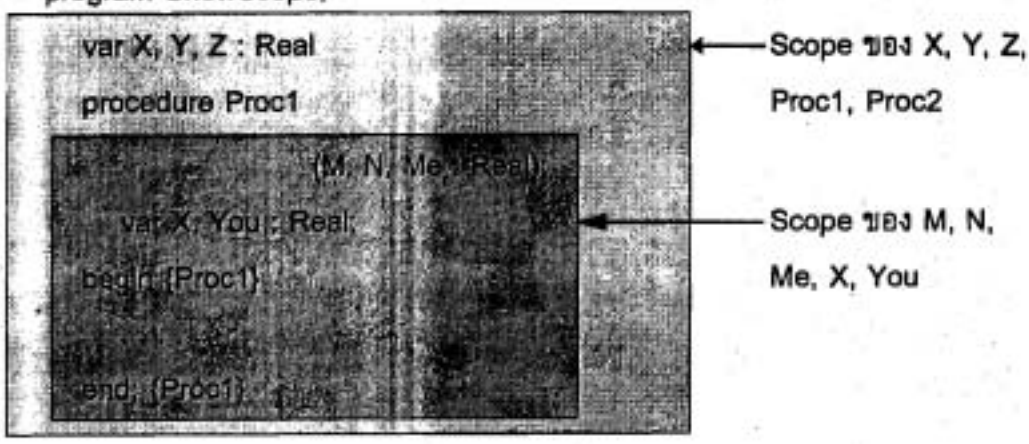
รูป 6.8 แสดงให้เห็นกฎสโคปข้อ 1. โดยการใส่กล่องบล็อกโปรแกรม และบล็อกกระบวนงานแต่ละชุด เราวางรายการพารามิเตอร์รูปนัยแต่ละชุดบนบรรทัดแยกต่างหากเพื่อแสดงว่ามันคือส่วนของบล็อกกระบวนงานของมัน

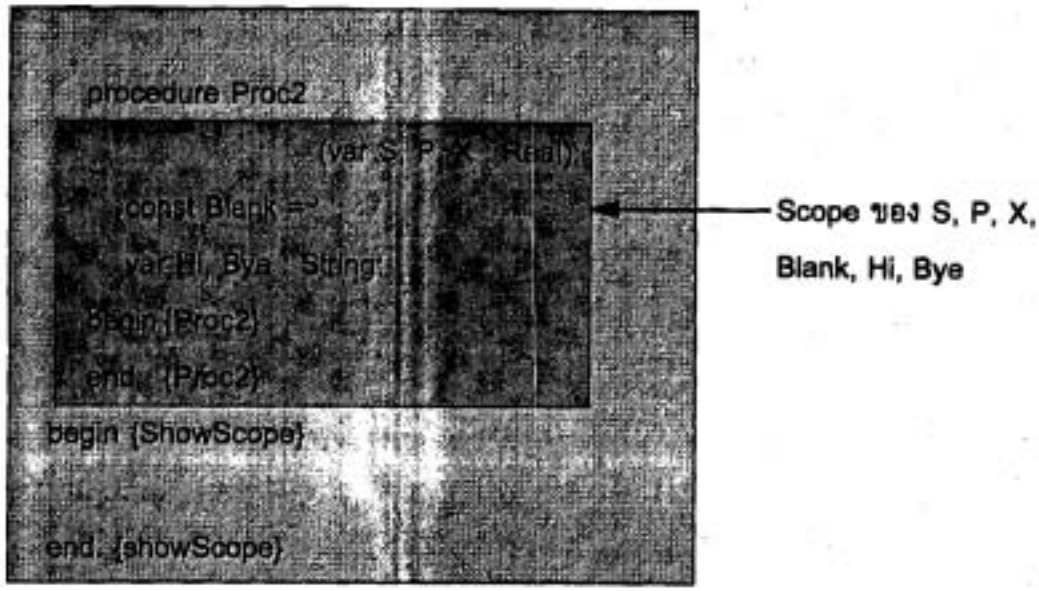
ตัวแปร Y และ Z เป็นตัวแปรส่วนกลาง หมายความว่า เราสามารถอ้างถึงตัวแปรเหล่านี้ที่ใดก็ได้ (เช่น ใน body ของโปรแกรมหลัก หรือ Proc1 หรือ Proc2)

ในโปรแกรมหลักและกระบวนงาน Proc2 ทั้งสองชุดนี้ ไม่สามารถเรียกตัวแปรเฉพาะที่ซึ่งประกาศใน Proc1 (พารามิเตอร์รูปนัย M, N, Me และตัวแปรเฉพาะที่ X, You)

เพราะว่า Proc1 และ Proc2 เป็นไอเดนติไฟเออร์ส่วนกลาง ดังนั้นกระบวนงานสองชุดนี้ถูกเรียกที่ใดก็ได้

ตาราง 6.4 แสดงให้เห็นว่ากระบวนงาน Proc2 เรียก Proc1 ได้ แต่กระบวนงาน Proc1 เรียก Proc2 ไม่ได้ เหตุผลสำหรับข้อยกเว้นนี้คือ Proc2 ถูกประกาศหลัง Proc1 และการประกาศของไอเดนติไฟเออร์ต้องอยู่ก่อนการใช้ครั้งแรกของมัน ถ้าเรามีหนึ่งกระบวนงานซึ่งเรียกอีกหนึ่งกระบวนงาน ต้องมั่นใจว่าได้ประกาศกระบวนงานซึ่งกำลังถูกเรียกก่อนผู้เรียก (caller)





รูป 6.8 สโคปของไอเดนติไฟเออร์ (Scope of Identifiers)

ตาราง 6.4 Valid Procedure Calls

Program or Procedure Body	Procedure That Can be Called
ShowScope	Proc1, Proc2
Proc2	Proc1, Proc2
Proc1	proc1

ตาราง 6.4 แสดงให้เห็นว่ากระบวนงานสามารถเรียกตัวมันเอง (a recursive call) เราจะอภิปรายการเรียกซ้ำในหัวข้อ 6.8

โปรดสังเกตว่า ไอเดนติไฟเออร์ X ในรูป 6.8 มีการประกาศ 3 ครั้งแยกจากกัน (ตัวแปรส่วนกลาง, ตัวแปรเฉพาะที่ใน Proc1 และพารามิเตอร์รูปนัย ใน Proc2) เพราะฉะนั้น X จึงมีสโคปสามแห่งแตกต่างกัน เราทราบแล้วว่าเราประกาศไอเดนติไฟเออร์เหมือนกันสองครั้งในหนึ่งบล็อกไม่ได้ แต่เราสามารถมีการประกาศมากกว่าหนึ่งแห่งสำหรับไอเดนติไฟเออร์หนึ่งตัว ซึ่งจะทำให้ทั้งหมดนี้อยู่ในบล็อกที่แตกต่างกัน กฎสโคปข้อที่ 2 จะบอกเราว่า Pascal จัดกระทำการประกาศหลายครั้งของไอเดนติไฟเออร์หนึ่งตัวอย่างไร

กฎสโคปข้อที่สอง : การประกาศหลายครั้งของไอเดนติไฟเออร์หนึ่งตัว (Scope Rule 2 : Multiple Declarations of an Identifier)

เมื่อไอเดนติไฟเออร์หนึ่งตัวมีการประกาศหลายครั้งและทั้งหมดนี้อยู่ในบล็อกแตกต่างกัน ดังนั้น การประกาศเฉพาะที่ชุดล่าสุดถูกใช้ในแต่ละครั้งที่ไอเดนติไฟเออร์ถูกอ้างถึง

กฎสโคปข้อที่สองบอกว่า ไอเดนติไฟเออร์ส่วนกลาง X ถูกเข้าถึง เมื่อเราใช้ X ในตัวโปรแกรมหลัก ตัวแปรเฉพาะที่ X ถูกเข้าถึงเมื่อเราใช้ X ใน body ของ Proc1 และพารามิเตอร์รูปนัย X ถูกเข้าถึง เมื่อใช้ X ใน body ของ Pascal เนื่องจากกฎสโคปข้อที่สอง เราไม่จำเป็นต้องเกี่ยวข้อง ไม่ว่าไอเดนติไฟเออร์นั้นจะมีการประกาศที่ใดก็ตาม เมื่อเราประกาศและใช้มันในกระบวนงาน ถ้าไอเดนติไฟเออร์มีการประกาศที่ใดก็ตาม Pascal ไม่สนใจการประกาศอื่นๆ เมื่อเราอ้างถึงไอเดนติไฟเออร์ภายในตัวกระบวนงาน คุณสมบัติข้อนี้ทำให้การเขียนกระบวนงานง่ายสำหรับการนำไปใช้ใหม่ (reuse) ในโปรแกรมอื่นๆ

วิศวกรรมซอฟต์แวร์ : หลีกเลี่ยงผลกระทบ (Software Engineering : Avoiding Side Effects)

ถึงแม้ว่าตัวแปรส่วนกลางสามารถอ้างถึงที่ใดก็ได้ในโปรแกรม เราควรหลีกเลี่ยงการอ้างถึงตัวแปรเหล่านี้ในตัวกระบวนงาน

การเปลี่ยนแปลงค่าของตัวแปรส่วนกลางในกระบวนงานเรียกว่า **ผลกระทบ** ของการกระทำการกระบวนงาน บ่อยครั้งที่ไม่มีการทำเอกสารเพื่อแสดงว่ากระบวนงานจัดดำเนินการตัวแปรส่วนกลาง ดังนั้น ในโปรแกรมที่มีกระบวนงานจำนวนมากจึงเป็นเรื่องยากที่จะหาที่อยู่ของข้อความสั่งซึ่งกำหนดค่าไม่ถูกต้องให้กับตัวแปรส่วนกลาง ถ้าข้อความสั่ง

```
Y := Y + 3.5; {Exanple of a side effect}
```

ปรากฏในกระบวนงาน Proc1 จะทำให้เกิดผลกระทบ (บวก 3.5 กับตัวแปรส่วนกลาง Y) เมื่อใดก็ตามที่กระบวนงานถูกเรียก

อย่างไรก็ตาม กระบวนงานสามารถเข้าถึงตัวแปรส่วนกลางได้โดยไม่เกิดผลกระทบถ้าตัวแปรส่วนกลางถูกส่งผ่านรายการพารามิเตอร์

รายการพารามิเตอร์รูปนัยและการประกาศเฉพาะที่ สำหรับกระบวนงานเป็นเอกสารอย่างชัดแจ้งของข้อมูลที่จะถูกจัดดำเนินการ เราควรจัดดำเนินการเฉพาะไอเดนติไฟเออร์เฉพาะที่ในกระบวนงาน มีข้อยกเว้นเฉพาะค่าคงตัวส่วนกลาง (global constants) ไอเดนติไฟเออร์ชนิด (ซึ่งจะอภิปรายในบทต่อไป) เพราะว่า Pascal ไม่ยอมให้มีการเปลี่ยนแปลงค่าของ

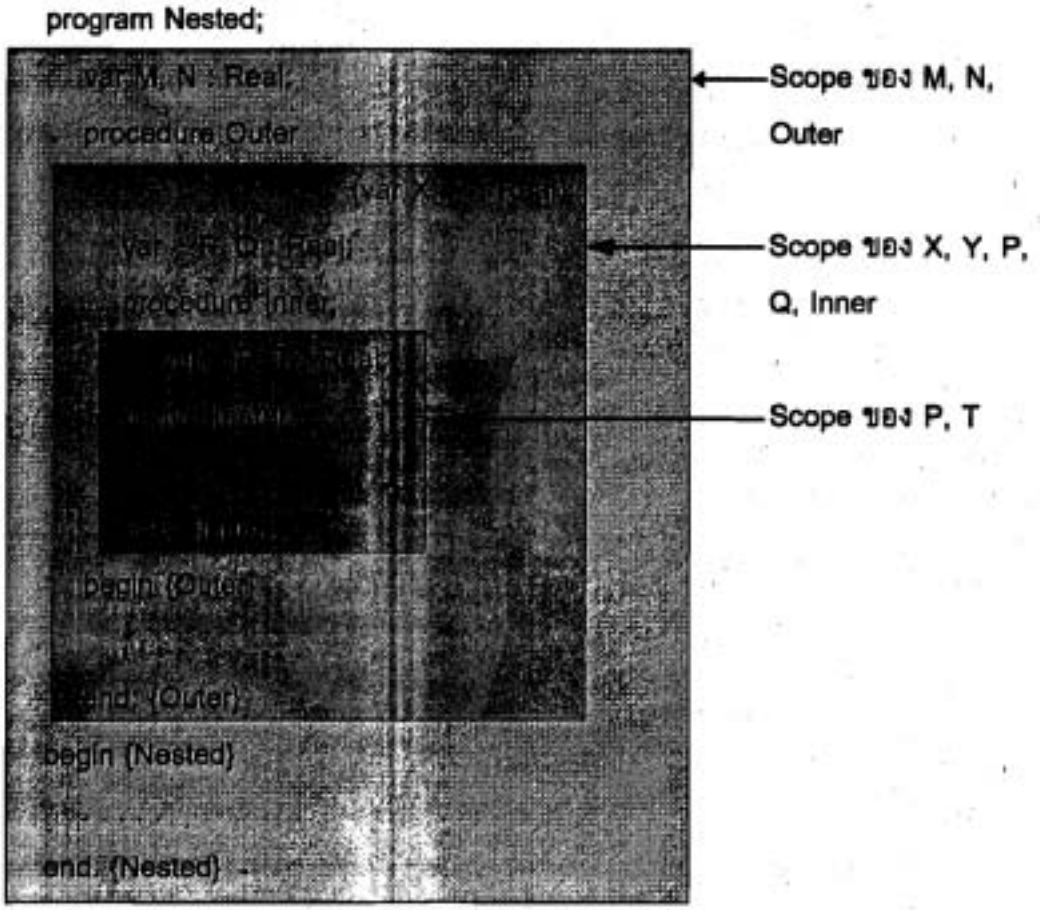
ค่าคงตัว (constant) เราสามารถอ้างถึงค่าคงตัวส่วนกลางในกระบวงนงานได้ โดยไม่มีผลกระทบ

ผลกระทบ หมายถึง การเปลี่ยนแปลงตัวแปร nonlocal ซึ่งไม่สมนัยกับพารามิเตอร์รูปนัยตัวแปร ผ่านการกระทำการของกระบวนงานหรือฟังก์ชัน (Side effect is the changing a nonlocal variabel that does not correspond to a variable formal parameter through a procedure or function execution.)

