บทที่ 3
โครงสร้างทางภาษา

โลกปัจจุบัน ได้นำคอมพิวเตอร์เข้ามาช่วยแก้ปัญหาต่าง ๆ ในระบบบริหารจัดการเกือบทุกด้านของมนุษย์ และสิ่งสำคัญที่ต้องนำมาใช้ในการแก้ปัญหาในระบบคอมพิวเตอร์ คือ การเลือกขั้นตอนวิธี (algorithm) ที่เหมาะสม เพื่อให้คอมพิวเตอร์อำนวยประโยชน์ต่อการดำเนินงานมากที่สุด โดยนักวิเคราะห์ระบบจัดการออกแบบและนำมาใช้ในการเขียนโปรแกรมปฏิบัติงาน ซึ่งผู้ออกแบบอาจเลือกขั้นตอนวิธีที่ง่ายแต่เสียค่าใช้จ่ายสูง หรืออาจสลับซับซ้อนยากต่อการเข้าใจแต่เสียค่าใช้จ่ายต่ำ หรือเลือกขั้นตอนวิธีที่สามารถทำให้คอมพิวเตอร์สามารถประมวลผลได้อย่างรวดเร็ว อย่างไรก็ตามจะเห็นได้ว่าแต่ละขั้นตอนวิธีอาจมีรูปแบบที่แตกต่างกันตามแนวทางออกแบบจากผู้เชี่ยวชาญหลายคน แต่ทุกวิธีการล้วนมีเป้าหมายเดียวกันเพียงแต่อาจมีข้อดี ข้อเสีย และข้อจำกัดที่แตกต่างกันได้

นอกจากการเลือกขั้นตอนวิธีที่เหมาะสมแล้วยังต้องคำนึงถึงองค์ประกอบอีกสิ่งหนึ่งที่มีความสำคัญ คือโครงสร้างข้อมูลที่นำมาใช้แก้ปัญหาการจัดเก็บข้อมูลและการจัดการ ซึ่งต้องสอดคล้องกับขั้นตอนวิธี เพราะความสอดคล้องดังกล่าวมีผลทำให้คอมพิวเตอร์ประมวลผลได้เร็วขึ้นหรือช้าลงได้ ดังนั้นการศึกษาแนวทางการแก้ปัญหาบนระบบคอมพิวเตอร์นั้น นอกจากจะศึกษาเกี่ยวกับโครงสร้างการจัดเก็บข้อมูลแล้วยังต้องศึกษาเกี่ยวกับขั้นตอนวิธีในแต่ละโครงสร้างข้อมูลด้วย และเพื่อให้สามารถนำไปใช้งานร่วมกันได้ สำหรับโครงสร้างข้อมูลบางรูปแบบจะต้องมีภาษาคอมพิวเตอร์ที่สามารถกำหนดและสร้างเพื่อใช้งานโครงสร้างตามความต้องการ และเพื่อให้เกิดความเข้าใจที่ตรงกันในโครงสร้างของภาษาจึงขอใช้โปรแกรมภาษาปาสกาล และโปรแกรมบอร์-แลนด์เดลไฟเป็นต้นแบบในการพัฒนาโปรแกรมสำหรับกำหนดโครงสร้างข้อมูล ซึ่งทั้ง 2 โปรแกรมมีโครงสร้างทางภาษาเหมือนกัน แต่ขอยึดโปรแกรมภาษาปาสกาลเป็นหลักและเรียนรู้พอสังเขปให้เพียงพอกับการเรียนรู้ด้านโครงสร้างข้อมูลและขั้นตอนวิธีของแต่ละโครงสร้าง เพื่อนำไปใช้ในการเขียนโปรแกรมต่อไป

3.1 โครงสร้างของภาษาคอมพิวเตอร์

สำหรับการเขียนด้วยโปรแกรมภาษาปาสกาลและโปรแกรมบอร์แลนด์เดลไฟ เมื่อประมวลผลอาจมีประสิทธิภาพแตกต่างกันได้ เนื่องจากดำเนินการผ่านระบบปฏิบัติการที่ต่างกัน แต่อย่างไรก็ตาม ทั้งสองภาษามีโครงสร้างทางภาษาเหมือนกัน คือ จัดแบ่งส่วนโปรแกรมแบบโครงสร้างบล็อก (block structure) กล่าวคือโครงสร้างโปรแกรมถูกแบ่งออกเป็นบล็อกในลักษณะโปรแกรมย่อย ซึ่งทางโครงสร้างภาษาเรียกกันว่า มอดูล (module) โดยมีรูปแบบการนำเสนอ 2 แบบคือ กระบวนงาน และฟังก์ชัน ซึ่งถูกเขียนขึ้นตามขั้นตอนวิธีที่ได้ออกแบบไว้ และแต่ละบล็อกยังสามารถแบ่งออกเป็นบล็อกย่อยต่อไปได้อีกโดยไม่มีขีดจำกัด นอกจากนี้ตัวแปรต่าง ๆ ที่กำหนดไว้ในแต่ละบล็อกจะไม่เกิดผลกระทบต่อตัวแปรที่ถูกกำหนดภายในโปรแกรมเดียวกัน แม้แต่นักเขียนโปรแกรมนั้นจะได้นิยามชื่อตัวแปรเหมือนกันก็ตาม ดังนั้นการเรียนรู้โครงสร้างของภาษาคอมพิวเตอร์ที่จะนำมาพัฒนาจึงมีความสำคัญ

3.1.1 ชนิดข้อมูล

ชนิดข้อมูล (data type) ของโปรแกรมภาษา ถูกนิยามเพื่อกำหนดลักษณะของข้อมูล โดยชนิดข้อมูลชนิดหนึ่งออกแบบให้สามารถบันทึกข้อมูลและใช้ประมวลผลข้อมูลแตกต่างกันไปตามการนิยาม โดยทั่วไปชนิดข้อมูลบนโปรแกรมคอมพิวเตอร์สามารถจำแนกได้ 2 ประเภท คือ ข้อมูลแบบง่าย และข้อมูลแบบโครงสร้าง (กฤษดา กรุดทอง, 2536, หน้า 3-5) ซึ่งมีรูปแบบการนิยามดังนี้

Type ชื่อตัวแปร = ค่าข้อมูล;

Var ชื่อตัวแปร : ชนิดข้อมูล;

3.1.1.1 ข้อมูลแบบง่าย (simple data type) คือชนิดข้อมูลที่ใช้กำหนดให้บันทึกค่าได้ครั้งละหนึ่งค่าและให้สามารถประมวลผลได้ด้วยตัวดำเนินการหนึ่งชุด ประกอบด้วยชนิดข้อมูลชนิดจำนวนเต็ม จำนวนจริง อักขระ และตรรกะ

(1) จำนวนเต็ม นิยามชนิดข้อมูลด้วยคำสำคัญ integer หรือ int ซึ่งตัวอย่างค่าข้อมูลที่เป็นไปได้ อาทิเช่น 1234 , 100 , 50 , 0 , -10 หรือ +1000 เป็นต้น และสามารถนิยามในโปรแกรมได้ดังนี้Var x,y : integer; //หรือ

Var x,y : int;

(2) จำนวนจริง นิยามชนิดข้อมูลด้วยคำสำคัญ real หรือ float ซึ่งตัวอย่างค่าข้อมูลที่เป็นไปได้ อาทิเช่น 1.2 , -18.35 , 14.99 หรือ 4.1E+3 เป็นต้น และสามารถนิยามในโปรแกรมได้ดังนี้ Var x,y : real; //หรือ

Var x,y : float;

(3) อักขระ นิยามชนิดข้อมูลด้วยคำสำคัญ char ซึ่งตัวอย่างค่าข้อมูลที่เป็นไปได้ อาทิเช่น ‘A’ , ‘B’, ‘C’ , ‘5’ , ‘?’ หรือ ‘%’ เป็นต้น และสามารถนิยามในโปรแกรมได้ดังนี้ Var x,y : char ;

(4) ตรรกะ นิยามชนิดข้อมูลด้วยคำสำคัญ boolean ซึ่งค่าข้อมูลที่เป็นไปได้มี 2 ค่าเท่านั้น คือ ค่าจริง (true) และค่าเท็จ (false) และนิยามในโปรแกรมได้ดังนี้

Var x,y : boolean;

3.1.1.2 ข้อมูลแบบโครงสร้าง (structured data type) คือชนิดข้อมูลที่กำหนดให้บันทึกค่าได้ครั้งละหนึ่งชุด และสามารถประมวลผลได้ด้วยกรรมวิธีชุดหนึ่ง ซึ่งค่าข้อมูลที่เป็นไปได้ขึ้นอยู่กับการนิยามชนิดข้อมูลภายในและค่าข้อมูลที่กำหนดไว้เบื้องต้น ได้แก่

(1) แจงนับ (enumerated) ซึ่งค่าข้อมูลที่เป็นไปได้ อาทิเช่น color := bule; weekday := Sat; หากนิยามในโปรแกรมไว้ดังนี้

Type color = (red, blue, green, yellow);

weekday = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

(2) แถวลำดับ (array) ซึ่งค่าข้อมูลที่เป็นไปได้ อาทิเช่น A[1] := 1; A[2] := 3; …; A[100] := 40; หากนิยามในโปรแกรมไว้ดังนี้

Var A : Array[1..100] of integer;

(3) สายอักขระ (string) ซึ่งค่าข้อมูลที่เป็นไปได้ อาทิเช่น x:='Pascal'; y:='Data Structure'; หรือ z:='Rajabhat Lampang; เป็นต้น และสามารถนิยามในโปรแกรมได้ดังนี้ Var x,y,z : string;

(4) ระเบียน (record) ซึ่งค่าข้อมูลที่เป็นไปได้ อาทิเช่น person.name := 'น.ส.ไพริน เพชรเทียนชัย'; person.sex := 'หญิง'; หากนิยามในโปรแกรมไว้ดังนี้

```
Type person = record
                name : string;
                sex : string;
              End;
```

3.1.2 ชื่อ

ชื่อ (identifier) เป็นคำที่กำหนดเพื่อใช้อ้างอิงชื่อโปรแกรม กระบวนงาน ฟังก์ชัน ตัวแปร หรือคำสั่งที่กำหนดขึ้นไว้แล้วใช้เป็นมาตรฐานสำหรับงานเฉพาะอย่างเช่น sin, read, write เป็นต้น สำหรับชื่อที่ตั้งขึ้นเองนั้น จะประกอบไปด้วย ตัวอักษร ตัวเลข และขีดล่าง (_) เท่านั้น และต้องเริ่มต้นด้วยอักษร เช่น X, matrix , id_no, name, cid555, address, persons_count_1 เป็นต้น

ตัวอักษรที่ใช้ในการกำหนดชื่อจะใช้ตัวพิมพ์เล็กหรือตัวพิมพ์ใหญ่ถือว่าไม่แตกต่างกัน เช่น TREE, Tree และ tree สามชื่อนี้ถือว่าเหมือนกัน

3.1.3 ตัวแปร

ตัวแปร (variable) คือข้อมูลที่ถูกกำหนดขึ้นใช้งานในโปรแกรม หรือแต่ละโปรแกรมย่อยเพื่อชี้ถึงค่าที่จัดเก็บ ตำแหน่งที่จัดการ ผลลัพธ์ที่ได้จากการประมวลผล ซึ่งตัวแปรเหล่านี้สามารถกำหนดเพื่อรับข้อมูลเข้า (input) และ/หรือประมวลผลหรือกำหนดเพื่อนำผลลัพธ์ออก (output) ด้วยเหตุนี้จึงมีการกำหนดชนิดของตัวแปรที่ใช้ในโปรแกรมย่อยได้ 5 ประเภท คือ ตัวแปรเสริมรับเข้า ตัวแปรเสริมนำออก ตัวแปรเสริมรับเข้า-นำออก ตัวแปรเฉพาะที่ และตัวแปรส่วนกลาง (Robert, 1989, p.46-47)

