บทที่ 3
ฟังก์ชันและกระบวนงาน
(Functions and Procedures)

3.1 การสร้างโปรแกรมจากสารสนเทศที่มีอยู่
กรณีศึกษา : การหาพื้นที่และเส้นรอบวงของวงกลม
3.2 ฟังก์ชัน
3.3 การออกแบบจากบนลงล่างและผังโครงสร้าง
กรณีศึกษา : วาดแผนภาพอย่างง่าย
3.4 กระบวนงาน
3.5 กระบวนงานคือบล็อกสร้างโปรแกรม
3.6 ข้อผิดพลาดร่วมของการเขียนโปรแกรม

โปรแกรมเมอร์ซึ่งใช้วิธีพัฒนาซอฟต์แวร์เพื่อแก้ปัญหาไม่ค่อยจะจัดการโปรแกรมใหม่แต่ละชุดเป็นเหตุการณ์หนึ่งเดียว (a unique event) สารสนเทศซึ่งอยู่ในประโยคปัญหาและสารสนเทศซึ่งรวบรวมระหว่างขั้นวิเคราะห์และออกแบบ ช่วยให้โปรแกรมเมอร์วางแผนและเสร็จสิ้นโปรแกรม

โปรแกรมเมอร์ใช้เซกเมนต์ (segments) ของผลเฉลยโปรแกรมก่อนหน้านั้น เป็นบล็อกการสร้าง (building blocks) เพื่อสร้างโปรแกรมใหม่

ในหัวข้อ 3.2 เราแสดงให้เห็นว่าจะเจาะ (tap) สารสนเทศที่มีอยู่ และลงรหัสในรูปแบบของฟังก์ชันซึ่งให้นิยามแล้ว เพื่อเขียนโปรแกรมอย่างไร นอกเหนือจากการใช้สารสนเทศที่มีอยู่ โปรแกรมเมอร์สามารถใช้เทคนิคการออกแบบจากบนลงล่าง เพื่อให้การพัฒนาอัลกอริทึมและโครงสร้างของโปรแกรม ผลลัพธ์ทำได้ง่ายขึ้น การประยุกต์ใช้การออกแบบจากบนลงล่างโปรแกรมเมอร์เริ่มต้นด้วยประโยคซึ่งกว้างที่สุดของผลเฉลยการแก้ปัญหา และทำงานลงไปยังรายละเอียดของปัญหาย่อยมากขึ้น

ในหัวข้อ 3.3 - 3.5 เราแสดงให้เห็นการออกแบบจากบนลงล่าง และเน้นบทบาทของการเขียนโปรแกรมส่วนจำเพาะโดยใช้กระบวนงาน

3.1 การสร้างโปรแกรมจากสารสนเทศที่มีอยู่ (Building Programs from Existing Information)

โปรแกรมเมอร์มีน้อยมากที่เริ่มต้นจากสถานะว่าง (หรือจอว่าง) เมื่อเขาพัฒนาโปรแกรม บ่อยครั้งหรือเกือบทั้งหมด การแก้ปัญหาพัฒนาจากสารสนเทศซึ่งมีอยู่เรียบร้อยแล้ว หรือจากผลเฉลยของอีกปัญหาหนึ่ง เช่นที่เราจะแสดงให้เห็นจากโปรแกรมเปลี่ยนหน่วยวัด

การทำตามวิธีการพัฒนาซอฟต์แวร์อย่างระมัดระวัง ซึ่งสร้างการทำเอกสารระบบที่สำคัญ ก่อนเริ่มต้นลงรหัสโปรแกรม การทำเอกสารเช่นนี้ ประกอบด้วย คำอธิบายการระบุความต้องการข้อมูลของปัญหา (พัฒนาระหว่างขั้นวิเคราะห์) และอัลกอริทึมแก้ปัญหาของมัน (พัฒนาระหว่างออกแบบ) บทสรุปความสนใจของเรา ตลอดจนการประมวลผล

เราสามารถใช้เอกสารนี้เป็นจุดเริ่มต้นในการลงรหัสโปรแกรม ตัวอย่างเช่น เริ่มต้นทำซ้ำส่วนที่เป็นการระบุความต้องการข้อมูลปัญหา ในส่วนการประกาศโปรแกรม (รูป 3.1) จากนั้นตรวจแก้ (edit) บรรทัดเหล่านี้ เพื่อให้ตรงกับวากยสัมพันธ์ของ Pascal สำหรับการประกาศค่าคงตัวและตัวแปร ทำส่วนการประกาศของโปรแกรมให้เสร็จสิ้น (รูป 3.2) วิธีนี้จะเป็นการช่วยถ้าการทำเอกสารสร้างไว้แล้วด้วยตัวประมวลผลคำอยู่ในไฟล์ซึ่งเราสามารถตรวจแก้ได้

```
Program Metric;
{Converts square meters to square yards}
relevent formula
   1  square meter = 1.196 square yards
problem input
   SqMeters {the fabric size in square meters}
problem output
   SqYards {the fabric size in square yards}
```

```
begin
end.
```

รูป 3.1 ส่วนการประกาศก่อนทำการตรวจแก้

```
program Metric;
{Converts square meters to square yards}

const
   MetersToYards = 1.196; {conversion constant}

var
   SqMeters,          {input-fabric size in meters}
   SqYards : Real;    {output-fabric size in yards}

begin
   {1. Read the fabric size in square meters.}
   {2. Convert the fabric size to square yards.}
      {2.1 Size in sq. yards is 1.196 * size in sq. meters.}
   {3. Display the fabric size in square yards.}
end.
```

รูป 3.2 การใช้อัลกอริทึมแบ่งละเอียดเป็นโครงร่าง (framework) ของโปรแกรม

การพัฒนาตัวโปรแกรม (program body) ขั้นแรกใช้อัลกอริทึมเริ่มต้น และการแบ่งละเอียดของมัน เป็นคอมเมนต์ของโปรแกรม คอมเมนต์จะอธิบายทุกขั้นตอนของอัลกอริทึม และถือว่าเป็นการทำเอกสารโปรแกรม ซึ่งเป็นแนวทางการลงรหัส Pascal รูป 3.2 แสดงให้เห็นว่า ณ จุดนี้โปรแกรมจะดูเป็นอย่างไร

หลังจากคอมเมนต์อยู่ในตัวโปรแกรมแล้ว เราจึงเริ่มต้นเขียนข้อความสั่ง Pascal ใส่รหัส Pascal โดยตรง สำหรับขั้นที่ไม่แบ่งละเอียดได้ขั้นตอนนั้น ส่วนขั้นที่มีการแบ่งละเอียด อาจมีการตรวจแก้การแบ่งละเอียดเพื่อเปลี่ยนจากภาษาอังกฤษให้เป็น Pascal หรือแทนที่ด้วยรหัส Pascal (ดูรูป 2.1)

กรณีศึกษา การหาพื้นที่และเส้นรอบวงของวงกลม

(1) ปัญหา (Problem)

อ่านรัศมีของวงกลม จากนั้นคำนวณและพิมพ์พื้นที่และเส้นรอบวงของวงกลม

(2) วิเคราะห์ (Analysis)

อินพุตของปัญหาคืออรัศมีของวงกลม เอาต์พุตที่ต้องการมีสองอย่าง คือ พื้นที่และเส้นรอบวงของวงกลม ตัวแปรเหล่านี้ควรมีชนิดเป็น Real เพราะว่าอินพุตและเอาต์พุตอาจมีภาคเศษส่วน (fractional part) ความสัมพันธ์เรขาคณิตระหว่างรัศมีของวงกลมและเส้นรอบวงของมันจะเขียนถัดไปพร้อมกับความต้องการข้อมูล

Data Requirements

Problem Constant

```
MyPi = 3.14159;
```

Problem Input

```
Radius : Real        {radius of a circle}
```

Problem output

```
Area : Real          {area of a circle}
Circum : Real        {circumference of a circle}
```

Relevant Formulas

area of a circle = π x radius^2

circumference of a circle = 2π x radius

(3) ออกแบบ (Design)

หลังจากการระบุอินพุตและเอาต์พุตของปัญหาแล้ว เขียนรายการขั้นตอนต่าง ๆ ที่จำเป็นเพื่อแก้ปัญหา

อัลกอริทึมเริ่มต้น

1. อ่านรัศมีของวงกลม

2. หาพื้นที่

3. หาเส้นรอบวง

4. พิมพ์พื้นที่และเส้นรอบวง

การแบ่งละเอียดอัลกอริทึม

ต่อไป แบ่งละเอียดขั้นตอนซึ่งยังขาดการแก้ปัญหาที่ชัดเจน (ขั้นที่ 2 และขั้นที่ 3)

ขั้นที่ 2 การแบ่งละเอียด

2.1 กำหนด MyPi * Radius * Radius ให้กับ Area

ขั้นที่ 3 การแบ่งละเอียด

3.1 กำหนด 2 * MyPi * Radius ให้กับ circums

(4) การปฏิบัติให้เกิดผล (Implementations)

รูป 3.3 แสดงให้เห็นโปรแกรม Pascal ส่วนที่เป็นตัวโปรแกรม ประกอบด้วยอัลกอริทึมเริ่มต้น และการแบ่งละเอียดของมัน

การเขียนโปรแกรมสุดท้าย เปลี่ยนการแบ่งละเอียด (ขั้น 2.1 และขั้น 3.1) ให้เป็น Pascal เขียนรหัส Pascal สำหรับขั้นซึ่งไม่แบ่งละเอียด (ขั้นที่ 1 และขั้นที่ 4) และลบเลขขั้นตอนออกจากคอมเมนต์ รูป 3.4 แสดงโปรแกรมสุดท้าย

```
Program Circle;
{Finds and prints the area and circumference of a circle}

const
    MyPi = 3.14159;

var
    Radius,          {input - radius of a circle}
    Area,            {output - area of a circle}
    Circum : Real;   {output - cirucmference of a circle}
```

```
begin
    {1. Read the circle radius.}
    {2. Find the area.}
        {2.1 Assign MyPi * Redius * Radius to Area.}
    {3.  Find the circumference.}
        {3.1 Assign 2 * MyPy * Radius to Circum.}
    {4.  Print the are and circumference.}
end.
```