สโคปสำหรับกระบวนงานซ้อนใน (Scope for Nested Procedures)

กระบวนงานซ้อนในเกิดขึ้นเมื่อเราประกาศหนึ่งกระบวนงานภายในอีกหนึ่งกระบวนงาน กฎสโคปข้อที่หนึ่ง บอกว่าสโคปของกระบวนงานชั้นใน (Inner procedure) หมายถึงบล็อกซึ่งมันถูกประกาศ เพราะฉะนั้นกระบวนงานชั้นในจึงถูกเรียกโดยกระบวนงานซึ่งประกาศมันและโดยตัวมันเอง (การเรียกซ้ำ) แต่กระบวนงานชั้นในไม่สามารถถูกเรียกโดยโปรแกรมหลักหรือโดยกระบวนงานซึ่งอยู่ภายนอกการซ้อนใน ข้อจำกัดนี้เป็นข้อจำกัดศักยภาพการนำกลับมาใช้ใหม่ของกระบวนงานชั้นใน ดังนั้นจึงให้ข้อแนะนำว่า เราไม่ควรทำเป็นกระบวนงานซ้อนใน

รูป 6.9 แสดงให้เห็นผลกระทบของการซ้อนในกระบวนงานบนสโคปของไอเดนติไฟเออร์ และตาราง 6.5 แสดงรายการของไอเดนติไฟเออร์ซึ่งสามารถถูกอ้างได้ในแต่ละบล็อกและกระบวนงานซึ่งถูกเรียกได้ ในตารางแสดงให้เห็นว่ากระบวนงาน Inner สามารถอ้างถึงตัวแปรเฉพาะที่ของมัน ไอเดนติไฟเออร์ทั้งหมดที่ประกาศในโปรแกรมหลัก และไอเดนติไฟเออร์ทั้งหมดที่ประกาศในกระบวนงาน Outer ยกเว้นตัวแปร P เหตุผลสำหรับข้อยกเว้นนี้ คือ Inner ประกาศตัวแปรเฉพาะที่ P ของมันเอง และการประกาศนั้นมีการทำก่อน (กฎสโคปข้อที่สอง)



รูป 6.9 สโคปสำหรับกระบวนงานซ้อนใน (Scope for Nested Procedures)

ตาราง 6.5 Valid Identifier Reference for Fig. 6.9

Program or Procedure Body	Identifiers That Can Be Referenced
Nested	M, N (global variables) Outer (global variable)
Outer	M, N (global variables) X, Y (parameters) P, Q (local variables)

Program or Procedure Body	Identifiers That Can Be Referenced
Inner	Outer (global procedure) Inner (local procedure) M, N (global variables) X, Y (parameters for Outer) Q (variable declared in Outer) P, T (local variables) Outer (global procedure) Inner (procedure declared in outer)

อธิบายกฎสโคป (Illustrating the Scope Rules)

ตัวอย่าง 6.6 อธิบายปฏิบัติการเขียนโปรแกรมที่แย่ เพราะว่าชื่อไอเดนติไฟเออร์ไม่มีความหมายแต่อย่างใด และไม่มีความจำเป็นต้องตั้งชื่อซ้ำ แต่การศึกษาตัวอย่างนี้จะช่วยให้เรามีความรอบรู้กฎสโคปของ Pascal

ตัวอย่าง 6.6

รูป 6.10 แสดงหนึ่งกระบวนงานซึ่งประกาศในโปรแกรมหลัก

W ถูกประกาศเป็นตัวแปร ทั้งในกระบวนงานและในโปรแกรมหลัก

X ถูกประกาศเป็นตัวแปรในโปรแกรมหลัก และเป็นพารามิเตอร์ในกระบวนงาน

Y ถูกประกาศเป็นตัวแปรในโปรแกรมหลักเท่านั้น

โปรแกรมหลักเริ่มต้นโดยการกำหนดค่าเริ่มต้นให้กับตัวแปรส่วนกลาง W, X, และ Y ซึ่งแสดงให้เห็นในรูป 6.11 ข้อความสั่งเรียกกระบวนงาน

Change (W);

เรียกกระบวนงาน Change ด้วยตัวแปร W ของโปรแกรมหลักสมนัยกับพารามิเตอร์ตัวแปร X รูป 6.11 แสดงให้เห็นพื้นที่ข้อมูลของโปรแกรมและของกระบวนงาน หลังจากเรียกกระบวนงานในกระบวนงาน Change ข้อความสั่งกำหนดค่า

```
X := 6.0; {Change parameter X}
```

เก็บ 6.0 ในตัวแปร W ของโปรแกรมหลัก และข้อความสั่งกำหนดค่า

```
Y := Y + 1.0; {Side effect - change global Y}
```

เพิ่มค่าตัวแปร Y ของโปรแกรมหลักให้เป็น 4.0 (ผลกระทบ) ข้อความสั่งกำหนดค่าอีกสองคำสั่งในกระบวนงาน Change มีผลเฉพาะกับตัวแปรเฉพาะที่ W และ Z ของมันเท่านั้น (W กลายเป็น 35.0 และ Z กลายเป็น 3.0)

ข้อความสั่ง WriteLn ชุดที่สองในกระบวนงาน Change แสดงผลค่าของไอเดนติ-ไฟเออร์เฉพาะที่ของมัน (W, X และ Z) และตัวแปรส่วนกลาง Y ก่อนกระบวนงานกลับคืน (return)

ข้อความสั่ง WriteLn ในโปรแกรมหลักแสดงผลค่าของตัวแปรในโปรแกรมหลักสามตัว หลังจากกลับคืน

W	X	Y
6.0	2.0	4.0

โปรดสังเกตว่า ค่าของตัวแปร X ของโปรแกรมหลักไม่เปลี่ยนแปลง และค่าของ W คือ 6.0 (ไม่ใช่ 35.0)

จงพิจารณาว่า จะเกิดอะไรขึ้นถ้า X หรือ Y ถูกใช้เป็นพารามิเตอร์จริง แทนที่จะเป็น W คำถามนี้ทิ้งให้เป็นแบบฝึกหัดที่ตอนท้ายของหัวข้อนี้

Edit Window

```
program ScopeRules;
  var
    W, X, Y : Real;
  procedure Change (var X {input/output} : Real);
    var
      W, Z : Real;
  begin {Change}
    W := 35.0;      {Change local W}
    X  := 6.0;      {Change parameter X)
    Y  := Y + 1.0; {Side effect - change global Y}
    Z  := 3.0;      {Change local Z}
    WriteLn ('w' : 5, 'X' :5, 'Y' : 5, 'Z' : 5);
```

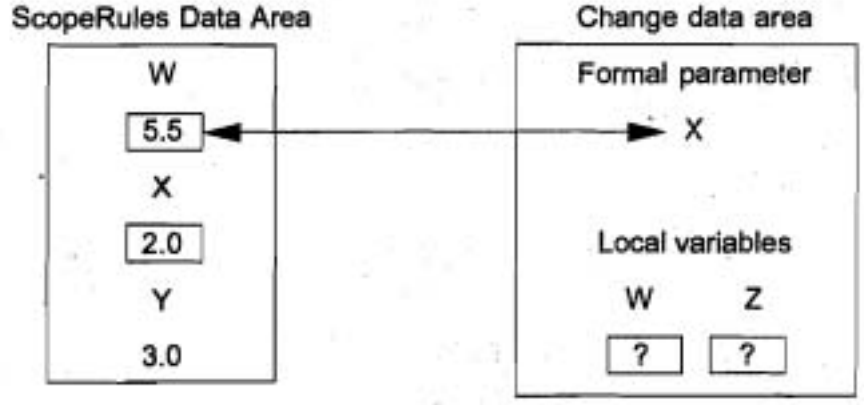
```
    WriteLn (W : 5 : 1, X : 5 : 1, Y : 5 : 1, Z : 5: 1, 'in Change')
  end; {Change}

begin {ScopeRules}
  W := 5.5;       {Initialize global W}
  X := 2.0;       {Initialize global X}
  Y := 3.0;       {Initialize global Y}
  Change (W);  {Update global W}
  WriteLn (W : 5 : 1, X : 5 : 1, Y : 5 : 1, 'in ScopeRules' : 19)
end. {ScopeRules}
```

Output Window

```
W    X    Y    Z
35.0 6.0  4.0  3.0  in Change
6.0  2.0  4.0       in ScopeRules
```

รูป 6.10 โปรแกรม ScopeRules



รูป 6.11 พื้นที่ข้อมูลหลังจากเรียกกระบวนงาน (Data Areas After Procedure Call)

แบบฝึกหัด 6.4 Self-Check

1. ในรูป 6.8 ทำไมตัวแปรซึ่งประกาศในกระบวนงาน Proc1 จึงถูกอ้างถึงโดยโปรแกรมหลัก หรือกระบวนงาน Proc2 ไม่ได้ ทำไมกระบวนงาน Proc2 จึงถูกเรียกภายใน body ของ Proc1 ไม่ได้

2. ในรูป 6.8 จะมีผล (effect) อะไรจากการกระทำการ body ของ Proc1 ข้างล่างนี้

```
begin {Proc1}
   X := 5.5;
   Y := 6.6;
   M := 2;
   N := 3;
 You  := M
end; {Proc1}
```

3. ถ้าลำดับข้อความสั่งในแบบฝึกหัดข้อ 2 ปรากฏในบล็อกที่แตกต่างกัน ข้อความสั่งกำหนดค่าบางคำสั่งจะผิดวากยสัมพันธ์ (syntactically incorrect) จงบอกว่าข้อความสั่งใดไม่ถูกต้อง และชี้ให้เห็นผล (effect) ของการกระทำอื่นๆ ถ้าลำดับข้อความสั่งปรากฏเป็น body ของ

(a) Proc2

(b) ShowScope

4. จงพิจารณาโปรแกรม ScopeRules ในรูป 6.10

(a) จะเกิดข้อผิดพลาดชนิดใด ถ้าข้อความสั่งกำหนดค่า

```
Z := 15.0;
```

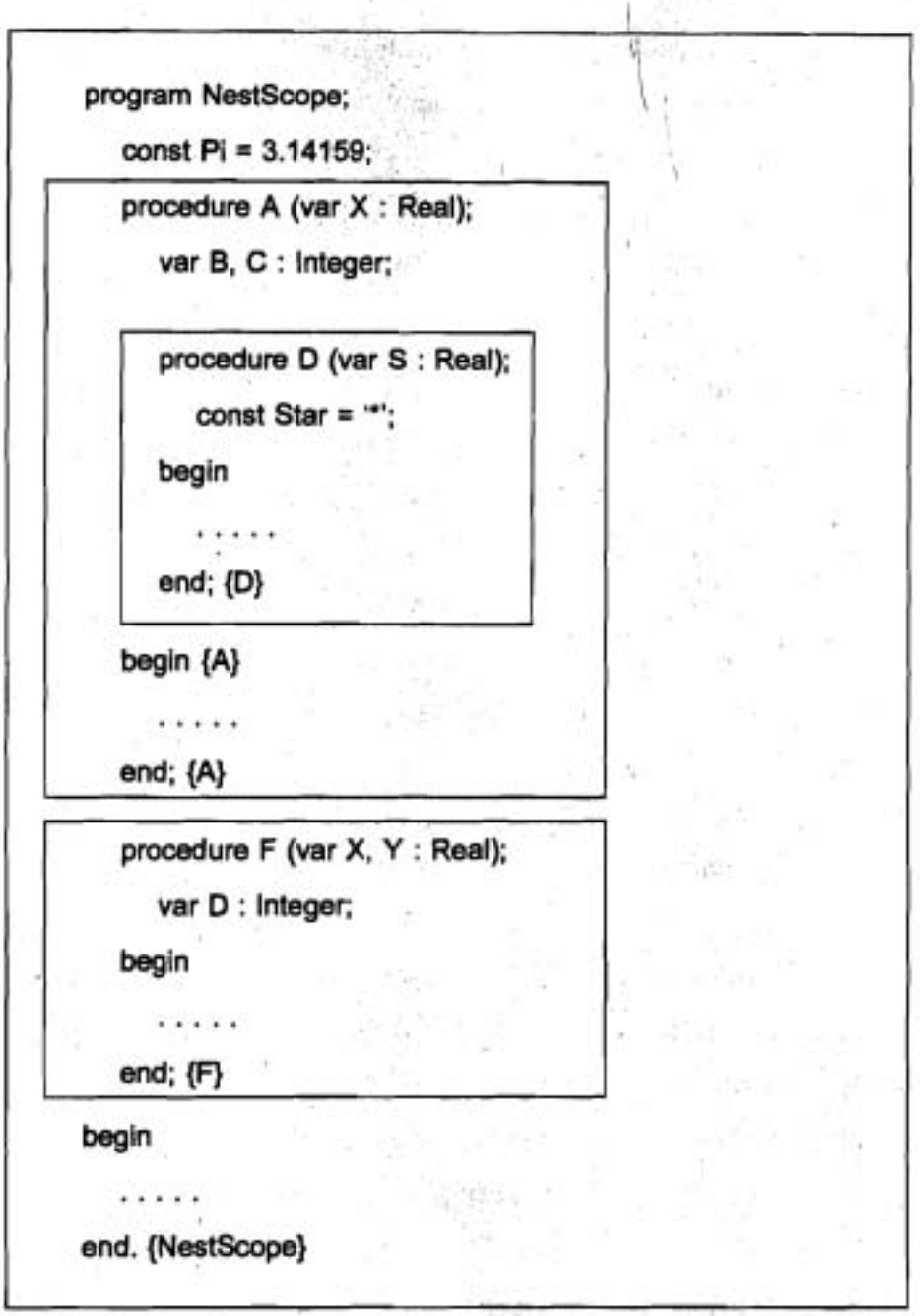
ใส่ในโปรแกรมหลัก

(b) จงแสดงค่าใหม่ของ W, X และ Y ถ้า X เป็นพารามิเตอร์จริงในการเรียกกระบวนงาน Change