3.1.3.1 ตัวแปรเสริมรับเข้า (input parameter) คือ ตัวแปรที่ถูกกำหนดเพื่อใช้ในโปรแกรมย่อยเท่านั้นแต่จะไม่มีผลทำให้เปลี่ยนแปลงค่าภายในโปรแกรมย่อย สำหรับภาษาปาสกาลตัวแปรเสริมรับเข้ามักถูกใช้กำหนดเป็นค่าข้อมูลเสมอเพื่อส่งค่าข้อมูลดังกล่าวไปประมวลผลในโปรแกรมย่อยนั้น

3.1.3.2 ตัวแปรเสริมนำออก (output parameter) คือ ตัวแปรที่บรรจุผลลัพธ์ของการประมวลผลในโปรแกรมย่อยเพื่อส่งค่าผลลัพธ์กลับมายังตำแหน่งที่ถูกเรียกใช้ ดังนั้นตัวแปรประเภทนี้จะสามารถเปลี่ยนแปลงค่าได้ในโปรแกรมย่อย

3.1.3.3 ตัวแปรเสริมรับเข้าและนำออก (in-out parameter) คือ ตัวแปรที่กำหนดขึ้นเพื่อให้สามารถดำเนินงานทั้งรับเข้าและนำออก ซึ่งหมายถึงตัวแปรที่ถูกกำหนดให้ส่งค่าข้อมูลไปยังโปรแกรมย่อยและสามารถประมวลผลทำให้ค่าข้อมูลเปลี่ยนแปลงเมื่อถูกส่งกลับมาได้

3.1.3.4 ตัวแปรเฉพาะที่ (local variable) คือ ตัวแปรที่ถูกกำหนดขึ้นเพื่อดำเนินงานโดยเฉพาะในโปรแกรมย่อยหนึ่ง โดยค่าข้อมูลในตัวแปรดังกล่าวจะไม่มีผลกระทบต่อตัวแปรใด ๆ ในระบบโปรแกรมโดยรวม แม้ชื่อของตัวแปรจะเหมือนกันก็ตาม

3.1.3.5 ตัวแปรส่วนกลาง (global variable) คือ ตัวแปรที่ถูกกำหนดขึ้นเพื่อดำเนินงานได้ทั้งระบบโปรแกรม โดยค่าข้อมูลของตัวแปรดังกล่าวสามารถถูกใช้และประมวลผลทำให้เกิดการเปลี่ยนแปลงในทุกที่ไม่ว่าจะเป็นโปรแกรมย่อยหรือโปรแกรมหลักก็ตาม ดังนั้นนักเขียนโปรแกรมที่ดีจึงไม่ควรใช้ตัวแปรส่วนกลางในการพัฒนาโปรแกรมมากนัก เพราะผลการทำงานมักเกี่ยวข้องกับค่าใช้จ่ายที่สูงทั้งในเรื่องของทรัพยากรหน่วยความจำ และการตรวจสอบ ข้อผิดพลาดของโปรแกรมโดยรวม

3.1.4 ตัวดำเนินการ

ตัวดำเนินการ (operator) ที่ถูกกำหนดในโครงสร้างภาษาตามขั้นตอนวิธี หรือนิพจน์ใด ๆ เช่น การบวก ลบ คูณ หาร และ/หรือ ปฏิเสธ ตรวจสอบค่ามากกว่า น้อยกว่า หรือเท่ากับ เป็นต้น ซึ่งการดำเนินการกับนิพจน์หมายถึง ค่าคงที่ ตัวแปร หรือฟังก์ชัน ได้ถูกกระทำโดยตัวดำเนินการหนึ่งนั้น ดังนั้นจึงได้จัดกลุ่มของตัวดำเนินการแบ่งได้ 3 ชนิด ตามการกระทำกับ

นิพจน์ที่มักพบเห็นและถูกใช้บ่อยในระบบคอมพิวเตอร์ ได้แก่ ตัวดำเนินการทางคณิตศาสตร์ ตัวดำเนิน-การเปรียบเทียบ และตัวดำเนินการตรรกะ

3.1.4.1 ตัวดำเนินการคณิตศาสตร์ (arithmetic operator) คือ ตัวดำเนิน-การที่ใช้ในการคำนวณค่าต่าง ๆ ทางคณิตศาสตร์ โดยตัวดำเนินการชนิดนี้จะกระทำกับชนิดข้อมูลชนิดตัวเลข คือ จำนวนจริงหรือจำนวนเต็ม ผลลัพธ์การกระทำโดยตัวดำเนินการทางคณิตศาสตร์จะได้ค่าชนิดข้อมูลชนิดตัวเลขเช่นกัน สำหรับการใช้ตัวดำเนินการทางคณิตศาสตร์จะต้องกระทำกับค่า 2 ค่า ซึ่งจะอยู่สองข้างของตัวดำเนินการ และเรียกค่า 2 ค่านี้ว่า ตัวโอเปอแรนด์ (operand) (วุฒิชัย สิทธิมาลากร และอธิคม ไช้ศรีทอง, 2544, หน้า 55) สามารถแสดงความหมายและตัวอย่างของตัวดำเนินการดังตารางที่ 3.1

ตารางที่ 3.1 ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการ	ความหมาย	ตัวอย่าง
-	นิเสธ	เช่น –X
*	การคูณ	เช่น X*Y
/	การหารจำนวนจริง	เช่น X/Y
DIV	การหารเอาส่วน	เช่น X DIV Y
MOD	การหารเอาเศษ	เช่น X MOD Y
+	การบวก, การลบ	เช่น X+Y
-	การลบ	เช่น X-Y

3.1.4.2 ตัวดำเนินการเปรียบเทียบ (comparison operator) คือ ตัวดำเนิน-การที่ใช้สำหรับการเปรียบเทียบข้อมูล เช่น ค่าของชนิดข้อมูลชนิดจำนวนจริงและจำนวนเต็ม หรือค่าของชนิดข้อมูลชนิดอักขระหรือสายอักขระ เป็นต้น สำหรับผลลัพธ์ที่ได้จากการเปรียบเทียบจะมีค่าเป็นจริงหรือเท็จ เท่านั้น สามารถแสดงตัวดำเนินการดังตารางที่ 3.2

ตารางที่ 3.2 ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการ	ความหมาย	ตัวอย่าง
=	เท่ากับ	เช่น X = Y
<>	ไม่เท่ากับ	เช่น X <> Y
<	น้อยกว่า	เช่น X < Y
>	มากกว่า	เช่น X > Y
<=	น้อยกว่าหรือเท่ากับ	เช่น X <= Y
>=	มากกว่าหรือเท่ากับ	เช่น X >= Y

3.1.4.3 ตัวดำเนินการตรรกะ (logical operator) คือ ตัวดำเนินการที่ใช้สำหรับการตรวจสอบข้อเท็จจริงสำหรับการเชื่อมนิพจน์ที่มักต้องมากกว่า 1 นิพจน์เป็นต้นไป ยกเว้นตัวดำเนินการ NOT ซึ่งผลลัพธ์ที่ได้จากนิพจน์ที่ถูกเชื่อมโดยตัวดำเนินการตรรกะจะมีค่าเป็นจริงหรือเท็จ ขึ้นอยู่กับตัวดำเนินการ สามารถแสดงตัวดำเนินการดังตารางที่ 3.3

ตารางที่ 3.3 ตัวดำเนินการตรรกะ

ตัวดำเนินการ	ความหมาย	ตัวอย่าง
NOT	เปลี่ยนนิพจน์เป็นตรงกันข้าม	เช่น NOT(2=3)
AND	ผลลัพธ์เป็นจริงเมื่อนิพจน์ทั้งสองเป็นจริงทั้งคู่	เช่น (X <2) AND (Y<1)
OR	ผลลัพธ์เป็นจริงเมื่อนิพจน์ทั้งสองเป็นจริง อย่างน้อย 1 นิพจน์	เช่น (X >1) OR (Y>2)
XOR	ผลลัพธ์เป็นเท็จเมื่อนิพจน์ทั้งสองมีค่าตรรกะเหมือนกัน	เช่น (X >1) XOR (Y>2)

3.1.5 ลำดับความสำคัญของตัวดำเนินการ

โดยปกติการทำงานของตัวดำเนินการจะเริ่มทำจากซ้ายไปขวา แต่บางกรณีที่ในนิพจน์หนึ่งอาจประกอบด้วยตัวดำเนินการหลายตัว ดังนั้นการประมวลผลนิพจน์จะพิจารณาตามลำดับความสำคัญของตัวดำเนินการก่อน โดยตรวจสอบในนิพจน์จากซ้ายไปขวาเช่นเดิม อย่างไรก็ตามลำดับความสำคัญของตัวดำเนินการตรรกะบางตัว เช่น AND OR และ XOR มักใช้

สำหรับประโยคที่มีหลายนิพจน์ ดังนั้นเพื่อประมวลได้ง่ายขึ้นทุกนิพจน์ควรกำหนดขอบเขตด้วยวงเล็บเสมอ อาทิเช่น

(2 + 4 / 2 > 4) **OR** (3 + 1 = 4)

หรือ (3 * 2 / 2 = 3) **AND** (2 / 2 +1 > 2) เป็นต้น

พื้นฐานลำดับความสำคัญของตัวดำเนินการสามารถพิจารณาการดำเนินการ 3 รูปแบบ โดยเรียงลำดับจากความสำคัญสูงสุดไปถึงต่ำสุดดังตารางที่ 3.4 อย่างไรก็ตาม มีข้อยกเว้นสำหรับทุกตัวดำเนินการ นั่นคือ หากพบเครื่องหมายวงเล็บ ‘( )’ ให้ประมวลผลภายใต้วงเล็บก่อนเสมอ

ตารางที่ 3.4 ลำดับความสำคัญของตัวดำเนินการ

ตัวดำเนินการ	ความสำคัญ
NOT	อันดับ 1 (สูงสุด)
* / DIV MOD AND	อันดับ 2
+ - OR XOR	อันดับ 3
= <> > < >= <=	อันดับ 4 (ต่ำสุด)

ที่มา (กมลมาศ กำจรกิจการ, 2543, หน้า 85)

เพื่อให้เกิดความเข้าใจในการจัดลำดับความสำคัญของตัวดำเนินการซึ่งมักพบเห็นบ่อยครั้งในปัจจุบันว่าในหนึ่งนิพจน์ทางคณิตศาสตร์มีการใช้ตัวดำเนินการมากกว่า 1 ตัว และในบางครั้งยังมีหลายนิพจน์ร่วมกัน ดังนั้นจึงขอยกตัวอย่างการประมวลผลกับนิพจน์ที่กำหนดให้ และการเขียนนิพจน์ที่ถูกต้องเพื่อไม่ให้เกิดการผิดพลาดในการประมวลผล ดังตัวอย่างที่ 3.1 และตัวอย่างที่ 3.2 (วุฒิชัย สิทธิมาลากร และอธิคม ไช้ศรีทอง, 2544, หน้า 75-76)

<u>ตัวอย่างที่</u> 3.1 พิจารณานิพจน์ต่อไปนี้ หากกำหนดให้ตัวแปรแต่ละตัวมีชนิดข้อมูลชนิดตัวเลขจำนวนจริง

(1) X < Y + Z มีความหมายเช่นเดียวกับ X < (Y+Z) เนื่องจากตัวดำเนินการ + มีค่าความสำคัญมากกว่าตัวดำเนินการเปรียบเทียบ <

(2) Y <= X AND X <= Z เป็นนิพจน์ตรรกะที่ไม่ถูกต้อง จะเกิดข้อผิดพลาดขึ้นเมื่อประมวลผล เนื่องจากตัวดำเนินการ AND มีค่าความสำคัญมากกว่าตัวดำเนินการเปรียบเทียบ <= ซึ่งคอมไพเลอร์จะไม่สามารถนำเอาเลขจำนวนจริง X มา AND กันได้ อย่างไรก็ตาม ปกติมักแก้ไขนิพจน์นี้ได้ถูกต้องโดยใส่เครื่องหมายวงเล็บ ดังนี้ (Y <= X) AND (X <= Z)

