บทที่ 7
แบบชนิดข้อมูลอย่างง่าย
(Simple Data Types)

7.1 แบบชนิดข้อมูลตัวเลข : Real และ Integer

7.2 แบบชนิดข้อมูล Boolean

7.3 แบบชนิดข้อมูล Character

7.4 ฟังก์ชัน Ordinal และฟังก์ชัน Character

7.5 ชนิดพิสัยย่อย (Subrange)

7.6 ชนิดใช้แทนกันได้และการกำหนดค่าใช้แทนกันได้ (Type Compatibility และ Assignment Compatibility)

7.7 ชนิดแจงนับ (Enumerated)

7.8 ข้อผิดพลาดร่วมของการเขียนโปรแกรม

ขณะนี้เราจะศึกษาแบบชนิดข้อมูลมาตรฐานของ Pascal อย่างใกล้ชิดมากขึ้น ได้แก่ Integer, Real, Char และ Boolean รวมทั้งอธิบายฟังก์ชันนิยามแล้วบางตัวสำหรับการประมวลผลชนิดเชิงอันดับที่

ไม่มีภาษาเขียนโปรแกรมใดๆ ที่จะสามารถให้นิยามก่อน (predefined) แบบชนิดข้อมูลทั้งหมดที่โปรแกรมเมอร์อาจต้องใช้ ดังนั้น Pascal จึงยอมให้โปรแกรมเมอร์สร้างแบบชนิดข้อมูลใหม่ได้ ในบทนี้เราจะเรียนรู้การประกาศแบบชนิดข้อมูลใหม่อีกสองชุดว่าทำอย่างไร ได้แก่ ชนิดพิสัยย่อย (subrange) และชนิดแจงนับ (enumerated)

แบบชนิดข้อมูลที่กล่าวข้างต้นทั้งหมดนี้ (standard types, subrange types และ enumerated types) เรียกว่า**แบบชนิดข้อมูลอย่างง่าย** (simple) **หรือสเกลาร์** (scalar) เพราะว่ามีเพียงหนึ่งค่าเท่านั้นซึ่งเก็บในตัวแปรหนึ่งตัวของแต่ละชนิด

แบบชนิดข้อมูลอย่างง่ายหรือสเกลาร์ หมายถึง แบบชนิดข้อมูลซึ่งใช้เก็บหนึ่งค่า (A simple or scalar data type is a data type use to store a single value.)

7.1 แบบชนิดข้อมูลตัวเลข : Real และ Integer (Numeric Data Types : Real and Integer)

แบบชนิดข้อมูล Integer และ Real แทนสารสนเทศตัวเลข ถึงแม้ว่า ข้อมูลตัวเลขส่วนใหญ่เป็นจำนวนจริง โดยปกติโปรแกรมเมอร์ใช้จำนวนเต็มเป็นตัวนับส่วนวนซ้ำ (loop counters) และใช้แทนข้อมูลจำนวนเต็ม เช่น คะแนนสอบ

ความแตกต่างระหว่างชนิดตัวเลข (Differences Between Numeric Types)

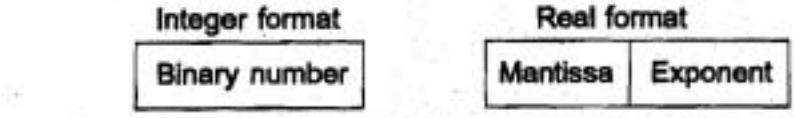
เราสามารถแทนจำนวนเต็มเป็นจำนวนจริงซึ่งมีภาคเศษส่วน (fractional part) เป็น 0

ทำไม Pascal จึงจำเป็นต้องมีแบบชนิดข้อมูลตัวเลขสองชนิด บนเครื่องคอมพิวเตอร์จำนวนมากการดำเนินการกับจำนวนเต็มเร็วกว่าการดำเนินการกับเลขจำนวนจริง และจำนวนเต็มใช้เนื้อที่หน่วยเก็บน้อยกว่าเลขจำนวนจริง การดำเนินการกับจำนวนเต็มเที่ยงตรง (precise) เสมอ ในขณะที่การดำเนินการกับเลขจำนวนจริงอาจมีบางส่วนของความแม่นยำ (accuracy) หายไป

ความแตกต่างเหล่านี้เป็นผลมาจากวิธีที่ตัวเลขถูกแทนในหน่วยความจำของคอมพิวเตอร์ ข้อมูลทั้งหมดถูกแทนในหน่วยความจำเป็นสายอักขระฐานสอง (binary strings) คือเป็นสายของเลข Os และเลข Is แต่สายอักขระฐานสองซึ่งเก็บจำนวนเต็ม เช่น 13 แตกต่างจากสายอักขระฐานสองซึ่งเก็บจำนวนจริง 13.0

การแทนที่ภายในจริงขึ้นอยู่กับเครื่องคอมพิวเตอร์ และปอยครั้งเลขจำนวนจริงใช้จำนวนไบต์ (bytes) ของหน่วยความจำคอมพิวเตอร์มากกว่า

เมื่อเปรียบเทียบรูปแบบของจำนวนเต็มและจำนวนจริงซึ่งแสดงในรูป 7.1 ในรูปแสดงให้เห็นว่าเลขจำนวนเต็มแทนด้วยเลขฐานสอง ตัวอย่างเช่น จำนวนเต็ม 13 แทนด้วยเลขฐานสอง 0...01101 บิตซ้ายมือสุดแทนเครื่องหมายของเลข เลขจำนวนเต็มบวกมีบิตเครื่องหมาย (sign bit) เป็น 0 แต่เลขจำนวนเต็มลบ มีบิตเครื่องหมาย เป็น 1, **Turbo Pascal ใช้ 16 บิต เพื่อเก็บเลขจำนวนเต็ม**



รูป 7.1 รูปแบบ Integer และ Real

รูปแบบ Real (เรียกว่า รูปแบบ floating - point) คล้ายกับสัญกรณ์วิทยาศาสตร์ (scientific notation) พื้นที่หน่วยเก็บของเลขจำนวนจริงแบ่งออกเป็นสองส่วน คือ แมนทิสซา (mantissa) และเลขชี้กำลัง (exponent) mantissa คือ เศษฐานสอง มีค่ามากกว่าหรือเท่ากับ 0.0 และน้อยกว่า 1.0 ส่วนที่เป็นเลขชี้กำลัง คือ กำลังสองของ 2 ดังนั้นสูตรที่ถูกต้อง ได้แก่

real number = mantissa x $2^{exponent}$

เนื่องจากเซลล์หน่วยความจำมีขนาดจำกัด เลขจำนวนจริงไม่ใช่ทั้งหมด ซึ่งอยู่ในพิสัย (range) ของ reals สามารถแทนได้อย่างถูกต้อง

Turbo Pascal ใช้ 48 บิตเพื่อเก็บเลขจำนวนจริง

เลขจำนวนจริงไม่เหมือนเลขจำนวนเต็ม เพราะมีภาคเศษส่วนและพิสัยของมันมีขนาดใหญ่กว่าพิสัยของจำนวนเต็ม ใน Turbo Pascal พิสัยของเลขจำนวนจริงมีค่าจาก 10^{-39} ถึง 10^{+38}

เลขจำนวนเต็มบวก เล็กที่สุดคือ 1, เลขใหญ่ที่สุด MaxInt คือ 32767

จำนวนเต็มลบ ใหญ่ที่สุดคือ -MaxInt-1 หรือ -32768

ความไม่ถูกต้องเชิงตัวเลข (Numerical Inaccuracies)

ปัญหาข้อหนึ่งของการประมวลผลเลขจำนวนจริง คือ โอกาสผิดพลาดในการแทนข้อมูลจริง เช่นกับเลขบางตัวไม่สามารถแทนอย่างแม่นยำในระบบเลขฐานสอง (ตัวอย่างเช่น 1/3 คือ 0.333333 ...) ดังนั้นเลขบางตัวไม่สามารถแทนอย่างแม่นยำในรูปแบบ real ขนาดของข้อผิดพลาดการแทนที่ (representational error) นี้ขึ้นอยู่กับจำนวนของเลขฐานสอง (bits) ซึ่งใช้ใน mantissa จำนวนบิตที่มากขึ้น ข้อผิดพลาดจะน้อยลง

ข้อผิดพลาดการแทนที่ หมายถึง ข้อผิดพลาดซึ่งเกิดจากการลงรหัสไม่แม่นยำของเลขฐานสิบในรูปแบบฐานสอง (Representation error is an error due to imprecise coding of a decimal number in binary form.)

ตัวอย่างเช่น เลขฐานสิบ 0.1 เป็นเลขจำนวนจริง ซึ่งไม่สามารถแทนอย่างแม่นยำในระบบเลขฐานสอง ให้ผลลัพธ์เป็นข้อผิดพลาดเล็กน้อย ซึ่งอาจจะขยาย (magnified) ผ่านการคำนวณแบบทำซ้ำ ด้วยเหตุนี้ เนื่องจากการบวก 0.1 สิบครั้ง ผลลัพธ์ไม่ใช่ 1.0 การวนซ้ำข้างล่างนี้อาจจบ (terminate) ไม่ได้บนคอมพิวเตอร์บางเครื่อง

```
Trial := 0.0;
while Trial < > 1.0 do
```

```
begin
  . . .
  Trial := Trial + 0.1
end {while}
```

ถ้าการทดสอบการทำซ้ำลูปเปลี่ยนเป็น Trial < 1.0 ลูปอาจกระทำการ 10 ครั้งบนคอมพิวเตอร์หนึ่งเครื่อง และกระทำการ 11 ครั้งบนคอมพิวเตอร์อีกหนึ่งเครื่อง เพื่อหลีกเลี่ยงปัญหานี้ ให้ใช้ตัวแปรจำนวนเต็ม เมื่อใดก็ตามที่เป็นไปได้ในการทดสอบการทำซ้ำลูป

ข้อผิดพลาดอื่นๆ เกิดขึ้นเมื่อการจัดดำเนินการกระทำกับเลขจำนวนจริงใหญ่มากและเล็กมาก การบวกเลขขนาดใหญ่และเลขขนาดเล็กมากๆ อาจทำให้เลขขนาดใหญ่กว่ายกเลิก (cancel out) เลขตัวเล็กกว่า ผลลัพธ์คือ ข้อผิดพลาดการยกเลิก ถ้า X มีขนาดใหญ่กว่า Y มาก แล้ว X+Y อาจมีผลลัพธ์เป็นค่าเดียวกับ X

ตัวอย่างเช่น 100000.0 + 0.000001234 มีค่าเท่ากับ 100000.0 บนคอมพิวเตอร์บางเครื่อง

ข้อผิดพลาดจากการยกเลิก หมายถึง ข้อผิดพลาดการคำนวณ เกิดจากการประมวลผลเลขที่มีขนาดใหญ่กับเลขที่มีขนาดเล็กมากๆ (Cancellation error is a computational error caused by processing a large number with a much smaller number.)