รูป 3.3 Output ของโปรแกรม Circle

Edit Window

```
program Circle;
{Finds and prints the area and circumference of a circle}

    const
        MyPi = 3.14159;

    var
        Radius,             {input - radius of a circle}
        Area,               {output - area of a circle}
        Circum : Real;      {output - circumference of a circle}

begin
    {Read the circle radius.}
    Write (' Enter radius > ');
    ReadLn (Radius);
```

```
{Find the area.}
Area := MyPi * Radius * Radius;

{Find the circumference.}
Circum := 2.0 * MyPi * Radius;

{Print the area and circumference.}
    WriteLn (' The area is ' ; Area : 4 : 2);
    WriteLn (' The circumference is ' ; Circum : 4 : 2)
end.
```

Output Window

```
Enter radius > 5.0
The area is 78.54
The circumfeence is 31.42
```

รูป 3.4 การหาพื้นที่และเส้นรอบวงของวงกลม

(5) การทดสอบ (Testing)

เอาต์พุตตัวอย่างในรูป 3.4 ถือว่าเป็นการทดสอบที่ดีของผลเฉลย เพราะว่าค่อนข้างง่ายในการคำนวณด้วยมือสำหรับพื้นที่และเส้นรอบวงของวงกลมที่มีรัศมีเท่ากับ 5 กำลังสองของรัศมีเท่ากับ 25 และ π ค่าโดยประมาณเท่ากับ 3 ดังนั้นค่าของพื้นที่ดูแล้วถูกต้อง เส้นรอบวงของวงกลมควรจะเป็น 10 เท่าของ π ซึ่งเป็นเลขที่คำนวณด้วยมือได้โดยง่าย

แบบฝึกหัด 3.1 Self - Check

1. จงอธิบายอินพุตและเอาต์พุตของปัญหา แล้วเขียนอัลกอริทึมสำหรับโปรแกรมคำนวณเงินเดือนรวม (gross salary) ของพนักงานเมื่อกำหนดจำนวนชั่วโมงทำงาน (hours worked) และอัตราค่าจ้างต่อชั่วโมง (hourly rate)

2. จงเขียนเวอร์ชันแรกของโปรแกรมจากผลเฉลยในแบบฝึกหัดข้อ 1 แสดงส่วนการประกาศของโปรแกรม และคอมเมนต์โปรแกรมซึ่งสมนัยกับอัลกอริทึมและการแบ่งละเอียดของมัน

3. จะมีการเปลี่ยนแปลงอะไรบ้าง เมื่อเราขยายอัลกอริทึมเงินเดือน (payrol algorithm) ในแบบฝึกหัดข้อ 1 ซึ่งรวมชั่วโมงทำงานล่วงเวลา (overtime hours) ซึ่งจ่ายในอัตรา 1.5 เท่าของอัตราค่าจ้างต่อชั่วโมงปกติ เมื่อคำนวณเงินเดือนรวมของพนักงาน สมมติว่าจำนวนชั่วโมงทำงานล่วงเวลา ใส่ (enter) แยกจากกัน

เขียนโปรแกรม

1. ใส่การแบ่งละเอียดให้กับร่างของโปรแกรมข้างล่างนี้ แล้วเขียนโปรแกรม Pascal สุดท้าย :

```
program SumAndAverage;
{Finds and prints the sum and average of two numbers}

  var
    One, Two,          {input - numbers to process}
    Sum,               {Output - Sum of One and Two}
    Average : Real;    {Output - average of One and Two}
begin
  {1. Real two numbers.}
  {2. Compute sum of numbers.}
  {3. Compute average of numbers.}
  {4. Print sum and average.}
end.
```

2. จงเขียนโปรแกรม Pascal ที่เสร็จสมบูรณ์ของแบบฝึกหัดข้อ 1

3. จงเขียนโปรแกรม Pascal ที่เสร็จสมบูรณ์ของ revised payroll algorithm ในแบบฝึกหัด 3.1

3.2 ฟังก์ชัน (Functions)

ฟังก์ชันและการนำกลับมาใช้ใหม่ (Functions and Reusability)

เป้าหมายหลักของการเขียนโปรแกรมเชิงโครงสร้าง คือ เขียนรหัสที่ไม่มีข้อผิดพลาด (error - free code) วิธีหนึ่งในการทำสิ่งนี้ เรียกว่า การนำกลับมาใช้ใหม่ หมายถึง นำกลับมาใช้ (revese) เมื่อใดก็ตามที่เป็นไปได้ซึ่งมีรหัสที่เขียนและทดสอบเรียบร้อยแล้ว Pascal สนับสนุนการนำกลับมาใช้โดยมีฟังก์ชันซึ่งให้นิยามแล้วจำนวนมากสำหรับการคำนวณเชิงคณิตศาสตร์

ฟังก์ชัน หมายถึง ส่วนจำเพาะของรหัสที่แยกจากกัน ซึ่งคำนวณแล้วให้ผลลัพธ์หนึ่งค่า (A function is a separate module of code that compute a single result.)

ตัวอย่างเช่น ฟังก์ชัน Pascal ชื่อ Sqrt ซึ่งคำนวณรากที่สอง (square root computation)

ส่วนนิพจน์ของข้อความสั่งกำหนดค่า ทำให้รหัสของฟังก์ชัน Sqrt ทำงานโดยส่งอาร์กิวเมนต์ X ให้กับฟังก์ชัน

Y := Sqrt (X)

ในที่นี้ Sqrt เป็นชื่อฟังก์ชัน, X เป็นอาร์กิวเมนต์ของฟังก์ชัน,

Sqrt (X) เป็น function designator ซึ่งเป็นส่วนของนิพจน์

ฟังก์ชันอาร์กิวเมนต์ หมายถึง ค่าซึ่งส่งไปยังฟังก์ชัน (A function argument is a value pascal into a function)

function designator หมายถึง ส่วนของนิพจน์ ซึ่งกระตุ้นให้ฟังก์ชันทำงานและส่งอาร์กิวเมนต์ไปยังฟังก์ชัน (A function designator is the part of an expression that activates a function and passes arguments to the function.)

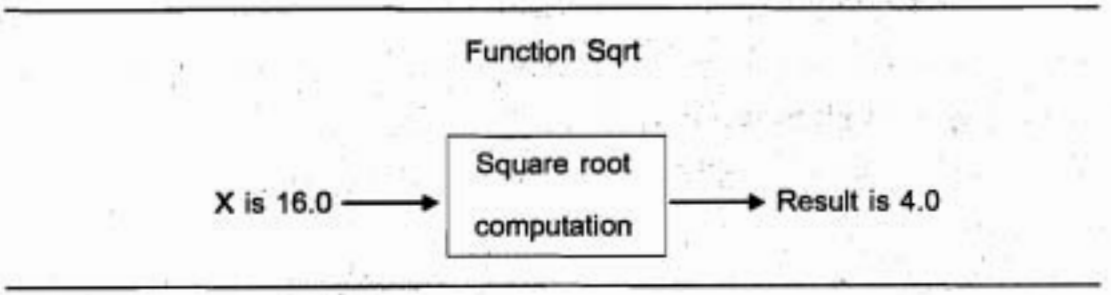
หลังจากฟังก์ชันกระทำการ ค่าของฟังก์ชันจะไปแทนที่ function designator ในนิพจน์ ถ้า X มีค่าเท่ากับ 16.0

ข้อความสั่งกำหนดค่าจะถูกประเมินผลดังนี้

1. X เท่ากับ 16.0 ดังนั้นฟังก์ชัน Sqrt คำนวณรากที่สองของ 16.0 ผลลัพธ์คือ 4.0
2. ผลลัพธ์ของฟังก์ชันคือ 4.0 ถูกกำหนดให้ Y

ฟังก์ชันเปรียบเสมือนกล่องดำ (black box) ซึ่งรับค่าอินพุตหนึ่งค่าหรือมากกว่าหนึ่งค่าที่ส่งมา และให้ค่าเอาต์พุตหนึ่งค่าส่งกลับอัตโนมัติ รูป 3.6 แสดงให้เห็นกระบวนการ

นี้ สำหรับเรียกฟังก์ชัน Sqrt ค่าของ X (คือ 16.0) เป็นอินพุตของฟังก์ชัน และผลลัพธ์ของฟังก์ชันหรือเอาต์พุตคือรากที่สองของ 16.0 (ผลลัพธ์คือ 4.0)



รูป 3.6 ฟังก์ชัน Sqrt เป็นกล่องดำ

อีกตัวอย่างหนึ่ง ถ้า W เท่ากับ 9.0 ข้อความสั่งกำหนดค่า

Z := 5.7 + Sqrt (W)

ประเมินผลดังนี้

1. W มีค่าเท่ากับ 9.0 ดังนั้นฟังก์ชัน Sqrt คำนวณรากที่สองของ 9.0 ผลลัพธ์คือ 3.0

2. ค่า 5.7 บวกกับ 3.0

3. ผลบวกคือ 8.7 เก็บใน Z

ตัวอย่าง 3.1

โปรแกรมในรูป 3.7 แสดงผลรากที่สองของเลขสองจำนวน ซึ่งเป็นข้อมูลอินพุต (First และ Second) และรากที่สองของผลบวกของเลขสองตัวนี้ ในการทำสิ่งนี้ ต้องเรียกฟังก์ชัน Sqrt สามครั้ง

```
Answer := Sqrt (First);
Answer := Sqrt (Second);
Answer := Sqrt (First + Second);
```

สำหรับการเรียกสองครั้งแรก อาร์กิวเมนต์คือตัวแปร (First และ Second) การเรียกครั้งที่สามแสดงให้เห็นว่า อาร์กิวเมนต์ของฟังก์ชันเป็นนิพจน์ (First + Second) การเรียกทั้งสามครั้งผลลัพธ์กลับคืน (return) โดยฟังก์ชัน Sqrt ถูกกำหนดให้กับตัวแปร Answer

ถ้าเราดูโปรแกรมในรูป 3.7 อย่างใกล้ชิด จะเห็นว่าแต่ละข้อความสั่งประกอบด้วยการเรียกโปรซีเดอร์ Pascal (Write, WriteLn, ReadLn) หรือฟังก์ชัน Sqrt - เราใช้กระบวนงานให้นิยามแล้วของ Pascal และฟังก์ชันเป็นบล็อกการสร้างเพื่อสร้างโปรแกรมใหม่