(c) จะเกิดอะไรขึ้น ถ้า Y เป็นพารามิเตอร์จริง ในการเรียกกระบวนงาน Change

(d) จะเกิดผลอะไรของการทำพารามิเตอร์รูปนัย X เป็นพารามิเตอร์ค่า

5. จากเค้าร่าง (outline) ของโปรแกรมที่มีกระบวนงานซ้อนในข้างล่างนี้ จงอธิบายสโคปของไอเดนติไฟเออร์แต่ละตัว รวมทั้งชื่อกระบวนงาน



6.5 ฟังก์ชัน : ส่วนจำเพาะซึ่งกลับคืนหนึ่งผลลัพธ์ (Functions : Modules That a Single Result)

ฟังก์ชัน หมายถึง ส่วนจำเพาะอิสระคล้ายกระบวนงาน ยกเว้นกระบวนงานสามารถกลับคืนผลลัพธ์กี่จำนวนก็ได้ ในขณะที่ฟังก์ชันกลับคืนผลลัพธ์หนึ่งจำนวนเสมอ

ในหัวข้อ 3.2 เราได้แสดงให้เห็นการเรียก**ฟังก์ชันนิยามแล้ว** (predefind function) ว่าทำอย่างไร โดยเขียน function designator ในนิพจน์หนึ่งตัว ตัวอย่างเช่น ฟังก์ชัน designator

Abs (X+Y) เรียกฟังก์ชันนิยามแล้ว Abs (absolute value)

เมื่อ นิพจน์ X+Y คือพารามิเตอร์จริง ส่งไปฟังก์ชัน Abs ในหัวข้อนี้ เราศึกษาว่าจะเขียนฟังก์ชันของเราเองอย่างไร

การเขียนฟังก์ชันผู้ใช้กำหนด (Writing User-Defined Functions)

การให้นิยามฟังก์ชันของเราเอง เขียนการประกาศฟังก์ชัน และใส่มันไว้ในส่วนประกาศของโปรแกรมเรียก (calling program)

เนื่องจากฟังก์ชันไม่ได้กลับคืนผลลัพธ์ของมันผ่านรายการพารามิเตอร์ของมัน การประกาศฟังก์ชันจึงแตกต่างจากการประกาศกระบวนงานในวิธีข้างล่างนี้

- หัวเรื่องของฟังก์ชันขึ้นต้นด้วยคำสงวน function แทนที่จะเป็น procedure

```
function Exponent (U, V : Real) : Real ;
         ↑          ↑          ↑
      Function   Formal     Type of
       name     parameter    result
                 list
```

- พารามิเตอร์ทั้งหมดของฟังก์ชันต้องเป็นพารามิเตอร์ค่า
- แบบชนิดข้อมูลของผลลัพธ์ของฟังก์ชัน (function result) กำหนดที่ตอนท้ายของหัวเรื่องฟังก์ชัน ตามหลังรายการพารามิเตอร์รูปนัย
- ภายใน body ของฟังก์ชัน ผลลัพธ์ของฟังก์ชันถูกนิยามโดยกำหนดค่าให้กับชื่อฟังก์ชัน ค่าสุดท้ายกำหนดให้กับชื่อฟังก์ชันถูกกลับคืนเป็นผลลัพธ์ของฟังก์ชัน

ตัวอย่าง 6.7

ปกติโปรแกรมเมอร์ใช้ฟังก์ชันเพื่อทำให้การคำนวณเชิงตัวเลขง่ายขึ้นสำหรับกรณีตัวอย่าง เนื่องจาก Pascal ไม่มีตัวดำเนินการเลขชี้กำลัง (exponentiation operator) ฟังก์ชันคลัง (library function) สำหรับการยกกำลังเลขน่าจะเป็นประโยชน์ ฟังก์ชัน Exponent (รูป 6.12) ยกกำลังอาร์กิวเมนต์ตัวแรกของมันคือ U ให้กับกำลังซึ่งแสดงด้วยอาร์กิวเมนต์ตัวที่สองของมัน คือ V (ผลลัพธ์ของฟังก์ชันคือ U^V) หัวเรื่องฟังก์ชันแสดงว่า Exponent มีพารามิเตอร์ค่าชนิด Real สองตัว ชนิดของผลลัพธ์คือ Real อยู่หลัง colon

ข้อความสั่ง if ในตัวฟังก์ชัน เลือกข้อความสั่งกำหนดค่า หรือข้อความสั่ง WriteLn ขึ้นอยู่กับค่าที่ส่งมาให้ U ถ้าค่าของ U เท่ากับ 0 ข้อความสั่งกำหนดค่า

```
Exponent := 0.0
```

ถูกกระทำการ และนิยามผลลัพธ์ของฟังก์ชันเป็น 0 ถ้าค่าของ U เป็นค่าบวก (positive) ข้อความสั่งกำหนดค่า

```
Exponent := Exp (V*Ln(U))
```

เรียกฟังก์ชันในตัว (built-in function) ของ Pascal ชื่อ Exp และ Ln เพื่อคำนวณค่าที่ต้องการ โดยใช้สูตรที่ได้มาในตัวอย่าง 3.4 กำหนดค่านี้ให้ Exponent นิยามเป็นผลลัพธ์ของฟังก์ชัน

ถ้าค่าของ U เป็นลบ (negative) ข้อความสั่ง WriteLn แสดงข้อความระบุความผิดพลาด แต่ไม่มีค่าใดๆ กำหนดให้กับ Exponent ดังนั้นผลลัพธ์ของฟังก์ชันจึงไม่ถูกนิยาม (undefined) สิ่งนี้คือความต้องกันกับเงื่อนไขก่อน (precondition) ของฟังก์ชัน ซึ่งกล่าวว่าฟังก์ชัน Exponent จะถูกเรียกเฉพาะเมื่อพารามิเตอร์ตัวแรกของมันไม่ใช่ค่าลบเท่านั้น

โปรดสังเกตว่า ข้อความสั่งกำหนดค่าทั้งสองคำสั่ง ใช้ชื่อฟังก์ชัน Exponent โดยไม่มีพารามิเตอร์ ถ้าเราใช้พารามิเตอร์หลังชื่อฟังก์ชัน Pascal จะถือว่านี่เป็นการเรียกซ้ำ (reversive call) ฟังก์ชัน

```
function Exponent (U, V : Real) : Real;
{
Returns its first argument raised to the power specified by its second
argument.
Pre  :  U >= 0.0 and V is defined.
Post :  Returns U raised to the power V.
}
begin {Exponent}
  if  U = 0.0 then
    Exponent := 0.0      {Result of 0.0 to any power is 0.0}
  else if U > 0.0 then
```

```
        Exponent := Exp (V*Ln(U))   {Result is U raised to power V}
    else
        WriteLn ('**** Error in first parameter of Exponent')
end; {Exponent}
```

รูป 6.12 ฟังก์ชัน Exponent

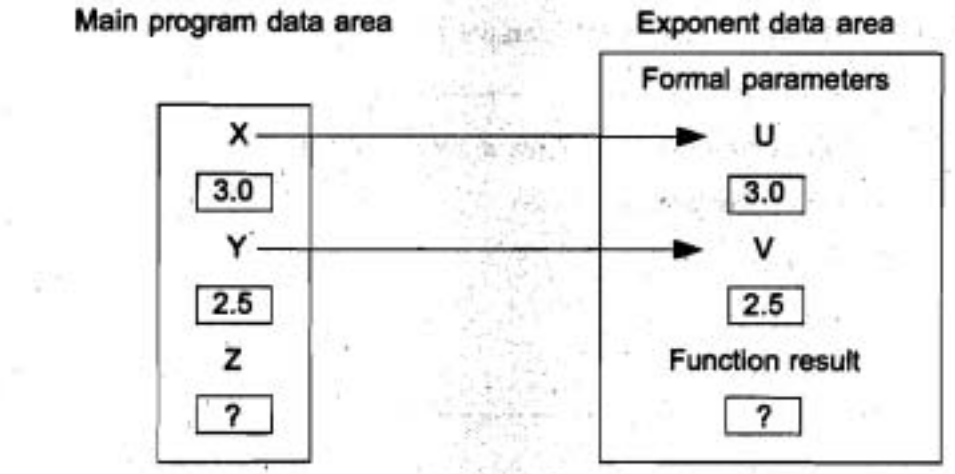
การเรียกฟังก์ชันผู้ใช้กำหนด (Calling a User-Defined Function)

ถ้าเรามีโปรแกรมหลักที่มีตัวแปรชนิด Real สามตัวคือ X, Y และ Z ข้อความสั่งในโปรแกรมหลัก

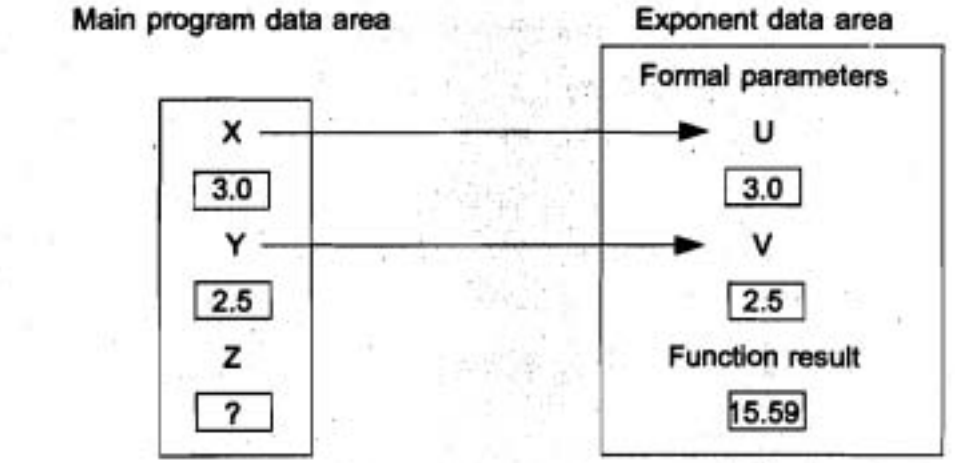
```
Z := Exponent (X, Y)
     ↑        ↑
  Function   Actual parameter
   name        list
```

เรียก Exponent โดยส่งค่าของพารามิเตอร์จริง X และ Y ตัวฟังก์ชันคำนวนค่าของ X^Y จากนั้นกลับคืนจากฟังก์ชันผลลัพธ์ของฟังก์ชันกำหนดให้ Z

รูป 6.13 แสดงพื้นที่ข้อมูลของโปรแกรมหลัก และพื้นที่ข้อมูลของฟังก์ชันหลังจากเรียกฟังก์ชัน สำหรับค่าของ X และ Y ที่แสดงไว้ (3.0 และ 2.5) การกระทำการฟังก์ชันนิยามผลลัพธ์ของฟังก์ชันเป็น 15.59 รูป 6.14 แสดงพื้นที่ข้อมูลของโปรแกรมและพื้นที่ข้อมูลของฟังก์ชัน หลังจากตัวฟังก์ชันเสร็จสิ้นการกระทำการ แต่ก่อนกลับคืนฟังก์ชัน โปรดสังเกตว่าไม่มีการเชื่อมต่อใดๆ ระหว่างตัวแปร Z ของโปรแกรมหลักและเซลล์หน่วยความจำในพื้นที่ข้อมูลของ Exponent ซึ่งแทนผลลัพธ์ของฟังก์ชัน ผลลัพธ์ของฟังก์ชัน จะถูกกำหนดให้ Z หลังจากการกลับคืนไปยังโปรแกรมหลัก



รูป 6.13 พื้นที่ข้อมูลหลังจากเรียกฟังก์ชัน



รูป 6.14 พื้นที่ข้อมูลหลังจากการกระทำการฟังก์ชัน แต่ก่อนกลับคืน

Syntax Display

การประกาศฟังก์ชัน (Function Declaration)

Form :

```
Function fname (formal parameters) : ftype;
   local declaration section
begin
   function body
end;
```

ตัวอย่าง

```
function InverseSum (X, Y : Real) : Real;
{Computer 1.0 divided by the sum of X, Y.}
   var
      SumXY : Real; {local storage for the sum of X, Y}
begin {InverseSum}
   SumXY := X+Y;
   InverseSum := 1.0/SumXY
end; {InverseSum}
```

มีความหมายดังนี้ ประกาศฟังก์ชัน ชื่อ fname ตามด้วย formal parameters อยู่ภายในเครื่องหมายวงเล็บ

ไอเดนติไฟเออร์ซึ่งประกาศใน local declaration section เป็นไอเดนติไฟเออร์เฉพาะที่ของฟังก์ชัน และถูกนิยามเฉพาะระหว่างการกระทำการของฟังก์ชันเท่านั้น formal parameters ไม่สามารถถูกประกาศเป็นไอเดนติไฟเออร์เฉพาะที่

function body อธิบายการจัดดำเนินการข้อมูลที่กระทำโดยฟังก์ชัน โดยใช้ formal parameters เป็นชื่อดัมมี (dummy names) ในการอธิบาย เมื่อ formal parameter ถูกอ้างถึงระหว่างการกระทำการของฟังก์ชัน ค่าของพารามิเตอร์จริงซึ่งสมนัยกัน จะถูกจัดดำเนินการ

ฟังก์ชันกลับคืน (returns) หนึ่งผลลัพธ์มีชนิดเป็น ftype ผลลัพธ์ของฟังก์ชัน คือ ค่าซึ่งกำหนดให้กับ fname ก่อนกลับคืน

ข้อสังเกตข้อ 1 ไอเดนติไฟเออร์ ftype ต้องเป็นชื่อแบบชนิดข้อมูลมาตรฐาน (Integer, Real, Boolean หรือ Char), ชนิด subrange หรือชนิด enumerated (อธิบายในบทที่ 7) Turbo Pascal (ไม่ใช่ Standard Pascal) ยอมให้ string และชนิดextended numeric (อธิบายในบทที่ 7) ใช้เป็น function return type ได้

ข้อสังเกตข้อที่ 2 ถ้าไม่มี parameters ส่วนที่เป็น formal parameters และเครื่องหมายวงเล็บปิดล้อม ไม่ต้องเขียน