ผลลัพธ์ของการคูณเลขที่มีขนาดเล็กมากๆ สองตัวอาจเป็นปริมาณที่เล็กมาก ซึ่งแทนเป็นศูนย์ (น้อยเกินเก็บเลข) (arithmetic underflow) ในทำนองเดียวกัน ถ้าเลขขนาดใหญ่มากสองตัวคูณกัน ผลลัพธ์อาจมีขนาดใหญ่มากเกินกว่าที่จะแทนได้ (มากเกินเก็บเลข (arithmetic overflow))

มากเกินเก็บเลข สามารถเกิดขึ้นได้ แม้แต่เมื่อการประมวลผลกระทำกับค่าจำนวนเต็มค่อนข้างเล็ก ตัวอย่างเช่น ถ้า Hours คือ 24 Min คือ 60 และ Sec คือ 60 ข้อความสั่ง

```
SecInDay := Hours * Mn * Sec
```

กำหนดจำนวนวินาทีในหนึ่งวัน (86400) ให้กับ SecInDay เนื่องจากผลลัพธ์มีค่าใหญ่เกินไปสำหรับชนิด Integer ค่าไม่ถูกต้อง (20864) จึงถูกเก็บแทนและไม่มีการแสดงข้อผิดพลาดใดๆ เพื่อหลีกเลี่ยงข้อผิดพลาดนี้ ใน Standard Pascal เราควรใช้ตัวแปรชนิด Real สำหรับ Turbo Pascal มีวิธีแก้ปัญหาอีกอย่างหนึ่งซึ่งจะได้อภิปรายในหัวข้อถัดไป

น้อยเกินเก็บเลข หมายถึง ข้อผิดพลาดเกิดจากการแทนที่ไม่ถูกต้องของผลลัพธ์การคำนวณเลขขนาดเล็กเป็นศูนย์ (Arithmetic underflow is an error cause by incorrectly representing a small computational result as zero.)

มากเกินเก็บเลข หมายถึง ข้อผิดพลาดเกิดจากผลลัพธ์ของการคำนวณมีค่าใหญ่เกินกว่าที่จะแทนที่ได้ (Arithmetic overflow is an error caused by a computational result that is too large to be represented.)

ชนิดตัวเลขเพิ่มเติมใน Turbo Pascal (Additional Numeric Types in Turbo Pascal)

Turbo Pascal มีแบบชนิดข้อมูล LongInt ซึ่งมีค่าระหว่าง -2147483648 และ 2147483647 ตัวแปรชนิด LongInt ใช้หน่วยเก็บ 32 บิตในการเก็บตัวเลข ในขณะที่ตัวแปรชนิด Integer ใช้เนื้อที่ 16 บิตในการเก็บตัวเลข

ตาราง 7.1 แสดงรายการแบบชนิดข้อมูล Integer เพิ่มเติมใน Turbo Pascal พร้อมกับพิสัยของค่าซึ่งสามารถเก็บในตัวแปรของแต่ละชนิด

ตาราง 7.1 แบบชนิดข้อมูล Integer

Type	Ranges
Byte	0..255
ShortInt	-128..127
Integer	-32768..32767
Word	0..65535
LongInt	-2147483648..2147483647

แบบชนิดจำนวนเต็มสองตัว, Byte และ ShortInt ใช้เนื้อที่เพียง 8 บิตต่อค่า ข้อมูลซึ่งมีชนิดเป็น Word เก็บได้เฉพาะจำนวนเต็มบวกเท่านั้น ดังนั้นค่าใหญ่สุดของมันจึงเป็นสองเท่าของแบบชนิด Integer ตัวดำเนินการทุกตัวของ Turbo Pascal ซึ่งใช้กับชนิด Integer และสามารถใช้กับแบบชนิดข้อมูลที่แสดงรายการในตาราง 7.1 ได้ด้วย

Turbo Pascal ยังมีแบบชนิดข้อมูลเพิ่มเติมอีกหลายตัว สำหรับเลขจำนวนจริง แบบชนิดข้อมูลเหล่านี้แสดงรายการในตาราง 7.2 พร้อมกับพิสัยของค่าสำหรับแต่ละชนิด และจำนวนเลขนัยสำคัญ (significant digits) สำหรับแต่ละชนิดตัวดำเนินการทุกตัวของ Turbo Pascal ซึ่งใช้กับชนิด Real สามารถใช้กับแบบชนิดข้อมูล ที่แสดงรายการในตาราง 7.2 ได้ด้วย

ตาราง 7.2 แบบชนิดข้อมูล Real

Type	Range	Significant Digits
Single	1.5 E-45 .. 3.4E38	7-8
Real	2.9E-39 .. 1.7E38	11-12
Double	5.0E-324 .. 1.7E308	15-16
Extended	1.9E-4957 .. 1.1E4932	19-20

ขณะที่ชนิด Integer เพิ่มเติมมีให้ใช้ได้เสมอใน Turbo Pascal เราอาจต้องมีคำสั่งชี้แนะคอมไพเลอร์ (compiler directives) พิเศษ เพื่อที่จะใช้ชนิด extended Real

คำสั่งชี้แนะคอมไพเลอร์มีความจำเป็นขึ้นอยู่กับว่าคอมพิวเตอร์ของเรามีตัวประมวลผลร่วมตัวเลข (numeric coprocessor) หรือไม่

ตัวประมวลผลร่วมตัวเลข หมายถึง ชิปคอมพิวเตอร์ซึ่งใช้สำหรับกระทำการดำเนินการคำนวณ floating-point (A numeric coprocessor is a computer chip used for performing floatng-point arithmetic operations.)

ถ้ามีการติดตั้งตัวประมวลผลร่วมตัวเลข ให้ใช้คู่คำสั่งชี้แนะคอมไพเลอร์ {$N+, E-} ซึ่งสั่งคอมไพเลอร์ให้กระทำการประมวลผลตัวเลข floating-point โดยเรียกกระบวนงาน ซึ่งใช้ประโยชน์ (utilize) ตัวประมวลผลร่วม

ถ้าไม่มีการติดตั้งตัวประมวลผลร่วม ให้ใช้คู่คำสั่งชี้แนะ {$N+, E+} ซึ่งสั่งคอมไพเลอร์กระทำการคำนวณ floating-point โดยเรียกกระบวนงาน ซึ่งเลียนแบบ (emulate) (จำลองด้วยซอฟต์แวร์) ตัวประมวลผลร่วม

Syntax Display

คำสั่งชี้แนะคอมไพเลอร์สนับสนุนตัวเลข (Numeric Support Compiler Directive)

Form : {$N-} หรือ {$N+}

Default : {$N-}

มีความหมายดังนี้ : เมื่อเป็น passive (ค่า -) คอมไพเลอร์สร้างรหัสให้กระทำการคำนวณ floating-point ซึ่งไม่ต้องใช่ตัวประมวลผลร่วมตัวเลข เมื่อเป็น active (ค่า +) คอมไพเลอร์สร้างรหัสเพื่อให้กระทำการคำนวณ floating-point โดยใช้ตัวประมวลผลร่วมตัวเลข

โปรดสังเกต {$N+} ต้องมีเสมอ เพื่อใช้แบบชนิดข้อมูล extended Real ของ Turbo Pascal : Single, Double หรือ Extended

Syntax Display

การเลียนแบบคำสั่งชี้แนะคอมไพเลอร์ (Emulation Compiler Directive)

Form : {$E-} หรือ {$E+}

Default : {$E+}

มีความหมายดังนี้ : เมื่อเป็น active (ค่า +) ตัวประมวลผลร่วมตัวเลขถูกเลียนแบบ เมื่อเป็น passive (ค่า -) ตัวประมวลผลร่วมตัวเลขถูกใช้โดยตรง

แบบฝึกหัด 7.1

1. สมมติว่าคอมพิวเตอร์แสดงข้อผิดพลาดยกเลิก (cancellation error) เช่น 1000.00 + 0.0001234 มีค่าเท่ากับ 1000.00

จงอธิบายและบอกผลลัพธ์ของข้อความสั่งกำหนดค่าข้างล่างนี้ พร้อมทั้งอธิบาย

X := 100.0001234;

เขียนโปรแกรม

1. เนื่องจากการวนซ้ำในหัวข้อความไม่ถูกต้องเชิงตัวเลข อาจดำเนินการไม่ถูกต้อง จงเขียนลูปซึ่งเอาต์พุตเป็นเศษส่วนฐานสิบ (decimal fractions) จาก 0.0 ถึง 1.0 เพิ่มขั้นละ 0.1 โดยใช้จำนวนเต็มเป็นตัวนับลูป (loop counter)

7.2 แบบชนิดข้อมูล Boolean (The Boolean Data Type)

ตัวแปรแบบบูลมีสองค่า ได้แก่ True หรือ Fals ซึ่ง Pascal แทนเป็นเลขฐานสอง 1 และ 0 ตามลำดับ เราสามารถใช้ตัวกำหนดค่า (assignment operator) กับข้อมูลแบบบูล และตัวดำเนินการแบบบูล (Boolean operators) and, or, not นิพจน์แบบบูล (Boolean expressions) ใช้ในข้อความสั่ง if, while และ repeat

โปรแกรมเมอร์เขียนนิพจน์แบบบูลซึ่งเรียกฟังก์ชันแบบบูล เป็นการสนับสนุนโปรแกรมให้น่าอ่าน

ฟังก์ชันแบบบูลกลับคืน (return) ค่า True หรือ False

ตัวอย่าง 7.1 ฟังก์ชันแบบบูล (Boolean Functions)

อัลกอริทึมข้างล่างนี้ประกอบด้วยขั้นตอนตัดสินใจในตัวอย่างนี้ เราไม่สนใจรายละเอียดของกระบวนงานทั้งสามชุด ซึ่งถูกเรียกในข้อความสั่ง if เราเพียงต้องการให้แน่ใจว่าเรียกกระบวนงานที่ถูกต้อง

```
if Ch is a letter then
   Call procedure ProcessLetter
else if Ch is a digit character then
   Call procedure ProcessDigit
else
   Call procedure ProcessSpecial
```

ข้อความสั่ง if ที่ตามมาปฏิบัติให้เกิดผลของอัลกอริทึมนี้ นิพจน์แบบบูลตัวแรกเรียกฟังก์ชัน IsLetter (รูป 7.2) เพื่อตรวจว่า Ch เป็นตัวอักษรหรือไม่ นิพจน์แบบบูลตัวที่สอง เรียกฟังก์ชัน IsDigit เพื่อตรวจว่า Ch เป็นเลขโดดหรือไม่

function designator IsLetter (Ch) กลับคืนค่า True ถ้า Ch เป็นตัวอักษร และกลับคืนค่า False ถ้า Ch ไม่ใช่ตัวอักษร ในทำนองเดียวกัน function designator IsDigit กลับคืนค่า True ถ้า Ch เป็นอักขระเลขโดด และกลับคืนค่า False ในกรณีอื่นๆ

```
if IsLetter(Ch) then
   ProcessLetter(Ch)
else if IsDigit(Ch) then
   ProcessDigit(Ch)
```

```
else
   ProcessSpecial(Ch)

function IsLetter(Ch : Char) : Boolean;
{
Returns True when its argument is a letter;
otherwise, returns False
}
begin {IsLetter}
   IsLetter := (('A' <=Ch) and (Ch <='Z')) or (('a' <= Ch) and (Ch <= 'z'))
end; {IsLetter}

function IsDigit (Ch : Char) : Boolean;
{
Returns True when its argument is a digit character; otherwise, returns False
}
begin {IsDigit}
   IsDigit := ('0' <= Ch) and (Ch <= '9')
end; {IsDigit}
```

รูป 7.2 ฟังก์ชัน IsLetter และฟังก์ชัน IsDigit

ฟังก์ชัน IsLetter และฟังก์ชัน IsDigit ประกอบด้วยหนึ่งข้อความสั่งซึ่งกำหนดค่าแบบบูลให้กับชื่อฟังก์ชัน (function name) นี่คือการนิยามผลลัพธ์ของฟังก์ชัน ค่าที่กำหนด (True หรือ False) ขึ้นอยู่กับผลลัพธ์ของการเปรียบเทียบตัวอักขระระบุโดยนิพจน์แบบบูล เราจะอภิปรายการเปรียบเทียบตัวอักขระในหัวข้อถัดไป