ตัวอย่างที่ 3.2 พิจารณานิพจน์ต่อไปนี้ หากกำหนดให้ X มีค่า 3, Y มีค่า 4, Z มีค่า 2 และ Test เป็นชนิดข้อมูลตรรกะมีค่าเท็จ

(1) (Z > Y) OR (Z > Y)	ผลลัพธ์เป็นค่าเท็จ
(2) (X > Z) AND (Y>Z)	ผลลัพธ์เป็นค่าจริง
(3) NOT Test	ผลลัพธ์เป็นค่าจริง
(4) (X + Y/Z) <= 2	ผลลัพธ์เป็นค่าเท็จ
(5) (X = 2) OR (X = 3)	ผลลัพธ์เป็นค่าจริง
(6) (X <= Z) OR (X = Y)	ผลลัพธ์เป็นค่าจริง
(7) (Z < X) AND (X < Y)	ผลลัพธ์เป็นค่าเท็จ
(8) (NOT Test) OR ((Y + Z) >= (X – Z))	ผลลัพธ์เป็นค่าจริง
(9) NOT (Test OR ((Y + Z) >= (X – Z)))	ผลลัพธ์เป็นค่าเท็จ
(10) Test XOR NOT ((Y-Z) <> 1)	ผลลัพธ์เป็นค่าเท็จ

ทดสอบ 3.1 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถแสดงผลลัพธ์ทางจอภาพ ดังตัวอย่างที่ 3.2

3.3 คำสั่งพื้นฐาน

คำสั่ง (statement) ในโปรแกรมภาษาปาสกาลและโปรแกรมบอร์แลนด์เดลไฟ แต่ละคำสั่งต้องปิดท้ายด้วยเครื่องหมายอัฒภาค (;) โดยคำสั่งถูกกำหนดให้ดำเนินงานได้ 2 รูปแบบ คือ คำสั่งเชิงเดี่ยว และคำสั่งเชิงโครงสร้าง

3.3.1 คำสั่งเชิงเดี่ยว

คำสั่งเชิงเดี่ยว (simple statement) เป็นคำสั่งประโยคเดี่ยว โดยไม่มีคำสั่งอื่นร่วม ได้แก่ คำสั่งกำหนดค่า คำสั่งอ่านข้อมูล คำสั่งแสดงผล คำสั่งไปที่บรรทัด คำสั่งโพรซีเยอร์ และคำสั่งว่าง

3.3.1.1 คำสั่งกำหนดค่า (assignment statement) คือ คำสั่งที่กำหนดค่าให้กับตัวแปร ซึ่งนิยามได้ดังรูปแบบที่ 3.1 และแสดงการใช้งานดังตัวอย่างที่ 3.3

รูปแบบที่ 3.1

```
< ตัวแปร >  := < นิพจน์ >;
```

ตัวอย่างที่ 3.3 การใช้งานคำสั่งกำหนดค่า

```
X  :=  100;
Y  :=  X+1;
Z  := 0.5 * Base * High;
```

ซึ่งในกรณีนี้ สามารถกำหนดให้การแสดงในคอมโพเนนท์หนึ่ง ๆ ได้ด้วยคำสั่งกำหนดค่า เช่น ดังตัวอย่างที่ 3.4

ตัวอย่างที่ 3.4 การใช้งานคำสั่งกำหนดค่าลงในคอมโพเนนท์

```
Label1.Caption := 'วิชา 5671301 โปรแกรมภาษาภาพ';
Edit1.text      := 'วิชา 5672303 พัฒนาซอฟต์แวร์เชิงรับ-ให้บริการ'
Edit2.text      := INTTOSTR(5000);
X               := Edit1.text;              //กรณีตัวแปร X เป็นชนิดตัวอักษร
B               := STRTOINT(Edit2.text);  //กรณีตัวแปร B เป็นชนิดตัวเลข
```

ซึ่งการกำหนดค่าดังกล่าวจึงเหมือนกับการใช้งานคำสั่ง Read และ Write ของโปรแกรมบอร์แลนด์เดลไฟผ่านคอมโพเนนท์เมื่อเกิดเหตุการณ์หนึ่ง ๆ หรือมีการเปลี่ยนสถานะของคอมโพเนนท์

ตัวอย่างที่ 3.6 ขั้นตอนวิธีสำหรับการคำนวณค่า X^Y

1. แสดงข้อความทางจอภาพ ‘รับค่าเลขจำนวนเต็มและค่ายกกำลัง ‘;
2. รับค่า X และ Y
3. เมื่อคลิกปุ่มผลลัพธ์ ให้คำนวณจากสูตร Exp(ln(X)*Y);
4. แสดงผลลัพธ์ที่ได้ทางจอภาพ

ผลลัพธ์ทางจอภาพของโปรแกรม (เมื่อส่งค่า X=2 และ Y=8) คือ

รับค่าเลขจำนวนเต็มและค่ายกกำลัง : 2 8

2.00^8.00 = 256.00

ทดสอบ 3.2 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถแสดงผลลัพธ์ทางจอภาพ ดังตัวอย่างที่ 3.6

3.3.1.2 โพซีเยอร์ (**procedure** statement) คือ คำสั่งที่เรียกใช้โพรซีเยอร์หรือฟังก์ชัน ซึ่งเป็นส่วนหนึ่งของโปรแกรมย่อยที่สามารถสร้างขึ้นได้เองในโปรแกรม ลักษณะของคำสั่งประกอบด้วยชื่อโพรซีเยอร์หรือฟังก์ชัน ตามด้วยรายชื่อตัวแปรหรือค่าข้อมูลที่ส่งไปประมวลผล แต่ถ้าโพรซีเยอร์หรือฟังก์ชันใดไม่ได้กำหนดตัวแปรสำหรับนำเข้าข้อมูลหรือส่งผลลัพธ์ออกแล้ว คำสั่งโพรซีเยอร์เพื่อเรียกใช้งานต้องไม่มีรายชื่อตัวแปรเหล่านั้นเช่นกัน ซึ่งนิยามได้ดังรูปแบบที่ 3.3 และแสดงการใช้งานดังตัวอย่างที่ 3.7

รูปแบบที่ 3.3

```
ชื่อโพรซีเยอร์ (< ค่าตัวแปรนำเข้าข้อมูลหรือส่งผลลัพธ์ออก >);
ชื่อตัวแปรรับค่า := ชื่อฟังก์ชัน (< ค่าตัวแปรประมวลผล >);
```

ตัวอย่างที่ 3.7 การใช้งานคำสั่งโพรซีเยอร์

```
Sort(List);                                  {วิธีการเรียกใช้โพรซีเยอร์}
X := Area(R,H);                              {เรียกใช้ฟังก์ชัน}
A := Copy('Rajabhat Lampang',10,7);
```

3.3.1.3 คำสั่งว่าง (empty statement) คือ คำสั่งที่ไม่มีการระบุใด ๆ ให้ปฏิบัติการโดยปล่อยให้เป็นที่ว่าง ดังนั้นรูปแบบการนิยามจึงไม่ต้องระบุคำสั่งใด ๆ ทั้งสิ้น แต่ทั้งนี้ขึ้นอยู่กับใช้งานในคำสั่งเชิงโครงสร้างแสดงได้ต่อไป แสดงการใช้งานตัวอย่างที่ 3.8

ตัวอย่างที่ 3.8 การใช้งานคำสั่งว่าง

```
Begin   End;
While (a  <> b)  do;
For i := 1 to 100 do;
```

3.3.2 คำสั่งเชิงโครงสร้าง

คำสั่งเชิงโครงสร้าง (structured statement) เป็นคำสั่งที่มีลักษณะเป็นโครงสร้างโดยแต่ละคำสั่งอาจจะประกอบด้วยคำสั่งอื่น ๆ ร่วมอยู่ด้วย ได้แก่ คำสั่งเชิงประกอบ คำสั่งเงื่อนไข และคำสั่งกระทำซ้ำ

3.3.2.1 คำสั่งเชิงประกอบ (compound statement) คือ กลุ่มคำสั่งที่อยู่ระหว่างคำสำคัญ Begin และ End โดยกลุ่มคำสั่งเหล่านี้จะถูกคั่นด้วยอัฒภาค แต่เมื่อประมวลผลโปรแกรมแล้วจะถือว่าทุกคำสั่งในกลุ่มนั้นเป็นคำสั่งเดียว ซึ่งนิยามได้ดังรูปแบบที่ 3.4 และแสดงการใช้งานดังตัวอย่างที่ 3.9

รูปแบบที่ 3.4

```
Begin
        <คำสั่งที่ 1; [,..,คำสั่งที่ n;]>
End;
```

ตัวอย่างที่ 3.9 การใช้งานคำสั่งเชิงประกอบ

```
Begin   A := B;  B := C ;  C := A; End;

Begin   A := 10;  B:= 1;
        For i := 1 to A  do B := B*A;
End;
```

นอกจากนั้นยังสามารถใช้คำสั่งเชิงประกอบในสถานการณ์โดยให้ปฏิบัติการกับกลุ่มคำสั่งก็ต่อเมื่อเงื่อนไขเป็นจริงหรือเท็จ โดยกำหนดร่วมกับคำสั่งเงื่อนไขหรือคำสั่งกระทำซ้ำ แสดงการใช้งานได้ดังตัวอย่างที่ 3.10

ตัวอย่างที่ 3.10 การใช้งานคำสั่งเชิงประกอบตามเงื่อนไข

```
IF A< B Then
Begin
        X := A; A := B; B := X;
End;
```

3.3.2.2 คำสั่งเงื่อนไข (conditional statement) คือ คำสั่งให้เลือกปฏิบัติการตามเงื่อนไขที่กำหนด โดยตรวจสอบพบว่าเงื่อนไขเป็นจริงให้ปฏิบัติการกับคำสั่งชุดหนึ่ง และถ้าเงื่อนไขเป็นเท็จให้ปฏิบัติการกับคำสั่งอีกชุดหนึ่งได้ ซึ่งคำสั่งแบบเงื่อนไขมีให้เลือก 2 รูปแบบ ได้แก่ คำสั่งเงื่อนไขแบบ IF...Then...Else และ คำสั่งเงื่อนไขแบบ Case...Of

(1) คำสั่งเงื่อนไขแบบ IF...Then...Else ใช้ในกรณีที่เลือกกระทำเมื่อเงื่อนไขเป็นจริงจะให้ปฏิบัติการอย่างหนึ่ง และเมื่อเงื่อนไขเป็นเท็จจะให้ปฏิบัติการอีกอย่างหนึ่ง ซึ่งนิยามได้ดังรูปแบบที่ 3.5 และแสดงการใช้งานคำสั่งในกรณีต่าง ๆ ดังตัวอย่างที่ 3.11 ถึงตัวอย่างที่ 3.13

รูปแบบที่ 3.5

```
IF <เงื่อนไข> Then
        <คำสั่งหรือกลุ่มคำสั่งเมื่อเงื่อนไขเป็นจริง>
[Else   <คำสั่งหรือกลุ่มคำสั่งเมื่อเงื่อนไขเป็นเท็จ>];
```

ตัวอย่างที่ 3.11 การใช้งานคำสั่งเงื่อนไขแบบ IF…Then

```
Var     A,B, Big : Integer;
Begin
        If A>B Then
                Big := A;
End;
```

ตัวอย่างที่ 3.12 การใช้งานคำสั่งเงื่อนไขแบบ IF…Then…Else

```
Var     A,B, Big : Integer;
Begin
        If A>B Then
                Big := A
        Else    Big := B;
End;
```

ตัวอย่างที่ 3.13 การใช้งานคำสั่งเงื่อนไขแบบ IF…Then…Else ซ้อนกัน

```
Var     Score : Integer;
Begin
        Readln(Score);
        If Score>=80 Then
                Writeln('เกรด A')
        Else    If Score >=60 Then
                        Writeln('เกรด B')
                Else
                        Writeln('เกรด C');
End;
```