Edti Window

```
Program SquareRoots;
{Performs three square root computations}

  var
    First, Second,          {input - two data vlues}
    Answer : Real;          {output - a square root value}

  begin
    {Get first number and display its square root.}
    Write ( 'Enter the first number > ');
    ReadLn (First);
    Answer := Sqrt (First);
  WriteLn (' The square root of the first number is ' ,  Answer : 4 : 2);

  {Get second number and display its square root.}
  Write (' Enter the second number > ');
  ReadLn (Second);
  Answer := Sqrt (Second);
  WriteLn (' The squrare root of the second number is ' , Answer : 4 : 2);

  {Display the square root of the sum of both numbers.}
  Answer := Sqrt (First + Second);
  WritLn (' The square root of the sum of both numbers is ' , Answer : 4 : 2)
end.
```

Output Window

Enter the first number > 9.0

The square root of the first number is 3.00

Enter the second number > 16.0

The square root of the second number is 4.00

The square root of the sum of both number is 5.00

รูป 3.7 โปรแกรม Square Roots

ฟังก์ชัน Pascal มาตรฐาน (Standard Pascal Functions)

ตาราง 3.1 แสดงรายการชื่อและคำอธิบายของฟังก์ชันให้นิยามแล้ว ของ Pascal บางชุด ยกเว้น Abs, Round, Sqr และ Tranc เท่านั้น ฟังก์ชันแต่ละชุดในตาราง 3.1 กลับคืน (return) ค่าชนิด Real โดยไม่สนใจชนิดอาร์กิวเมนต์ของมัน (Real หรือ Integer) ชนิดของผลลัพธ์กลับคืนของฟังก์ชัน Abs หรือ Sqr จะเหมือนกับชนิดของอาร์กิวเมนต์ของมัน

ตาราง 3.1 ฟังก์ชันมาตรฐาน

Function	Purpose	Argument	Result
Abs (X)	Returns the absolute value of X	Real or Integer	Same as argument
ArcTan (X)	Returns the angle y in radians satisfy X = tan (y), where $-\frac{\pi}{2} < y < \frac{\pi}{2}$	Real or Integer	Real (radians)
Cos (X)	Returns the cosine of angle X	Real or Integer (radians)	Real

Function	Purpose	Argument	Result
Exp (X)	Returns e^x, where e = 2.71828...	Real or Integer	Real
Ln (X)	Returns the natural logarithm of X for X>0.0	Real or Integer	Real
Round (X)	Returns the closet Integer value to X	Real	Integer
Sin (X)	Returns the sine of angle X	Real or Integer (radians)	Real
Sqr (X)	Returns the square of X	Real or Integer	Same as argument
Sqrt (X)	Returns the positive square root of X for X>0.0	Real or Integer	Real
Trunc (X)	Returns the integer part of X	Real	Integer

สำหรับฟังก์ชัน Round และ Trunc นั้น ส่วนที่เป็นอาร์กิวเมนต์ต้องเป็นชนิด Real และกลับคืนค่าชนิด Integer ฟังก์ชันสองชุดนี้ต้องการหาส่วนที่เป็นจำนวนเต็มของนิพจน์ ซึ่งมีค่าเป็น Real ด้วยเหตุนี้ นิพจน์

Trunc (1.5 * Gross);

Round (Cents / 100)

มีค่าเป็นชนิด Integer และอาจกำหนดให้กับตัวแปรชนิด Integer สำหรับ Trunc หมายถึง ปัดเศษทิ้ง (truncates) หรือลบภาคเศษส่วน (fractional part) ของอาร์กิวเมนต์ของมัน

Round หมายถึง ปัดเศษ (rounds) อาร์กิวเมนต์ให้เป็นเลขจำนวนเต็มใกล้ที่สุด ตัวอย่างเช่น Trunc (17.5) คือ 17 ในขณะที่ Round (17.5) คือ 18

Trunc (-3.8) คือ -3 ในขณะที่ Round (-3.8) คือ -4 ฟังก์ชันส่วนใหญ่ในตาราง 3.1 กระทำการคำนวณเชิงคณิตศาสตร์ร่วม อาร์กิวเมนต์ของ Ln และ Sqrt ต้องเป็นค่าบวก อาร์กิวเมนต์ของ Sin และ Cos ต้องมีหน่วยเป็นเรเดียน (radians) ไม่ใช่องศา (degree) ฟังก์ชัน ArcTan ผลลัพธ์ของมันมีหน่วยเป็นเรเดียน

.ตัวอย่าง 3.2

เราใช้ฟังก์ชัน Sqr และ Sqrt เพื่อคำนวณราก (roots) ของสมการกำลังสอง (quadratic equation) ใน X ของรูปแบบ

$AX^2 + BX + C = 0$

รากสองตัวนิยามดังนี้

$$Root_1 = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

$$Root_2 = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

เมื่อดิสคริมิแนนต์ (discriminant) $B^2 - 4AC$ มีค่ามากกว่าศูนย์ ถ้าเราสมมติว่านี่คือกรณีศึกษาเราสามารถใช้ข้อความสั่งกำหนดค่าต่อไปนี้ กำหนดค่าให้กับ $Root_1$ และ $Root_2$

```
{compute two roots, Root1 และ Root2, for Disc > 0.0}
   Disc := Sqr (B) - 4 * A * C;
   Root1 := (-B + sqrt(Disc)) / (2 * A);
   Root2 := (-B - Sqrt(Disc)) / (2 * A)
```

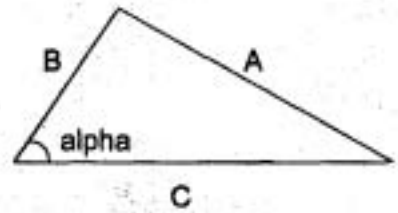
ตัวอย่าง 3.3

ถ้าเราทราบความยาวของด้านสองด้าน (B และ C) ของสามเหลี่ยมรูปหนึ่ง และทราบค่ามุมระหว่างด้านทั้งสองซึ่งมีหน่วยเป็นองศา (Alpha) ดูรูป 3.8 เราสามารถคำนวณหาความยาวของด้านที่สาม (A) โดยใช้สูตร

$A^2 = B^2 + C^2 - 2BCCosAlpha$

การใช้ฟังก์ชัน cosin ของ Pascal เราต้องแสดงมุมอาร์กิวเมนต์ของมันให้มีหน่วยเป็นเรเดียน (radian) ไม่ใช่องศา การเปลี่ยนมุมจากหน่วยองศาเป็นเรเดียน ให้คูณมุมองศาด้วย $\frac{\pi}{180}$ ถ้าสมมติว่า Pi (ดูตาราง 3.2) กลับคืนค่าคงตัว π ข้อความสั่งกำหนดค่าของ Pascal ข้างล่างนี้คำนวณความยาวของด้านซึ่งเรายังไม่ทราบค่า

```
A := Sqrt (Sqr(B) + Sqr(C) - 2 * B * C * Cos(Alpha * Pi / 180.0))
```



รูป 3.8 รูปสามเหลี่ยมซึ่งไม่ทราบค่า A

ตัวอย่าง 3.4

เนื่องจาก Pascal ไม่มีตัวดำเนินการเลขชี้กำลัง (exponentiation operator) จึงเป็นไปไม่ได้ที่จะเขียน u^v โดยตรง เมื่อ u และ v เป็นตัวแปรชนิด Real อย่างไรก็ตาม ทฤษฎีลอการิทึม (theory of logarithms) บอกว่า

$$\ln (u^v) = v \times \ln (u)$$

และ

$$z = e^{\ln(z)}$$

เมื่อ e มีค่าเท่ากับ 2.71828... ดังนั้นถ้าเราแทนที่ u^v ตรง z ในสมการข้างต้น จะได้

$$u^v = e^{(v \times \ln(u))}$$

สูตรนี้สามารถทำให้เกิดผล (implemented) ใน Pascal ดังนี้

```
UToPowerV := Exp(v * Ln(u))
```

ฟังก์ชันของ Turbo Pascal เพิ่มเติม (Additional Turbo Pascal Functions)

ตาราง 3.2 คือรายการฟังก์ชันเชิงคณิตศาสตร์เพิ่มเติม ซึ่งมีอยู่ใน Turbo Pascal (ไม่มีใน standard Pascal)

ฟังก์ชัน Frac และ Int สามารถนำมาใช้เพื่อตัดทอน (extract) ภาคเศษส่วนและภาคจำนวนเต็มตามลำดับของเลขจำนวนจริง

ตัวอย่างเช่น ถ้า X มีค่าเท่ากับ 5.16123

Frac(X) กลับคืนค่าจำนวนจริง 0.16123

และ Int(X) กลับคืนค่าจำนวนจริง 5.0

ตาราง 3.2 ฟังก์ชันของ Turbo Pascal

Function	Purpose	Argument	Result
Frac (num)	Returns the fractional part of its argument	Real	Real
Int (num)	Returns the whole number part of its argument	Real	Real
Pi	Returns an approximation to π	None	Real
Random	Returns a real random number $\geq$ 0.0 and < 1.0	None	Real
Random (num)	Returns a random integer $\geq$ 0 and < num	Integer	Integer

ฟังก์ชัน Pi ไม่มีอาร์กิวเมนต์ มันกลับคืนค่าโดยประมาณของ π (3.1415926536) ดังนั้นเราจึงใช้ Pi แทนที่ค่าคงตัว MyPi ในรูป 3.4 และ 3.5 ได้

ฟังก์ชัน Random ก่อกำเนิดเลขสุ่ม

เลขสุ่ม หมายถึง เลขซึ่งถูกเลือกแบบสุ่มจากพิสัยที่กำหนดของเลขต่างๆ เลขทุกตัวมีโอกาสถูกเลือกเท่าๆ กัน (A random is a number that is selected at random from a specified range of numbers, each with an equally likely chance for selection.)