แบบฝึกหัด 7.2

1. จงประเมินผลนิพจน์แบบบูลแต่ละข้อข้างล่างนี้เมื่อ Ch1 คือ 'a' และ Ch2 คือ '3'

a) IsLetter(Ch1) and IsDigit(Ch2)

b) IsLetter(Ch1) or IsDigit(Ch2)

c) IsLetter(Ch1) and IsDigit(Ch1)

d) IsLetter(Ch1) or IsDigit(Ch1)

เขียนโปรแกรม

1. จงเขียนฟังก์ชันแบบบูลซึ่งมีพารามิเตอร์ชนิด Integer สองตัว ชื่อ M และ N ให้กลับคืน True เมื่อ M เป็นตัวหารของ N (M หาร N ลงตัว) และกลับคืน False กรณีอื่นๆ

ข้อสังเกต : ศูนย์ไม่ใช่ divisor ของเลขใดๆ

7.3 แบบชนิดข้อมูล Character (The Character Data Type)

แบบชนิดข้อมูลตัวอักขระ (Char) สามารถเก็บและจัดดำเนินการอักขระแต่ละตัว เช่น ประกอบกันเป็นชื่อคน ที่อยู่ และข้อมูลส่วนตัวอื่นๆ สัญพจน์ (literal) ชนิด Char ประกอบด้วยอักขระหนึ่งตัว (ตัวอักษร, เลขโดด เครื่องหมายกำกับวรรคตอน เป็นต้น) อยู่ภายในเครื่องหมาย apostrophes

ตัวอย่างเช่น ถ้า Next มีชนิดเป็น Char, ข้อความสั่งกำหนดค่า

Next := 'A'

เก็บตัวอักษร A ในตัวแปร Next

เราสามารถเปรียบเทียบข้อมูลตัวอักขระโดยใช้ตัวดำเนินการสัมพันธ์ (relational operators) ถ้า Next และ First มีชนิดเป็น Char, นิพจน์แบบบูล

Next = First

Next <> First

คือการตรวจสอบว่าตัวแปรอักขระสองตัวนี้มีค่าเหมือนกัน หรือมีค่าแตกต่างกัน ตัวดำเนินการสัมพันธ์ <, <=, > และ >= มีไว้ให้เราทำการเปรียบเทียบอันดับบนข้อมูลตัวอักขระ

การแปลความหมายผลลัพธ์ของการเปรียบเทียบอันดับ เราต้องทราบบางสิ่งเกี่ยวกับวิธีซึ่งตัวอักขระต่างๆ ถูกแทนภายในคอมพิวเตอร์ ตัวอักขระแต่ละตัวมีรหัสตัวเลขที่เป็นหนึ่งเดียวของมันเอง ซึ่งรูปแบบฐานสองของมันเก็บในเซลล์หน่วยความจำ มีค่าเป็นตัวอักขระ (Each character has its own unique numeric code whose binary form is stored in a memory cell that has a character value.)

เลขฐานสองเหล่านี้ถูกเปรียบเทียบโดยตัวดำเนินการสัมพันธ์

รหัสตัวอักขระที่ใช้โดย Turbo Pascal เรียกว่า **แอสกี** (รหัสมาตรฐานของสหรัฐอเมริกาเพื่อการสับเปลี่ยนสารสนเทศ) (ASCII (American Standard Code for Information Interchange))

ค่าของรหัสแอสกี แสดงรายการในภาคผนวก D ตัวอักขระเลขโดด '0' ถึง '9' เป็นเลขลำดับเพิ่มขึ้น (increasing sequence) ของตัวอักขระติดต่อกัน ใน ASCII และ**มีค่ารหัสเป็น 48 ถึง 57 (เลขฐานสิบ)** ความสัมพันธ์เชิงอันดับข้างล่างนี้เป็นคุณสมบัติสำหรับอักขระเลขโดด

'0' < '1' < '2' < '3' < '4' < '5' < '6' < '7' < '8' < '9'

ตัวอักษรตัวใหญ่ (uppercase letters) และอักษรตัวเล็ก (lowercase letters) เป็นลำดับเพิ่มขึ้นของตัวอักขระติดต่อกัน ใน ASCII เช่นกัน ใน ASCII, **อักษรตัวใหญ่มีค่ารหัสเลขฐานสิบเป็น 65 ถึง 90** และ**อักษรตัวเล็กมีรหัสเลขฐานสิบเป็น 97 ถึง 122** ความสัมพันธ์เชิงอันดับข้างล่างนี้เป็นคุณสมบัติของตัวอักษรใหญ่และตัวอักษรเล็ก

'A' < 'B' < 'C' < 'D' < 'E' < ... < 'X' < 'Y' < 'Z'

'a' < 'b' < 'c' < 'd' < 'e' < ... < 'x' < 'y' < 'z'

ใน ASCIIม ตัวอักขระพิมพ์ได้ (printable characters) มีรหัสเป็นเลขจาก 32 (รหัสสำหรับตัวว่าง (blank) หรืออักขระว่าง (space)) ถึง 126 (รหัสสำหรับสัญลักษณ์ ~) รหัสส่วนที่เหลือปกติแทนตัวอักขระควบคุมพิมพ์ไม่ได้ (nonprintable control characters) ถึงแม้ว่า Turbo Pascal มีสัญลักษณ์พิมพ์ไม่ได้ สำหรับตัวอักขระเหล่านี้เมื่อมันถูกแสดงผลบนจอภาพ

การส่งตัวอักขระควบคุมไปยังอุปกรณ์เอาต์พุตทำให้อุปกรณ์ตัวนั้นกระทำการดำเนินการพิเศษ เช่น การกลับคืนตัวชี้ตำแหน่ง (cursor) ไปยังสดมภ์ที่ 1, เลื่อนตัวชี้ตำแหน่งไปยังบรรทัดถัดไป หรือทำให้เกิดเสียงกระดิ่ง

แบบฝึกหัด 7.3

1. จงประเมินผลนิพจน์แบบบูลข้างล่างนี้

   'A' < 'a'

   'A' <> 'a'

   'Z' > 'A'

   'Z' > 'a'

   'O' <> 'o'

   '0' <= '0'

   '3' <= '9'

   '9' <= 'A'

2. ข้อความสั่งข้างล่างนี้ทำอะไร

```
for Ch := 'A' to 'Z' do
   Write (Ch)
```

7.4 ฟังก์ชันเชิงอันดับที่และฟังก์ชันตัวอักขระ (Ordinal Functions and Character Functions)

แบบชนิดข้อมูลเชิงอันดับที่ (Ordinal data types)

แบบชนิดข้อมูล Integer, Boolean, และ Character จัดอยู่ในกลุ่ม แบบชนิดข้อมูลเชิงอันดับที่

ค่าของแบบชนิดข้อมูลเชิงอันดับที่ สามารถแสดงรายการเรียงอันดับได้เสมอ แต่ละค่ามีตัวหน้าหนึ่งตัว (ยกเว้นตัวแรก) และมีตัวหลังหนึ่งตัว (ยกเว้นตัวสุดท้าย)

ตัวอย่างเช่น ตัวหน้าของ 5 คือ 4 และตัวหลังของ 5 คือ 6 อันดับหรือลำดับของแบบชนิดข้อมูลเชิงอันดับที่ถูกนิยามแล้ว

ตัวอย่างเช่น -MaxInt-1 คือจำนวนเต็มค่าเล็กที่สุด

MaxInt คือจำนวนเต็มค่าใหญ่ที่สุด

และเลขจำนวนเต็ม ได้แก่

-MaxInt-1, -MaxInt, ..., -1, 0, 1, ... MaxInt-1, MaxInt

อันดับของค่าแบบบูลคือ False, True

แบบชนิดข้อมูลเชิงอันดับที่ หมายถึง แบบชนิดข้อมูลซึ่งมีเซตจำกัดของค่าต่างๆ ที่สามารถแสดงรายการเรียงอันดับจากตัวแรกไปยังตัวสุดท้าย (Ordinal data type is a data type having a finite set of values that can be listed in order from the first to the last.)

ฟังก์ชันเชิงอันดับที่ : Ord, Pred และ Suce (Ordinal Functions : Ord, Pred, and Suce)

ฟังก์ชัน Ord ของ Pascal กำหนดเลขเชิงอันดับที่ (ordinal number) หรือตำแหน่งสัมพัทธ์ (relative position) ของค่าเชิงอันดับที่ในลำดับของค่าของมัน ถ้าพารามิเตอร์ของ Ord เป็นจำนวนเต็ม จะกลับคืนเลขเชิงอันดับที่คือเลขจำนวนเต็มตัวมันเอง สำหรับค่าเชิงอันดับที่อื่นๆ ทั้งหมด เลข ordinal ของค่าแรกในลำดับคือ 0, เลข ordinal ของค่าที่สองคือ 1 เช่นนี้เรื่อยไป

สำหรับแบบชนิดข้อมูล Boolean,

Ord(False) คือ 0 และ Ord(True) คือ 1

ถ้าตัวแปร A และ B มีชนิด ordinal เหมือนกัน และ A < B เป็นจริง แล้ว Ord(A) < Ord(B) ต้องเป็นจริงด้วย

ฟังก์ชัน Pred ของ Pascal กลับคืน (returns) ตัวหน้า (predecessor) ของพารา-มิเตอร์ของมัน และฟังก์ชัน Succ กลับคืนตัวหลัง (successor) ฟังก์ชันเหล่านี้คล้าย Ord คือใช้ได้เฉพาะกับพารามิเตอร์ซึ่งเป็นชนิด ordinal เท่านั้น

ตัวอย่าง 7.2 ฟังก์ชัน Ordinal กับข้อมูล Integer และข้อมูล Boolean

ตาราง 7.3 แสดงผลลัพธ์ของการใช้ฟังก์ชัน Ord, Succ และ Pred กับพารามิเตอร์จำนวนเต็มและพารามิเตอร์แบบบูล จากตารางแสดงให้เห็นว่าค่าสุดท้ายในชนิด ordinal แต่ละตัวจะไม่มีตัวหลัง (MaxInt, True) และค่าแรกชนิด ordinal จะไม่มีตัวหน้า (-MasInt-1, False)

ตาราง 7.3 ผลลัพธ์ของฟังก์ชัน Ord, Succ และ Pred

Parameter	Ord	Succ	Pred
15	15	16	14
0	0	1	-1
-30	-30	-29	-31

Parameter	Ord	Succ	Pred
MaxInt	MaxInt	Undefined	MaxInt-1
-MaxInt-1	-MaxInt-1	-MaxInt	Undefined
False	0	True	Undefined
True	1	Undefined	False

ถึงแม้ว่าเราสามารถใช้ฟังก์ชัน ordinal กับ ordinal type ได้ทุกชนิด แต่ส่วนใหญ่เราใช้กับชนิด Char และชนิด enumerated ซึ่งจะอภิปรายในหัวข้อ 7.7

เลข ordinal ของตัวอักขระขึ้นอยู่กับรหัสชุดอักขระที่ใช้ (Turbo Pascal ใช้รหัส ASCII)