Syntax Display

Function Designator (Function Call)

Form : fname (actual parameters)

ตัวอย่าง InverseSum (3.0, Z)

มีความหมายดังนี้ actual parameter อยู่ภายในเครื่องหมายวงเล็บ เมื่อฟังก์ชัน fname ถูกเรียกไปกระทำการ พารามิเตอร์จริงตัวแรกสมนัยกับพารามิเตอร์รูปนัยตัวแรก พารามิเตอร์จริงตัวที่สองสมนัยกับพารามิเตอร์รูปนัยตัวที่สอง เช่นนี้เรื่อยไป

function designator ต้องปรากฏภายในนิพจน์เสมอ

หลังจากกระทำการแล้ว ผลลัพธ์ของฟังก์ชัน แทนที่ function designator ในนิพจน์นั้น

ข้อสังเกต ถ้าไม่มีพารามิเตอร์, ไม่ต้องเขียนพารามิเตอร์จริง และเครื่องหมายวงเล็บ

วิศวกรรมซอฟต์แวร์ : หลีกเลี่ยงผลกระทบของฟังก์ชัน (Software Engineering : Avoiding Function Side Effects)

เนื่องจากพารามิเตอร์ฟังก์ชันทั้งหมดในหนังสือเล่มนี้เป็นอินพุตพารามิเตอร์ เราจึงไม่เขียนคอมเมนต์ {input} ในรายการพารามิเตอร์รูปนัยของฟังก์ชัน ข้อจำกัดนี้ถูกบังคับโดยสไตล์ของการเขียนโปรแกรม ไม่ใช่โดย Pascal เราสามารถกลับคืน (return) ผลลัพธ์เพิ่มเติมจากฟังก์ชัน โดยการทำหนึ่งในพารามิเตอร์ของมันให้เป็นพารามิเตอร์ตัวแปร แต่การเปลี่ยนแปลงใดๆ ของค่าของพารามิเตอร์ตัวนี้ จะไม่ถือว่าเป็น side effect เพราะว่าไม่ได้ถูกคาดคิดไว้ และจะขัดแย้งกับการปฏิบัติการเขียนโปรแกรมมาตรฐาน

การทดสอบฟังก์ชัน (Testing Functions)

ในตัวอย่าง 6.8 เราเขียนฟังก์ชันซึ่งคำนวณภาษีเงินได้ที่ต้องชำระและโปรแกรมขนาดเล็กซึ่งมีวัตถุประสงค์อย่างเดียวคือเพื่อเรียกและทดสอบฟังก์ชันเท่านั้น โปรแกรมที่ใช้ทดสอบการดำเนินการของฟังก์ชันหรือกระบวนงานเรียกว่า โปรแกรมตัวขับ

โปรแกรมตัวขับ หมายถึงโปรแกรมขนาดเล็ก เขียนขึ้นมาเพื่อใช้ทดสอบกระบวนงานหรือฟังก์ชัน (Driver program is a small program written to test a procedure or function.)

ตัวอย่าง 6.8

รูป 4.13 ประกอบด้วยข้อความสั่ง if ซึ่งคำนวณภาษีเงินได้สำหรับเงินเดือน คิดตามตารางภาษีที่แสดงในตาราง 4.10 เนื่องจากตารางนี้ปรากฏในโปรแกรมต่างๆ จำนวนมาก นักบัญชีของเราตัดสินใจว่าจะใส่มันในฟังก์ชัน FindTax (รูป 6.15) ข้อความสั่งกำหนดค่า

```
MyTax := FindTax (MySalary);
```

เรียกฟังก์ชัน FindTax เพื่อส่งค่าของ MySalary ไปยังอินพุตพารามิเตอร์ salary ถ้าค่าซึ่งส่งไปยัง Salary อยู่ในพิสัย (range) ของตาราง ภาษีซึ่งต้องชำระถูกคำนวณและกลับคืนเป็นผลลัพธ์ของฟังก์ชัน กรณีอื่นๆ ฟังก์ชันกลับคืน -1.0 เพื่อให้สัญญาณว่า เงินเดือนที่เป็นอินพุตอยู่นอกพิสัยตาราง

โปรดสังเกตว่า งานที่ประมวลผลถูกแบ่งอย่างไร ในรูป 6.15

ขั้นแรก ข้อความสั่ง ReadLn ใส่ค่าเงินเดือนในโปรแกรมหลัก ไม่ใช่ใส่ในฟังก์ชัน FindTax ค่าที่อ่านเข้ามาส่งไปยัง FindTax ผ่านทางรายการพารามิเตอร์ของมัน ไม่ต้องใส่ข้อความสั่งใดๆ เพื่ออ่านข้อมูลอินพุตของฟังก์ชันในตัวฟังก์ชัน

Edit Window

```
program Driver;
{Tests function FindTax}
  var
    MySalary,              {input-salary}
    MyTax : Real;          {output-tax}
```

```
function FindTax (Salary : Real) : Real;
{
   Returns tax anount owed for a salary < $15000.
   Pre  :  Salary is assigned a value.
   Post :  If Salary is within range, returns the tax owed; otherwise,
           returns -1.0
}
   const
      MaxSalary     =  15000.00;  {maximum salary for table}
      OutofRange    =  -1.0;      {"tax" for an out - of - range salary}

begin {FindTax}
   if  Salary < 0.0 then
      FindTax := OutofRange       {Salary too small}
   else if Salary < 1500.00 then  {first range}
      FindTax := 0.15 * Salary
   else if Salary < 3000.00 then  {second range}
      FindTax := (Salary - 1500.00) * 0.16 + 225.00
   else if Salary < 5000.00 then  {third range}
      FindTax := (Salary - 3000.00) * 0.18 + 465.00
   else if Salary < 8000.00 then  {fourth range}
      FindTax := (Salary - 5000.00) * 0.20 + 825.00
   else if Salary <= MaxSalary then {fifth range}
      FindTax := (Salary - 8000.00) * 0.25 + 1425.00
   else
      FindTax := OutofRange       {Salary too large}
end;  {FindTax}
```

```
begin {Driver}
   Write ('Enter a salary less than or equal to $15000.00 > $');
   ReadLn (MySalary);
   MyTax := FindTax (MySalary);
   if  MyTax >= 0.0 then
      WriteLn ('The tax on $', MySalary : 4 : 2, 'is $', MyTax : 4 : 2)
   else
      WriteLn ('Salary $', Salary : 4 : 2, 'is out of table range')
end. {Driver}
```

Output Window

```
Enter a Salary less than or equal to $15000.00 > $6000.00
The tax on $6000.00 is $1025.00
```

รูป 6.15 โปรแกรมตัวขับที่มีฟังก์ชัน FindTax

ข้อที่สอง ผลลัพธ์ของฟังก์ชัน ถูกคำนวณในฟังก์ชัน แต่แสดงผลโดยข้อความสั่ง WriteLn ในโปรแกรมเรียก โดยทั่วไปเราควรใช้ข้อความสั่ง WriteLn ในฟังก์ชันเฉพาะเพื่อแสดงข้อความระบุความผิดพลาดเท่านั้น

ข้อที่สาม โปรดสังเกตว่า ตัวฟังก์ชันให้นิยามผลลัพธ์ของฟังก์ชันโดยการกำหนดค่าให้กับชื่อฟังก์ชัน FindTax หลังจากกลับคืนฟังก์ชันแล้ว ข้อความสั่งกำหนดค่า

```
MyTax := FindTax (Salary);
```

ในโปรแกรมหลัก กำหนด (assigns) ผลลัพธ์ของฟังก์ชันให้ MyTax บางครั้งโปรแกรมเมอร์มือใหม่กำหนดค่าให้ MyTax ไม่ถูกต้อง แทนที่จะเป็น FindTax ในตัวฟังก์ชัน เพราะว่า MyTax เป็นตัวแปรส่วนกลาง ฟังก์ชันจะถูกคอมไพล์โดยไม่มีข้อผิดพลาด แต่ผลลัพธ์ของฟังก์ชันจะไม่ถูกนิยาม ดังนั้นขยะ (garbage) จะถูกกำหนดค่าให้ MyTax หลังจากฟังก์ชันกลับคืน

สไตล์โปรแกรม (Program Style)

การตรวจสอบความสมเหตุสมผลของอินพุตพารามิเตอร์ (Validating Input Parameters)

ข้อความสั่ง if ในฟังก์ชัน FindTax ทดสอบค่าที่ถูกต้องของอินพุตพารามิเตอร์ Salary ก่อนกระทำการคำนวณภาษี กระบวนงานและฟังก์ชันทั้งหมดควรจะตรวจสอบความสมเหตุสมผลของอินพุตพารามิเตอร์ของมัน ไม่มีการรับประกันใดๆ ว่าค่าต่างๆ ที่ส่งไปยังอินพุตพารามิเตอร์จะมีความหมาย

สไตล์โปรแกรม (Program Style)

ส่วนจำเพาะติดกัน (Cohesive modules)

ฟังก์ชัน FindTax คำนวณเฉพาะภาษีเงินเดือนเท่านั้น

ฟังก์ชันนี้ไม่ได้อ่านค่าสำหรับ Salary และไม่ได้แสดงผลผลลัพธ์ของการคำนวณ และไม่ได้แสดงข้อความระบุความผิดพลาด ถ้าค่าที่ส่งไปยัง Salary อยู่นอกพิสัย มันกลับคืนค่าพิเศษ (-1.0) และโปรแกรมเรียกแสดงผลข้อความระบุความผิดพลาด

ส่วนจำเพาะซึ่งกระทำการดำเนินการหนึ่งอย่างเรียกว่า **ส่วนจำเพาะติดกัน** (Modules that perform a single operation are called cohesive modules.)

การเขียนส่วนจำเพาะติดกัน เป็นสไตล์ของการเขียนโปรแกรมที่ดี ซึ่งจะช่วยทำให้ฟังก์ชันกระบวนงานค่อนข้างจะกระชับ และง่ายต่อการอ่าน เขียน และแก้จุดบกพร่อง

สไตล์โปรแกรม (Program Style)

การเขียนโปรแกรมตัวขับเพื่อทดสอบส่วนจำเพาะ (Writing Driver Programs to Test Modules)

ตัวโปรแกรมหลักในรูป 6.15 ประกอบด้วยข้อความสั่งสำหรับการรับเข้า (entry) ข้อมูล ข้อคามสั่งกำหนดค่าที่มี function designator ในส่วนนิพจน์ของมัน และข้อความสั่ง if เพื่อแสดงผลผลลัพธ์ของฟังก์ชัน วัตถุประสงค์หนึ่งเดียวของโปรแกรมคือการทดสอบฟังก์ชัน FindTax โปรแกรมเช่นนี้เรียกว่า **โปรแกรมตัวขับ**

โปรแกรมเมอร์ที่มีประสบการณ์ บ่อยครั้งใช้โปรแกรมตัวขับ เพื่อการทดสอบก่อน (pretest) ทั้งฟังก์ชันและกระบวนงาน โดยทั่วไปการลงทุนเล็กๆ ในเวลาและความพยายามที่จะเขียนโปรแกรมตัวขับสั้นๆ ให้ผลคุ้มค่า (pay off) กับการลดเวลาทั้งหมดที่ใช้ไปในการแก้จุดบกพร่องระบบโปรแกรมขนาดใหญ่ ซึ่งประกอบด้วยส่วนจำเพาะหลายชุด

แบบฝึกหัด 6.5 Self - Check

1. เนื่องจากทั้งกระบวนงานและฟังก์ชัน กลับคืนผลลัพธ์ และกระบวนงานไม่ได้จำกัดว่าต้องเป็นหนึ่งเอาต์พุต ทำไมเราจึงต้องการฟังก์ชัน

2. ในฟังก์ชัน FindTax ในรูป 6.15 ทำไมเราจึงไม่แทนที่ข้อความสั่งกำหนดค่า

```
FindTax := OutofRange
```

ด้วยข้อความสั่ง WriteLn

```
WriteLn (Salary : 4 : 2, 'is out of range')
```

3. a) ฟังก์ชันข้างล่างนี้ทำอะไร

```
function Hypot (X, Y : Real) : Real;
begin {Hypot}
   Hypot := Sqrt (Sqr (X) + Sqr (Y))
end {Hypot}
```

b) จงเขียนข้อความสั่งซึ่งเรียกฟังก์ชันนี้ที่มีอาร์กิวเมนต์เป็น A และ B และเก็บผลลัพธ์ของฟังก์ชันใน C

4. งานต่อไปนี้ชุดใดควรจะ implemented ดีที่สุด เป็นฟังก์ชัน และชุดใดเป็นกระบวนงาน ให้เหตุผลอธิบายคำตอบ

a) คำนวณปริมาตรของรูปทรงกรวย (cone)

b) แสดง user instructions

c) ทดสอบว่าค่าของข้อมูลอยู่ในพิสัยที่กำหนดไว้

d) การคำนวณมุมและความเร็วต้นของกระสุน (muzzle velocity) สำหรับปืนใหญ่ตามพิสัยที่ต้องการ

เขียนโปรแกรม (Programming)

1. จงเขียนโปรแกรมตัวขับ ซึ่งทดสอบฟังก์ชัน FindTax สำหรับทุกค่าของ Salary จาก -400.00 ถึง 15100.00 เพิ่มขึ้นครั้งละ 500.00

6.6 การออกแบบการแบ่งละเอียดที่มีฟังก์ชันและกระบวนงาน (Stepwise Design with Functions and Procedures)

การใช้รายการพารามิเตอร์เพื่อส่งสารสนเทศไปและกลับ จากกระบวนงานและฟังก์ชันจะปรับปรุงทักษะการแก้ปัญหาของเราให้ดีขึ้น ถ้าผลเฉลยให้กับปัญหาย่อย ไม่

สามารถเขียนได้ง่ายเพียงแค่ข้อความสั่ง Pascal ไม่กี่ประโยคให้ลงรหัส (code) เป็นกระบวนงานหรือฟังก์ชัน กรณีศึกษาข้างล่างนี้แสดงให้เห็นการออกแบบการแบ่งละเอียดของโปรแกรมโดยใช้กระบวนงานและฟังก์ชัน