จากตัวอย่างการใช้งานในกรณีข้างต้น นักเขียนโปรแกรมสามารถนำมาประยุกต์ใช้เพื่อแก้ปัญหาที่ซับซ้อนได้ ดังโปรแกรมตัวอย่างที่ 3.14

ตัวอย่างที่ 3.14 โปรแกรมหาจำนวนที่มีค่าน้อยที่สุดในข้อมูล 3 จำนวน

```
Program  Find_Min(Input, Output);
Var     A, B, C , Min : Integer;
Begin
        Readln(A, B, C);
        IF A < B        Then Min := A Else Min := B;
        IF C < Min      Then  Min := C;
        Writeln('ค่าที่น้อยที่สุดคือ ',Min);
End.
```

ผลลัพธ์ทางจอภาพจากโปรแกรม (เมื่อส่งค่า A = 6, B=10 และ C=3) คือ

```
6 10 3
ค่าที่น้อยที่สุดคือ 3
```

ทดสอบ 3.3 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถแสดงผลลัพธ์ทางจอภาพ

ดังตัวอย่างที่ 3.14

(2) คำสั่งเงื่อนไขแบบ Case ...Of ใช้ในกรณีที่มีการกำหนดทางเลือกที่ใช้ในการตัดสินใจมีหลายเงื่อนไข การใช้คำสั่ง IF ซ้อนกันหลาย ๆ ชั้นอาจจะทำให้โปรแกรมดูซับซ้อนและตรวจสอบยาก ดังนั้นเพื่อช่วยให้การเขียนโปรแกรมง่ายขึ้น โครงสร้างของภาษาจึงกำหนดให้ใช้คำสั่ง Case...of แทนการใช้ IF ซ้อนกันเพราะให้ผลลัพธ์ที่เหมือนกัน และยังทำให้เขียนโปรแกรมได้กระชับและเข้าใจได้ง่ายกว่า แต่มีข้อจำกัดบางประการเพราะสามารถตรวจสอบค่าข้อมูลกับนิพจน์เดียวและใช้ได้กับชนิดข้อมูลชนิดอักขระ ตรรกะ และเลขจำนวนเต็มเท่านั้น ซึ่งนิยามได้ดังรูปแบบที่ 3.6 และแสดงการใช้งานดังตัวอย่างที่ 3.15

รูปแบบที่ 3.6

```
Case <นิพจน์> Of
        <ค่าของนิพจน์ 1> : <คำสั่งเมื่อตรงกับค่าของนิพจน์ 1>
        <ค่าของนิพจน์ 2> : <คำสั่งเมื่อตรงกับค่าของนิพจน์ 2>
                ....
        <ค่าของนิพจน์ N> : <คำสั่งเมื่อตรงกับค่าของนิพจน์ N>
Else
        <คำสั่งเมื่อไม่ตรงกับค่าของนิพจน์ใด>
End;
```

ตัวอย่างที่ 3.15 การใช้งานคำสั่งเงื่อนไขแบบ Case…Of

```
Var     Scale : Integer;
Begin
        Case Scale Of
                0 : Writeln('Low');
                1 : Writeln('Medium');
        Else    Writeln('High');
        End;
End;
```

บางกรณีที่มีค่าของนิพจน์ที่ทำงานซ้ำกัน สามารถระบุอยู่ในบรรทัดเดียวกันแล้วคั่นด้วยเครื่องหมายจุลภาค หรือกำหนดเป็นช่วงข้อมูลด้วยเครื่องหมายจุดจุด(...) แสดงการใช้งานดังตัวอย่างที่ 3.16 และสามารถนำมาประยุกต์ใช้ในการเขียนโปรแกรมดังตัวอย่างที่ 3.17

ตัวอย่างที่ 3.16 การใช้งานคำสั่งเงื่อนไขแบบ Case...Of

```
Var     Score : Integer;
Begin
        Case Score Of
                80..100 : Writeln('เกรด A');      {คะแนนระหว่าง 80-100}
                60..79  : Writeln('เกรด B');      {คะแนนระหว่าง 60-79}
        Else    Writeln('เกรด C');
        End;
End;
```

ตัวอย่างที่ 3.17 โปรแกรมคำนวณหาภาษีตามอัตราสินทรัพย์

```
Program  Taxes(Input, Output);
Var     Category , Value : Integer;
        Rate, Tax : Real;
Begin
        Readln(Value);
        Category := Value Div 1000;
        IF Category > 6 Then Category := 6;
        Case Category of
                0,1     : Rate := 0.03;
                  2     : Rate := 0.04;
            3, 4, 5     : Rate := 0.05;
                  6     : Rate := 0.06;
        End;
        Tax := Value * Rate;
        Writeln(' มูลค่าสินทรัพย์ = ',Value);
        Writeln('  ชำระภาษี = ',Tax:12:2);
End.
```

ผลลัพธ์ทางจอภาพจากโปรแกรม (เมื่อส่งค่า Value = 6500) คือ

6500

มูลค่าสินทรัพย์ = 6500

ชำระภาษี = 390

ทดสอบ 3.4 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถแสดงผลลัพธ์ทางจอภาพ

ดังตัวอย่างที่ 3.17

3.3.2.3 คำสั่งกระทำซ้ำ (repetitive statement) คือ คำสั่งให้โปรแกรมปฏิบัติการกับกลุ่มคำสั่งซ้ำกันหลาย ๆ รอบ โดยมีการระบุช่วงการปฏิบัติการที่เป็นวงวน (loop) โดยคำสั่งแบ่งเป็น 3 รูปแบบ ได้แก่ คำสั่ง While…Do คำสั่ง Repeat…Until และคำสั่ง For…To/DownTo..Do

(1) คำสั่ง While...Do คือ คำสั่งกระทำซ้ำโดยต้องตรวจสอบเงื่อนไขที่อยู่หลัง While ก่อนดำเนินการใด ถ้าเงื่อนไขเป็นจริงจึงสามารถทำตามคำสั่งที่กำหนดไว้หลัง Do ได้ภายในขอบเขตวงวนที่กำหนด และเมื่อจบคำสั่งให้กลับมาตรวจสอบเงื่อนไขอีก และให้ดำเนินการในลักษณะเช่นนี้จนกว่าเงื่อนไขเป็นเท็จหรือกล่าวได้ว่า กลุ่มคำสั่งหลัง Do จะถูกปฏิบัติการซ้ำกันจนกระทั่งเงื่อนไขของ While เป็นเท็จ ซึ่งนิยามได้ดังรูปแบบที่ 3.7 แสดงการใช้งานดังตัวอย่างที่ 3.18 และเมื่อนำมาประยุกต์ใช้งานโปรแกรมสามารถแสดงได้ดังตัวอย่างที่ 3.19

รูปแบบที่ 3.7

```
While <เงื่อนไข> Do
        <คำสั่งภายในวงวน>;
```

ตัวอย่างที่ 3.18 การใช้งานคำสั่ง While…Do

```
While A < B do B := B-A;
While i <= 100 Do
Begin
        Sum := Sum+i;  i := i+1;
End
```

ตัวอย่างที่ 3.19 โปรแกรมหาค่าหารร่วมมาก (หรม.) ของจำนวนเต็ม 2 จำนวน

```
Program  Gcd(Input, Output);
Var      A, B : Integer;
Begin
         Readln(A, B);
         While A <> B Do
                  IF A > B Then A := A-B
                  Else  B := B-A;
         Write('ค่า หรม. = ',A);
End.
```

ผลลัพธ์ทางจอภาพจากโปรแกรม (เมื่อส่งค่าเลขจำนวนเต็ม A = 10 และ B = 15) คือ

```
10 15
ค่า หรม. = 5
```

ทดสอบ 3.5 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถแสดงผลลัพธ์ทางจอภาพดังตัวอย่างที่ 3.19

(2) คำสั่ง Repeat...Until คือ คำสั่งกระทำซ้ำโดยปฏิบัติการกลุ่มคำสั่งตามขอบเขตของวงวนตั้งแต่คำสั่ง Repeat จนกระทั่งถึง Until ด้วยวิธีการตรวจสอบเงื่อนไขที่อยู่หลัง Until ถ้าเงื่อนไขเป็นเท็จให้ย้อนกลับไปเริ่มปฏิบัติการกับคำสั่งที่อยู่หลัง Repeat ใหม่ แต่ถ้าเงื่อนไขเป็นจริงให้หยุดการทำงานซ้ำและออกจากวงวน หรือกล่าวได้ว่ากลุ่มคำสั่งที่อยู่ระหว่าง Repeat และ Until จะถูกปฏิบัติการซ้ำกันจนกระทั่งเงื่อนไขของ Until เป็นจริง ซึ่งนิยามได้ดังรูปแบบที่ 3.8 แสดงใช้งานดังตัวอย่างที่ 3.20 และเมื่อนำมาเขียนโปรแกรมสามารถแสดงได้ดังตัวอย่างที่ 3.21

รูปแบบที่ 3.8

```
Repeat
         <คำสั่งภายในวงวน>
Until <เงื่อนไข> ;
```

ตัวอย่างที่ 3.20 การใช้คำสั่ง Repeat...Until

```
Repeat B := B-A Until A >= B
Repeat
        Sum := Sum+i;
        i := i+1;
Until (i > 100);
```

ตัวอย่างที่ 3.21 โปรแกรมหาค่าเฉลี่ยของจำนวน N จำนวน

```
Program  Mean(Input, Output);
Var      X, Sum, Mean : Real;
         N, Count : Integer;
Begin
         Sum := 0.0;  Count := 0;
                  Write(' กรุณาระบุตัวเลขที่มีค่ามากกว่า 0 เท่านั้น = ');
                  Readln(N);
                  Repeat
                           Readln(X);
                           Sum := Sum+X;
                           Count := Count+1;
                  Until (Count = N);
                  Mean := Sum/N;
                  Writeln('ค่าเฉลี่ยของเลข ',N,' จำนวน คือ ',Mean:7:3);
         End.
```

ผลลัพธ์ทางจอภาพจากโปรแกรม (เมื่อส่งค่า N = 5 และ X = 1,2,3,4,5) คือ

```
กรุณาระบุตัวเลขที่มีค่ามากกว่า 0 เท่านั้น =5
1
2
3
4
5
ค่าเฉลี่ยของเลข 5 จำนวน คือ 3.000
```

ทดสอบ 3.6 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถแสดงค่าเฉลี่ยของเลขทางจอภาพ โดยกำหนดให้รับค่าตัวเลขในคอมโพเนนท์เดียวกัน ดังตัวอย่างที่ 3.21

(3) คำสั่ง For…To/DownTo..Do คือ คำสั่งกระทำซ้ำโดยปฏิบัติการตามคำสั่งหลัง Do และปฏิบัติการซ้ำกันเป็นจำนวนคงที่ โดยมีตัวแปรที่กำหนดอยู่หลัง For เป็นตัวนับจำนวนรอบ ดังนั้นตัวแปรดังกล่าวต้องมีชนิดข้อมูลเป็นจำนวนเต็มเท่านั้น ซึ่งจะเริ่มนับตั้งแต่ค่าของนิพจน์เริ่มต้น <ค่าเริ่มต้น> และจะเพิ่มขึ้นรอบละ 1 ค่า เมื่อใช้คำสำคัญว่า To แต่หากใช้คำสำคัญว่า DownTo แต่ละรอบจะถูกลดทีละ 1 ค่า จนกระทั่งถึงค่าของนิพจน์สิ้นสุด <ค่าสิ้นสุด> ซึ่งนิยามได้ดังรูปแบบที่ 3.9 แสดงการใช้งานดังตัวอย่างที่ 3.22 และสามารถนำมาเขียนโปรแกรมดังตัวอย่างที่ 3.23

รูปแบบที่ 3.9

```
{กระทำซ้ำโดยนับจำนวนรอบเพิ่มขึ้นที่ละ 1}
For <ตัวแปร> := <ค่าเริ่มต้น> To <ค่าสิ้นสุด> Do
        <คำสั่งภายในวงวน>;

{กระทำซ้ำโดยนับจำนวนรอบลดลงที่ละ 1}
For <ตัวแปร> := <ค่าเริ่มต้น> DownTo <ค่าสิ้นสุด> Do
        <คำสั่งภายในวงวน>;
```