ตัวอย่าง 7.3 ฟังก์ชัน Ordinal กับข้อมูล Character

ตาราง 7.4 แสดงผลลัพธ์ของฟังก์ชัน Ord, Succ และ Pred สำหรับ ASCII จากตารางแสดงให้เห็นว่า เลขโดด '0' มีเลขเชิงอันดับที่ (ordinal number) เป็น 48 และเลขโดด '7' มีเลขเชิงอันดับที่เป็น 55 ใน ASCII

ไม่ต้องคำนึงถึงรหัสตัวอักขระที่ใช้ค่าของนิพจน์

Ord('7') - Ord('0')

เท่ากับ 7 เพราะว่าอักขระเลขโดดต้องอยู่ในลำดับติดต่อกัน

ตาราง 7.4 ผลลัพธ์ของฟังก์ชัน Ord, Succ และ Pred สำหรับ ASCII

Parameter	Ord	Succ	Pred
'C'	67	'D'	'B'
'c'	99	'd'	'b'
'0'	48	'1'	'/'
'7'	55	'8'	'6'
'y'	121	'z'	'x'
' '	32	'!'	Unprintable

ตาราง 7.4 แสดงให้เห็นว่าอักขระ 'C' มีเลขเชิงอันดับที่เป็น 67 ใน ASCII เนื่องจากอักขระ 'D' เป็นตัวหลังของอักขระ 'C' ดังนั้นอักขระ 'D' จึงต้องมีเลขเชิงอันดับที่เป็น 68 และเนื่องจากตัวอักษรอยู่ในลำดับติดต่อกันใน ASCII, ค่าของนิพจน์

Ord('C') - Ord('A')

เท่ากับ 2 ใน ASCII, อักษรตัวเล็กมีค่าของรหัสใหญ่กว่าอักษรตัวใหญ่ และความแตกต่างในค่ารหัส สำหรับทั้งสองกรณีของอักษรตัวเดียวกันเท่ากับ 32

ตัวอย่างเช่น Ord('c') - Ord('C') เท่ากับ 32

ฟังก์ชัน Character (Character Functions)

ฟังก์ชัน Chr กลับคืนตัวอักขระหนึ่งตัวเป็นผลลัพธ์ของมัน ตัวอักขระที่กลับคืนคือตัวที่มีเลขเชิงอันดับที่ของมันเป็นอาร์กิวเมนต์ของฟังก์ชัน ตัวอย่างเช่น ผลลัพธ์ของ function reference Chr(67) คืออักขระที่มีเลขเชิงอันดับที่เท่ากับ 67 (ตัวอักษร C ในรหัส ASCII)

ถ้า Ch เป็นตัวแปรชนิด Char, nested function reference Chr(Ord(Ch))

จะมีค่าเหมือนกับ Ch เพราะฉะนั้น ฟังก์ชัน Chr คือ ตัวผกผัน (inverse) ของฟังก์ชัน Ord สำหรับตัวอักขระ

ตัวอย่าง 7.4

ฟังก์ชัน LowerCase ในรูป 7.3 กลับคืนผลลัพธ์ชนิด Char-รูปแบบตัวพิมพ์เล็กของอักขระตัวพิมพ์ใหญ่ซึ่งส่งไปพารามิเตอร์ Ch ของมัน (ตัวอย่างเช่น ถ้า Ch คือ 'C', Lower Case กลับคืน "c') ถ้า Ch คือ 'c' นิพจน์แบบบูลเป็นจริง และข้อความสั่งกำหนดค่าชุดแรกถูกประเมินผลเป็น

```
Lowercase := Chr(Ord('C') - Ord('A') + ord ('a')
             Chr(67 – 65 + 97)
             Chr(99) = 'c'
```

ถ้า Ch ไม่ใช่อักษรตัวใหญ่ ฟังก์ชัน Lowercase กลับคืน Ch เป็นผลลัพธ์ของมัน

ตัวอย่าง 7.5 ลำดับรวมแฟ้มเรียง (Collating Sequence)

ลำดับรวมแฟ้มเรียง หมายถึง ลำดับของตัวอักขระจัดเรียงโดยเลขเชิงอันดับที่ โปรแกรมในรูป 7.4 พิมพ์ส่วนของลำดับรวมแฟ้มเรียงของ Turbo Pascal ตัวอักขระที่มีเลขเชิงอันดับที่ 32 ถึง 90 นับทั้งต้นและท้าย (inclusive) อักขระตัวแรกที่พิมพ์คือ อักขระว่าง (เลขเชิงอันดับที่เท่ากับ 32)

ฟังก์ชัน UpCase

ฟังก์ชันที่เป็นประโยชน์อีกตัวหนึ่งใน Turbo Pascal (แต่ไม่มีใน Standard Pascal) คือ UpCase ฟังก์ชัน UpCase กลับคืนตัวพิมพ์ใหญ่ซึ่งสมมูลกับอาร์กิวเมนต์ชนิด Chra ของมัน

ตัวอย่างเช่น UpCase ('a') กลับคืน 'A' ฟังก์ชัน UpCase กลับคืนอาร์กิวเมนต์ของมันโดยไม่มีการเปลี่ยนแปลง ถ้าอาร์กิวเมนต์ตัวนั้นไม่ใช่อักษรตัวเล็ก ตาราง 7.5 คือ การสรุปฟังก์ชันซึ่งได้แนะนำในหัวข้อนี้

```
Function LowerCase (Ch : Char) : Char;
{
   Returns the Lowercase form of its argument.
   Pre  :  None
   Post :  Returns the Lowercase equivalent of Ch if Ch is an uppercase
           letter; otherwise, returns Ch.
}
begin {LowerCase}
   if(Ch >= 'A') and (Ch <= 'Z') then
      LowerCase := Chr(Ord(Ch) – Ord ('A') + Ord('a'))
   else
      LowerCase := Ch
End; {LowerCase}
```

รูป 7.3 ฟังก์ชัน LowerCase

Edit Window

```
program Collate;
{Prints part of the collating sequence}
   const
      Min = 32;   {smallest ordinal number}
      Max = 90;   {largest ordinal number}
   var
      NextOrd : Integer;

begin {Collate}
   {Print characters Chr(32) through Chr(90).}
   for NextOrd := Min to Max do
      Write (Chr(NextOrd)); {Print next character.}
   WriteLn
end. {Collate}
```

Output Window

```
! ="#$%&'( )*+,-./0123456789:; <=>@ABCDEFGHIGKLMNOPQRSTUVWXYZ
```

รูป 7.4 การพิมพ์ส่วนของลำดับการรวมแฟ้มเรียง

ตาราง 7.5 ฟังก์ชัน Ordinal และฟังก์ชัน Character

Function	Purpose	Argument	Resuls
Chr(n)	Returns the character whose ordinal number is N	Integer	Char
Ord (N)	Returns the ordinal number of its argument	Any ordinal type	Integer
Pred(N)	Returns the predecessor of its argument	Any ordinal type	Same as argument
Succ (N)	Returns the successor of Its argument	Any ordinal type	Same as argument
UpCase(Ch)	Returns uppercase equivalent of argument	Char	Char

แบบฝึกหัด 7.4 Self-Check

1. จงประเมินผลสิ่งต่อไปนี้
   a) Ord(True)
   b) Pred (True)
   c) Succ (False)
   d) Ord(True) – Ord (False)
2. จงประเมินผลสิ่งต่อไปนี้ สมมติว่าตัวอักษรเป็นอักขระติดต่อกัน
   a) Ord('D') – Ord('A')
   b) Ord('d') – Ord('a')
   c) Succ(Pred('a'))
   d) Chr (Ord('C'))
   e) Chr(Ord('Z') – Ord('A') + Ord('a'))
   f) Ord('7') + Ord('6')

g) Ord('7') – Ord('6')

h) Succ(Succ(Succ('d')))

i) Chr(Ord('A') + 5)

3. จงเขียน while ลูปซึ่งสมมูลกับ for ลูปข้างล่างนี้

```
for Ch := 'A' to 'Z' do
    WriteLn (Ch, Ord(Ch));
```

เขียนโปรแกรม

1. จงเขียนฟังก์ชัน UpperCase (เวอร์ชันของ UpCase ของเราเอง) ซึ่งกลับคืนเป็นค่าตัวพิมพ์ใหญ่ซึ่งสมมูลกับอาร์กิวเมนต์ ตัวอักขระของมัน หรือถ้าไม่มีให้กลับคืนค่าของอาร์กิวเมนต์ของมันเอง

7.5 ชนิดพิสัยย่อย (Subrange Types)

Pascal ยอมให้เรานิยามแบบชนิดข้อมูลของเราเอง แบบชนิดข้อมูลซึ่งนิยามโดยผู้ใช้ (user-defined data type) อันดับแรกคือชนิด subrange เมื่อ subrange นิยามเซตย่อยของค่าต่างๆ ที่เกี่ยวข้องกับแบบชนิดเชิงอันดับที่เฉพาะหนึ่งชุด (เรียกว่าชนิดแม่ข่าย (host type) หรือชนิดฐาน (base type))

subranges แสดงถึง พิสัยของค่าต่างๆ ที่ยอมให้ใช้สำหรับตัวแปรเชิงอันดับที่ เพื่อที่ Turbo Pascal จะตรวจพบได้เมื่อตัวแปรถูกกำหนดให้ด้วยค่าซึ่งไม่สมเหตุสมผลในสิ่งแวดล้อมของปัญหา

ชนิดพิสัยย่อย หมายถึง แบบชนิดข้อมูลเชิงอันดับที่ ซึ่งค่าต่างๆ ของมันเป็นเซตย่อยของแบบชนิดเชิงอันดับที่อีกหนึ่งชุด (Subrange type is an ordinal data type whose values are a subset of another ordinal type.)

ชนิดแม่ข่าย ชนิดฐาน หมายถึง แบบชนิดข้อมูล ซึ่งพิสัยของค่าต่างๆ ของมันถูกจำกัดในการประกาศชนิด subrange (Host type (base type) is the data type whose range of values is restricted in a subrange type declaration.)

ตัวอย่าง 7.6

การนิยาม subrange หรือแบบชนิดข้อมูลใหม่ใดๆ ก็ตาม เราเขียนการประกาศชนิดซึ่งขึ้นต้นด้วยคำสงวน type ตัวอย่างต่อไปนี้แสดงให้เห็นการประกาศ subrange สองชุดเหมือนกับตัวแปรของชนิดใหม่แต่ละชุด

```
type
   Letter = 'A' . . 'Z';
   DaysInMonth = 1 . . 31;
var
   NextChar : Letter ; {NextChar is an uppercase Letter.}
   InDay : DaysInMonth ; {InDay is an integer <= 31}
```

subrange ชุดแรก คือ Letter มี host type เป็น Char ดังนั้นค่าตัวอักขระใดๆ จาก 'A' ถึง 'Z' เท่านั้น จึงสามารถเก็บในตัวแปรชนิด Letter ได้ คอมพิวเตอร์อาจแสดงข้อความระบุความผิดพลาด (error message) และหยุดการกระทำการโปรแกรมถ้าเราพยายามอ่านอักขระอื่นๆ ตัวใดก็ตามไว้ในตัวแปรชนิด Letter ข้อความสั่งกำหนดค่า

```
NextChar := 'a';
```

เกิดข้อผิดพลาดขณะแปลโปรแกรม เพราะว่า ค่าตัวอักขระ 'a' ไม่อยู่ในแบบชนิดข้อมูล Letter

DaysInMonth เป็นชนิด subrange ที่มีชนิด host เป็น Integer ตัวแปรชนิด Days InMonth สามารถเก็บวันของเดือน ซึ่งเป็นเลขระหว่าง 1 ถึง 31 นับทั้งต้นและท้าย ข้อความสั่ง

```
ReadLn (InDay)
```

อ่านค่าข้อมูลไว้ใน InDay (ชนิด DaysInMonth)

ชนิดแม่ข่าย (host type) สำหรับ subrange ถูกกำหนดโดยคู่ของค่าซึ่งนิยาม subrange เลขเชิงอันดับที่ของค่าแรกต้องน้อยกว่าหรือเท่ากับเลขเชิงอันดับที่ของค่าที่สอง (the ordinal number of the first value must be less than or equal to the ordinal number of the second value.)