กรณีศึกษา ปัญหา ผลบวกและค่าเฉลี่ย โดยทั่วไป (Case Study : General Sum-and-Average Problem)

ปัญหา (Problem)

สะสมผลบวกและค่าเฉลี่ย รายการของค่าข้อมูล โดยใช้ส่วนจำเพาะกระบวนงานและฟังก์ชัน เนื่องจากงานเหล่านี้มีอยู่ในปัญหาจำนวนมาก การออกแบบเชตทั่วไปของส่วนจำเพาะ เราสามารถนำกลับมาใช้ใหม่ในโปรแกรมอื่นๆ

วิเคราะห์ (Analysis)

ส่วนวนซ้ำ (loop) ในรูป 5.4 คำนวณเงินเดือนรวมของบริษัทแห่งหนึ่ง เราสามารถใช้ส่วนวนซ้ำที่คล้ายกันเพื่อคำนวณผลบวกของกลุ่มของค่าข้อมูล การคำนวณค่าเฉลี่ย เราหารผลบวกด้วย จำนวนหน่วยข้อมูลทั้งหมด ให้ระวังจะไม่กระทำการหารนี้ ถ้าจำนวนหน่วยข้อมูลเท่ากับ 0

ความต้องการข้อมูล (Data Requirements)

อินพุตปัญหา (Problem Inputs)

```
NumItem : Integer   {number of data items to be summed}
Item : Real         {each data item}
```

เอาต์พุตปัญหา (Problem Outputs)

```
Sum : Real          {sum of data items}
Average : Real      {average of data}
```

สูตรที่เกี่ยวข้อง (Relevant Formula)

$$\text{average} = \frac{\text{sum of data}}{\text{number of data items}}$$

ออกแบบ (Design)

อัลกอริทึมเริ่มต้น

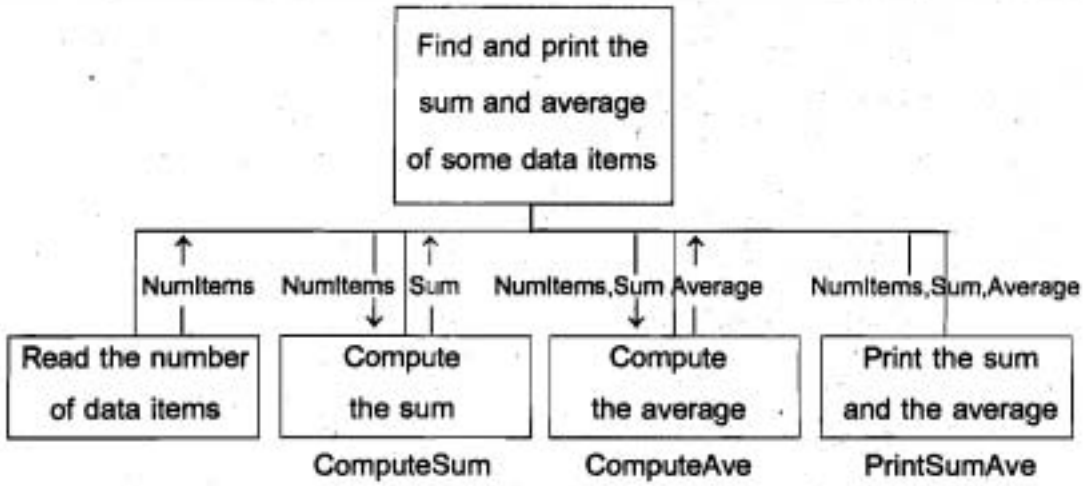
1. อ่านจำนวนหน่วยข้อมูล
2. อ่านข้อมูลและคำนวณผลบวกของข้อมูล

3. คำนวณค่าเฉลี่ยของข้อมูล

4. พิมพ์ผลบวกและค่าเฉลี่ย

ผังโครงสร้าง (structure chart) ในรูป 6.16 แสดงให้เห็นกระแสข้อมูล (data flow) ระหว่างปัญหาหลักกับปัญหาย่อยของมัน เราจะ implement เป็นส่วนจำเพาะแยกจากกัน แต่ละขั้นตอนซึ่งมีผลเฉลยที่สำคัญ label ได้ขั้นตอนหมายถึงชื่อของส่วนจำเพาะซึ่ง implement ขั้นตอนนี้ แต่ละขั้นตอน ยกเว้นขั้นตอนแรก ถูก implemented ในส่วนจำเพาะแยกจากกัน

รูป 6.16 ทำให้เห็นกระแสข้อมูล ชัดเจนระหว่างโปรแกรมหลักกับส่วนจำเพาะแต่ละชุด ตัวแปรทั้งหมด ซึ่งค่าของมันถูกกำหนดโดยส่วนจำเพาะ เรียกว่า**เอาต์พุตส่วนจำเพาะ** (module outputs)



รูป 6.16 ผังโครงสร้างที่มีสารสนเทศกระแสข้อมูล

(แสดงด้วยลูกศรชี้ออกจากส่วนจำเพาะ) ตัวแปรทั้งหมดซึ่งค่าของมันใช้ในการคำนวณ แต่ไม่มีการเปลี่ยนแปลง โดยส่วนจำเพาะเรียกว่า **อินพุตส่วนจำเพาะ** (module inputs) (แสดงด้วยลูกศรชี้ไปยังส่วนจำเพาะ) บทบาทของตัวแปรโปรแกรมแต่ละตัว ขึ้นอยู่กับการใช้ของมันในส่วนจำเพาะ และเปลี่ยนแปลงจากขั้นตอนหนึ่งไปยังอีกขั้นตอนหนึ่งในผังโครงสร้าง

เนื่องจากขั้น Real the number of data items ให้นิยามค่าของ NumItems ตัวแปรตัวนี้จึงเป็นเอาต์พุตของขั้นตอนนี้ ส่วนจำเพาะ ComputeSum ต้องการค่าของ NumberItems เพื่อทราบจำนวนของหน่วยข้อมูล (data item) ที่จะอ่านและคำนวณผลบวก ดังนั้น NumItems จึงเป็นอินพุตของส่วนจำเพาะ ComputeSum ตัวแปร Sum คือเอาต์พุตของส่วนจำเพาะ ComputeSum แต่เป็นอินพุตของส่วนจำเพาะ ComputeAve และ PrintSumAve ตัวแปร Average เป็นเอาต์พุตของส่วนจำเพาะ ComputeAve แต่เป็นอินพุตของส่วนจำเพาะ PrintSumAve

การทำให้เกิดผล (Implementation)

การใช้สารสนเทศกระแสข้อมูลในผังโครงสร้าง เราเขียนโปรแกรมหลักก่อน การแบ่งละเอียดอัลกอริทึม ให้ทำตามวิธีซึ่งอธิบายไว้ในหัวข้อ 3.1 เพื่อเขียนโปรแกรมหลักเริ่มต้นโดยการทำสำเนาความต้องการข้อมูลไว้ในส่วนการประกาศของโปรแกรม ประกาศตัวแปรทั้งหมดซึ่งปรากฏในผังโครงสร้างในโปรแกรมหลัก เพราะว่ามันเก็บข้อมูลที่จะส่งไปยังส่วนจำเพาะ หรือเก็บผลลัพธ์ซึ่งกลับคืนจากส่วนจำเพาะ ไม่ต้องประกาศตัวแปร Itme เพราะว่ามันไม่ปรากฏในผังโครงสร้าง แต่ต้องแน่ใจว่าได้ประกาศตัวแปรนี้ในส่วนจำเพาะซึ่งใช้มัน (ComputeSum) ถัดไปย้ายอัลกอริทึมแรกไปไว้ในตัวโปรแกรม เขียนขั้นตอนอัลกอริทึมแต่ละชุดเป็นคอมเมนต์ (รูป 6.17)

```
program SumItems;
{Finds and prints the sum and average of a list of data items}
  var
    NumItems : Integer ;   {input - number of items to be added}
    Sum,                   {output - sum being accumulated}
    Average : Real         {output - average of the data}
  {
  Insert declarations for procedures ComputeSum and PrintSumAve and
  function ComputeAve here.
  }
```

```
begin    {SumItems}
   {Real the number of items.}
   Write ('How many numbers will be added> ');
   ReadLn (NumItems);

   {Real the data items and compute their sum.}
   ComputeSum (NumItems, Sum);

   {Compute the average of the data.}
   Average := ComputeAve (NumItems, Sum);

   {Print the sum and average.}
   PrintSumAve (NumItems, Sum, Average)
end. {SumItems}
```

รูป 6.17 โปรแกรมหลักสำหรับปัญหาผลบวก-และ-ค่าเฉลี่ยโดยทั่วไป

เพื่อทำให้โปรแกรมหลักเสร็จบริบูรณ์ ลงรหัสขั้นอัลกอริทึมแต่ละชุด in-line (เป็นส่วนรหัสของโปรแกรมหลัก) หรือเป็นการเรียกกระบวนงานหรือเรียกฟังก์ชัน ลงรหัสขึ้นนำข้อมูลเข้า in-line เนื่องจากมันประกอบด้วย Write (สำหรับข้อความพร้อมรับ) และ ReadLn

ส่วนจำเพาะ ComputeSum และ ComputeAve กลับคืนหนึ่งผลลัพธ์ ดังนั้นเราอาจเขียนเป็นฟังก์ชัน อย่างไรก็ตามในที่นี้รหัส ComputeSum เป็นกระบวนงานเพราะว่ามันอ่านหน่วยข้อมูล ซึ่งจะรวมในผลบวก การปฏิบัติการนำข้อมูลเข้าอาจมีผลกระทบ (side effect) ถ้า ComputeSum เป็นฟังก์ชัน

สารสนเทศกระแสข้อมูลในรูป 6.16 บอกถึง พารามิเตอร์จริงที่ใช้ในการเรียกกระบวนงานหรือฟังก์ชันแต่ละชุด ในกรณีของฟังก์ชัน มันบอกชื่อตัวแปรของโปรแกรมหลักซึ่งจะเก็บผลลัพธ์ของฟังก์ชัน ตัวอย่างเช่น

ข้อความสั่งกำหนดค่า

Average := ComputeAve (NumItems, Sum);

เรียก ComputeAve และกำหนดให้เป็นค่าของ Average ในการเรียกนี้ NumItems และ Sum ถูกส่งเป็นอินพุตพารามิเตอร์ไปยังฟังก์ชัน ComputeAve

เรียก ComputeSum โดยใช้ข้อความสั่งเรียกกระบวนงาน

ComputeSum (NumItems, Sum);

ในที่นี้ NumItems เป็นอินพุตพารามิเตอร์ และ Sum เป็นเอาต์พุตพารามิเตอร์

เรียก PrintSumAve โดยใช้ข้อความสั่งเรียกกระบวนงาน

PrintSumAve (NumItems, Sum, Average)

ในที่นี้ตัวแปรทั้งสามตัว ทั้งหมดเป็นอินพุตของกระบวนงาน

กระบวนงาน ComputeSum

เมื่อโปรแกรมหลักเสร็จบริบูรณ์แล้ว เราสามารถเน้นที่ส่วนจำเพาะแต่ละชุดของมัน เริ่มต้นจากกระบวนงาน ComputeSum เริ่มต้น ความต้องการข้อมูล สำหรับ ComputeSum ด้วยอินพุตและเอาต์พุตของกระบวนงาน ComputeSum ต้องการตัวแปรเฉพาะที่สองตัว ตัวหนึ่งสำหรับเก็บหน่วยข้อมูล (Item) แต่ละตัว และอีกตัวหนึ่งสำหรับควบคุมการวนซ้ำ (Count)

Data Requirements for ComputeSum

อินพุตของกระบวนงาน (Procedure input)

NumItems : Integer {number of data items to be sumed}

เอาต์พุตของกระบวนงาน (Procedure output)

Sum : Real {the sum of the data items}

ตัวแปรเฉพาะที่ (Local variabels)

Item : Real {each data item}

Count : Integer {count of data items summed}

ก่อนสะสมผลบวกในการวนซ้ำ (loop) เริ่มต้นให้ sum มีค่าเท่ากับศูนย์ ก่อนเข้าการวนซ้ำ (ดูหัวข้อ 5.2) ขั้น loop-control ต้องมั่นใจว่า จำนวนหน่วยข้อมูลที่อ่านถูกต้อง และนำมาสะสมในผลบวก เนื่องจากเราทราบจำนวน items (NumItems) ที่จะบวก เราจึงใช้ลูปชนิดนับจำนวน (counting loop) ใช้ขั้นตอนเหล่านี้เพื่อเขียนอัลกอริทึม สำหรับ ComputeSum รูป 6.18 แสดงรหัสของ ComputeSum

อัลกอริทึมสำหรับ ComputeSum

1. Initialize Sum to 0
2. for each value of Count from 1 to NumItems do
   begin
      3. Read in the next item.
      4. Add it to Sum
   end

รูป 6.19 แสดงการสมนัยของพารามิเตอร์ ระบุโดยข้อความสั่งเรียกกระบวนงาน

ComputeSum (NumItems, Sum);

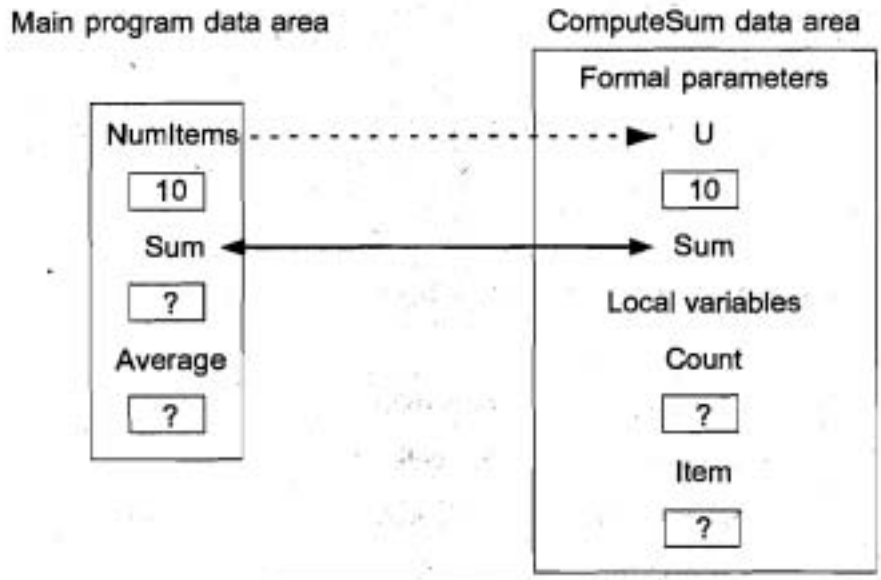
สมมติว่า ค่า 10 ถูกอ่านเข้าไปยัง NumItems ก่อนเรียกกระบวนงาน ตัวแปรเฉพาะที่ Count และ Item ไม่ถูกนิยาม (undefined) เมื่อกระบวนงานถูกเรียก กระบวนงานเริ่มต้นด้วยตัวแปรของโปรแกรมหลัก Sum มีค่าเป็น 0 ซึ่งสมนัยกับพารามิเตอร์ตัวแปร Sum