ตัวอย่างที่ 3.22 การใช้งานคำสั่ง For..To/DownTo…Do

```
For i := 1 To 10 Do  Writeln(i);
For k := i DownTo j Do
Begin
        Readln(X)
        Sum := Sum+X;
End;
```

ตัวอย่างที่ 3.23 โปรแกรมการสร้างรูปสามเหลี่ยมจากสัญลักษณ์ ★

```
Program  Dawning;
Var      i, j, n : Integer;
Begin
        n := 5;
        For  i := n DownTo  1 Do
        Begin
                for j := 1 to i do   Write('★');
                Writeln;
        End;
End.
```

ผลลัพธ์ทางจอภาพจากโปรแกรม คือ

★★★★★
★★★★
★★★
★★
★

ทดสอบ 3. 7 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถแสดงผลลัพธ์ทางจอภาพ ดังตัวอย่างที่ 3.23 โดยแสดงผ่านคอมโพเนนท์ Label 5 ตัว

3.4 การกำหนดแถวลำดับ

ในโปรแกรมภาษาทั่วไปสามารถกำหนดตัวแปรให้เก็บค่าข้อมูลมากกว่า 1 ค่าได้ด้วยการกำหนดรูปชนิดข้อมูลให้เป็นแถวลำดับด้วยคำสั่ง Array ตามด้วยขอบเขตของดัชนีซึ่งเป็นตัวชี้บอกลำดับและชนิดข้อมูลที่ต้องการเก็บในสมาชิกของแถวลำดับนั้น โดยการกำหนดแถวลำดับอาจนิยามไว้ในส่วน Type หรือ Var ได้ สำหรับขอบเขตของดัชนีสามารถกำหนดผ่านตัวแปรได้ แต่ต้องกำหนดค่าตัวแปรนั้นมาก่อนในแบบค่าคงที่ (constant) นอกจากนี้ดัชนีอาจจะเป็นอักขระหรือมีชนิดข้อมูลที่กำหนดขึ้นเองได้ แถวลำดับโดยปกติสามารถกำหนดเป็นมิติ (dimension) ได้

มากกว่า 1 มิติ จึงมีชื่อเรียกแถวลำดับตามมิติ ได้แก่ แถวลำดับ 1 มิติ แถวลำดับ 2 มิติ และ แถวลำดับ 3 มิติ

3.4.1 การกำหนดแถวลำดับ 1 มิติ

แถวลำดับ 1 มิติ คือการกำหนดตัวแปรเพื่อจัดเก็บค่าข้อมูลที่ถูกจัดเรียงในหน่วยความจำอย่างต่อเนื่องตามลำดับ โดยกำหนดขอบเขตของดัชนีเพียงช่วงเดียวเท่านั้น และจำนวนสมาชิกพิจารณาจากค่าที่กำหนดในขอบเขตล่างและขอบเขตบนของดัชนี ซึ่งนิยามได้ดังรูปแบบที่ 3.10 และแสดงการใช้งานดังตัวอย่างที่ 3.24

รูปแบบที่ 3.10

```
Type   <ตัวแปร> = Array[l..u] of <ชนิดข้อมูล>
Var    <ตัวแปร> :  Array[l..u] of <ชนิดข้อมูล>
```

โดยที่ l หมายถึงขอบเขตล่างของมิติ (lower bound)

u หมายถึงขอบเขตบนของมิติ (upper bound)

(Aaron & Moshe, 1986, p.23)

ตัวอย่างที่ 3.24 การใช้กำหนดแถวลำดับ 1 มิติ

```
Conts  min = 1; max = 7;
Type   Weekday = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
       List    = Array[1..100] of Real;
Var    Week  :  Array[1..7] of Weekday;
       a     :  Array[1..100] of Real;
       b     :  Array[min..max] of Integer;
```

จากตัวอย่างที่ 3.24 การกำหนดข้างต้นจะได้ค่าข้อมูลผลลัพธ์ของตัวแปรแถวลำดับ สามารถอธิบายได้ดังต่อไปนี้

3.4.1.1 แถวลำดับ List มีจำนวน 100 สมาชิก คือ List[1], List[2],..., List[100] สามารถเก็บค่าข้อมูลเป็นจำนวนจริงโดยอ้างอิงจากลำดับสมาชิก เช่น List[50] := 45.11 หมายถึง แถวลำดับ List สมาชิกตัวที่ 50 เก็บค่า 45.11 ไว้

3.4.1.2 แถวลำดับ Week มีจำนวน 7 สมาชิก คือ Week[1], Week[2],..., Week[7] สามารถเก็บข้อมูลแบบเดียวกับที่กำหนดให้กับ Weekday โดยอ้างอิงจากลำดับสมาชิก เช่น Week[3] := Wed; หรือ Week[6] := Sat;

3.4.1.3 แถวลำดับ a มีจำนวน 100 สมาชิก คือ a[1], a[1],..., a[100] โดยเก็บค่าข้อมูลเป็นจำนวนจริงเช่นเดียวกับแถวลำดับ List

3.4.1.4 แถวลำดับ b มีจำนวน 7 สมาชิก คือ b[1], b[2],..., b[7] โดยเก็บค่าข้อมูลเป็นจำนวนเต็ม

ทดสอบ 3.8 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถรับชื่อนักศึกษาในห้องเรียนจำนวน 10 ชื่อ เมื่อรับครบถ้วนให้แสดงทางจอภาพ

ทดสอบ 3.9 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถรับคะแนนสอบของนักศึกษาในห้องเรียนจำนวน 10 ครั้ง เมื่อรับครบถ้วนให้คำนวณหาค่าเฉลี่ยและแสดงทางจอภาพ

3.4.2 การกำหนดแถวลำดับ 2 มิติ

แถวลำดับ 2 มิติ คือการกำหนดตัวแปรเพื่อจัดเก็บค่าข้อมูลที่ถูกจัดเรียงในหน่วยความจำอย่างต่อเนื่องตามลำดับ โดยกำหนดขอบเขตของดัชนีไว้ 2 ตัวตามมิติและจำนวนสมาชิกพิจารณาจากค่าที่กำหนดในขอบเขตล่างและขอบเขตบนของดัชนีตัวที่ 1 และดัชนีตัวที่ 2 ซึ่งโปรแกรมภาษาปาสกาลกำหนดให้ดัชนีตัวที่ 1 อ้างถึงแถวของแถวลำดับและดัชนีตัวที่ 2 อ้างถึงคอลัมน์ของแถวลำดับ จึงนิยามได้ดังรูปแบบที่ 3.11 และแสดงการใช้งานดังตัวอย่างที่ 3.25

<u>รูปแบบที่</u> 3.11

Type <ตัวแปร>=Array[l_1..u_1,l_2..u_2] of <ประเภทข้อมูล>

Var <ตัวแปร>: Array[l_1..u_1,l_2..u_2] of <ประเภทข้อมูล>

โดยที่ l_1 และ u_1 เป็นขอบเขตล่างและบนของดัชนีตัวที่ 1

l_2 และ u_2 เป็นขอบเขตล่างและบนของดัชนีตัวที่ 2

ตัวอย่างที่ 3.25 การกำหนดแถวลำดับ 2 มิติ

```
Conts   min = 1; max = 7;
Type    Weekday = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
        List    = Array[1..2,1..100] of Real;
Var     a       : Array[1..7,min..max] of Integer;
        b       : Array[min..max, Weekday] of Real;
```

จากตัวอย่าง 3.25 การกำหนดข้างต้นจะได้ค่าข้อมูลผลลัพธ์ของตัวแปรแถว-ลำดับ 2 มิติ สามารถอธิบายได้ดังต่อไปนี้

3.4.2.1 แถวลำดับ List มีจำนวน 200 สมาชิก คือ List[1,1], List[1,2],...,List[1,100], List[2,1], List[2,2],..., List[2,100] และสามารถเก็บค่าข้อมูลเป็นจำนวนจริงโดยอ้างอิงจากลำดับสมาชิกตามขอบเขตดัชนีตัวที่ 1 และขอบเขตดัชนีตัวที่ 2 เช่น List[1,50] := 45.11 หมายถึง แถวลำดับ List สมาชิกแถวที่ 1 คอลัมน์ที่ 50 เก็บค่า 45.11 ไว้

3.4.2.2 แถวลำดับ a มีจำนวน 49 สมาชิก คือ a[1,1], a[1,2], ..., a[1,7], a[2,1], a[2,2], ..., a[2,7], a[3,1], a[3,2], ..., a[3,7], a[4,1], a[4,2], ..., a[4,7], a[5,1], a[5,2], ..., a[5,7], a[6,1], a[6,2], ..., a[6,7], a[7,1], a[7,2], ..., a[7,7]

3.4.2.3 แถวลำดับ b มีจำนวน 49 สมาชิก คือ b[1,Mon], b[1,Tue],..., b[1,Sun], b[2,Mon], b[2,Tue],..., b[2,Sun], b[3,Mon], b[3,Tue],..., b[3,Sun], b[4,Mon], b[4,Tue],..., b[4,Sun], b[5,Mon], b[5,Tue],..., b[5,Sun], b[6,Mon], b[6,Tue],..., b[6,Sun], b[7,Mon], b[7,Tue],..., b[7,Sun]

ในการกำหนดตัวแปรสำหรับเก็บข้อความ หรืออักขระที่มีมากกว่า 1 ตัวนั้น โปรแกรมภาษาปาสกาลมาตรฐานไม่ได้กำหนดไว้ อย่างไรก็ตามหากต้องการใช้ตัวแปรในโปรแกรมแล้ว สามารถกำหนดชนิดข้อมูลโดยใช้แถวลำดับเข้ามาช่วย ดังเช่น

```
Var  name : array[1..30] of char;
```

สำหรับการกำหนดดังกล่าว ตัวแปร name สามารถจัดเก็บตัวอักขระได้ 30 ตัว สำหรับโปรแกรมภาษาปาสกาลที่พัฒนาใหม่และโปรแกรมบอร์แลนด์เดลไฟ พบว่ามีการพัฒนาชนิดข้อมูลเพื่อจัดเก็บกลุ่มอักขระดังกล่าวขึ้นใหม่ในชื่อว่าสายอักขระโดยใช้คำสำคัญว่า String ดังเช่น

```
Var name : String[30];
Var name : String;              {ในโปรแกรมบอร์แลนด์เดลไฟ}
```

หรือหากไม่ระบุขนาดของสายอักขระ หมายความว่าสามารถจัดเก็บตัวอักขระได้ตามมาตรฐานแอสกีคือ 255 ตัวอักขระ

ทดสอบ 3.10 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถรับชื่อ และอายุของนักศึกษาในห้องเรียนจำนวน 10 ชื่อ เมื่อรับครบถ้วนให้แสดงทางจอภาพ

ทดสอบ 3.11 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถรับชื่อและคะแนนสอบของนักศึกษาในห้องเรียนจำนวน 10 ครั้ง เมื่อรับครบถ้วนให้แสดงชื่อและคะแนนของนักศึกษาที่มีคะแนนมากที่สุดและน้อยที่สุดทางจอภาพ

อย่างไรก็ตาม แถวลำดับ 2 มิติมีการใช้งานกันมากเพราะมีรูปแบบการจัดเก็บที่พบเห็นในชีวิตประจำวัน อาทิเช่น สมุดบัญชี คะแนนสอบของนักศึกษาหรือการประยุกต์ทางคณิตศาสตร์ เช่น การคำนวณเมตริกซ์ เป็นต้น แสดงได้ดังโปรแกรมตัวอย่างที่ 3.26