การดำเนินการซึ่งใช้ได้รับชนิดแม่ข่าย สามารถกระทำกับชนิด sburange ของมันได้ทุกตัว ไอเดนติไฟเออร์ชนิด subrange มีกฎสโคป (scope rules) เหมือนกับไอเดนติไฟเออร์อื่นๆ ของ Pascal

Syntax Display

การประกาศชนิดพิสัยย่อย (Subrange Type Declaration)

Form : `type subrange-type = minvalue .. maxvalue;`

ตัวอย่าง : type LowCase = 'a' .. 'z';

มีความหมายดังนี้ : แบบชนิดข้อมูลใหม่ชื่อ subrange-type ถูกนิยาม ตัวแปรชนิด subrange-type สามารถถูกกำหนดค่าจาก minvalue ถึง maxvalue นับทั้งต้นและท้าย ค่า minvalue และ maxvalue ต้องอยู่ในชนิดเชิงอันดับที่เหมือนกัน (เรียกว่า ชนิดแม่ข่าย และ Ord(minvalue) ต้องน้อยกว่าหรือเท่ากับ Ord(maxvalue)

ข้อสังเกต : minvalue และ maxvalue อาจเป็นไอเดนติไฟเออร์ค่าคงตัว (constant identifiers) ของแบบชนิดข้อมูลเหมือนกัน

การตรวจสอบพิสัย (Range Checking)

ข้อผิดพลาดการตรวจสอบพิสัยเกิดขึ้นเมื่อโปรแกรมพยายามที่จะเก็บค่าซึ่งเล็กเกินไปหรือใหญ่เกินไปในตัวแปร ถึงแม้ว่าชนิด subrange สามารถตรวจพบค่า out-of-range ได้ Turbo Pascal (ไม่เหมือน standard Pascal) ไม่กระทำหน้าที่นี้อัตโนมัติ การปฏิบัติให้เกิดผลการตรวจสอบพิสัย ต้องใส่ตัวชี้แนะคอมไพเลอร์ {$R+} ในโปรแกรมและคอมไพเลอร์ของ Turbo Pascal จะสร้างรหัสการตรวจสอบพิสัยจากจุดนั้นในโปรแกรม เราสามารถเปลี่ยน (turn) การตรวจสอบพิสัยทาง Options menu ได้ด้วย โดยการเลือก Compiler submenu และใส่ X ใน check box สำหรับการตรวจสอบพิสัย

Syntax Display

Range Checking Compiler Directive

Form : {$R-} หรือ {$R+}

Default : {$R-}

มีความหมายดังนี้ : ในการกำหนด {$R-}

Turbo Pascal ไม่ตรวจสอบข้อผิดพลาด subrange ระหว่างการกระทำการโปรแกรม ตัวชี้แนะคอมไพเลอร์ {$R+} ทำให้คอมไพเลอร์สร้างรหัสการตรวจสอบพิสัย และควรใช้ระหว่างการแก้จุดบกพร่อง (debugging) โปรแกรม เมื่อตัวเลือกนี้ใช้งาน การทดสอบค่าซึ่งอยู่นอกพิสัย (out-of-range values) เกิดขึ้นก่อนการกำหนดค่าแต่ละครั้งให้กับตัวแปร subrange

สไตล์โปรแกรม (Program Style)

แรงจูงใจสำหรับการใช้ subranges และการตรวจสอบพิสัย (Motivation for Using Subranges and Range Checking) โปรแกรมเมอร์ส่วนใหญ่ต้องการให้โปรแกรมหยุดการ

กระทำการทันทีขณะที่เก็บข้อมูลไม่ดี (bad data) ในตัวแปร และไม่ต้องทำการคำนวณซึ่งจะนำไปสู่ผลลัพธ์ที่ไม่มีความหมาย (meaningless results) หรือเกิดข้อผิดพลาดเวลาดำเนินงาน (run-time error) ณ จุดอื่นภายหลัง ชนิด subrange ซึ่งรวมกับการตรวจสอบพิสัย จัดหา ความสามารถนี้ใน Turbo Pascal ทันทีที่โปรแกรมพยายามเก็บค่าซึ่งอยู่นอกพิสัยในตัวแปรซึ่งชนิดเป็น subrange โปรแกรมจะหยุดและบอกว่า Range check error ระหว่างการแปลโปรแกรม Turbo Pascal จะแสดงผลข้อผิดพลาดวากยสัมพันธ์ Constant out of range ถ้าค่าคงตัวไม่ถูกต้อง (invalid constant) ถูกกำหนดให้กับตัวแปรซึ่งมีชนิดเป็น subrange

ชนิด subrange เพิ่มสมรรถนะการทำเอกสารโปรแกรม มันบอกอย่างชัดเจนให้กับผู้อ่านโปรแกรมว่าตัวแปรตัวใดมีข้อจำกัดเรื่องพิสัยของค่าต่างๆ

การเรียงอันดับการประกาศของ Pascal (The Order of Pascal Declarations)

ใน Standard Pascal การประกาศชนิดสำหรับชนิด subrange หรือแบบชนิดข้อมูลอื่นๆ ซึ่งผู้ใช้ให้นิยามใดๆ ก็ตามต้องอยู่ระหว่างการประกาศ constant และการประกาศตัวแปรในบล็อก Pascal รูปแบบของส่วนการประกาศสำหรับ standard Pascal block เป็นดังนี้

constant declarations

type declarations

variable declarations

procedure and function declarations

ถึงแม้ว่า Turbo Pascal จะไม่บังคับว่าต้องเรียงอันดับตามนี้ แต่เรายังคงต้องประกาศไอเดนติไฟเออร์แต่ละตัวก่อนการใช้ครั้งแรกของมัน สิ่งนี้หมายความว่า เราต้องประกาศแบบชนิดข้อมูลซึ่งผู้ใช้ให้นิยามก่อนการประกาศตัวแปรของชนิดนั้น

แบบฝึกหัด 7.5 Self - Check

1. Subranges ต่อไปนี้ชุดใดไม่ถูกต้อง (illegal) ให้อธิบาย

   a) 1 . . MaxInt

   b) 'A' . . 'Z'

   c) -15 . . 15

   d) 'A' . . 'z'

e) -50 . . 5.0

f) 0 . . '9'

g) 15 . . -15

h) 'ACE' . . 'HAT'

i) 'a' . . 'Z'

j) - MaxInt . . - MaxInt + 5

2. ประกาศชนิดและกระบวนงานต่อไปนี้เพื่อให้แน่ใจว่าพิสัยของเลขอินพุตถูกต้อง แต่มันให้ข้อผิดพลาดการตรวจสอบพิสัย จงอธิบายว่าทำไม

```
{$R+}
type Year = 1990 . . 2000 ; {valid years}
procedure SafeYear (var Y : Year);
begin {SafeYear}
   repeat
      Write ('Input year 1990-2000> ');
      ReadLn(Y)
   until (Y >= 1990) and (Y <= 2000)
end; {SafeYear}
```

เขียนโปรแกรม

1. จงเขียนการประกาศชนิดสำหรับข้อมูลชนิด Month ให้ถูกต้องเพื่อแทนเลขเดือน (month number) และเขียนกระบวนงาน ReadMonth ซึ่งมีพารามิเตอร์เอาต์พุต ชื่อ M (ชนิด Month) ในกระบวนงานควรแจ้งผู้ใช้ให้ใส่เดือนเป็นคำ (word) และกลับคืนเลขเดือนที่ถูกต้อง

7.6 ชนิดใช้แทนกันได้และการกำหนดค่าใช้แทนกันได้ (Type Compatibility and Assignment Compatibility)

เราได้อภิปรายการกำหนดค่าใช้แทนกันได้ในหัวข้อ 2.5 แล้ว ขณะนี้เราสามารถใช้แบบชนิดข้อมูลได้มากขึ้น จึงต้องตรวจสอบกฎต่าง ๆ ใหม่ สำหรับแบบชนิดข้อมูลใช้แทนกันได้

ชนิดเหมือนกัน (Type Identical)

ใน Turbo Pascal แบบชนิดข้อมูลสองชุด จะเป็น**ชนิดเหมือนกัน** (แบบชนิดข้อมูลเหมือนกัน (same data type)) เมื่อเงื่อนไขข้อหนึ่งข้างล่างนี้เป็นจริง

1) มีการประกาศให้แต่ละชุดสมมูล (equivalent) ซึ่งกันและกัน

2) แบบชนิดข้อมูลแต่ละชุดถูกประกาศให้สมมูลกับไอเดนติไฟเออร์ชนิดที่สาม

การประกาศชนิด

```
type
   Numbers     = Integer ;
   PosAndNeg   = Numbers ;
   IntType     = PosAndNeg ;
```

มีผลคือ ทำให้ Numbers, PosAndNeg, IntType และ Integer เป็นแบบชนิดข้อมูลเหมือนกัน อย่างไรก็ตาม Percent และ Hundred ซึ่งจะประกาศถัดไปไม่ใช่ชนิดเหมือนกัน เพราะว่าแต่ละตัวถูกประกาศให้เป็น subrange เหมือนกัน (1 . . 100) แต่ไม่ใช่ไอเดนติไฟเออร์ชนิดเหมือนกัน (not the same type identifier)

```
type
   Percent = 1 . . 100 ;
   Hundred = 1 . . 100 ;
```

ข้อสังเกต

แบบชนิดข้อมูลเหมือนกัน หมายถึง แบบชนิดข้อมูลซึ่งสมมูลกัน (Type identical data types are data types that are equivalent.)

ชนิดใช้แทนกันได้ (Type Compatibility)

แบบชนิดข้อมูลสองชุด เป็นชนิดใช้แทนกันได้ใน Turbo Pascal ถ้าเงื่อนไขต่อไปนี้มีหนึ่งเงื่อนไขเป็นจริง :

1) ทั้งคู่มีชนิดเหมือนกัน

2) แบบชนิดข้อมูลทั้งคู่เป็นชนิดจำนวนเต็ม (Byte, ShortInt, Integer, Word, LongInt) โดยไม่จำเป็นต้องเป็นแบบชนิดจำนวนเต็มเหมือนกัน

3) แบบชนิดข้อมูลทั้งคู่เป็นชนิด real (Single, Real, Double, Extended) โดยไม่จำเป็นต้องเป็นแบบชนิด real เหมือนกัน

4) ชนิดหนึ่งเป็น subrange ของอีกชนิดหนึ่ง (ตัวอย่างเช่น Letter และ char ในตัวอย่าง 7.6)

5) แบบชนิดข้อมูลทั้งคู่เป็น subrange ของชนิดแม่ข่าย (host type) เหมือนกัน

ตัวถูกดำเนินการ (operands) ซึ่งชนิดใช้แทนกันได้สามารถจัดดำเนินการด้วยกัน ตัวอย่างเช่น นิพจน์

```
NextChar < > '3'
```

ถูกต้องเชิงวากยสัมพันธ์ ตราบใดที่ NextChar เป็นชนิด Char หรือ Letter ในทางตรงกันข้าม นิพจน์

```
NextChar < > 3
```

เกิดข้อผิดพลาดวากยสัมพันธ์เพราะว่าเลขจำนวนเต็ม 3 ไม่ใช่ชนิดใช้แทนกันได้กับ NextChar

แบบชนิดข้อมูลใช้แทนกันได้ หมายถึง แบบชนิดข้อมูลต่างๆ ซึ่งสามารถใช้กับตัวดำเนินการเดียวกันได้ (Type compatible data types are data types that can be used with the same operator.)