ลูป for อ่านหน่วยข้อมูลครั้งละหนึ่งตัวเข้าไปยังตัวแปรเฉพาะที่ Item และบวกข้อมูลตัวนี้กับตัวแปร Sum ของโปรแกรมหลัก ออกจาก loop และกลับคืนกระบวนงานหลังจากบวก Item 10 ตัว

```
procedure ComputeSum (NumItems {input} : Integer;
                      var Sum {output} : Real);
{
   Computes the sum of a list of NumItems data items.
   Pre  :  NumItems is assigned a value.
   Post :  NumItems data items are read; their sum is stored in Sum.
}
   var
      Count : Integer; {count of items added so far}
      Item  : Real;  {the next data item to be added}
```

```
begin {ComputeSum}
   {Read each data item and add it to Sum.}
   Sum := 0.0;
   for Count := 1 to NumItems do
      begin
         Write ('Next number to be added > ' );
         ReadLn (Item);
         Sum := Sum + Item
      end {for}
end; {ComputeSum}
```

รูป 6.18 กระบวนงาน ComputeSum



รูป 6.19 การสมนัยของพารามิเตอร์สำหรับ ComputeSum (NumItems, Sum)

ฟังก์ชัน ComputeAve และกระบวนงาน PrintSumAve

ComputeAve และ PrintSumAve ทั้งคู่ค่อนข้างตรงไปตรงมา จะเห็นได้จากรายการแสดงความต้องการข้อมูลและอัลกอริทึมของมัน อัลกอริทึมทั้งสองชุดทดสอบ NumItems เพราะว่ามันจะไม่มีความหมายแต่อย่างใด ที่คำนวณหรือแสดงผลค่าเฉลี่ยของหน่วยข้อมูล ถ้า NumItems ไม่ใช่ค่าบวก รูป 6.20 แสดงฟังก์ชัน ComputeAve และรูป 6.21 แสดงกระบวนงาน PrintSumAve

Data Requirements for ComputeAve

อินพุตของฟังก์ชัน (Function Inputs)

NumItems : Integer {the number of data items}

Sum : Real {the sum of all data}

เอาต์พุตของฟังก์ชัน (Function Output)

ComputeAve : Real {the average of the data}

Algorithm for ComputeAve

1. if NumItems is positive then
   2. Set ComputeAve to Sum/NumItems
   else
   3. Set ComputeAve to 0

```
function ComputeAve (NumItems : Integer; Sum : Real) : Real;
{
   Returns the average of NumItems data items with sum of Sum.
   Pre  :  NumItems and Sum are defined.
   Post :  If NumItems is positive, returns Sum / NumItems; otherwise,
              return zero
}
begin   {ComputeAve}
   {Compute the average of the data.}
```

```
    if NumItems > 0 then
        ComputeAve := Sum / NumItems
    else
        ComputeAve := 0.0
end;  {ComputeAve}
```

รูป 6.20 ฟังก์ชัน ComputeAve

Data Requirements for PrintSumAve

อินพุตของโปรซีเดอร์ (Procedure Inputs)

NumItems : Integer {the number of data items}

Sum : Real {the sum of all data}

Average : Real {the average of the data}

อัลกอริทึมสำหรับ PrintSumAve

1. If NumItems is positive then
   2. Display the sum and the average of the data

```
procedure PrinsSumAve (NumItems {input} : Integer;
                       Sum, Average {input} : Real);
{
   Displays the sum and average of NumItems data items.
   Pre  :  NumItems, Sum, and Average are defined.
   Post :  Displays Sum and also Average if NumItems > 0.
}
begin {PrintSumAve}
   if  NumItems > 0 then
      begin
```

```
        WriteLn ('The sum is ' , Sum : 4 : 2);
        WriteLn ('The average is ' , Average : 4 : 2)
      end
    else
      WriteLn ('Sum and average are not defined')
end;  {PrintSumAve}
```

รูป 6.21 กระบวนงาน PrintSumAve

การทดสอบ (Testing)

เราต้องใส่การประกาศฟังก์ชันและกระบวนงานในส่วนการประกาศของโปรแกรม SumItems (หลังการประกาศตัวแปร) ก่อนที่จะวิ่ง (run) โปรแกรม การทดสอบ SumItems ทำให้มั่นใจว่าโปรแกรมแสดงผล ผลบวก และค่าเฉลี่ยถูกต้อง เมื่อ NumItems มีค่าเป็นบวก และแสดงข้อความระบุความผิดพลาด เมื่อ NumItems มีค่าเท่ากับศูนย์หรือเป็นค่าลบ รูป 6.22 แสดงการวิ่งตัวอย่าง (sample run)

```
How many numbers will be added > 3
Next number to be added > 5.0
Next number to be added > 6.0
Next number to be added > -7.0
The sum is 4.00
The average is 1.33
```

รูป 6.22 Sample Run for SumItems

เมื่อไรใช้ฟังก์ชันเมื่อไรใช้กระบวนงาน (When to Use a Function or Procedure)

ในสี่ขั้นตอนของโปรแกรม sum-and-average โดยทั่วไป (ดูรูป 6.16) ขั้นตอนทั้งหมดยกเว้นขั้นตอนแรก ซึ่งสามารถกระทำโดยส่วนจำเพาะแยกจากกัน จะเห็นชัดว่า ขั้นที่ 1 สามารถปฏิบัติให้เกิดผลโดยใช้ Write และ ReadLn ดังนั้น ขั้นตอนนี้จึงเขียนโดยตรงในโปรแกรมหลัก เราใช้กระบวนงาน (ComputeSum) สำหรับขั้นที่ 2 เพราะว่าอัลกอริทึมของมันค่อนข้างซับซ้อน ขั้นที่ 3 และขั้นที่ 4 ค่อนข้างปฏิบัติให้เกิดผลได้ง่าย เราใช้ฟังก์ชัน (ComputeAve) สำหรับขั้นที่ 3 และใช้กระบวนงาน (PrintSumAve) สำหรับขั้นที่ 4 เพราะว่าการทำให้เกิดผลค่อนข้างยาว นี่คือเหตุผลในการกำหนดว่า การ implement หนึ่งขั้นตอนจะเป็นส่วนจำเพาะแยกจากกันหรือไม่ จากจุดนี้ ในตัวโปรแกรมหลักจึงประกอบด้วยลำดับของการเรียกกระบวนงานและฟังก์ชัน

การประกาศหลายครั้งของไอเดนติไฟเออร์ในโปรแกรม (Multiple Declarations of Identifiers in a Program)

ไอเดนติไฟเออร์ Sum และ NumItems ถูกประกาศเป็นตัวแปรส่วนกลาง ในโปรแกรมหลัก และเป็นตัวแปรรูปนัยในส่วนจำเพาะสามชุด ซึ่งเรียกโดยโปรแกรมหลัก จากการอภิปรายเรื่องสโคปของไอเดนติไฟเออร์ เราทราบแล้วว่า การประกาศแต่ละชุดมีสโคปของมันเอง และสโคปสำหรับพารามิเตอร์รูปนัยแต่ละตัวคือส่วนจำเพาะซึ่งประกาศตัวมัน รายการพารามิเตอร์เกี่ยวข้องกับตัวแปรส่วนกลาง Sum กับไอเดนติไฟเออร์อื่นๆ แต่ละตัวชื่อ Sum ค่าของตัวแปรส่วนกลาง Sum ถูกนิยามตั้งแต่แรก เมื่อกระบวนงาน ComputeSum กระทำการ

เพราะว่าตัวแปรส่วนกลาง Sum สมนัยกับเอาต์พุตพารามิเตอร์ Sum ของกระบวนงาน ComputeSum ค่านี้จึงถูกส่งไปยังฟังก์ชัน ComputeAve เพราะว่าตัวแปรส่วนกลาง Sum สมนัยกับอินพุตพารามิเตอร์ Sum ของฟังก์ชัน ComputeAve เช่นนี้เรื่อยไป

เพื่อหลีกเลี่ยงความสับสนที่เห็นการประกาศไอเดนติไฟเออร์ Sum ในส่วนจำเพาะหลายชุด ขอแนะนำว่าให้ตั้งชื่อแตกต่างกัน ในส่วนจำเพาะแต่ละชุด (ตัวอย่างเช่น Total, MySum) อย่างไรก็ตาม โปรแกรมจะอ่านง่ายกว่าถ้าใช้ชื่อ Sum ตลอด

เมื่ออ้างถึงผลบวก ของค่าข้อมูล ให้มั่นใจว่าเราจำการเชื่อมโยงการใช้แยกจากกันเหล่านี้ของไอเดนติไฟเออร์ Sum ได้ตลอดรายการพารามิเตอร์

แบบฝึกหัด 6.6 Self-Check

1. กระบวนงาน ComputeSum กลับคืนหนึ่งค่า ทำไมเราจึงไม่ implemented เป็นฟังก์ชัน

2. จงวาดรูปพื้นที่ข้อมูลของโปรแกรมหลัก และพื้นที่ข้อมูลของฟังก์ชัน สำหรับการเรียก ComputeAve สมมติว่า Sum มีค่าเท่ากับ 100.00 และ NumItems มีค่าเท่ากับ 10

3. จงวาดรูปพื้นที่ข้อมูลของโปรแกรมหลักและพื้นที่ข้อมูลของกระบวนงานสำหรับการเรียก PrintSumAve

6.7 การแก้จุดบกพร่องและการทดสอบโปรแกรมที่มีส่วนจำเพาะ (Debugging and Testing Programs with Modules)

เมื่อจำนวนข้อความสั่งในโปรแกรม หรือส่วนจำเพาะเพิ่มขึ้น มีความเป็นไปได้ที่ข้อผิดพลาดจะเพิ่มขึ้น การเฝ้าระวัง ส่วนจำเพาะแต่ละชุด ให้มีขนาดที่สามารถจัดการได้ ดูเหมือนจะทำให้ข้อผิดพลาดต่ำลง และสนับสนุนการอ่านและการทดสอบส่วนจำเพาะแต่ละชุด

เราสามารถทำให้กระบวนการเขียนโปรแกรมภาพรวมง่ายขึ้น โดยการเขียนโปรแกรมขนาดใหญ่เป็นเซตของส่วนจำเพาะซึ่งเป็นอิสระจากกัน การทดสอบและการแก้จุดบกพร่องของโปรแกรมที่มีส่วนจำเพาะหลายชุด จะง่ายขึ้นถ้าเราทดสอบตามระยะ (stages) เช่น การเกิดโปรแกรม (program evolves)

การทดสอบแบ่งออกเป็นสองชนิด ได้แก่ การทดสอบจากบนลงล่าง และการทดสอบจากล่างขึ้นบน (top-down testing and bottom-up testing)

เราสามารถใช้วิธีเหล่านี้รวมกัน (combination) เพื่อทดสอบโปรแกรมและส่วนจำเพาะของมัน

การทดสอบจากบนลงล่างและ Stubs (Top-Down Testing and Stubs)

ไม่ว่าจะเป็นโปรแกรมเมอร์หนึ่งคนหรือทีมผู้เขียนโปรแกรม ซึ่งกำลังพัฒนาระบบโปรแกรม (โปรแกรมหลักและส่วนจำเพาะของมัน) ส่วนจำเพาะทั้งหมดนั้นจะไม่เสร็จเรียบร้อย ณ เวลาเดียวกัน อย่างไรก็ตาม สิ่งที่เป็นไปได้ คือ ทดสอบสายงานรวมของการควบคุม (overall flow of control) ระหว่างโปรแกรมหลัก กับส่วนจำเพาะระดับที่ 1 ของมัน และทดสอบ และแก้ไขจุดบกพร่องส่วนจำเพาะระดับที่ 1 ซึ่งเสร็จสมบูรณ์แล้ว

กระบวนการของการทดสอบสายงานการควบคุมระหว่างโปรแกรมหลักและส่วนจำเพาะย่อยของมันเรียกว่า **การทดสอบจากบนลงล่าง** (The process of testing the flow of control between a main program and its subordinate modules is called top-down testing.)