ฟังก์ชัน Random อาจถูกเรียกโดยมีอาร์กิวเมนต์หรือไม่มีอาร์กิวเมนต์ ถ้ามีอาร์กิวเมนต์ Random จะกลับคืนเลขจำนวนเต็มสุ่มจาก 0 จนถึงอาร์กิวเมนต์ของมัน แต่ไม่รวม

อาร์กิวเมนต์ ทั้งนี้ใน Turbo Pascal จะต้องเรียกกระบวนงาน Randomize ก่อนที่จะเรียกฟังก์ชัน Random ชุดแรก

ตัวอย่าง 3.5

โปรแกรม MultQuest ในรูป 3.9 เป็นการฝึกฝนและปฏิบัติเรื่องการคูณ มีการเรียกฟังก์ชัน Random สองครั้ง ในการเรียกแต่ละครั้งเลือกเลขจำนวนเต็มจาก 0 ถึง 9 (ไม่รวม 9) ค่าสุ่มเหล่านี้คือตัวถูกดำเนินการ (Factor 1, Factor 2) ในคำถามการคูณ ถ้า 7 และ 8 เป็นค่าซึ่งกลับคืนจาก Random ตัวอย่างคำถาม

What is the value 7 * 8 ?

แสดงขึ้นมา คำตอบของผู้ใช้อ่านไว้ใน Response และคอมพิวเตอร์จะแสดงผลผลคูณของ 7 และ 8 ดังนั้น ผู้ใช้สามารถตรวจคำตอบได้

เพื่อให้การฝึกฝนและโปรแกรมฝึกหัดสมบูรณ์มากขึ้น คอมพิวเตอร์ควรเปรียบเทียบผลคูณจริงจาก Factor 1 และ Factor 2 กับค่าที่ใส่ใน Response แล้วมีข้อความที่ถูกต้องแสดงผลว่าค่าใน Response ถูกต้องหรือไม่

Edit Window

```
program multQuest;
{
   Asks a random multiplication question and displays the correct product
   after the student responds
}
   const
      Limit = 10;              {operand is < Limit}

   var
      Factor 1, Factor 2       {two operands}
      Answer,                  {correct answer}
      Response : Integer;      {student response}
```

```
begin {MultQuest}
   Randomize;                    {Initialize the random number generator}
   Factor 1 := Random (Limit);   {Get first operand.}
   Factor 2 := Random (Limit);   {Get second operand.}
   Write (' What is the value of ; Factor 1 : 1, ' * ' , Factor 2 : 1, ' ? ');
   ReadLn (Response);
   Answer := Factor 1 * Factor 2;
   WriteLn (' Correct answer is ' , Answer : 1)
end.  {Multquest}
```

Output Window

```
What is the value of 7 * 8 ? 82
Correct answer is 56
```

รูป 3.9 โปรแกรม MultQuest

ดูว่าเรากำลังอยู่หัวข้อใด (A look at Where We Are Heading)

Pascal ยอมให้เราเขียนฟังก์ชันของเราเอง สมมติว่าเรามีฟังก์ชันเขียนเรียบร้อยแล้วคือ FindArea และ FindCircum

Function FindArea (R) กลับคืนพื้นที่ของวงกลมที่มีรัศมีเท่ากับ R

Function FindCircum (R) กลับคืนเส้นรอบวงของวงกลมที่มีรัศมีเท่ากับ R

เราสามารถนำฟังก์ชันเหล่านี้กลับมาใช้ใหม่ (reuse) จากสองโปรแกรมก่อนหน้าในบทนี้ (ดูรูปที่ 3.4 และรูปที่ 3.5)

โปรแกรมในรูป 3.4 แสดงผลพื้นที่และเส้นรอบวงของวงกลม ซึ่งรัศมีเป็นข้อมูลอินพุต รูป 3.10 แสดงโปรแกรมปรับแก้ไขใหม่ ซึ่งใช้ฟังกชัน FindArea และฟังก์ชัน Find Circum ในรูป 3.10 ส่วนนิพจน์ของข้อความสั่งกำหนดค่าแต่ละชุด

```
Area = FindArea (Radius);
Circum := FindCircum (Radius);
```

คือ function designator มีอาร์กิวเมนต์ Radius (รัศมีของวงกลม) ผลลัพธ์ส่งกลับโดยการกระทำการฟังก์ชันแต่ละชุด แล้วเก็บในตัวแปรเอาต์พุต สำหรับโปรแกรม (Area หรือ Circum) การวิ่งโปรแกรมนี้ เราจำเป็นต้องเขียนฟังก์ชันให้เสร็จบริบูรณ์ และใส่ฟังก์ชันทั้งคู่หลังคอมเมนต์ข้างล่างนี้

{Insert functions FindArea and FindCircum here.}

หัวข้อ 6.2 จะแสดงให้เห็นการเขียนฟังก์ชันของเราเอง

นอกจากข้อดีของการนำรหัสกลับมาใช้ใหม่แล้ว การใช้สองฟังก์ชันนี้ทำให้เราเป็นอิสระจากรายละเอียดของการคำนวณ พื้นที่หรือเส้นรอบวงของวงกลม เมื่อเราเขียนโปรแกรมหลัก (main program) นี่คือวิธีหนึ่งที่เราสามารถจัดการและลดความซับซ้อนของการเขียนโปรแกรม

```
program CircleFunction;
{Finds area and circumference of a circle using functions}
  var
    Radius,              {input - radius of a circlr}
    Area,                {output - area of a circle}
    Circumference :      {output - circumference of a circle}

  {Insert functions FindArea and FindCircum here.}

begin {main control section}
  {Read the circle radius.}
  Write (' Enter radius > ');
  ReadLn (Radius);
  {Find the area.}
    Area := FindArea (Radius);

  {Find the circumference.}
```

```
    Circum := FindCircum (Radius);

    {Print the area and circumference.}
    WriteLn (' The area is ' , Area : 4 : 2);
    WriteLn (' The circumference is ' , Circum : 4 : 2)
end.
```

รูป 3.10 การหาพื้นที่และเส้นรอบวงโดยใช้ฟังก์ชัน

แบบฝึกหัด 3.2 Self-Check

1. จงเขียนนิพจน์คณิตศาสตร์ข้างล่างนี้ใหม่โดยใช้ฟังก์ชัน Pascal
   a) $\sqrt{U + V} \times W^2$
   b) $\log_n (X^Y)$
   c) $\sqrt{(X - Y)^3}$
   d) $|XY - W/Z|$
2. จงประเมินผล function designators ข้างล่างนี้ และให้บอกชนิดของผลลัพธ์
   a) Tranc (- 15.8)
   b) Round (- 15.8)
   c) Round (6.8) * Sqr (3)
   d) Int (- 15.8) * Sqr (3)
   e) Sqrt (Abs (Round (- 15.8)))
   f) Round (3.5)
   g) Sqr (3.0)
   h) Trunc (22.1) * Sqr (3)

เขียนโปรแกรม (Programming)

1. จงเขียนข้อความสั่งอ่านค่าสองค่าไว้ใน X และ Y จากนั้นคำนวณและแสดงผลผลต่างสัมบูรณ์ (absolute difference)

   ตัวอย่าง ถ้า X เท่ากับ 9 และ Y เท่ากับ 7 ผลต่างสัมบูรณ์ เท่ากับ 2

2. จงใช้ฟังก์ชัน Round เขียนข้อความสั่ง Pascal เพื่อปัดเศษ ค่าจำนวนจริง X ให้ใกล้สุด มีจุดทศนิยมสองตำแหน่ง

คำแนะนำ : ให้คูณ X ด้วย 100 ก่อนทำการปัดเศษ จากนั้นปัดเศษผลลัพธ์และต่อไปหารด้วย 100)

3. จงเขียนโปรแกรม Pascal ที่เสร็จบริบูรณ์แจ้งผู้ใช้ให้ใส่พิกัดคาร์ทีเซียน (Cartesian coordinates) ของสองจุด (X1, Y1) และ (X2, Y2) จากนั้นให้แสดงผลระยะทางระหว่างสองจุดโดยใช้สูตรต่อไปนี้

$$\text{distance} = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

3.3 การออกแบบจากบนลงล่างและผังเชิงโครงสร้าง (Top-Down Design and Structure Charts)

บ่อยครั้งที่อัลกอริทึมซึ่งใช้แก้ปัญหามีความซับซ้อนมากกว่าอัลกอริทึมทั้งหลายโปรแกรมเมอร์จึงต้องแบ่งปัญหาให้เป็นปัญหาย่อยเพื่อพัฒนาผลเฉลยโปรแกรม ในความพยายามที่จะแก้ปัญหาย่อยที่หนึ่งระดับ เราแนะนำปัญหาย่อยชุดใหม่ที่ระดับต่ำกว่ากระบวนการนี้เรียกว่า การออกแบบจากบนลงล่าง ดำเนินการจากปัญหาเดิม (original) ที่ระดับบนไปสู่ปัญหาย่อยที่แต่ละระดับซึ่งต่ำกว่า การแบ่งหนึ่งปัญหาให้เป็นปัญหาย่อยที่เกี่ยวข้องกันคล้ายกับกระบวนการแบ่งละเอียดของอัลกอริทึม กรณีศึกษาต่อไปนี้จะแนะนำเครื่องมือเชิงเอกสารชนิดหนึ่งเรียกว่าผังเชิงโครงสร้าง ซึ่งจะช่วยเราเก็บร่องรอย (track) ของความสัมพันธ์ระหว่างปัญหาย่อย

การออกแบบจากบนลงล่าง หมายถึง วิธีการแก้ปัญหา ซึ่งขั้นแรกเราแบ่งปัญหาให้เป็นปัญหาย่อยที่สำคัญของมัน จากนั้นแก้ปัญหาย่อย เพื่อให้ได้มาซึ่งผลเฉลยของปัญหาเดิม (Top-down design is a problem-solving method in which you first break a problem up into its major subproblems and them solve the subproblems to derive the solution to the original problem.)

ผังเชิงโครงสร้าง หมายถึง เครื่องมือเชิงเอกสารซึ่งแสดงให้เห็นความสัมพันธ์ระหว่างปัญหาย่อยต่างๆ ของหนึ่งปัญหา (A structure chart is a documentation tool that shows the relationships among the subproblems of a problem.)