การกำหนดค่าใช้แทนกันได้ (Assignment Compatibility)

นิพจน์จะเป็นการกำหนดค่าที่ใช้แทนกันได้กับตัวแปรใน Turbo Pascal ถ้าเงื่อนไขต่อไปนี้มีหนึ่งเงื่อนไขเป็นจริง

1) ชนิดของพวกมันเป็นชนิดเดียวกัน

2) พวกมันเป็นแบบชนิดข้อมูลใช้แทนกันได้และค่าของนิพจน์อยู่ภายในพิสัยของค่าที่เป็นไปได้สำหรับตัวแปร

3) ตัวแปรมีหนึ่งตัวเป็นชนิด real, นิพจน์หนึ่งตัวเป็นชนิด integer และค่าของนิพจน์อยู่ภายในพิสัยของค่าที่เป็นไปได้สำหรับตัวแปร

ถ้าตัวแปรและนิพจน์เป็นการกำหนดค่าใช้แทนกันได้แล้ว นิพจน์สามารถถูกกำหนดค่าให้กับตัวแปรโดยไม่มีข้อผิดพลาด

สมมติมีการประกาศดังนี้

```
type
  Letter = 'A' . . 'Z';
var
  NextCh : Letter; {NestCh is an upperease letter.}
```

ข้อความสั่งกำหนดค่า

```
NextCh := '3';
```

ทำให้เกิดข้อผิดพลาดวากยสัมพันธ์ Constant out of range เพราะว่าสัญพจน์ '3' ไม่ใช่ชนิดใช้แทนกันได้กับตัวแปร NextCh (ชนิด Letter)

ถ้า Ch เป็นชนิด Char และการตรวจสอบพิสัยทำได้โดยใช้ {$R+} ข้อความสั่งกำหนดค่า

```
NextCh := Ch;
```

จะคอมไพล์ผ่าน แต่อาจเกิด Range check error ณ เวลาดำเนินงาน ข้อผิดพลาดนี้เกิดขึ้นถ้าตัวอักขระซึ่งเก็บใน Ch ไม่ใช่อักษรตัวใหญ่

การสมนัยของพารามิเตอร์กับชนิดใช้แทนกันได้ (Parameter Correspondence and Type Compatibility)

อะไรคือสิ่งที่จำเป็นสำหรับชนิดของการสมนัยพารามิเตอร์ตัวแปร, พารามิเตอร์จริงแต่ละตัว ต้องเป็นชนิดเหมือนกับพารามิเตอร์รูปนัยซึ่งสมนัยกันของมัน สำหรับพารามิเตอร์ค่า, พารามิเตอร์จริงแต่ละตัวต้องการกำหนดค่าใช้แทนกันได้กับพารามิเตอร์รูปนัยซึ่งสมนัยกันของมัน

แบบฝึกหัด 7.6 Self-Check

1. สมมติว่า I เป็นชนิด 1 . . 10, J เป็นชนิด Integer และ K เป็นชนิด Real จงบอกว่านิพจน์ข้างล่างนี้แต่ละชุดเป็นการกำหนดค่าใช้แทนกันได้กับตัวแปรทางซ้ายมือหรือไม่ และมีข้อจำกัดอะไรบ้างที่จำเป็น เพื่อหลีกเลี่ยงข้อผิดพลาด out-of-range

a) K := 3 * I + J

b) I := 15

c) J := Trunc(K) + 2 * I

d) I := I div 2

e) I := I/J

f) I := J mod 11

g) J := 2 * K + 3

2. จงอธิบายว่าทำไมคอมไพเลอร์จึงไม่สามารถตรวจสอบ Range check error ว่าอาจเกิดขึ้น ณ เวลาดำเนินงานหรือไม่ สำหรับข้อความสั่งกำหนดค่า

7.7 ชนิดแจงนับ (Enmumerated Types)

ผลเฉลยของปัญหาเขียนโปรแกรมจำนวนมากต้องการแบบชนิดข้อมูลใหม่ (new data types) ตัวอย่างเช่น

โปรแกรมทำงบประมาณ เราอาจต้องการแยกความแตกต่างระหว่างประเภทของค่าใช้จ่าย : บันเทิง, ค่าเช่า, ค่าน้ำ-ค่าไฟ, อาหาร, เสื้อผ้า, รถยนต์, ประกันภัย, อื่นๆ ถึงแม้ว่าเราสามารถสร้างรหัสใดๆ ซึ่งเกี่ยวข้องกับบันเทิงด้วยค่าตัวอักขระของ 'e' ค่าเช่าด้วยค่าตัวอักขระ 'r' เป็นต้น Pascal ยอมให้เราสร้างชนิดแจงนับ ซึ่งแต่ละตัวแสดงรายการค่าที่มีความหมายของมันเอง

ตัวอย่างเช่น ชนิดแจงนับ Expenses มีค่าที่เป็นไปได้แปดค่าอยู่ภายในวงเล็บ

```
type
   Expense = (Entertainment, Rent, Utilities, Food, Clothing, Automobile,
              Insurance, Miscellanceous);
var
   ExpenseKind : Expenses;
```

ตัวแปร ExpenseKind (ชนิด Expenses) สามารถเก็บค่าใดค่าหนึ่งของทั้งหมดแปดค่า ข้อความสั่ง if ข้างล่างนี้ ทดสอบค่าซึ่งเก็บใน ExpenseKind

```
if  ExpenseKind = Entertainment then
  WriteLn ('Postpone until after your payday.')
else if ExpenseKind = Rent then
   WriteLn ('Pay before the fifth of the month!')
...
```

ชนิดแจงนับ หมายถึง แบบชนิดข้อมูลซึ่งรายการค่าต่างๆ ถูกกำหนดโดยโปรแกรมเมอร์ในการประกาศชนิด (Enummerated type is a data type whose list of values is specified by the programmer in a type declaration.)

ตัวอย่าง 7.7

แบบชนิดแจงนับ Day มีค่าเป็น Sunday, Monday, เช่นเรื่อยไป

```
type
   Day =   (Sunday, Monday, Tuesday, Wednesday, Thrusday, Friday,
            Saturday); {days of week}
```

ค่าที่เกี่ยวข้องกับชนิดแจงนับต้องเป็นไอเดนติไฟเออร์เสมอ เป็นตัวเลข, ตัวอักขระ หรือสัญพจน์สายอักขระ ไม่ได้

(ตัวอย่างเช่น 'Sunday' เป็นค่าของชนิดแจงนับไม่ได้)

กฎสโคปสำหรับไอเดนติไฟเออร์ใช้กับชนิดแจงนับและค่าของมัน ค่าชนิดแจงนับแต่ละตัวถือว่าเป็นไอเดนติไฟเออร์ค่าคงตัว (constant identifier) ในบล็อกที่มีข้อความสั่ง

การประกาศชนิด การประกาศชนิดต้องอยู่ก่อนการประกาศตัวแปรใดๆ ซึ่งอ้างถึงมัน

Syntax Display

การประกาศชนิดแจงนับ (Enumerated Type declaration)

Form : type enumerated-type = (identifier-list);

ตัวอย่าง

```
type
   Class = (Freshrman, Sophomore, Junior, Senior);
```

มีความหมายดังนี้ : แบบชนิดข้อมูลใหม่ชื่อ enumerate-type ถูกประกาศค่าต่างๆ เกี่ยวข้องกับชนิดนี้กำหนดใน identifier-list ค่าแต่ละตัวถูกนิยามเป็นไอเดนติไฟเออร์ค่าคงตัว (constant identifier) ในบล็อกที่มีข้อความสั่งการประกาศชนิด

ข้อสังเกต ไอเดนติไฟเออร์หนึ่งตัวปรากฏใน identifier-list เพียงหนึ่งครั้งเท่านั้นในบล็อกที่กำหนดให้

ไอเดนติไฟเออร์ไม่สามารถปรากฏในการประกาศชนิดแจงนับมากกว่าหนึ่งชุด

ถ้าชนิด Day มีการประกาศไปเรียบร้อยแล้ว การประกาศชนิด

```
type
   TDay = (Tuesday, Thursday);
```

ไม่ถูกต้อง (invalid) เพราะว่า Tuesday และ Thursday เกี่ยวข้องกับชนิด Day

ตัวดำเนินการชนิดแจงนับ (Enumerated Type Operators)

คล้ายกับชนิดมาตรฐาน Integer, Boolean และ Char ชนิดแจงนับแต่ละตัวเป็นชนิดเชิงอันดับที่ ดังนั้นอันดับของค่าของมันจึงคงที่ เมื่อชนิดแจงนับถูกประกาศ สำหรับชนิด Day ค่าแรกในรายการของมัน (Sunday) มีเลขเชิงอันดับที่เท่ากับ 1 ค่าถัดไป (Monday) มีเลขเชิงอันดับที่เท่ากับ 2 เช่นนี้เรื่อยไป ตัวดำเนินการซึ่งสามารถใช้กับชนิดแจงนับ มีตัวดำเนินการสัมพันธ์และตัวดำเนินการกำหนดค่าเท่านั้น

ความสัมพันธ์ต่อไปนี้ทั้งหมดเป็นจริง

Sunday < Monday

Wednesday = Wednesday

Wednesday > Tuesday

Entertainment < Rent

ข้อความสั่งกำหนดค่าสามารถให้นิยามค่าของตัวแปรซึ่งชนิดของมันคือชนิดแจงนับ ในตัวอย่างต่อไปนี้ สมมติว่า Day มีการให้นิยามแล้วก่อนหน้านี้เป็นชนิดแจงนับ การประกาศตัวแปรระบุว่า Today และ Tomorrow เป็นชนิด Day

```
var
   Today,              {current day of the week}
   Tomorrow : Day;     {day after today}
```

ตัวแปรสามารถถูกกำหนดเป็นค่าใดๆ ก็ได้ในรายการ ซึ่งประกาศชนิด Day ดังนั้น ข้อความสั่งกำหนดค่า

```
Today := Friday ;
Tomorrow := Satursday ;
```

กำหนดค่า Friday ให้กับตัวแปร Today และค่า Satursday ให้กับตัวแปร Tomorrow หลังจากกำหนดค่าแล้ว, ความสัมพันธ์เชิงอันดับข้างล่างนี้ทั้งหมดเป็นจริง

```
Today     =  Friday
Tomorrow  =  Satursday
Today     <  Tomorrow
Today     <> Wednesday
Today     >= Sunday
```

เราสามารถใช้ฟังก์ชันเชิงอันดับที่ Succ, Pred และ Ord กับชนิดแจงนับได้ ตัวอย่างต่อไปนี้ สมมติว่า Today คือ Friday และ Tomorrow คือ Satursday

Ord (Today) คือ 5

Ord (Tomorrow) คือ 6

Succ (Today) คือ Satursday

Pred (Today) คือ Tursday

Pred (Succ (Today)) คือ Friday

Succ (Tomorrow) คือ ไม่ถูกนิยาม (undefined)