ตัวอย่างที่ 3.26 โปรแกรมหาผลรวมแต่ละแถว แต่ละหลักของเมตริกซ์ขนาด 5x5

```
Program  Row_Col_Sum(Input, Output);
Type     matrix = Array[1..5,1..5] of Integer;
Var      a    : matrix;
         row, col : Array[1..5] of Integer;
         i, j  : Integer;
         Begin
          For i := 1 to 5 do Begin
```

```
            row[i] := 0;
            col[i] := 0;
        End;
        Writeln('รับค่า Matrix a ');
        For i := 1 to 5 do
        Begin
            For  j := 1 to 5 do Begin Read(a[i,j]); Write(' '); End;
            Writeln;
        End;
        For i := 1 to 5 do
            For  j := 1 to 5 do row[i] := row[i]+a[i,j];
        For j := 1 to 5 do
            For  i := 1 to 5 do col[j] := col[j]+a[i,j];
        Writeln('ผลรวมของแถว ');
        For i := 1 to 5 do Write(row[i],'   ');
        Writeln;Writeln('ผลรวมของหลัก ');
        For j := 1 to 5 do Write(col[j],'   ');
End.
```

ผลลัพธ์ทางจอภาพจากโปรแกรม (เมื่อจัดเก็บค่าแถวลำดับ a จำนวน 25 สมาชิก) คือ

```
รับค่า Maxtrix a

1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
ผลรวมของแถว  15  15  15  15  15
ผลรวมของหลัก   5  10  15  20  25
```

ทดสอบ 3.12 จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้สามารถแสดงผลลัพธ์ทางจอภาพดังตัวอย่างที่ 3.26

3.5 การดีบักและจัดการกับข้อผิดพลาด

การพัฒนาแอพพลิเคชันหนึ่งขึ้นมา แม้จะมีการออแบบและใช้ความระมัดระวังในการเขียนโปรแกรมอย่างดีแล้วก็ยังมีความเป็นไปได้ที่การทำงานของโปรแกรมที่ได้จะไม่เป็นไปตามความต้องการของผู้ใช้งาน ดังนั้นการตรวจสอบหาข้อผิดพลาดของโปรแกรมหรือที่เรียกกันว่าดีบักโปรแกรมจึงเป็นส่วนสำคัญที่จะทำให้โปรแกรมมีความถูกต้องสมบูรณ์ยิ่งขึ้น

เดลไฟมีเครื่องมือช่วยในการตรวจสอบข้อผิดพลาดที่เกิดขึ้นในโปรแกรม ซึ่งช่วยให้ผู้เขียนโปรแกรมสามารถแก้ไขข้อผิดพลาดต่าง ๆ ได้ง่าย นอกจากนี้ยังมีคำสั่งที่สามารถระบุไว้ในโปรแกรมเพื่อตรวจสอบและจัดการกับข้อผิดพลาดที่อาจเกิดขึ้นในขณะรันโปรแกรมด้วย โดยปกติข้อผิดพลาดของโปรแกรม สามารถแบ่งได้ 3 ประเภท

1. Compile Error

   คือข้อผิดพลาดที่เกิดขึ้นในขณะคอมไฟล์โปรแกรม สาเหตุจากการเขียนโปรแกรมผิดไวยากรณ์ เช่น ไม่มีเครื่องหมาย ; เมื่อจบคำสั่งหรือมีการเรียกใช้ตัวแปรที่ไม่ได้ประกาศไว้ก่อน เป็นต้น การแก้ไขทำได้ง่ายเพราะในขณะคอมไฟล์โปรแกรมถ้าเกิดข้อผิดพลาดประเภทนี้ เดลไฟจะเลื่อนไปอยู่ที่บรรทัดที่ผิด พร้อมกับแสดงสาเหตุของข้อผิดพลาดที่เกิดขึ้นไว้ในกรอบด้านล่างของวินโดว์โค้ดเอดิเตอร์

2. Run-Time Error

   คือข้อผิดพลาดที่เกิดขึ้นในขณะที่โปรแกรมทำงาน โดยที่คำสั่งต่าง ๆ เขียนไว้อย่างถูกต้องตามหลักไวยากรณ์แล้ว ตัวอย่างข้อผิดพลาดประเภทนี้ได้แก่ การหารด้วยศูนย์ หรือลบข้อมูลใน ListBox ที่ไม่มีข้อมูล เป็นต้น โดยเดลไฟจะแจ้งข้อผิดพลาดด้วยไดอะล็อกซ์บ๊อกซ์

3. Logic Error

   คือข้อผิดพลาดที่เกิดขึ้นโดยที่การเขียนคำสั่งในโปรแกรมถูกต้องแล้ว แต่การทำงานของโปรแกรมให้ผลลัพธ์ไม่ตรงกับที่ต้องการ เช่น อาจจะกำหนดคำสั่งให้ทำงานไม่ครบ ดังตัวอย่างโปรแกรมต่อไปนี้

การตรวจสอบข้อผิดพลาด

สำหรับข้อผิดพลาดประเภท Compile Error สามารถตรวจสอบได้ในขณะออกแบบและกำลังคอมไฟล์โปรแกรม ส่วนข้อผิดพลาดประเภท Run-Time Error หรือ Login Error นั้นจะต้องตรวจสอบหาข้อผิดพลาดได้ในขณะรันโปรแกรมเท่านั้น โดยใช้เครื่องมือในการตรวจสอบหรือที่เรียกว่า ดีบักเกอร์ (debugger) ซึ่งมีการทำงานหลายอย่าง ดังนี้

1. **เลือก Run to Cursor**
2. การใช้ Trace Into
3. การใช้ Trace to Next Source Line
4. การใช้ Run Until Return
5. การใช้ Step Over
6. การใช้เบรคพอยท์ (Breakpoint)
7. **การใช้ Program Pause**
8. **การใช้ Program Reset**
9. การใช้ Watch
10. การใช้ Evaluate/Modify
11. การใช้ Call Stack

3.6 กรณีตัวอย่าง

3.6.1 การกำหนดตัวแปร และรูปแบบภาษา

ภาษาออปเจ็คปาสคาล เป็นภาษาที่สามารถใช้ตัวอักษรพิมพ์เล็กหรือใหญ่ก็ได้ในการเขียน ซึ่งจะมีความหมายเหมือนกัน และไม่จำกัดว่าจะต้องอยู่ในบรรทัดเดียวกัน เพียงแต่ต้องเขียนให้ถูกต้องตามหลักไวยากรณ์ของภาษาเท่านั้น โดยที่แต่ละคำสั่งจะต้องปิดท้ายด้วยเครื่องหมาย ; (semi-colon)

<u>กรณีตัวอย่างการกำหนดค่าให้กับออปเจ็ค</u>

```
PROCEDURE TForm.Button1CLICK(SENDER : TOBJECT);
BEGIN
        Label1.Caption := ' ยินดีที่ได้รู้จัก !!';
END;
```

<u>กรณีตัวอย่างการกำหนดค่าให้กับออปเจ็ค</u>

```
PROCEDURE TForm.Button1CLICK(SENDER : TOBJECT);
BEGIN
        Label1.Caption := ' ยินดีที่ได้รู้จัก !!'+Edit1.text+' และ '+Edit2.text;
END;
```

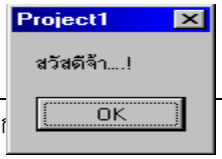
<u>กรณีตัวอย่างการกำหนดตัวแปร</u>

```
  var   Name1, Name2      : String;
        Score             : Integer;
        Grade             : Real;
```

3.6.2 โพรซีเยอร์และฟังก์ชันที่ควรรู้จักในบอร์แลนด์เดลไฟ

สำหรับการควบคุมการใช้งานโปรแกรมภาษา บางขณะอาจจำเป็นต้องสร้างโพรซีเยอร์และฟังก์ชันขึ้นมาเอง เพื่ออำนวยความสะดวกในการใช้งานกรณีที่มีการใช้บ่อยครั้ง แต่

บางครั้งโพซีเยอร์หรือฟังกัชันที่ถูกเตรียมใช้เฉพาะงานอาจมีให้บริการไว้ในโปรแกรมภาษาอยู่แล้ว ดังนั้นผู้ใช้งานสามารถเรียกใช้งานได้ตามรูปแบบของแต่ละโพซีเยอร์หรือฟังก์ชันนั้น ๆ ซึ่งในบอร์แลนด์เดลไฟได้เตรียมโพซีเยอร์และฟังก์ชันไว้มากมาย จึงขอนำเสนอเฉพาะที่ใช้งานบ่อยครั้งเท่านั้น ดังต่อไปนี้

<table>
<tr><td>Showmessage</td><td>โพซีเยอร์แสดงไดอะล๊อกบ๊อกซ์ข้อความทางจอภาพโดยผู้ใช้กำหนดและปุ่ม OK เท่านั้น
<u>รูปแบบ</u> Showmessage('ข้อความ');
ตัวอย่าง Showmessage('สวัสดีจ้า....!');
  </td></tr>
<tr><td>MessageDlg</td><td>ฟังก์ชันใช้แสดงข้อความเหมือน Showmessage แต่สามารถ
ไดอะล๊อกบ๊อกซ์ที่แสดงได้
<u>รูปแบบ</u> MessageDlg('ข้อความ',ประเภทไดอะล๊อกบ๊อกซ์,[ปุ่มที่ต้องการให้แสดง],
หมายเลขแสดงข้อมูลช่วยเหลือ);
<u>ประเภทไดอะล๊อกบ๊อกซ์</u>
- mtWarning สำหรับแสดงข้อความเตือน โดยใช้เครื่องหมาย ! บนสามเหลี่ยมสีเหลืองและไตเติลบาร์แสดงคำว่า Warning
- mtError สำหรับแสดงข้อผิดพลาด โดยใช้เครื่องหมายกากบาทบนวงกลมสีแดงและไตเติลบาร์แสดงคำว่า Error
- mtInformation สำหรับแสดงข้อมูลทั่วไป โดยใช้อักษรตัว ! บนสัญลักษณ์การพูด และไตเติลบาร์แสดงคำว่า Information
- mtConfirmation สำหรับแสดงข้อความยืนยัน โดยใช้เครื่องหมาย ? บนสัญลักษณ์การพูด และไตเติลบาร์แสดงคำว่า Confirm
- mtCustom สำหรับแสดงข้อความ โดยไม่มีรูปภาพใด ๆ และไตเติลบาร์มีชื่อของโปรเจ็คปัจจุบัน
<u>ปุ่มที่ต้องการแสดง</u>
- mbYes บนปุ่มแสดงข้อความ Yes
- mbNo บนปุ่มแสดงข้อความ No
- mbOK บนปุ่มแสดงข้อความ OK
- mbCancel บนปุ่มแสดงข้อความ Cancel
- mbHelp บนปุ่มแสดงข้อความ Help</td></tr>
</table>

	- mbAbort บนปุ่มแสดงข้อความ Abort - mbRetry บนปุ่มแสดงข้อความ Retry - mbIgnore บนปุ่มแสดงข้อความ Ignore - mbAll บนปุ่มแสดงข้อความ All โดยต้องกำหนดค่าเหล่านี้อย่างน้อยหนึ่งค่าลงระหว่างสัญลักษณ์ [ ] เช่น [mbYes,mbNo,mbCancel] หมายเลขช่วยเหลือ - เป็นค่าตัวเลขสำหรับระบุหมายเลข Help ที่ใช้กับไดอะล็อกบ๊อกซ์นี้เมื่อมีการกด F1 ถ้าไม่มี Help ให้ใช้การกำหนดค่า 0 แทน *ตัวอย่าง* MessageDlg('คุณต้องการเข้าสู่ระบบ..',mtConfirmation,[mbYes,mbNo],0); MessageDlg('ยืนยันการจัดเก็บ...หรือไม่',mtWarning,[mbOK, mbCancel],0);
StrToInt	ฟังก์ชันแปลงชนิดข้อมูลจากตัวอักษร เป็น ตัวเลขจำนวนเต็ม รูปแบบ StrToInt(ข้อความ); *ตัวอย่าง* X := StrToInt('30'); // ค่าตัวแปร X = 30 ;
IntToStr	ฟังก์ชันแปลงชนิดข้อมูลจากตัวเลขจำนวนเต็มเป็นตัวอักษร รูปแบบ IntToStr(ตัวเลข); *ตัวอย่าง* X := IntToStr(0030); // ค่าตัวแปร X = '30' ;
StrToFloat	ฟังก์ชันแปลงชนิดข้อมูลจากตัวอักษร เป็นตัวเลขแบบมีทศนิยม รูปแบบ StrToFloat(ข้อความ); *ตัวอย่าง* Y := StrToFloat('7.5212'); // ค่าตัวแปร Y = 7.5212 ;
FloatToStr	ฟังก์ชันแปลงชนิดข้อมูลจากตัวเลขแบบมีทศนิยมเป็นตัวอักษร รูปแบบ FloatToStr(ตัวเลขมีทศนิยม); *ตัวอย่าง* Y := FloatToStr(7.5212); // ค่าตัวแปร Y = '7.5212' ;
FloatToStrF	ฟังก์ชันแปลงชนิดข้อมูลจากตัวเลขแบบมีทศนิยมเป็นตัวอักษรตามรูปแบบที่กำหนด รูปแบบ FloatToStrF(ตัวเลขมีทศนิยม,ประเภทรูปแบบ,ขนาด,ทศนิยม); ประเภทรูปแบบ (FloatFormat Type) - ffGeneral รูปแบบตัวเลขตามปกติ เช่น 5670 - ffFixed รูปแบบตัวเลขแบบมีทศนิยมกำกับท้าย เช่น 5670.00