กรณีศึกษา การวาดรูปแผนภาพอย่างง่าย (Drawing Simple Diagrams)

(1) ปัญหา (Problem)

ต้องการวาดแผนภาพอย่างง่ายบนเครื่องพิมพ์ (printer) หรือจอภาพ (screen) มีสองตัวอย่าง คือ รูปบ้าน และรูปผู้หญิง ในรูป 3.11

(2) วิเคราะห์ (Analisys

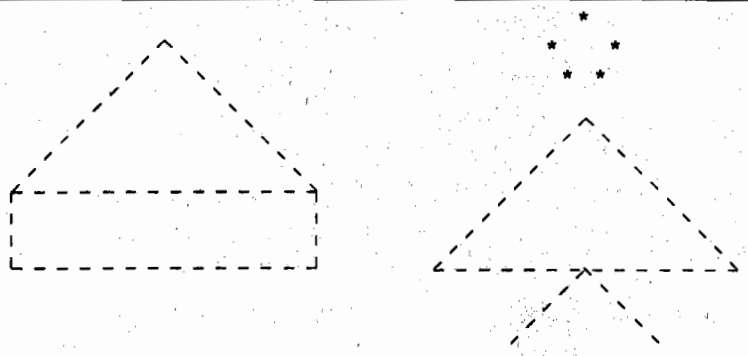
รูปบ้านประกอบด้วยรูปสามเหลี่ยมไม่มีฐานอยู่บนรูปสี่เหลี่ยมผืนผ้า ส่วนรูปผู้หญิง ประกอบด้วย รูปวงกลม รูปสามเหลี่ยม และรูปสามเหลี่ยมไม่มีฐาน เราสามารถวาดรูปทั้งสองรูปโดยมีส่วนประกอบพื้นฐานสี่อย่างคือ

วงกลม (a circle)

ฐานของรูปสามเหลี่ยม (a base line)

เส้นขนาน (parallel lines)

เส้นตัด (intersection lines)



รูป 3.11 รูปบ้านและรูปผู้หญิง

(3) ออกแบบ (Design)

การสร้างรูปผู้หญิง เราต้องแบ่งปัญหาออกเป็นปัญหาย่อยสามชุด

อัลกอริทึมเริ่มต้น

1. วาดรูปวงกลม
2. วาดรูปสามเหลี่ยม
3. วาดรูปเส้นตัด

การแบ่งละเอียดอัลกอริทึม

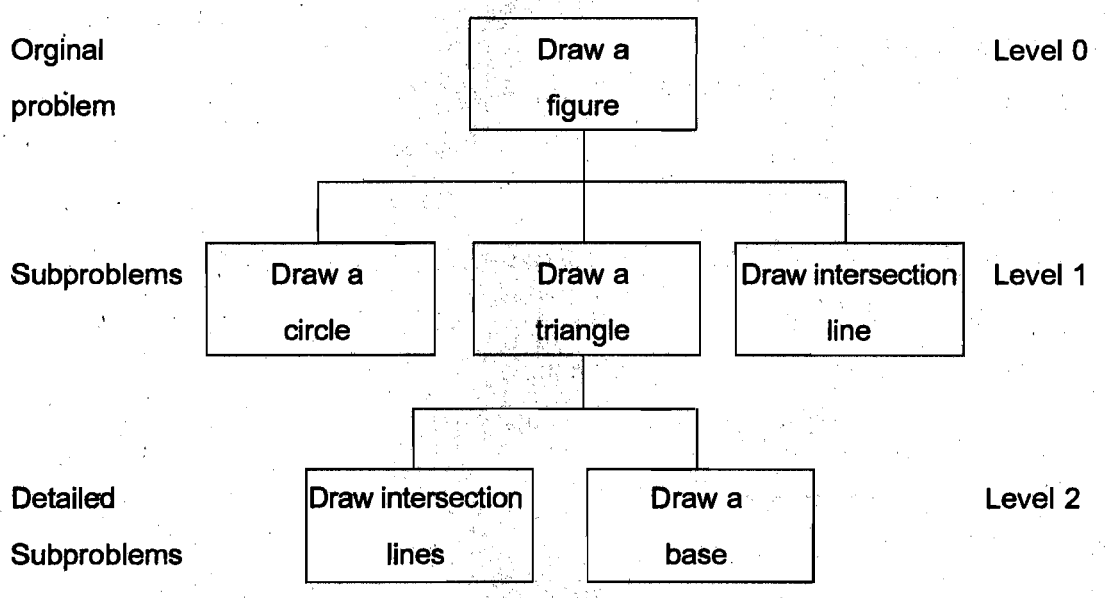
เพราะว่ารูปสามเหลี่ยมไม่ใช่ส่วนประกอบหลัก เราจึงต้องแบ่งละเอียดขั้นที่ 2 ทำให้เกิดปัญหาย่อย ดังนี้

การแบ่งละเอียดขั้นที่ 2

2.1 วาดรูปเส้นตัด

2.2 วาดรูปฐาน

เราสามารถใช้ผังเชิงโครงสร้างเพื่อแสดงความสัมพันธ์ระหว่างปัญหาเดิมและปัญหาย่อยของมัน ดังแสดงในรูป 3.12 ซึ่งปัญหาเดิม ระดับ 0 และปัญหาย่อยอีก 3 ชุด แสดงที่ระดับ 1 ปัญหาย่อยวาดรูปสามเหลี่ยมแบ่งเป็นปัญหาย่อยของมันเองอีก อยู่ที่ระดับ 2



รูป 3.12 ผังเชิงโครงสร้างสำหรับวาดรูปผู้หญิง

ปัญหาย่อยปรากฏอยู่ในอัลกอริทึมและผังเชิงโครงสร้างทั้งคู่ในอัลกอริทึมแสดงอันดับ (order) ซึ่งเราต้องทำแต่ละขั้นตอนให้ประสบผลสำเร็จเพื่อแก้ปัญหา ส่วนในผังเชิงโครงสร้างเพียงแสดงให้เห็นส่วนย่อยของปัญหาย่อยของแต่ละชุดและของปัญหาย่อยเดิม

3.4 กระบวนงานหรือกระบวนคำสั่งหรือโปรซีเดอร์ (Procedures)

บ่อยครั้งโปรแกรมเมอร์ใช้กระบวนงานเพื่อ implement การออกแบบจากบนลงล่างในโปรแกรม กระบวนงานคล้ายฟังก์ชัน คือเป็นหน่วยโปรแกรมหรือมอดูลแยกจากกัน ประกอบด้วยกลุ่มของข้อความสั่งโปรแกรมเพื่อแก้ปัญหาเฉพาะเรื่อง

โปรซีเดอร์ หมายถึง ส่วนจำเพาะโปรแกรมเขียนแยกจากกันเพื่อแก้ปัญหาอย่างหนึ่ง (A procedure is a separate program module solving a particular subproblem.)

โปรแกรมเมอร์จะเขียนกระบวนงานหนึ่งชุดสำหรับแต่ละปัญหาย่อยในผังเชิงโครงสร้าง กระบวนงานมีลักษณะทั่วไปมากกว่าฟังก์ชันเพราะว่ากระบวนงานกลับคืน (return) ผลลัพธ์จำนวนเท่าใดก็ได้

ในบทที่ 2 ได้แนะนำกระบวนงาน ซึ่งนำเสนอโดยคอมไพเลอร์ Pascal ไปแล้วสามชุด ได้แก่ ReadLn, Write และ WriteLn การเรียกใช้งานกระบวนงานแต่ละชุด ให้ใช้ข้อความสั่งเรียกกระบวนงาน (procedure call statement) ในหัวข้อนี้เราจะเรียนรู้การเขียนและเรียกกระบวนงานด้วยตนเอง

ข้อความสั่งเรียกกระบวนงาน หมายถึง คำสั่งซึ่งเรียกหรือใช้งานกระบวนงาน (A procedure call statement is an instruction that calls or activates a procedure.)

ตัวอย่างเช่น การออกแบบจากบนลงล่างที่มีกระบวนงาน เราสามารถใช้ตัวโปรแกรมในรูป 3.13 เพื่อวาดรูปผู้หญิง ขั้นตอนอัลกอริทึมทั้งสามชุดเขียนรหัสเรียกกระบวนงาน Pascal สามชุด ตัวอย่างเช่น ข้อความสั่ง

```
DrawTriangle;        {Draw a triangle.}
```

เรียกกระบวน (Drew Triangle) เพื่อ implement อัลกอริทึมขั้นวาดรูปสามเหลี่ยม

```
begin {StickFigure}
   DrawCircle;       {Draw a circle.}
   DrawTriangle;     {Draw a triangle.}
   DrawIntersect     {Draw intersection lines.}
end.  {StickFigure}
```

รูป 3.13 ตัวโปรแกรมเพื่อวาดรูปผู้หญิง

กระบวนงาน หมายถึง หน่วยโปรแกรมอิสระ ซึ่งรูปแบบของมันคล้ายกับโปรแกรมอย่างมาก กระบวนงานขึ้นต้นด้วย procedure heading ซึ่งประกอบด้วยคำว่า procedure ตามด้วยชื่อของ procedure (ซึ่งเป็นไอเดนติไฟเออร์) และเครื่องหมาย semicolon ส่วนการประกาศอาจละเว้นได้ (optional) และจำเป็นต้องเขียนเฉพาะเมื่อกระบวนงานมีค่าคงตัวและตัวแปรของมันเอง ทุกกระบวนงานมีตัวโปรซีเดอร์ (procedure body) ซึ่งเริ่มต้นด้วย begin และ end;

ชื่อโปรซีเดอร์เป็นไอเดนติไฟเออร์ซึ่งให้นิยามโดยผู้ใช้ (user-defined indentifier) ดังนั้นจึงต้องถูกประกาศในส่วนการประกาศของหน่วยโปรแกรมซึ่งเรียกมัก เราใช้การประกาศโปรซีเดอร์ เพื่อประกาศกระบวนงาน

Syntax Display

การประกาศกระบวนงาน (Procedure Declaration)

Form:

```
procedure pname;
declaration part
begin
    procedure body
end;
```

ตัวอย่าง

```
procedure Skip3Lines;
{Skips three lines}

begin {skip3Lines}
   WriteLn;
   WriteLn;
   WriteLn
end;        {Skip3Lines}
```