Pred (Tomorrow) คือ Friday

ตัวอย่างก่อนตัวอย่างสุดท้าย ไม่ถูกนิยาม เพราะว่าไม่มีค่าของชนิด Day ตามหลัง Satursday ในทำนองเดียวกัน ค่าของ Pred (Sunday) ไม่ถูกนิยาม การดำเนินการ Succ หรือ Pred ซึ่งนำไปสู่ผลลัพธ์ ไม่ถูกนิยามอาจทำให้เกิด Range check error ระหว่างการกระทำการโปรแกรม

ตัวอย่าง 7.8

ข้อความสั่ง if ต่อไปนี้กำหนดค่าของ Tomorrow บนฐานของค่าของ ToDay (ทั้งคู่ชนิด Day)

```
if  ToDay = Satursday then
    Tomorrow := Sunday
else
    Tomorrow := Succ (Today)
```

เนื่องจากวันในหนึ่งสัปดาห์เป็นวงรอบ ถ้า Today คือ Satursday, ดังนั้น Tomorrow จึงเป็น Sunday ค่าสุดท้าย (Satursday) ในชนิดแจงนับ Day ถือว่าแยกต่างหากจากกัน เนื่องจาก Succ (Today) ไม่ถูกนิยามเมื่อ ToDay คือ Satursday

เพราะว่าชนิดแจงนับเป็นชนิดเชิงอันดับที่ (ordinal type) เราสามารถใช้ตัวแปรซึ่งอยู่ในชนิดแจงนับ เป็น **ตัวแปรนับ** (counter variables) ในข้อความสั่ง for หรือเป็นตัวเลือก case (case selectors) ในข้อความสั่ง case ได้

สองตัวอย่างถัดไปแสดงให้เห็นข้อความสั่ง for และข้อความสั่ง case

ตัวอย่าง 7.9

for ลูปในรูป 7.5 อ่านจำนวนชั่วโมงทำงานต่อหนึ่งสัปดาห์ของพนักงานหนึ่งคน และสะสมผลบวกของจำนวนชั่วโมงทำงานใน WeekHours ถ้าตัวแปรนับ Today ประกาศเป็นชนิดแจงนับ Day, การวนซ้ำกระทำการสำหรับ Today เท่ากับ Monday จนถึง Friday ระหว่างการทำซ้ำแต่ละครั้ง เรียก Write และ WriteDay แสดงตัวพร้อม เมื่อ WriteDay (ดูแบบฝึกหัดข้อ 3 ตอนท้ายของหัวข้อนี้) แสดงชื่อวัน เมื่อ Today มีค่าเป็น Monday, ตัวพร้อมจะเป็นดังนี้

```
Enter hours for Monday >
```

ถัดไป อ่านแต่ละค่าไว้ใน DayHours ซึ่งจะบวกกับ WeekHours หลังจากออกจากลูป แสดงผลค่าสุดท้ายของ WeekHours ถัดไปจะอธิบายว่าทำไมเราจึงต้องใช้กระบวนงาน WriteDay

```
WeekHours := 0.0;
for Today := Monday to Friday do
   begin
      Write ('Enter hours for ');
      WriteDay (Today);
      Write ('>');
      ReadLn (DayHours);
      WeekHours := WeekHours + DayHours
   end; {for}
WriteLn ('Total weekly hours are ', WeekHours : 4 : 2)
```

รูป 7.5 การสะสมชั่วโมงทำงาน

การอ่านและการเขียนค่าชนิดแจงนับ (Reading and Writing Enumerate Type Values)

เนื่องจากชนิดแจงนับและค่าที่แตกต่างกันสามารถใช้ได้ในแต่ละโปรแกรม, กระบวนงานอินพุต/เอาต์พุตของ Pascal ไม่ได้ถูกออกแบบให้อ่านหรือเขียนค่าชนิดแจงนับ อย่างไรก็ตามเราสามารถเขียนรหัสกระบวนงานของเราเองเพื่อวัตถุประสงค์นี้ได้

ตัวอย่าง 7.10

กำหนดการประกาศดังนี้

```
type
   Color = (Red, Green, Blue, Yellow);
var
   Eyes : Color;
```

ข้อความสั่ง

Write (Ord (Eyes) : 1)

สามารถใช้เพื่อการวินิจฉัยการพิมพ์ระหว่างการแก้จุดบกพร่อง มันไม่พิมพ์ค่าของ Eyes แต่แสดงผลเลขเชิงอันดับที่ของค่า นั่นคือ เป็นเลขจำนวนเต็มจาก 0 (สำหรับ Red) ไปถึง 3 (สำหรับ Yellow)

กระบวนงาน WriteColor ในรูป 7.6 พิมพ์สายอักขระซึ่งแทนค่าของชนิด Color ถ้าค่าของ Eyes ถูกนิยาม, ข้อความสั่ง

WriteColor (Eyes)

แสดงผลค่าของ Eyes เป็นสายอักขระ ให้แน่ใจว่าเราเข้าใจความแตกต่างระหว่างสายอักขระ 'Blue' และ ไอเดนติไฟเออร์ค่าคงตัว Blue

บ่อยครั้งที่โปรแกรมเมอร์ใช้ข้อความสั่ง case เช่นในรูป 7.6 ซึ่ง case label คือค่าที่ประกาศในชนิดแจงนับ โปรดระวังไม่ให้ใช้สายอักขระ เช่น 'Red' เป็น case label สิ่งนี้อาจทำให้เกิดข้อผิดพลาดวากยสัมพันธ์ constant and case type do not match โปรดจำว่าเฉพาะค่าเชิงอันดับที่หรือค่าคงตัวเท่านั้น (รวมทั้งค่าคงตัวแจงนับ) สามารถเป็น case labels (Remumber that only ordinal values or constants (including enumerated constants) can be case labels.)

```
procedure WriteColor (Incolor {input} : Color);
{
   Display the value of InColor.
   Pre  :  InColor is assigned a value.
   Post :  The value of InColor is displayed as a string.
}
begin {WriteColor}
   case InColor of
      Red : Write ('Red');
      Green : Write ('Green');
      Blue : Write ('Blue');
```

```
        Yellow : Write ('Yellow')
    end {case}
end; {WriteColor}
```

รูป 7.6 กระบวนงานพิมพ์ค่าของชนิด Color

ตัวอย่าง 7.11

กระบวนงาน ReadInColor ในรูป 7.7 กลับคืนหนึ่งค่าของชนิด Color ผ่านพารามิเตอร์เอาต์พุตของมัน ชื่อ ItemColor

ลูปในรูป 7.7 ทำซ้ำจนกระทั่งสีถูกต้อง (valid color) ถูกกำหนดให้กับ ItemColor ตัวบ่งชี้แบบบูล (Boolean Flag) ชื่อ ValidColor ควบคุมการทำซ้ำลูปและค่าเริ่มต้นคือ True ถ้าตัวอักขระข้อมูลถูกต้อง (R, G, B หรือ Y) ถูกอ่านข้อความสั่ง if กำหนดค่าสีซึ่งสมนัยกันของมันให้กับ ItemColor ถ้าตัวอักขระข้อมูลไม่ถูกต้องถูกอ่าน ValidColor ถูกกำหนดเป็น False และลูปถูกทำซ้ำ

```
procedure ReadInColor (var ItemColor {output} : Color);
{
   Assigns a value to ItemColor based on an input character.
   Pre  :  None
   Post :  Itemcolor is defined as the color value whose first letter is the
           same as the data character.
   Calls :  UpCase
}
   var
      ColorChar : Char;       {first letter of color name}
      ValidColor : Boolean;   {flag for valid color read}

begin {ReadInColor}
```

```
repeat
    ValidColor := Ture; {Assume a valid color will be read.}
    Write ('Enter first letter of color (R, G, B, or Y) > ');
    ReadLn (ColorChar);
    ColorChar := UpCase (ColorChar); {Convert to uppercase.}

    {Assign the color value or reset ValidColor to False.}
    if      ColorChar = 'R' then
            ItemColor := Red
    else  if ColorChar = 'G' then
            ItemColor := Green
    else  if ColorChar = 'B' then
            ItemColor := Blue
    else  if ColorChar = 'Y' then
            ItemColor := Yellow
    else
            ValidColor := False {repeat-valid color was not read.}
  until     ValidColor
end; {ReadInColor}
```

รูป 7.7 กระบวนงาน ReadInColor

ถ้า Black และ Brown ถูกใส่เพิ่มในรายการค่าของ Color จำเป็นจะต้องอ่านตัวอักขระเพิ่มเติมเมื่ออักษรตัวแรกที่อ่านเป็น B นี่เป็นแบบฝึกหัดเขียนโปรแกรมข้อ 2 ที่ตอนท้ายของหัวข้อนี้

พิสัยย่อยของชนิดแจงนับ (Subrange of Enumerated Types)

เราสามารถประกาศพิสัยย่อยของชนิดแจงนับ การประกาศต่อไปนี้ระบุว่า weekDay (ค่า Monday ถึง Friday) เป็นพิสัยย่อยของชนิด Day และตัวแปร SchoolDay เป็นชนิด WeekDay

```
type
   Day =   (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday,
           Satursday); {days of week}
   WeekDay = Monday . . Friday; {weekdays only}
var
   SchoolDay : Weekday;
```

ข้อความสั่งกำหนดค่า

```
   SchoolDay := Monday;
```

ถูกต้อง แต่ ข้อความสั่งกำหนดค่า

```
   SchoolDay := Sunday;
```

เกิดข้อผิดพลาดวากยสัมพันธ์ Constant out of range

ทำไมใช้ชนิดแจงนับ (Why Use Enumerated Type?)

ณ จุดนี้ เราอาจสงสัยว่า การใช้ชนิดแจงนับคุ้มค่า (worth) หรือไม่ พิจารณาที่ต้องใช้ความพยายาม (effort) เพื่ออ่านและเขียนค่าของมัน ถ้าเราจำเป็นต้องมีรหัสเพื่อใส่ค่าของตัวแปรชนิดแจงนับ ทำไมจึงไม่ใส่รหัสนั้นตลอดทั้งโปรแกรม เหตุผลคือชนิดแจงนับทำให้การอ่านและทำความเข้าใจตัวโปรแกรมง่ายขึ้น

ตัวอย่าง 7.12

ข้อความสั่ง if

```
if    DayNum = 1 then
         PayFactor := 2.0 {double pay for sunday}
else  if DayNum = 7 then
         PayFactor := 1.5 {time and a half for saturday}
else
         PayFactor := 1.0 {regular pay}
```

ปรากฏในโปรแกรมบัญชีเงินเดือนโดยไม่มีชนิดแจงนับ ถ้า Sunday และ Saturday ถูก "ลงรหัส" เป็นจำนวนเต็ม 1 และ 7 ตามลำดับ ถ้าเราใช้ชนิดแจงนับ Day และตัวแปร Today (ชนิด Day) เราเขียนข้อความสั่งนี้ใหม่เป็นดังนี้

```
if    Today = Sunday then
         PayFactor := 2.0
```

```
else  if Today = Saturday then
        PayFactor := 1.5
else
        PayFactor := 1.0
```

ตัวอย่างที่สองเห็นชัดว่าอ่านง่ายกว่าเพราะว่ามันแทนที่ค่า (Saturday และ Sunday) ซึ่งมีความหมายให้กับปัญหา