เนื่องจากโปรแกรมหลักเรียกส่วนจำเพาะระดับที่ 1 ทั้งหมดเราจำเป็นต้องมีตัวแทน (substitute) เรียกว่า Stub สำหรับส่วนจำเพาะทั้งหมด ซึ่งยังไม่ได้ลงรหัส

Stub หมายถึง ส่วนจำเพาะ ซึ่งประกอบด้วยหัวเรื่องกระบวนงาน หรือฟังก์ชัน ตามด้วย minimal body ซึ่งจะแสดงข้อความระบุถึงส่วนจำเพาะที่กำลังกระทำการ และควรกำหนดค่าอย่างง่ายให้กับเอาต์พุต รูป 6.23 แสดงให้เห็น stub สำหรับกระบวนงาน ComputeSum ซึ่งใช้ในการทดสอบโปรแกรมหลัก รูป 6.17 ใน Stub กำหนดค่า 100.0 ให้กับพารามิเตอร์เอาต์พุต Sum ซึ่งเป็นข้อมูลที่สมเหตุสมผลสำหรับส่วนจำเพาะที่เหลือ เพื่อประมวลผลการตรวจสอบ เอาต์พุตโปรแกรมบอกเราว่า โปรแกรมหลักเรียกส่วนจำเพาะระดับ 1 ของมัน ในลำดับที่ต้องการหรือไม่ และข้อมูลไหล (flows) ระหว่างโปรแกรมหลักกับส่วนจำเพาะระดับ 1 ของมันอย่างถูกต้องหรือไม่

การทดสอบจากล่างขึ้นบนและตัวขับ (Bottom-up Testing and Drivers)

ส่วนจำเพาะที่เสร็จบริบูรณ์แล้วควรจะถูกแทนที่สำหรับ Stub ของมัน แต่ขั้นแรกกระทำการทดสอบเบื้องต้นของส่วนจำเพาะใหม่เสมอ การหาตำแหน่งและแก้ไขข้อผิดพลาดให้ถูกต้องในหนึ่งส่วนจำเพาะจะง่ายกว่าการหาตำแหน่งและแก้ไขข้อผิดพลาดในระบบโปรแกรมที่เสร็จบริบูรณ์ เราสามารถทดสอบส่วนจำเพาะใหม่ โดยการเขียนโปรแกรมตัวขับสั้นๆ คล้ายกับในรูป 6.15

```
procedure ComputeSum (NumItems {input} : Integer;
                      var Sum {output} : Real);
{
   Compute the sum of a list of NumItems data items.
   Pre  :  NumItems is assigned a value.
   Post :  NumItems data items are read ; their sum is stored in Sum.
}
```

```
begin {ComputeSum Stub}
   WriteLn ('Procedure ComputeSum entered');
   Sum := 100.0
end; {ComputeSum stub}
```

รูป 6.23 Stub สำหรับกระบวนงาน ComputeSum

อย่าใช้เวลามากในการสร้างโปรแกรมตัวขับที่สวยงาม เพราะว่าเราจะตัดมันทิ้งทันทีที่ส่วนจำเพาะชุดใหม่ถูกทดสอบ โปรแกรมตัวขับควรมีเฉพาะการประกาศและข้อความสั่งกระทำการเท่าที่จำเป็น เพื่อทดสอบหนึ่งส่วนจำเพาะเท่านั้น นอกจากนี้ มันควรเริ่มต้นด้วยการอ่าน หรือการกำหนดค่าให้กับพารามิเตอร์อินพุต และให้กับพารามิเตอร์อินพุต/เอาต์พุตทั้งหมด จากนั้นเรียกส่วนจำเพาะที่ต้องการทดสอบ หลังจากเรียกส่วนจำเพาะแล้ว โปรแกรมตัวขับควรแสดงผลลัพธ์ของส่วนจำเพาะ

เมื่อเรามีความมั่นใจแล้วว่า ส่วนจำเพาะทำงานอย่างถูกต้อง แทนที่ส่วนจำเพาะนั้นกับ Stub ของมัน ในระบบโปรแกรม กระบวนการของการทดสอบส่วนจำเพาะแต่ละส่วนแยกต่างหากจากกัน ก่อนใส่มันในระบบโปรแกรมเรียกว่า **การทดสอบจากล่างขึ้นบน** (The process of separately testing individual modules befere inserting them into a program system is called bottom-up testing.)

ตลอดการรวมกันของการทดสอบจากบนลงล่างและจากล่างขึ้นบน เรามีความเชื่อมั่นว่าระบบโปรแกรมที่เสร็จสมบูรณ์จะไม่มีข้อผิดพลาด เมื่อรวมเข้าด้วยกันในตอนสุดท้ายด้วยเหตุนี้ ขั้นตอนการแก้จุดบกพร่องควรจะประมวลผลได้อย่างรวดเร็วและราบเรียบ

คำแนะนำการแก้จุดบกพร่องสำหรับโปรแกรมส่วนจำเพาะ (Debugging Tips for Modular Programs)

ในที่นี้คือคำแนะนำสำหรับการแก้จุดบกพร่องของระบบโปรแกรม

1. ขณะที่เขียนรหัส ใส่คอมเมนต์อย่างรอบคอบเพื่อทำเอกสารให้กับพารามิเตอร์ส่วนจำเพาะแต่ละตัว และไอเดนติไฟเออร์เฉพาะที่ รวมทั้งใช้คอมเมนต์อธิบายการดำเนินการส่วนจำเพาะ

2. การตามรอย (trace) ของการกระทำการโดยใส่ข้อความ WriteLn ซึ่งแสดงชื่อส่วนจำเพาะที่ตอนเริ่มต้นของตัวส่วนจำเพาะ (module body) ให้ทิ้งไว้ที่เดิม

3. ใส่ข้อความสั่งซึ่งแสดงผลค่าของพารามิเตอร์อินพุตและอินพุต/เอาต์พุตทั้งหมดที่เข้ามายังส่วนจำเพาะ ตรวจสอบว่าค่าเหล่านี้มี ความหมายถูกต้อง

4. ใส่ข้อความสั่งซึ่งแสดงผลค่าของเอาต์พุตส่วนจำเพาะทั้งหมด หลังจากกลับคืน (return) จากส่วนจำเพาะ คำนวณด้วยมือค่าเหล่านี้เพื่อทวนสอบว่ามันถูกต้อง สำหรับกระบวนงาน จงมั่นใจว่าพารามิเตอร์อินพุต/เอาต์พุต และเอาต์พุตทั้งหมดประกาศเป็นพารา-มิเตอร์ตัวแปร

5. จงมั่นใจว่า module stub กำหนดค่าให้กับเอาต์พุตของมันทุกตัว

เป็นความคิดที่ดีในการวางแผนแก้จุดบกพร่องขณะที่เราเขียนส่วนจำเพาะแต่ละชุด ไม่ใช่ใส่ขั้นตอนการแก้จุดบกพร่องภายหลังใส่ข้อความสั่งเอาต์พุตที่แนะนำจากข้อ 2 ถึงข้อ 4 ในรหัส Pascal

สำหรับส่วนจำเพาะ เมื่อพอใจว่าส่วนจำเพาะทำงานได้ตามต้องการแล้ว ลบข้อความสั่งแก้ไขจุดบกพร่องออก วิธีที่ง่ายที่สุดคือเปลี่ยนมันให้เป็นคอมเมนต์โดยใส่เครื่องหมายวงเล็บปีกกาปิดล้อม ถ้าเรามีปัญหาทีหลัง เราสามารถลบวงเล็บปีกกาออก นั่นคือการเปลี่ยนคอมเมนต์กลับมาเป็นข้อความสั่งกระทำการ

อีกวิธีหนึ่งคือการหมุน (turning) ข้อความสั่งแก้จุดบกพร่องเปิด (on) และปิด (off) โดยใช้ค่าคงตัวแบบบูลส่วนกลาง (global Boolean constant) สมมติว่าตั้งชื่อ Debug ซึ่งประกาศในโปรแกรมหลัก การประกาศ

```
const
   Debug = Ture ; {Turn debugging on}
```

ระหว่างวิ่งการแก้จุดบกพร่อง (debugging run) และประกาศ

```
const
   Debug = False ; {Turn debugging off}
```

ระหว่างวิ่งปกติ (normal runs)

ภายในตัวโปรแกรมหลักและกระบวนงานของมันฝังตัว (embed) ข้อความสั่งวินิจฉัย WriteLn แต่ละชุด ในข้อความสั่ง if ที่มี Debug เป็นเงื่อนไข ของมัน ถ้ากระบวนงาน ComputeSum เริ่มต้นด้วยข้อความสั่ง if ข้างล่างนี้ ข้อความสั่ง WriteLn จะกระทำการเฉพาะระหว่างวิ่งแก้จุดบกพร่องเท่านั้น (Debug เป็น True) ตามต้องการ

```
if Debug then
  begin
    WriteLn ('Procedure ComputeSum entered');
    WriteLn ('Input parameter NumItems has value ', NumItems : 1)
  end; {if}
```

6.8 ฟังก์ชันเรียกซ้ำ (Recursive Functions)

Pascal ยอมให้ฟังก์ชันหรือกระบวนงานเรียกตัวมันเอง (call itself) ส่วนจำเพาะซึ่งเรียกตัวมันเอง เรียกว่า **ส่วนจำเพาะเรียกซ้ำ** (recursive modules) ในหัวข้อนี้เราอธิบายฟังก์ชันเรียกซ้ำชุดหนึ่ง ซึ่งกลับคืนค่าจำนวนเต็มหนึ่งค่าแทนแฟกทอเรียลของอาร์กิวเมนต์ของมัน แฟกทอเรียล N หมายถึง ผลคูณของจำนวนเต็มบวกทั้งหมดที่มีค่าน้อยกว่าหรือ เท่ากับ N ในวิชาคณิตศาสตร์ ใช้สัญลักษณ์ N! ตัวอย่างเช่น 4! หมายถึง 4 x 3 x 2 x 1 หรือเท่ากับ 24

ส่วนจำเพาะเรียกซ้ำ หมายถึง กระบวนงานหรือฟังก์ชัน ซึ่งเรียกตัวมันเอง (Recursive module is a procedure or function that calls itself.)

แฟกทอเรียลของ N หมายถึง ผลคูณของจำนวนเต็มบวกทั้งหมดซึ่งมีค่าน้อยกว่าหรือเท่ากับ N (Factorial of N is the product of all positive integer $\leq$ N0)

รูป 6.25 แสดงให้เห็นฟังก์ชัน Factorial แบบไม่เรียกซ้ำ (nonrecursive) ซึ่งใช้การวนซ้ำเพื่อสะสมส่วนของผลคูณในตัวแปรเฉพาะที่ ProductSoFar ข้อความสั่ง For ทำซ้ำขั้นตอนการคูณ เมื่อ N มีค่ามากกว่า 1 ถ้า N เท่ากับ 0 หรือ 1 ข้อความสั่ง for จะไม่ถูกกระทำการ ดังนั้น ProceductSoFar จึงยังคงเก็บค่าเริ่มต้นเท่ากับ 1 หลังจากออกจากลูป ค่าสุดท้ายของ ProductSoFar ถูกกำหนดให้ Factorial นั่นคือการนิยามผลลัพธ์ของฟังก์ชัน

รูป 6.25 แสดงให้เห็นฟังก์ชัน Factorial เขียนใหม่เป็นฟังก์ชันแบบเรียกซ้ำ ซึ่งข้อความสั่ง if ปฏิบัติให้เกิดผลตามสูตรข้างล่างนี้ ซึ่งเป็นบทนิยามของ N!

N! = 1 for N = 0 or 1

N! = N x (N-1)! for N > 1

เมื่อ N มีค่ามากกว่า 1 แทนที่การกระทำการลูป ให้ทำซ้ำการคูณ เช่น ในรูป 6.26 ข้อความสั่ง

Factorial := N* Factorial (N-1)

```
function Factorial (N : Integer) : Integer;
{
   Compute N!
   Pre  :  N >= 0
   Post :  Returns the product
           1 * 2 * 3 * . . . * n for N > 1;
           returns 1 when N is 0 or 1
}
   var
      I,                          {loop-control variable}
      ProductSoFar : Integer;     {accumulated product}

begin {Factorial}
   ProductSoFar := 1;             {Initialize accumulated product.}

   {Perform the repeated multiplication for N > 1.}
   for I := 2 to N do
      ProductSoFar := ProductSoFar * I;

   {Define function result.}
   Factorial := ProductSoFar
end; {Factorial}
```

รูป 6.25 ฟังก์ชันแฟกทอเรียล

กระทำการ ซึ่งเป็นรูปแบบ Pascal ของสูตรที่สอง ส่วนนิพจน์ของข้อความสั่งนี้ ประกอบด้วย function designator, Factorial (N-1) ซึ่งเรียกฟังก์ชัน Factorial ที่มีอาร์กิว-

เมนต์ มีค่าน้อยลง 1 จากอาร์กิวเมนต์ปัจจุบัน function call นี้ เรียกว่าการ เรียกซ้ำ (recursive call) ถ้า อาร์กิวเมนต์ในการเรียก Factorial ครั้งแรก คือ 3 เกิดลูกโซ่ (chain) ต่อไปนี้ของการเรียกซ้ำ

Factorial (3) → 3 * Factorial (2) → 3 * (2 * Factorial (1))

ในตอนทุดท้ายของการเรียกเหล่านี้ N มีค่าเท่ากับ 1 ดังนั้น ข้อความสั่ง

Factorial := 1

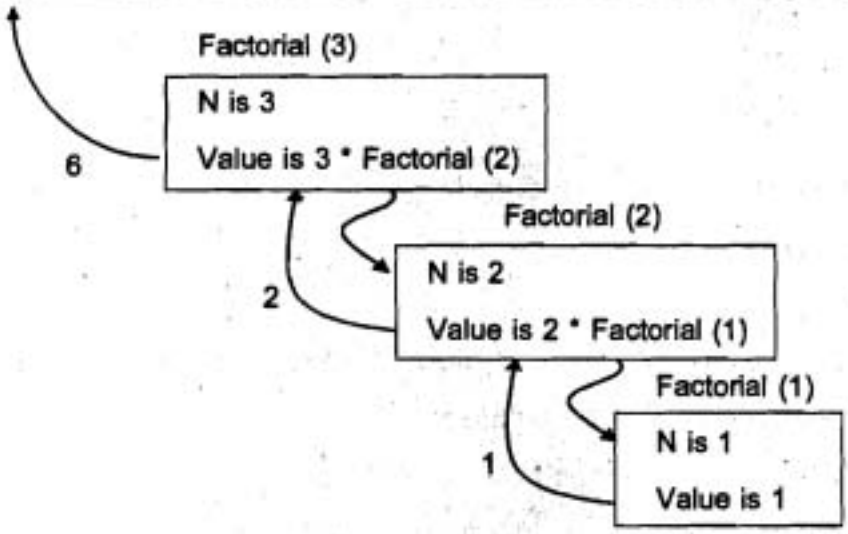
กระทำการเป็นการหยุดลูกโซ่ของการเรียกซ้ำ

เมื่อจบการเรียกฟังก์ชันครั้งสุดท้าย Pascal ต้องกลับคืนหนึ่งค่าจากการเรียกซ้ำแต่ละครั้ง เริ่มต้นด้วยค่าสุดท้าย การเรียกครั้งสุดท้ายคือ Factorial (1) และมันกลับคืนคา 1

```
function Factorial (N : Integer) : Integer;
{
   Compate N!
   Pre  :  N >= 0
   Post :  Returns the product
              1 * 2 * 3 * * . . . * N for N > 1;
           returns 1 when N is 0 or 1
}
begin {Factorial}
   if N <= 1 then
      Factorial := 1
   else
      Factoral := N * Factorial (N-1)
end;  {Factorial}
```