	- ffNumber รูปแบบตัวเลขจำนวนแบบมีทศนิยมกำกับท้าย เช่น 5,670.00 - ffCurrency รูปแบบตัวเลขแบบตัวเงิน เช่น ฿5,670.00 *ตัวอย่าง* X := FloatToStrF(5670,ffNumber,10,2); //ค่าตัวแปร X = '5,670.00';
Copy	ฟังก์ชันนำบางส่วนของสายอักขระ(ข้อความ) รูปแบบ Copy(สายอักขระ, ตำแหน่งเริ่มต้น, จำนวนที่อักขระที่ต้องการ); *ตัวอย่าง* Y := Copy('I Love You',3,4); // ค่าตัวแปร Y = 'Love' ;
Date และ DateToStr	ฟังก์ชันแสดงค่าวันที่ปัจจุบัน และแปลงเป็นตัวอักษร รูปแบบ DateToStr(Date); ตัวอย่าง Label1.Caption := 'วันนี้คือ '+DateToStr(Date); //บางทีใช้ Now
Time และ TimeToStr	ฟังก์ชันแสดงค่าเวลาปัจจุบัน และแปลงเป็นตัวอักษร รูปแบบ TimeToStr(Time); ตัวอย่าง Label1.Caption := 'ขณะนี้เวลา : '+TimeToStr(Time); //บางทีใช้ Now
FormatDateTime	ฟังก์ชันแสดงวันที่/เวลาตามรูปแบบที่กำหนด รูปแบบ FormatDateTime(รูปแบบวันที่/เวลา, ค่าวันที่/เวลา); ตัวอย่าง Label1.Caption := FormatDateTime('dd/mmm/yyyy',Now);
Length	ฟังก์ชันนับจำนวนของสายอักขระ(ข้อความ) รูปแบบ Length(สายอักขระ); *ตัวอย่าง* Y := Length('Rajabhat'); // ค่าตัวแปร Y = 8;
Pos	ฟังก์ชันค้นหาตำแหน่งอักขระที่ต้องการในสายอักขระ (ข้อความ) รูปแบบ Pos(อักขระที่ต้องการค้นหา, สายอักขระ); *ตัวอย่าง* X := Pos('k','Suksomboon'); // ค่าตัวแปร X = 3;
Break	โพรซีเยอร์สำหรับออกจากการวนซ้ำ อาทิ For, While, Repeat Until รูปแบบ Break; *ตัวอย่าง* For i:=1 To 10 If i>5 Then Break;
การใช้งาน External File	AssignFile โพรซีเยอร์ที่เรียกใช้ External File รูปแบบ AssignFile(ชื่ออ้างอิง,แหล่งที่ตั้งพร้อมชื่อไฟล์); Reset โพรซีเยอร์ที่เปิด External File ที่มีอยู่ รูปแบบ Reset(ชื่ออ้างอิง); Rewrite โพรซีเยอร์ที่เปิด External File แบบสร้างใหม่

	<u>รูปแบบ</u> Rewrite(ชื่ออ้างอิง); Write/Writeln โพรซีเยอร์ที่บันทึกข้อมูลลงใน External File ที่เปิดอยู่ <u>รูปแบบ</u> Writeln(ชื่ออ้างอิง,ข้อมูล1[,ข้อมูล2,…]); Read/Readln โพรซีเยอร์ที่อ่านข้อมูลจากใน External File ที่เปิดอยู่ <u>รูปแบบ</u> Readln(ชื่ออ้างอิง,ตัวแปร1[,ตัวแปร2,…]); CloseFile โพรซีเยอร์ปิดการเรียกใช้ External File ที่เปิดอยู่ <u>รูปแบบ</u> CloseFile(ชื่ออ้างอิง); EOF ฟังก์ชันสำหรับตรวจสอบการจบไฟล์ (End of File) <u>รูปแบบ</u> EOF(ชื่ออ้างอิง) *ตัวอย่าง1* Var Fi : TextFile; AssignFile(Fi,'C:\Data.txt'); *{$I-} Reset(Fi);{$I+} // I/O Error* Writeln(Fi,'Paijit','Suksomboon','Software Engineering'); CloseFile(Fi); *ตัวอย่าง2* Var Fi : TextFile; Fname, Lname, Depart : String; AssignFile(Fi,'C:\Data.txt'); *{$I-} Rewritet(Fi);{$I+}* While Not EOF(Fi) Do Begin Readln(Fi, Fname, Lname, Depart); End; CloseFile(Fi);

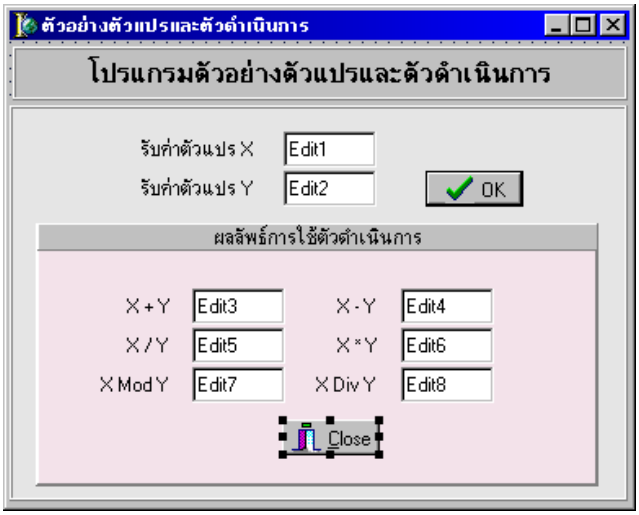
<u>ทดสอบ 3.13</u> จงเขียนโปรแกรมด้วยเดลไฟเพื่อให้ทดลองเรียกใช้ฟังก์ชันที่แสดงในตารางข้างต้น

<u>ทดสอบ 3.14</u> จงเขียนโปรแกรมด้วยเดลไฟเพื่อทดลองเปิด External File ใหม่ชื่อ MyFriend.txt เพื่อจัดเก็บข้อมูลเพื่อนประกอบด้วย รหัส ชื่อ นามสกุล และชื่อเล่น ไม่น้อยกว่า 5 คน

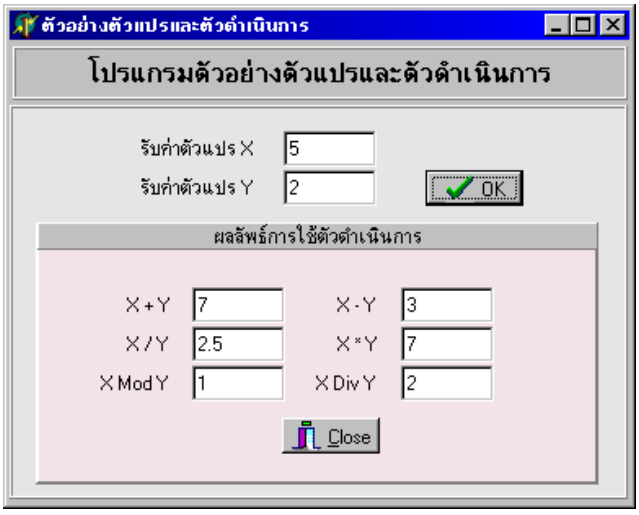
ทดสอบ 3.15 จงเขียนโปรแกรมด้วยเดลไฟเพื่อทดลองเปิด External File ที่มีอยู่แล้วชื่อ MyFriend.txt เพื่อดึงข้อมูลเพื่อที่จัดเก็บทั้งหมดมาแสดงทางจอภาพทีละคน

3.6.3 ทดสอบกรณีตัวอย่าง

ทดสอบ 3.16 จงเขียนโปรแกรมด้วยเดลไฟสำหรับทดสอบตัวดำเนินการทางคณิตศาสตร์ โดยกำหนดให้รับตัวแปร X และ Y ใน Edit1 และ Edit2 ตามลำดับ โดยกำหนดปุ่ม OK เพื่อแสดงผลของการใช้ตัวดำเนินการดังรูปที่ 3.1 ใน Edit3, Edit4, Edit5, Edit6, Edit7 และ Edit8 ดังต่อไปนี้



รูปที่ 3.1 ตัวอย่างการใช้ตัวแปรและตัวดำเนินการ



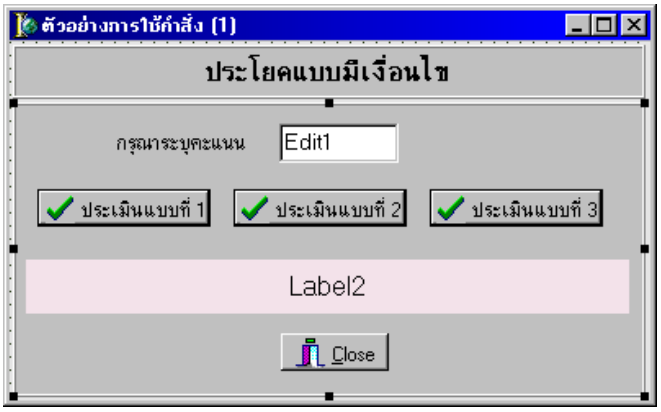
รูปที่ 3.2 ตัวอย่างผลลัพธ์การใช้ตัวแปรและตัวดำเนินการ

ตัวอย่างคำสั่ง

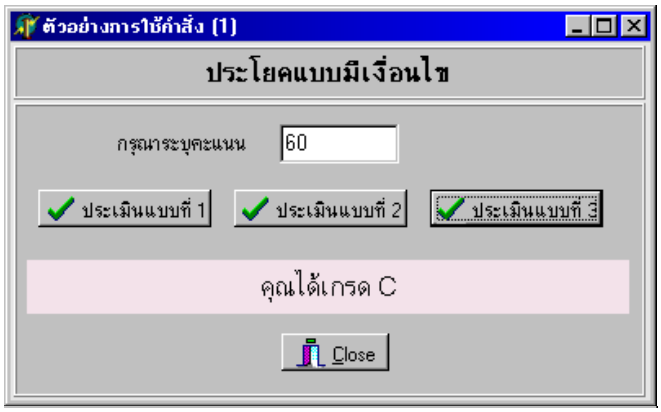
```
procedure TForm1.BitBtn1Click(Sender: TObject); ← สร้างเหตุการณ์เมื่อคลิกปุ่ม OK
var X, Y : Integer;
begin
        X := StrToInt(Edit1.text);
        Y := StrToInt(Edit2.text);
        Edit3.text := IntToStr(X+Y);
        Edit4.text := IntToStr(X-Y);
        Edit5.text := FloatToStr(X/Y);
        Edit6.text := IntToStr(X+Y);
        Edit7.text := IntToStr(X Mod Y);
        Edit8.text := IntToStr(X Div Y);
end;
procedure TForm1.BitBtn2Click(Sender: TObject); ← สร้างเหตุการณ์เมื่อคลิกปุ่ม Close
begin
        Close;
end;
```