มีความหมายดังนี้ ไอเดนติไฟเออร์ ซึ่งผู้ใช้ให้นิยาม pname ถูกประกาศให้เป็นชื่อของกระบวนงาน ไอเดนติไฟเออร์ใดๆ ก็ตาม ซึ่งประกาศในส่วนการประกาศจะถูกนิยาม

เฉพาะระหว่างการกระทำการของกระบวนงาน และสามารถถูกอ้างถึงได้เฉพาะภายในกระบวนงานเท่านั้น ตัวกระบวนงานอธิบายการจัดดำเนินการข้อมูล ซึ่งกระทำเมื่อกระบวนงานถูกเรียกผ่านทางข้อความสั่งเรียกกระบวนงาน

Syntax Display

ข้อความสั่งเรียกกระบวนงาน (Procedure Call Statement)

Form: | pname |

ตัวอย่าง DrawCircle

มีความหมายดังนี้ ข้อความสั่งเรียกกระบวนงานให้เริ่มต้นการกระทำการของกระบวนงาน pname หลังจากการกระทำการ pname เสร็จสิ้นแล้ว ข้อความสั่งโปรแกรมซึ่งตามหลัง procedure call จะถูกกระการ

ตัวอย่าง 3.6

รูป 3.14 แสดงให้เห็นการประกาศของกระบวนงาน DrawCircle ตัวกระบวนงานมีข้อความสั่ง WriteLn สามบรรทัดซึ่งทำให้คอมพิวเตอร์วาดรูปวงกลม ข้อความสั่งเรียกกระบวนงาน DrawCircle ทำให้ข้อความสั่ง WriteLn เหล่านี้ถูกกระทำการ

```
procedure DrawCircle;
{Draw a circle}

begin {DrawCircle}
   WriteLn    (' * ');
   WriteLn   (' *   * ');
   WriteLn    (' * * ');
end;  {Drawcircle}
```

รูป 3.14 กระบวนงาน DrawCircle

ผังเชิงโครงสร้างรูป 3.12 แสดงให้เห็นว่าปัญหาย่อย Draw a Triangl (ระดับ 1) ขึ้นอยู่กับผลเฉลยของส่วนย่อยของปัญหาย่อย DrawIntersecting Lines และ Draw a Base (ระดับ 2 ทั้งคู่)

รูป 3.15 แสดงให้เห็นว่าจะใช้การออกแบบจากบนลงล่าง เพื่อลงรหัสกระบวนงาน DrawTriangle อย่างไร แทนที่จะใช้ข้อความสั่ง WriteLn เพื่อแสดงผลรูปสามเหลี่ยม ตัวกระบวนงาน DrawTriangle เรียกกระบวนงาน DrawIntersect และ DrawBase เพื่อวาดรูปสามเหลี่ยม

```
precedure DrawTriangle; ← Procedure heading
{Draw a triangle}

begin {DrawTriangle}
   DrawIntersect;
                          } ← Procedure body
   DrawBase
end;  {DrawTriangle}
```

รูป 3.15 กระบวนงาน DrawTriangle

การวางการประกาศกระบวนงานในโปรแกรม (Placement of Procedure Declarations in Program)

รูป 3.13 แสดงตัวโปรแกรมของโปรแกรมวาดรูปผู้หญิง ในส่วนการประกาศ เราต้องประกาศกระบวนงานทั้งสามชุด ซึ่งถูกเรียกในโปรแกรม ส่วนประกาศกระบวนงานจะตามหลังการประกาศตัวแปรในโปรแกรม Turbo Pascal

อันดับสัมพัทธ์ (relative order) ของการประกาศกระบวนงานไม่มีผลต่ออันดับการกระทำการของมัน ซึ่งถูกกำหนดโดยอันดับของการกระทำการของข้อความสั่งเรียกกระบวนงาน อย่างไรก็ตามกระบวนงานทุกชุดต้องมีการประกาศก่อนที่เราจะเขียนข้อความสั่งเรียกกระบวนงาน ซึ่งเรียกใช้กระบวนงานนั้น ด้วยเหตุผลนี้ เราจึงต้องประกาศกระบวนงานสองชุดซึ่งถูกเรียกโดย DrawTriangle (คือ DrawIntersect และ DrawBase) ก่อน DrawTriangle

รูป 3.16 แสดงโปรแกรมวาดรูปผู้หญิงที่เสร็จสมบูรณ์ในโปรแกรมซึ่งใช้กระบวนงาน เราเรียกตัวโปรแกรมว่า โปรแกรมหลัก (main program)

```
program StickFigure;
{Displays a stick figure}
   procedure DrawCircle;
   {Draws a circle}

   begin {Draw a circle}
      WriteLn  ( ' * ' );
      WriteLn  (' *   * ');
      WriteLn  ( ' * * ' );
   end; {DrawCircle}

   procedure DrawInterset;
   {Draw intersecting lines}

   begin {DrawIntersect}
      WriteLn  (' / \ ');
      WriteLn  (' /   \ ');
      WriteLn (' /     \ ');
   end; {DrawIntersect}

   procedure DrawBase;
   {Draw a base}

   begin {DrawBase}
      WriteLn ('.....')
```

```
    end; {DrawBase}

    procedure DrawTriangle;
    {Draw a triangle}

    begin {DrawTriangle}
       DrawIntersect;
       DrawBase
    end; {DrawTriangle}

begin {SticFigure}
    DrawCircle;        {Draw a circle.}
    DrawTriangle;      {Draw a triangle.}
    DrawIntersect      {Draw intersecting lines.}
end. {StickFigure}
```

รูป 3.16 โปรแกรมวาดรูปผู้หญิง

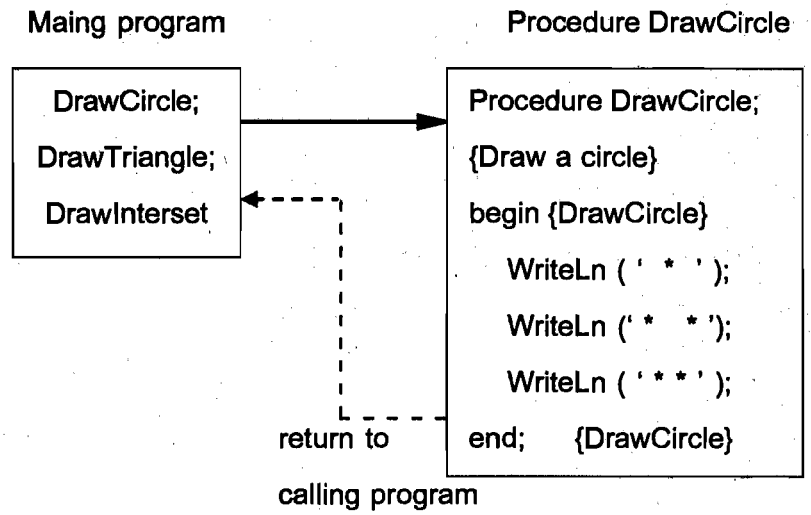
อันดับของการกระทำการของกระบวนงานและโปรแกรมหลัก (Order of Execution of Procedures and Main Program)

เนื่องจากกระบวนงานปรากฏในส่วนการประกาศของโปรแกรม ซึ่งอยู่ก่อนหน้าโปรแกรมหลัก คอมไพเลอร์จึงแปลการประกาศกระบวนงานก่อนแปลโปรแกรมหลัก ระหว่างการแปลนั้นคอมไพเลอร์ใส่หนึ่งข้อความสั่งที่ตอนท้ายของกระบวนงานซึ่งทำให้มีการส่งกลับคืน (return) จากกระบวนงานกลับไปยังข้อความสั่งซึ่งเรียกมัน

รูป 3.17 แสดงโปรแกรมหลักและกระบวนงาน DrawCircle ของโปรแกรมรูปผู้หญิงในเนื้อที่ของหน่วยความจำแยกจากกัน ถึงแม้ว่าข้อความสั่ง Pascal ที่แสดงในรูป 3.17 ที่จริงเป็นรหัสจุดหมาย (object code) สมนัยกับแต่ละคำสั่ง ซึ่งเก็บในหน่วยความจำ

เมื่อเราวิ่งโปรแกรม ข้อความสั่งแรกในโปรแกรมหลัก คือข้อความสั่งแรกที่ถูกกระทำการ (เรียก DrawCircle ในรูป 3.17) เมื่อคอมพิวเตอร์กระทำการ ข้อความสั่งเรียก

กระบวนงานส่งการควบคุม (control) ไปยังกระบวนงานที่อ้างถึง (แสดงโดยเส้นทึบในรูป 3.17) คอมพิวเตอร์จัดสรรหน่วยความจำซึ่งอาจจำเป็นต้องใช้สำหรับการประกาศค่าคงตัวและตัวแปรในกระบวนงานหลังจากนั้นกระทำข้อความสั่งต่างๆ ในตัวกระบวนงานหลังจากข้อความสั่งสุดท้ายในกระบวนงาน DrawCircle ถูกกระทำการ control คืนกลับไปยังโปรแกรมหลัก (แสดงโดยเส้นประในรูป 3.17) และคอมพิวเตอร์ปล่อย (releases) เนื้อที่หน่วยความจำใดๆ ซึ่งได้ถูกจัดสรรให้กับกระบวนงาน หลังจากคืนกลับไปยังโปรแกรมข้อความสั่งถัดไปจะถูกกระทำการ (เรียก DrawTriangle)



รูป 3.17 การควบคุมสายงาน (flow of control) ระหว่างโปรแกรมหลักและกระบวนงาน

ข้อดีของการใช้กระบวนงาน (Advantages of Using Procedures)

มีข้อดีหลายประการของการใช้กระบวนงาน การเปลี่ยนแปลงสภาพพร้อมใช้งานของมัน โปรแกรมเมอร์แต่ละคนจัดระเบียบผลเฉลยให้กับการเขียนโปรแกรมปัญหาอย่างไรสำหรับทีมของโปรแกรมเมอร์ซึ่งทำงานด้วยกันบนโปรแกรมขนาดใหญ่

กระบวนงานทำให้งานเขียนโปรแกรมง่ายขึ้น โปรแกรมเมอร์แต่ละคนจะรับผิดชอบเฉพาะเซตของกระบวนงานเฉพาะเรื่องสุดท้าย เขาทำให้งานเขียนโปรแกรมง่ายขึ้น เป็นกระบวนงานซึ่งมีอยู่จริงสามารถนำมาใช้เป็นบล็อกการสร้าง (building blocks) สำหรับโปรแกรมใหม่