รูป 6.26 ฟังก์ชันการเรียกซ้ำ Factorial

การหาค่ากลับคืนโดยการเรียกแต่ละครั้ง เมื่อ N มีค่ามากกว่า 1 คูณ N ด้วยค่ากลับคืนจาก Factorial (N-1) เพราะฉะนั้นค่ากลับคืนจาก Factorial (2) เท่ากับ 2* ค่ากลับคืนจาก Factorial (1) หรือ 2; ค่ากลับคืนจาก Factorial (3) เท่ากับ 3 * ค่ากลับคืนจาก Factorial (2) หรือ 6 (ดูรูป 6.27)



รูป 6.27 การประเมินผล Factorial (3)

แบบฝึกหัด 6.8 Self - Check

1. จงแสดงลูกโซ่ของฟังก์ชัน Mystery เมื่อ M เท่ากับ 4 และ N เท่ากับ 3 จากนั้นให้บอกว่าฟังก์ชัน Mystery คำนวณค่าอะไร

```
function Mystery (M, N : Integer) : Integer
  begin {Mystery}
    if N = 1 then
    Mystery := M
  else
    Mystery := M * Mystery (M, N-1)
end; {Mystery}
```

เขียนโปรแกรม

1. จงเขียนฟังก์ชันแบบเรียกซ้ำ กำหนดค่าอินพุตของ N แล้วให้คำนวณ

   N + N-1 + . . . + 2 + 1

2. จงเขียนฟังก์ชัน C(N,R) ซึ่งกลับคืนจำนวนวิธีที่แตกต่างกันของข้อมูล R ตัว เลือกมาจากกลุ่มข้อมูล N ตัวสูตรคณิตศาสตร์สำหรับ C(N,R) คือ

$$C(N,R) = \frac{N!}{R!\,(N-R)!}$$

ทดสอบ C(N, R) โดยใช้เวอร์ชันเรียกซ้ำ (recursive) และแบบไม่เรียกซ้ำ (nonrecursive versions) ของฟังก์ชัน Factorial

6.9 ข้อผิดพลาดร่วมของการเขียนโปรแกรม (Common Programming Errors)

การใช้พารามิเตอร์ให้ถูกต้องเป็นเรื่องยากสำหรับโปรแกรมเมอร์มือใหม่ที่จะควบคุม แต่มันเป็นทักษะที่จำเป็น เมื่อเราใช้ส่วนจำเพาะที่มีรายการพารามิเตอร์ มีโอกาสมากที่จะเกิดข้อผิดพลาด หลุมพรางที่เห็นชัดเจนเกิดขึ้น คือ รายการพารามิเตอร์จริงไม่เท่ากับจำนวนพารามิเตอร์ที่เป็นรายการพารามิเตอร์รูปนัย ข้อผิดพลาดวากยสัมพันธ์ " , " expected และ ") " expected แสดงถึงปัญหานี้

พารามิเตอร์จริงแต่ละตัวต้องเข้ากันได้ การกำหนดค่า (assignment compatible) กับพารามิเตอร์รูปนัย (สำหรับพารามิเตอร์ค่า) ที่สมนัยกัน หรือแบบชนิดข้อมูลเหมือนกัน (สำหรับพารามิเตอร์ตัวแปร) พารามิเตอร์จริงซึ่งสมนัยกับพารามิเตอร์รูปนัยตัวแปรต้องเป็นตัวแปร การฝ่าฝืนกฎข้อแรกผลลัพธ์เป็นข้อผิดพลาดวากยสัมพันธ์ mismatch การฝ่าฝืนกฎข้อที่สอง ผลลัพธ์เป็นข้อผิดพลาดวากยสัมพันธ์ variavle indentifier expected

การกลับคืน (return) ผลลัพธ์ของกระบวนงานไปยังส่วนจำเพาะเรียก กำหนดโดยค่าของพารามิเตอร์ตัวแปร ค่าใดๆ ซึ่งกำหนดให้กับพารามิเตอร์ค่าจะเก็บเฉพาะที่ (stored locally) ในกระบวนงานและไม่มีการกลับคืน ถ้ากระบวนงานดูเหมือนคำนวณอย่างถูกต้อง แต่ไม่กลับคืน ผลลัพธ์ที่คาดไว้อาจจะเป็นเพราะว่าเราลืมประกาศพารามิเตอร์เอาต์พุต หรือพารามิเตอร์อินพุต/เอาต์พุต ให้เป็นพารามิเตอร์ตัวแปร

จำไว้ว่าการกลับคืนผลลัพธ์ของฟังก์ชันทั้งหมดกระทำโดยการกำหนดหนึ่งค่าให้กับชื่อฟังก์ชัน การกลับคืนผลลัพธ์ฟังก์ชันตัวที่สองผ่านทางพารามิเตอร์ตัวแปร (เกิด func-

tion side effect) และเป็นการฝึกปฏิบัติเขียนโปรแกรมที่ไม่ดี (bad) การดำเนินการตัวแปรส่วนกลางในกระบวนงานหรือฟังก์ชันโดยตรงถือว่าเป็นการฝึกปฏิบัติเขียนโปรแกรมที่แย่เช่นกัน ตัวแปรส่วนกลางทั้งหมดควรส่งไปยังส่วนจำเพาะผ่านรายการพารามิเตอร์ของมัน

กฎสโคปของ Pascal กำหนดว่า ไอเดนติไฟเออร์สามารถอ้างถึงได้ที่ใด ถ้าไอเดนติไฟเออร์ถูกอ้างนอกสโคปของมัน ผลลัพธ์เกิดข้อผิดพลาดวากยสัมพันธ์ identifier not declared

ข้อสรุปตัวสร้างใหม่ของ Pascal

Construct	Effect
Function Declaration	
function Sigh (X : Real) : Char;	Returns a character
begin {Sigh}	value that indicates
if X > = 0.0 then	the sign (' + ' or ' - ')
Sign := ' + '	of its type Real Argument X.
else	
Sign := ' - '	
end, {Sign}	
Procedure Declaration	
procedure DoIt (X : Real,	If Op is ' + ' returns X + X
Op : Char;	through Y;
var Y : Char;	if Op is " * ' returns
var XSign : Char);	X * X through Y.
begin {DoIt}	Calls Sign to assign
case Op of	a character value that
' + ' : Y := X+X;	Indicate the sign (' + '
' * ' : Y := X * X	or ' - ') of X to Sign.
end; {case}	
XSign = Sign(X)	
.end; {DoIt}	

Construct	Effect
Procedure Call Statement DoIt(-5.0, ' * ', Y, Mysign)	Call DoIt. Passes - 5.0 into X, passes ' * ' into Op, returns 25.0 to Y, returns ' - ' to Mysign.

แบบฝึกหัด Quick - Check

1. พารามิเตอร์ชนิดใดปรากฏในการเรียกกระบวนงาน และพารามิเตอร์ชนิดใดปรากฏในการประกาศกระบวนงาน

2. ค่าคงตัวและนิพจน์สามารถสมนัยกับพารามิเตอร์รูปนัย เป็นพารามิเตอร์ชนิดใด

3. พารามิเตอร์รูปนัยซึ่งเป็นพารามิเตอร์ตัวแปร ต้องมีพารามิเตอร์จริงเป็นชนิดใด

4. พารามิเตอร์รูปนัยซึ่งเป็นพารามิเตอร์ตัวแปร ต้องมีพารามิเตอร์จริงเป็นแบบชนิดข้อมูลประเภทใด

5. แบบชนิดข้อมูลของพารามิเตอร์ค่าซึ่งสมนัยกันต้องเป็น.....

.....

6. ตัวขับโปรแกรม (driver) หรือตัว stub ซึ่งใช้ทดสอบส่วนจำเพาะชุดใหม่

7. ตัวขับโปรแกรมหรือตัว stub ซึ่งใช้ทดสอบการไหลของโปรแกรมหลัก (mian program flow)

8. จะเกิดอะไรขึ้น เมื่อฟังก์ชันกำหนดค่าให้กับพารามิเตอร์ตัวแปร หรือเมื่อกระบวนงานเปลี่ยนแปลงค่าของตัวแปรส่วนกลาง

9. จงบอกค่าของตัวแปร X และ Y ในโปรแกรมหลัก หลังจากโปรแกรมข้างล่างนี้ถูกกระทำการ

```
program Nonsense;
  var X, Y : Real;
  procedure Silly (X : Real);
```

```
        var Y : Real;
    begin
        Y := 25.0;
        X := Y
    end; {Silly}
begin {Nonsense}
    Silly (X)
end. {Nonsense}
```

10. จงตอบคำถามข้อ 9 ครั้งนี้สมมติว่า พารามิเตอร์ X ของ Silly เป็นพารามิเตอร์ตัวแปร

11. พารามิเตอร์ของฟังก์ชันควรเป็นพารามิเตอร์ชนิดใด

12. ส่วนของโปรแกรมซึ่งไอเดนติไฟเออร์สามารถถูกอ้างถึงได้ เรียกว่า..... ของไอเดนติไฟเออร์

13. จริงหรือเท็จ : ภายในตัวกระบวนงานนั้นกระบวนงานอื่นอีกหนึ่งชุด ซึ่งประกาศในโปรแกรมสามารถถูกเรียกได้

14. ฟังก์ชันกลับคืนผลลัพธ์ของมันอย่างไร

คำถามทบทวน (Review Questions)

1. จงเขียนหัวเรื่องของกระบวนงานสำหรับกระบวนงานชื่อ Script ซึ่งรับพารามิเตอร์สามตัวที่ส่งเข้ามา พารามิเตอร์ตัวแรกคือจำนวนตัวว่าง (spaces) ที่จะให้พิมพ์ตอนเริ่มต้นบรรทัด พารามิเตอร์ตัวที่สองคือตัวอักขระ (character) ที่จะให้พิมพ์หลังตัวว่าง และพารามิเตอร์ตัวที่สามคือจำนวนครั้งของการพิมพ์พารามิเตอร์ตัวที่สองบนบรรทัดเดียวกัน

2. จงเขียนฟังก์ชันชื่อ LetterGrade ซึ่งมีพารามิเตอร์หนึ่งตัวชื่อ Grade และฟังก์ชันนี้กลับคืนเกรดเป็นตัวอักษร โดยมาตรฐานระดับ 90-100 เป็น A, 80-89 เป็น B, 60-79 เป็น C, 50-59 เป็น D และต่ำกว่า 59 เป็น F

3. จงอธิบายว่าทำไมเราจึงเลือกทำพารามิเตอร์รูปนัยเป็นพารามิเตอร์ค่า ไม่ทำเป็นพารามิเตอร์ตัวแปร

4. จงเขียนกระบวนงานรับค่าจำนวนจริงสองค่าเป็นอินพุต และให้เอาต์พุตเป็นผลบวก และผลต่างของสองค่านั้น

5. จงอธิบายสถานการณ์ซึ่งกระบวนงานจะให้ผลลัพธ์แตกต่าง ถ้าพารามิเตอร์ค่าในกระบวนงานถูกเปลี่ยนให้เป็นพารามิเตอร์ตัวแปร

6. จงให้เหตุผลสองข้อสำหรับการ implement ส่วนจำเพาะเป็นกระบวนงาน ไม่ใช่ฟังก์ชัน

7. ในแผนภูมิ (chart) ข้างล่างนี้ จงเขียน Yes สำหรับกระบวนงานแต่ละชุดทางขวามือซึ่งถูกอ้างถึงได้ (called) โดยกระบวนงานทางซ้ายมือ และเขียน No สำหรับกระบวนงานแต่ละชุดซึ่งอ้างถึงไม่ได้ (inaccessible)

```
program ProcScope;
  procedure A;
    procedure B;
      procedure C;
      . . .
      end; {C}
      procedure D;
      . . .
      end; {D}
    . . .
    end; {B}
  . . .
  end; {A}
. . .
end. {ProcScope}
```

Calling Procedure	Callable Procedure A	B	C	D
A				
B				
C				
D				

เขียนโปรแกรม

1. คณะกรรมการอำนวยการของวิทยาลัยขนาดเล็กแห่งหนึ่ง พิจารณาจะขึ้นเงินเดือน 5.5% ให้กับพนักงานทุกคน แต่ขั้นแรกเขาต้องการทราบว่าจะต้องใช้เงินเพื่อเพิ่มเงินเดือนครั้งนี้จำนวนมากเท่าใด จงเขียนโปรแกรมพิมพ์เงินเพิ่มของพนักงานแต่ละคน และจำนวนเงินเพิ่มทั้งหมดที่จะต้องใช้ จากนั้นพิมพ์เงินเดือนรวมทั้งหมดของพนักงาน ก่อนและหลังการคิดเงินเดือน โดยใช้ข้อมูลเงินเดือนข้างล่างนี้ทดสอบโปรแกรม

$32,500	$24,029.50	$36,000	$43,250
$35,500	$22,800	$30,000.00	$28,900
$43,780	$24,029.50	$44,120.50	$24,100

ให้นักศึกษาตัดสินใจว่าส่วนใดควรเขียนเป็นกระบวนงาน และส่วนใดควรเขียนเป็นฟังก์ชัน

2. จงดัดแปรโปรแกรมแบบฝึกหัดข้อ 1 โดยสมมติว่าพนักงานที่มีเงินเดือนน้อยกว่า $30,000 ขึ้น 7% พนักงานที่มีเงินเดือนมากกว่า $40,000 ขึ้น 4% และพนักงานอื่นๆ ทั้งหมดขึ้น 5.5% จงเขียนฟังก์ชันชุดใหม่เพื่อคำนวณเปอร์เซนต์ที่เพิ่มขึ้น จากนั้นพิมพ์เปอร์เซนต์ที่เพิ่มและจำนวนเงินที่เพิ่มของพนักงานแต่ละคน