ทดสอบ 3.17 จงเขียนโปรแกรมด้วยเดลไฟสำหรับทดสอบคำสั่งเงื่อนไข โดยกำหนดให้รับคะแนนใน Edit1 โดยกำหนดป้ายข้อความ Label วางไว้ด้านล่างปุ่มประเมินผล 3 แบบดังรูปที่ 3.3 เตรียมพร้อมสำหรับการกำหนดเหตุการณ์ในการประเมินเมื่อคลิกปุ่มดังนี้ต่อไปนี้

- ปุ่ม ประเมินแบบที่ 1 ถ้าคะแนนใน Edit1.text > 50 ปรากฏข้อความ ว่า *คุณสอบผ่าน*
- ปุ่ม ประเมินแบบที่ 2 ถ้าคะแนนใน Edit1.text > 50 ปรากฏข้อความ ว่า *คุณสอบผ่าน* แต่ถ้าไม่ใช่ปรากฏข้อความว่า คุณสอบตก!!
- ปุ่ม ประเมินแบบที่ 3 กำหนดเกรด *A, B, C, D, สอบตก* ตามคะแนนดังตัวอย่างของ CASE..OF



รูปที่ 3.3
ตัวอย่างประโยคแบบมีเงื่อนไข



รูปที่ 3.4
ตัวอย่างผลลัพธ์แบบมีเงื่อนไข

ตัวอย่างคำสั่ง

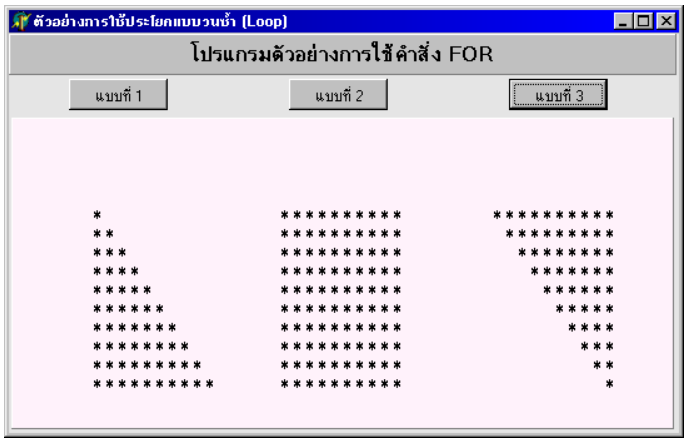
```
procedure TForm1.BitBtn1Click(Sender: TObject);  ← สร้างเหตุการณ์เมื่อคลิกปุ่มประเมินแบบที่ 1
begin
        if StrToInt(Edit1.text)>50 Then
                Label2.Caption := 'คุณสอบผ่าน';
end;

procedure TForm1.BitBtn2Click(Sender: TObject); ← สร้างเหตุการณ์เมื่อคลิกปุ่มประเมินแบบที่ 3
begin
        if StrToInt(Edit1.text)> 50 Then
                Label2.Caption := 'คุณสอบผ่าน'
        else
                Label2.Caption := 'คุณสอบตก!!';
end;
```

```
procedure TForm1.BitBtn3Click(Sender: TObject);←  สร้างเหตุการณ์เมื่อคลิกปุ่มประเมินแบบที่ 3
begin
        Case StrToInt(Edit1.text) of
                80..100 : Label2.Caption :=  'คุณได้เกรด A';
                70..79  : Label2.Caption :=  'คุณได้เกรด B';
                60..69  : Label2.Caption :=  'คุณได้เกรด C';
                50..59  : Label2.Caption :=  'คุณได้เกรด D';
                0..49   : Label2.Caption :=  'คุณสอบตก!!';
        End;
End;
procedure TForm1.BitBtn4Click(Sender: TObject);←  สร้างเหตุการณ์เมื่อคลิกปุ่ม Close
begin
        Close;
end;
```

ทดสอบ 3.18 จงเขียนโปรแกรมด้วยเดลไฟสำหรับทดสอบคำสั่งทำซ้ำ โดยกำหนดให้วาดรูปสามเหลี่ยม และ สี่เหลี่ยม โดยใช้สัญลักษณ์ * ระบายเป็นรูปทรงเมื่อคลิกปุ่ม 3 แบบ ดังรูปที่ 3.5

- ปุ่ม แบบที่ 1 วาดรูปสามเหลี่ยมมุมฉากล่างซ้าย
- ปุ่ม แบบที่ 2 วาดรูปสี่เหลี่ยม
- ปุ่ม แบบที่ 3 วาดรูปสามเหลี่ยมมุมฉากบนขวา



รูปที่ 3.5 ตัวอย่างผลลัพธ์ของการใช้ประโยคแบบวนซ้ำ

ตัวอย่างคำสั่ง

```
procedure TForm1.BitBtn1Click(Sender: TObject);  ← สร้างเหตุการณ์เมื่อคลิกปุ่ม แบบที่ 1
var i,j : integer;
begin
        for i := 1 to 10 do
                for j := 1 to i do
                PaintBox1.Canvas.TextOut(50+(j*10),50+(i*15),'*');
end;
procedure TForm1.BitBtn2Click(Sender: TObject); ← สร้างเหตุการณ์เมื่อคลิกปุ่ม แบบที่ 2
var i,j : integer;
begin
   for i := 1 to 10 do
     for j := 1 to 10 do
        PaintBox1.Canvas.TextOut(200+(j*10),50+(i*15),'*');
end;
procedure TForm1.BitBtn3Click(Sender: TObject); ← สร้างเหตุการณ์เมื่อคลิกปุ่ม แบบที่ 3
var i,j : integer;
begin
   for i := 1 to 10 do
     for j := 10 Downto i do
        PaintBox1.Canvas.TextOut(370+(j*10),50+(i*15),'*');
end;
```

แบบฝึกหัดที่ 3

1. จงเขียนโปรแกรมชื่อ EX1_1 เพื่อวัดระดับน้ำหนักของผู้ใช้บริการโดยรับค่าน้ำหนัก (Weight) เป็นค่าตัวเลขจำนวนเต็ม (หน่วยเป็นกิโลกรัม) และรับค่าส่วนสูง (Height) เป็นตัวเลขมีทศนิยม (หน่วยเป็นเมตร) เมื่อรับครบถ้วนให้ดำเนินการคำนวณหาค่าระดับด้วยสมการดังนี้

   ค่าระดับ = Weight / (Height^2)

   จากนั้นนำมาเปรียบเทียบและแสดงข้อความดังนี้

   - ถ้า ค่าระดับมากกว่า 23 แสดงข้อความ '**คุณอ้วนเกินไป กรุณาลดหุ่น!!**'
   - ถ้า ค่าระดับอยู่ระหว่าง 18 – 23 แสดงข้อความ '**คุณมีร่างกายสมส่วนดี**'
   - ถ้า ค่าระดับน้อยกว่า 18 แสดงข้อความ '**คุณผอมเกินไป กรุณาบำรุงร่างกายหน่อย**'
2. จงเขียนโปรแกรมชื่อ EX1_2 เพื่อหาพื้นที่วงกลม 1 วงกลม โดยคำนวณการรับค่ารัศมีจากผู้ใช้
3. จงเขียนโปรแกรมชื่อ EX1_3 เพื่อหาพื้นที่วงกลมหลาย ๆ วง โดยคำนวณการรับค่ารัศมีจากผู้ใช้จนกระทั่งผู้ใช้ระบุค่ารัศมี = 0
4. จงเขียนโปรแกรมชื่อ EX1_4 เพื่อแปลงตัวเลขจำนวนเต็มให้เป็นตัวเลขโรมัน โดยให้ผู้ใช้ระบุค่าตัวเลขแสดงตัวเลขโรมันแบบโต้ตอบ ตัวอย่างเช่น ระบุ 12 แปลงเป็น XII , ระบุ 14 แปลงเป็น XIV , ระบุ 40 แปลงเป็น XL, ระบุ 3500 แปลงเป็น MMMD เป็นต้น โดยออกแบบให้มีการวนรับค่าจนกว่าผู้ใช้จะป้อนตัวเลข 0 โดยกำหนดค่าอักษรสำหรับการแปลงดังนี้

I	มีค่า 1,	V	มีค่า 5,	X	มีค่า 10
L	มีค่า 50,	C	มีค่า 100,	D	มีค่า 500
M	มีค่า 1000				

5. จงเขียนโปรแกรมชื่อ EX1_5 เพื่อสร้างรูปทรงเรขาคณิตด้วยสัญลักษณ์ * ด้วยการระบุตัวเลขเพื่อเลือกการแสดงดังนี้

ระบุ 1 แสดง	ระบุ 2 แสดง	ระบุ 3 แสดง
*********	*	*
*******	***	***
*****	*****	*****
***	*******	***
*	*********	*

6. จงเขียนโปรแกรมชื่อ EX1_6 เพื่อตัดเกรดนักเรียนจำนวน 5 คน โดยกำหนดให้ผู้ใช้ระบุชื่อ สกุล และคะแนน จากนั้นโปรแกรมตัดเกรดแบบ 5 ระดับ เมื่อรับครบ 5 คนแล้วให้แสดงทางจอภาพ พร้อมคะแนนเฉลี่ยทางจอภาพ
7. จงเขียนโปรแกรมชื่อ EX1_7 เพื่อคิดค่าไฟฟ้า โดยรับค่าเลขอ่านไฟฟ้าครั้งก่อน และค่าเลขอ่านไฟฟ้าครั้งหลัง พร้อมทั้งค่า FT เมื่อรับค่าครบถ้วนให้คำนวณตามลำดับดังนี้
   (1) หาค่าฐานไฟฟ้า = ค่าเลขอ่านไฟฟ้าครั้งหลัง – ค่าเลขอ่านไฟฟ้าครังก่อน
   (2) หาค่าไฟฟ้าจริง โดยพิจารณาจากช่วงค่าฐานไฟฟ้าดังนี้

ถ้าฐานไฟฟ้า > 300	คิดหน่วยละ	3	บาท
ถ้าฐานไฟฟ้า 201-300	คิดหน่วยละ	2.50	บาท
ถ้าฐานไฟฟ้า 101-200	คิดหน่วยละ	1	บาท
ถ้าฐานไฟฟ้า 0-100	คิดหน่วยละ	0.50	บาท

   (3) หาค่าเงิน FT เพิ่ม = ค่าไฟฟ้าจริง (ข้อ 2) มาคูณกับค่า FT
   (4) หาค่าไฟฟ้าที่ต้องชำระทั้งสิ้น = ค่าไฟฟ้าจริง + ค่าเงิน FT เพิ่ม
8. จงเขียนโปรแกรมชื่อ EX1_8 เพื่อคิดโบนัสของพนักงานประจำปี กำหนดให้รับชื่อพนักงาน เงินเดือน เพศ อายุ และจำนวนวันลา โดยคิดโบนัสตามขั้นตอนดังนี้
   (1) ถ้าพนักงานเป็นเพศชาย ให้โบนัสเป็น 5 เท่าของเงินเดือน แต่ไม่เกิน 50,000 บาท ถ้าพนักงานเป็นหญิง ให้โบนัส 4 เท่าของเงินเดือน แต่ไม่เกิน 40,000 บาท
   (2) สำหรับพนักงานอายุ > 50 ปี และเป็นเพศหญิงเพิ่มเงินโบนัสพิเศษอีก 20% ของเงินเดือน แต่ถ้าเป็นเพศชายให้ 15% ของเงินเดือน ส่วนพนักงานอายุไม่เกิน 50 ปี ให้เงินโบนัสพิเศษเพียง 10% ของเงินเดือน

เมื่อคิดโบนัสครบถ้วนให้แสดงจำนวนเงินโบนัสที่ควรได้ทั้งหมด ทั้งแบบตัวเลข และอักษรตัวเงิน เช่น 5,700 บาท (ห้าพันเจ็ดร้อยบาทถ้วน)