การนิยามนามธรรมเชิงกระบวนงาน (Procedural Abstraction)

กระบวนงานทำให้เราลบทิ้งรหัสซึ่งเป็นผลเฉลยในรายละเอียดให้กับปัญหาย่อยออกจากโปรแกรมหลัก

เนื่องจากรายละเอียดเหล่านี้อยู่ในกระบวนงานและไม่อยู่ในโปรแกรมหลัก เราสามารถเขียนโปรแกรมหลักเป็นลำดับของข้อความสั่งเรียกกระบวนงานทันทีที่เรามีอัลกอริทึมเริ่มต้นที่กำหนดและก่อนที่เราแบ่งละเอียดขั้นตอนใดๆ เราควรประวิง (delay) การเขียนโปรแกรมสำหรับขั้นอัลกอริทึม จนกระทั่งเรามีการแบ่งละเอียดที่เสร็จสิ้นแล้วของขั้นนั้น วิธีการออกแบบเช่นนี้เรียกว่า การนิยามนามธรรม เราประวิงรายละเอียดของการปฏิบัติให้เกิดผลจนกระทั่งเราพร้อมที่จะเขียนมอดูลกระบวนงานแต่ละชุด

การเน้นที่ครั้งละหนึ่งกระบวนงานจะง่ายกว่าความพยายามที่จะเขียนโปรแกรมที่เสร็จบริบูรณ์ทั้งหมดในครั้งเดียว

การนิยามนามธรรมเชิงกระบวนงาน หมายถึง เทคนิคของการเขียนโปรแกรมซึ่งโปรแกรมหลักประกอบด้วยลำดับของการเรียกกระบวนงานและแต่ละกระบวนงานถูก implement เป็นมอดูลโปรแกรมแยกจากกัน (A procedural abstraction is a programming technique in which a main program consits of a sequence of procedure calls and each procedure is implemental as a separate program module.)

การนำกลับมาใช้ใหม่ของกระบวนงาน (Reuse of Procedures)

ข้อดีอีกประการหนึ่งของการใช้กระบวนงาน คือ มันสามารถถูกกระทำการได้มากกว่าหนึ่งครั้งในโปรแกรม ตัวอย่างเช่น กระบวนงาน DrawIntersect ถูกเรียกสองครั้งในรูป 3.16 (ครั้งหนึ่งโดย DrawTriangle และอีกครั้งหนึ่งโดยโปรแกรมหลัก) ทุกครั้งที่ DrawIntersect ถูกเรียก รายการของข้อความสั่งเอาต์พุตแสดงให้เห็นในรูป 3.16 ถูกกระทำการ และคู่ของเส้นตัดถูกวาด ถ้าไม่มีกระบวนงาน ข้อความสั่ง WriteLn ซึ่งวาดเส้นจะต้องอยู่ในรายการสองครั้งในโปรแกรมหลัก ซึ่งเป็นเหตุให้ความยาวของโปรแกรมหลักเพิ่มขึ้น และมีโอกาสผิดพลาด

สุดท้าย เมื่อเราเขียนและทดสอบกระบวนงาน เราสามารถใช้มันในโปรแกรมอื่นหรือกระบวนงานอื่นๆ ตัวอย่างเช่น กระบวนงานในโปรแกรม StickFigure สามารถนำไปใช้ในโปรแกรมซึ่งวาดรูปแผนภูมิอื่นๆ

แบบฝึกหัด 3.4

1. สมมติว่าเรามีกระบวนงาน PrintM และ PrintO กระบวนงานแต่ละชุดวาดรูปกล่องขนาดใหญ่ของตัวอักษร (ตัวอย่างเช่น PrintO วาดรูปกล่องตัวอักษร O) ทำไมตัวโปรแกรมข้างล่างนี้จึงมีประสิทธิภาพโดยการใช้กระบวนงานเหล่านี้มากกว่าที่จะไม่ใช้กระบวนงานเลย

```
bigin {main}
   PrintM;
   PrintO;
   PrintM
end. {main}
```

เขียนโปรแกรม

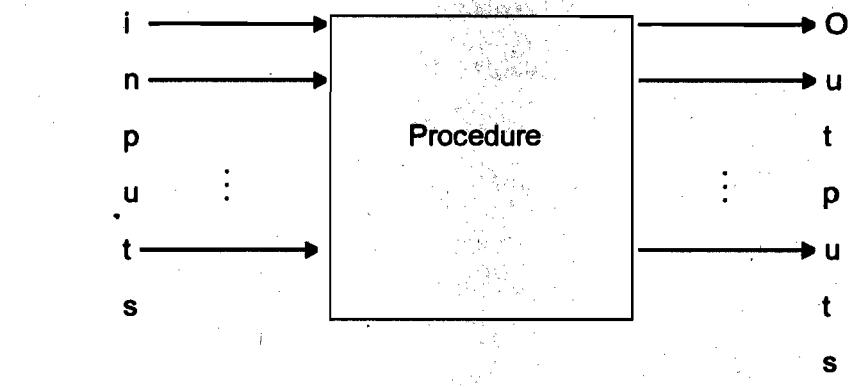
1. จงเขียนกระบวนงาน DrawParallel ซึ่งวาดรูปเส้นขนาน และกระบวนงาน DrawRectangle ซึ่งใช้ DrawParallel และ DrawBase เพื่อวาดรูปสี่เหลี่ยมผืนผ้า

3.5 กระบวนงานคือบล็อกการสร้างโปรแกรม (Procedures as Program Building Blocks)

โปรแกรมเมอร์ใช้กระบวนงานคล้ายกับบล็อกการสร้างเพื่อสร้างโปรแกรมขนาดใหญ่ กระบวนงานคล้าย Lego®blocks ไม่ใช่บล็อกไม้มีด้านเรียบ ซึ่งใช้เรียงต่อกันแต่เชื่อมต่อกันไม่ได้

การเขียนโปรแกรมเมื่อเปรียบเทียบกับบล็อกการสร้างสองประเภทข้างต้น กระบวนงาน DrawCircle และ DrawParallel คล้ายกับบล็อกไม้อย่างง่าย เราสามารถเขียนโปรแกรมขนาดเล็กที่สวยงามด้วยกระบวนงานเหล่านี้ และไม่ได้เป็นประโยชน์เฉพาะเรื่อง การสร้างโปรแกรมที่น่าสนใจมากกว่าคือเราต้องจัดกระบวนงานที่มีส่วนยื่นและครอบเพื่อให้มันสามารถเชื่อมต่อกันอย่างสากลได้โดยง่าย

พารามิเตอร์ (parameters) ของกระบวนงานจัดการเชื่อมโยงระหว่างกระบวนงานและโปรแกรมหลัก หรือระหว่างกระบวนงานสองชุดหรือมากกว่าสองชุดขึ้นไป พารามิเตอร์ของกระบวนงานรับค่า (โปรซีเดอร์อินพุต) ส่งเข้าไปยังกระบวนงานจากมอดูลเรียกหรือมันกลับคืนผลลัพธ์ (โปรซีเดอร์เอาต์พุต) ซึ่งคำนวณโดยกระบวนงานกลับไปยังมอดูลเรียก รูป 3.19 คือแผนภาพของกระบวนงานที่มีอินพุตและเอาต์พุต



รูป 3.19 กระบวนงานที่มีอินพุตและเอาต์พุต

โปรซีเดอร์พารามิเตอร์ หมายถึง สารสนเทศซึ่งส่งผ่านระหว่างกระบวนงานและหน่วยโปรแกรมซึ่งเรียกมัน (The procedure parameters are information passed between a procedure and the program unit that calls it.)

โปรซีเดอร์อินพุต หมายถึง ค่าซึ่งส่งเข้าไปยังกระบวนงานโดยหน่วยโปรแกรมเรียก (The procedure inputs are values passed into a procedure by the calling program unit.)

โปรซีเดอร์เอาต์พุต หมายถึง ผลลัพธ์ซึ่งกลับคืนไปยังหน่วยโปรแกรมเรียก โดยกระบวนงาน (The procedure outputs are results returned to the calling program unit by a procedure.)

เราเคยพบพารามิเตอร์มาแล้วในการทำงานกับกระบวนงาน WriteLn และ ReadLn ข้อความสั่งเรียกกระบวนงาน

```
WriteLn   (' The area is ', Area : 4 : 2)
  ↑                  ↑
Procedure          Parameter
  name               list
```

ประกอบด้วยสองส่วน : ชื่อของกระบวนงาน Pascal ซึ่งกำลังถูกเรียก คือ WriteLn และรายชื่อพารามิเตอร์ (parameter list) ภายในเครื่องหมายวงเล็บ ในที่นี้มีพารามิเตอร์

สองตัวคือ string และชื่อตัวแปรคั่นด้วย comma เพราะว่ามันมีรายชื่อพารามิเตอร์กระบวนงาน WriteLn จึงคล่องตัวมากกว่าและเป็นประโยชน์มากกว่ากระบวนงาน DrawCircle

กระบวนงาน DrawCircle แสดงผลได้เฉพาะรูปวงกลมหนึ่งวง ในขณะที่กระบวนงาน WriteLn สามารถแสดงผลอะไรก็ได้ที่เราต้องการให้ทำ

บทที่ 6 จะอภิปรายพารามิเตอร์ในรายละเอียดมากขึ้น สำหรับขณะนี้เราจะจำกัดการใช้กระบวนงานโดยไม่มีพารามิเตอร์เพื่อแสดงผลข้อความที่มีความยาวมากหรือคำสั่งให้กับผู้ใช้โปรแกรม และจะต่อเนื่องการใช้กระบวนงาน ReadLn, Write และ WriteLn สำหรับใส่ข้อมูลและแสดงผลข้อมูล

ตัวอย่าง 3.7 การแสดงผลคำสั่งผู้ใช้

จงเขียนกระบวนงานแสดงผลคำสั่งให้ผู้ใช้โปรแกรมซึ่งคำนวณพื้นที่และเส้นรอบวงของวงกลม (ดูรูป 3.4)

กระบวนงานชุดนี้แสดงให้เห็นข้อดีประการหนึ่งของการแยกข้อความสั่ง ซึ่งแสดงผลคำสั่งผู้ใช้จากตัวโปรแกรมหลัก การตรวจแก้คำสั่งเหล่านี้จะง่ายขึ้น เมื่อมันถูกแยกออกจากรหัส ซึ่งกระทำการคำนวณ

ถ้าเราใส่กระบวนงาน Instruct (ดูรูป 3.20) ในส่วนการประกาศของโปรแกรมเดิม เราสามารถเริ่มต้นปรับใหม่(revised) โปรแกรมหลักด้วยข้อความสั่งเรียกกระบวนงาน

Instruct;

ส่วนที่เหลือของโปรแกรมหลักประกอบด้วยข้อความสั่งกระทำการที่แสดงก่อนหน้าแล้ว รูป 3.21 แสดงให้เห็นเอาต์พุตซึ่งแสดงผลโดยการเรียกกระบวนงาน Instruct

---

```
procedure Instruct;
{Displays instructions to a user of program Circle}

begin {Instruct}
   WriteLn ('This program computes the area');
   WriteLn ('and cricumference of a circle.');
   WriteLn;
   WriteLn ('To use this program, enter the radius of');
```

```
    WriteLn ('the circle after the prompt : Enter radius>');
    WriteLn;
    WriteLn ('The circumference will be computed in the');
    WriteLn ('sam units of measurement as the radius'.);
    WriteLn ('The area will be computed in the same', ' units squared');
    WriteLn
end; {Instruct}
```

รูป 3.20 กระบวนงาน Instruct

แบบฝึกหัด 3.5

1. การใช้โปรซีเดอร์พารามิเตอร์ทำให้เป็นไปได้ที่จะเขียนโปรแกรมที่มีขนาดใหญ่กว่าและเป็นประโยชน์มากกว่าได้อย่างไร

2. ทำไมพารามิเตอร์สำหรับโปรซีเดอร์เอาต์พุต WriteLn จึงเป็นโปรซีเดอร์อินพุต และพารามิเตอร์สำหรับอินพุตโปรซีเดอร์ ReadLn จึงเป็นโปรซีเดอร์เอาต์พุต

เขียนโปรแกรม

1. จงแสดงให้เห็นโปรแกรมปรับปรุงใหม่ Circle ที่มีการเรียก Instruct

```
This program computes the area
and circumference of a circle.

To use this program. enter the radius of
the circle after the prompt : Enter radius>

The circumference will be computed in the
same units of measurement as the radius.
The area will be computed in the same units squared
```

รูป 3.21 แสดงผลเอาต์พุตโดยโปรซีเดอร์ Instruct

3.6 ข้อผิดพลาดร่วมของการเขียนโปรแกรมร่วม (Common Programming Errors)

ข้อควรจำคือ ให้ประกาศกระบวนงานทุกข้อที่ใช้ในโปรแกรม การประกาศกระบวนงานต้องอยู่ก่อนการเรียกกระบวนงาน และปกติพบในส่วนการประกาศของโปรแกรมหลังการประกาศตัวแปร ถ้ากระบวนงาน A เรียกกระบวนงาน B เราต้องประกาศกระบวนงาน B เป็นอันดับแรก

ข้อผิดพลาดวากยสัมพันธ์ หรือข้อผิดพลาดเวลาดำเนินงานสามารถเกิดขึ้นได้เมื่อเราใช้ฟังก์ชันซึ่งนิยามแล้วของ Pascal เพราะฉะนั้นเราต้องมั่นใจว่าอาร์กิวเมนต์ของทุกฟังก์ชันมีแบบชนิดข้อมูลถูกต้อง

ตัวอย่างเช่น อาร์กิวเมนต์สำหรับฟังก์ชัน Round และ Trune ควรจะเป็นชนิด Real ถ้าอาร์กิวเมนต์ของฟังก์ชัน Sqrt หรือ Ln เป็นค่าลบ จะเกิดข้อผิดพลาดเวลาดำเนินการ

ข้อสรุปของตัวสร้าง Pascal ตัวใหม่ (Summary of New Pascal Constructs)

Construct	Effect
Procedure Declaration procedure Display; {Prints 3 lines} const Star = ' * ' ; begin {Display} WriteLn (Star); WriteLn (Star); WriteLn (Star) end; {Display}	กระบวนงาน Display ถูกนิยาม และเมื่อถูกเรียกจะพิมพ์สามบรรทัด แต่ละบรรทัดมีหนึ่ง * ค่าคงตัว star จะถูกนิยามเฉพาะเมื่อ Display กำลังกระทำการ
Procedure Call Statement Display	กระบวนงาน Display ถูกเรียกและเริ่มต้นกระทำการ
Function Designator Z := Sqrt (X + Y)	ฟังก์ชัน Sqrt คำนวณรากที่สองของนิพจน์ X + Y จากนั้นผลลัพธ์ถูกกำหนดให้กับ Z

แบบฝึกหัด Quick - Check

1. การพัฒนาโปรแกรมจากเอกสารของมัน หมายความว่า ข้อความสั่งทุกคำสั่งในโปรแกรมต้องมีคอมเมนต์ ถูกหรือผิด

2. หลักของการนำกลับมาใช้ใหม่ กล่าวว่า ทุกกระบวนงานในโปรแกรมต้องใช้มากกว่าหนึ่งครั้ง ถูกหรือผิด

3. จงเขียนสมการข้างล่างนี้ให้เป็นข้อความสั่ง Pascal โดยใช้ฟังก์ชัน EXP, LN และ Sqr

$y = (e^n \ln b)^2$

4. การออกแบบจากบนลงล่าง หมายความว่า ขั้นแรก ให้เขียนหัวข้อการประกาศ Pascal จากนั้นเขียนตัวโปรแกรม ถูกหรือผิด

5. กระบวนงานแต่ละชุดถูกกระทำการในอันดับซึ่งมันถูกประกาศในโปรแกรมหลัก ถูกหรือผิด

6. วากยสัมพันธ์อะไร ซึ่งนำมาใช้ในโปรแกรม Pascal เพื่อเรียกกระบวนงาน

7. จงเขียนรายการอันดับของการประกาศในโปรแกรม Pascal มาตรฐาน

8. โปรซีเดอร์พารามิเตอร์หมายถึงอะไร

9. จงอธิบายว่าผังโครงสร้าง (structure chart) แตกต่างจากอัลกอริทึม (algorithm) อย่างไร

10. กระบวนงานสามารถมีอินพุต เอาต์พุต หรือมีทั้งคู่ ถูกหรือผิด

11. กระบวนงานข้างล่างนี้ทำอะไร

```
procedure Nonsense;
begin {Nonsense}
   WriteLn ('******');
   WriteLn ('*    *');
   WriteLn ('******')
end; {Nonsense}
```

12. ตัวโปรแกรมข้างล่างนี้ทำอะไร

```
begin
   Nonsense;
   Nonsense;
   Nonsense
end.
```

Programming Projects

1. จงเขียนกระบวนงานสองชุด ชุดแรกแสดงผลรูปสามเหลี่ยม และอีกชุดหนึ่งแสดงผลรูปสี่เหลี่ยมผืนผ้า โดยใช้ภาวะข้อความ (text mode) จากนั้นใช้กระบวนงานเหล่านี้เขียนโปรแกรม Pascal ที่เสร็จสมบูรณ์จากเค้าโครงข้างล่างนี้

```
program StackHonses;
begin
   {1. Draw triangle.}
   {2. Draw rectangle.}
   {3. Print 2 blank lines.}
   {4. Draw triangle.}
   {5. Draw rectangle.}
end;
```

2. ใส่กระบวนงานจากรูป 3.16 ในแบบฝึกหัดข้อ 1 ใช้กระบวนงานเหล่านี้ในโปรแกรมเพื่อวาดรูปเรือเหาะ (rocket ship) ซึ่งเป็นรูปสามเหลี่ยมเหนือรูปสี่เหลี่ยมผืนผ้าเหนือเส้นตัด, รูปผู้ชาย (รูปวงกลมเหนือสี่เหลี่ยมผืนผ้าเหนือเส้นตัด) และรูปผู้หญิงยืนบนศีรษะของรูปผู้ชาย

จงเขียนกระบวนงาน Skip5Lines และเรียกกระบวนงานชุดนี้เพื่อใส่บรรทัดว่าง 5 บรรทัดระหว่างแต่ละรูป

3. จงเขียนโปรแกรมคอมพิวเตอร์คำนวณระยะเวลาของการบินของกระสุน (projectile's flight) และความสูงของมันเหนือพื้นดินเมื่อมันถึงเป้าหมาย (target) เป็นส่วนของผลเฉลย ให้เขียนและเรียกกระบวนงานซึ่งแสดงคำสั่งให้กับผู้ใช้โปรแกรม

Problem constant

G = 32.17 (gravitatin constant)

Problem input

Theta : Real {input - angle (radians) of elevation}

Distance: Real {input - distance (ft) to target}

Velocity : Real {input - projectile velocity (ft/sec)}

Problem output

Time : Real {output - time (sec) of flight}

Height : Real {output - height at impact}

Relevant formulas

time = distance / (velocity x cosTheta)

height = velocity x sintheta x time - (g x time^2) / 2

4. มีนักวิ่งแข่ง 4 คน ซึ่งสามารถวิ่งระยะทางหนึ่งไมล์ครบถ้วน จงเขียนโปรแกรมอ่านข้อมูลเวลาของนักวิ่งแข่งแต่ละคน ซึ่งมีหน่วยเป็นนาที (Minutes) และวินาที (Seconds) จากนั้นให้คำนวณและพิมพ์ความเร็วมีหน่วยเป็นฟุตต่อวินาที (FPS) และเป็นเมตรต่อวินาที (MPS)

(**ข้อแนะนำ** หนึ่งไมล์เท่ากับ 5280 ฟุต และหนึ่งกิโลเมตรเท่ากับ 3282 ฟุต)

จงเขียนและเรียกกระบวนงานซึ่งแสดงคำสั่งให้ผู้ใช้โปรแกรม

Minutes	Seconds
3	52.83
3	59.83
4	00.03
4	16.22