ในโปรแกรมขนาดยาว ส่วนเกินที่ต้องใช้เพื่อปฏิบัติให้เกิดผลกระบวนงานสำหรับการอ่านและการเขียนค่าต่างๆ ที่เกี่ยวข้องกับชนิดแจงนับ ไม่มีนัยสำคัญ (insignificant) การใส่กระบวนงานเหล่านี้ในคลังส่วนจำเพาะ (library of modules) ของเราเอง จะทำให้ง่ายต่อการนำมาใช้ใหม่ (reuse) ในโปรแกรมในอนาคต (future programs)

ข้อดีอีกประการหนึ่งของการใช้ชนิดแจงนับ คือ การสร้างอย่างอัตโนมัติของพิสัยของค่าต่างๆ ที่กำหนดให้กับตัวแปร ด้วยรหัสจำนวนเต็ม ในทางกันข้าม ค่าจำนวนเต็มใดๆ ก็ตามสามารถถูกกำหนดค่าให้ ถ้าเราไม่มีปัญหาเรื่องการประกาศชนิด subrange ในตัวอย่างก่อนหน้านี้ ค่าจำนวนเต็มใดๆ ก็ตามสามารถกำหนดให้กับตัวแปร DayNum แต่มีเฉพาะหนึ่งค่าในเจ็ดค่าเท่านั้น ซึ่งแสดงรายการในการประกาศชนิดแจงนับ Day ที่สามารถถูกกำหนดให้กับตัวแปร Today

แบบฝึกหัด 7.7

1. จงประเมินผลนิพจน์ข้างล่างนี้ สมมติว่าก่อนการดำเนินการแต่ละชุด Today (ชนิด Day) คือ Thursday

    a) Ord (Monday)

    b) Ord (Today)

    c) Today < Tuesday

    d) Succ (Wednesday)

    e) Today + 1

    f) Ord (Today) + 1

    g) Pred (Today)

    h) Today >= Thursday

i) Pred (Sunday)

j) Ord (Succ(Succ(Today)))

2. จงบอกว่าการประกาศชนิดต่อไปนี้แต่ละชุดถูกต้อง (valid) หรือไม่ถูกต้อง (invalid) ถ้าชุดใดไม่ถูกต้องให้อธิบายว่าทำไมไม่ถูกต้อง

a) type Letters = ('A', 'B', 'C');

b) type Letters = (A, B, C);

TwoLetters = (A, C);

c) type Letters = ('A' . . 'Z');

d) type Statements = (begin, end, while, for);

e) type

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

WeekDay = Mon . . Fri;

WeekEnd = Sat . . Sun;

เขียนโปรแกรม

1. จงประกาศชนิดแจงนับ Month และเขียนข้อความสั่ง if ข้างล่างนี้ใหม่ สมมติว่า CurMonth เป็นชนิด Month แทนที่จะเป็นชนิด Integer จากนั้นเขียนข้อความสั่ง case ซึ่งมีความหมายเหมือนกัน

```
if  CurMonth = 1 then
    WriteLn ('Happy New Year')
else if CurMonth = 6 then
    WriteLn ('Summer begins')
else if CurMonth = 9 then
    WriteLn ('Back to school')
else if CurMonth = 12 then
    WriteLn ('Happy Holidays');
```

2. จงเขียนกระบวนงาน ReadLnColor ใหม่ (ดูรูป 7.7) สมมติว่า Black และ Brown เป็นค่าชนิดแจงนับ Color ด้วย

3. จงเขียนกระบวนงาน WriteDay สำหรับชนิดแจงนับ Day

7.8 ข้อผิดพลาดร่วมของการเขียนโปรแกรม (Common Programming Errors)

ให้รอบคอบเมื่อทำงานกับนิพจน์ที่ซับซ้อน หรือเราอาจพลั้งเผลอไม่ใส่วงเล็บหรือตัวดำเนินการ ถ้าไม่ใส่ตัวดำเนินการหรือวงเล็บไม่จับคู่กัน จะเป็นข้อผิดพลาดวากยสัมพันธ์ แต่ถ้าไม่ใส่คู่วงเล็บ นิพจน์นั้นถ้าแม้จะถูกต้องเชิงวากยสัมพันธ์อาจคำนวณแล้วค่าที่เป็นผลลัพธ์อาจไม่ถูกต้อง

เทคนิคที่เป็นประโยชน์สำหรับการทำงานกับนิพจน์ซับซ้อนคือการแบ่งนิพจน์ให้เป็นนิพจน์ย่อย กำหนดนิพจน์ย่อยให้กับตัวแปรชั่วคราว และจัดดำเนินการกับตัวแปรชั่วคราวเหล่านั้น ตัวอย่างเช่น เขียนข้อความสั่งกำหนดค่าสามชุดอย่างถูกต้อง

```
Temp1 := Sqrt(X + Y);     {Assign 1st subexpression to Temp1}
Temp2 := 1.0 + Temp1;     {Assign 2nd subexpression to Temp2}
    Z := Temp1/Temp2      {Divide the subexpression}
```

จะง่ายกว่าเขียนหนึ่งข้อความสั่งกำหนดค่า

```
Z := Sqrt(X + Y) / (1.0 + Sqrt(X + Y))
```

ซึ่งมีผลลัพธ์เหมือนกัน การใช้ข้อความสั่งกำหนดค่าสามชุด มีประสิทธิภาพมากกว่า เพราะว่า Sqrt(X + Y) ประเมินผลเพียงครั้งเดียว ส่วนการเขียนข้อความสั่งกำหนดค่าหนึ่งชุด การประเมินผลทำสองครั้ง

ตัวดำเนินการซึ่งสามารถใช้กับข้อมูลชนิด Char คือ

ตัวดำเนินการสัมพันธ์เท่านั้น นิพจน์

```
3 < > '3'          {incompatible operands}
```

ไม่ถูกต้อง เพราะว่า ตัวถูกดำเนินการ (operand) หนึ่งตัวเป็นจำนวนเต็ม และตัวถูกดำเนินการอีกหนึ่งตัวเป็นอักขระเลขโดด

ให้มั่นใจว่าเราใช้วงเล็บอย่างถูกต้อง ในนิพจน์ประกอบแบบบูล (compound Boolean expressions) ตัวดำเนินการแบบบูล and, or และ not มีการทำก่อน (precedence) สูงกว่าตัวดำเนินการสัมพันธ์ ดังนั้นจึงต้องใช้วงเล็บในนิพจน์ เช่น

```
(-5.0 <= X) and (X <= 5.0)
```

ข้อผิดพลาดวากยสัมพันธ์หรือข้อผิดพลาดเวลาดำเนินงาน สามารถเกิดขึ้นได้เมื่อเราใช้ฟังก์ชันในตัว (built-in functions) อาร์กิวเมนต์ของฟังก์ชัน Chr ต้องเป็นชนิด Integer, อาร์กิวเมนต์ของฟังก์ชัน Ord, Succ และ Pred ต้องเป็นชนิดเชิงอันดับที่ (ไม่ใช่ชนิด Real)

ผลลัพธ์ของฟังก์ชัน Succ, Pred และ Chr อาจไม่ถูกนิยาม (undefined) สำหรับอาร์กิวเมนต์เฉพาะตัว

พิสัยย่อย (Subranges) สามารถช่วยเราตรวจหาข้อผิดพลาดการคำนวณหรือข้อมูลถ้าค่าซึ่งถูกกำหนดอยู่ภายนอก พิสัยย่อย และการตรวจสอบพิสัยทำให้เกิด Range check error การดำเนินการซึ่งสามารถกระทำบนตัวแปรชนิด subrange ถูกกำหนดโดยแม่ข่าย (host type) สำหรับพิสัยย่อยนั้น อย่างไรก็ตามตัวแปรซึ่งชนิดของมันเป็นพิสัยย่อย ไม่สามารถสมนัยกับพารามิเตอร์ตัวแปรรูปนัย ซึ่งชนิดของมันคือ ชนิดแม่ข่ายสำหรับพิสัยย่อยนั้น

เมื่อเราประกาศชนิดแจงนับ โปรดจำว่า เฉพาะไอเดนติไฟเออร์เท่านั้นที่สามารถปรากฏในรายการแสดงค่าสำหรับชนิดแจงนับ ทั้งนี้เป็น strings, characters และ numbers ไม่ได้

ไอเดนติไฟเออร์ค่าคงตัว (constrant identifier) เหมือนกัน ปรากฏในการประกาศชนิดแจงนับมากกว่าหนึ่งครั้ง ในบล็อกที่กำหนดให้ไม่ได้แต่ยอมให้ไอเดนติไฟเออร์ค่าคงตัวเหมือนกัน ปรากฏในการประกาศชนิด subrange ได้มากกว่าหนึ่งชุด

โปรดจำว่า ไม่มีกระบวนงานมาตรฐานให้ใช้สำหรับอ่านหรือเขียนค่าของชนิดแจงนับ

บทสรุปของตัวสร้างใหม่ของ Pascal (Summary of New Pascal Constructs)

Construct	Effect
การประกาศ subrage	
type	ประกาศ subrange ของตัวอักขระพิสัยย่อยนี้ชื่อ Digit
Digit = '0' . . '9' :	ประกอบด้วยค่าตัวอักขระ '0' ถึง '9'
การประกาศชนิด enumerated	
type	ประกาศชนิดแจงนับชื่อ BColor ด้วยค่า Blue, Black
BColor = (Blue, Black, Brown);	และ Brown

แบบฝึกหัด Quick-Check

1. ฟังก์ชันข้างล่างนี้กลับคืนค่าอะไร

   a) เมื่ออาร์กิวเมนต์ของมันคือ 5 และ 7

   b) เมื่ออาร์กิวเมนต์ของมันคือ 7 และ 5

   c) ฟังก์ชัน ThisDoesWhat ทำงานอะไร

```
function ThisDoesWhat (First, Second : Integer) : Boolean;
begin
   ThisDoesWhat := (First div Second) = 0
end;  {ThisDoesWhat}
```

2. สิ่งใดต่อไปนี้สามารถเป็นค่าชนิด enumerated ได้

   an integer

   a real number

   an identifier

   a Boolean value

   a string value

3. จงอธิบายว่าชนิด subrange สามารถใช้ตรวจหาค่าจำนวนเต็ม out-of-range อย่างไร และ out-of-range ของค่า real เป็นอย่างไร

4. จงหาค่าของสิ่งต่อไปนี้

   a) Chr (Ord('a'))

   b) Chr (Ord('a') + 3)

   c) Ord ('7' - Ord ('0')

   d) Ord ('z') - Ord ('a')

5. ชนิด subrange ใช้ประโยชน์สำหรับการป้องกันผู้ใช้จากการใส่ค่า out-of-range ในข้อความสั่ง ReadLn หรือไม่ จงอธิบาย

6. a) ตัวแปรซึ่งชนิดของมันเป็น subrange สามารถสมนัยกับพารามิเตอร์ตัวแปรรูปนัย ซึ่งชนิดของมันเป็น host type ได้หรือไม่

   b) ถ้าพารามิเตอร์รูปนัยเป็นพารามิเตอร์ค่าจะเกิดอะไรขึ้น

7. ถ้าตัวแปรสองตัวมีชนิดใช้แทนกันได้ ตัวแปรตัวหนึ่งจะสามารถกำหนดค่าให้กับตัวแปรอีกตัวหนึ่งได้เสมอหรือไม่

8. ถ้าค่า 1/3 คำนวณโดยใช้ตัวแปร Real จงอธิบายว่า ทำไมผลลัพธ์จึงไม่เป็น 1/3 อย่างแม่นยำ (exactly)

9. การประกาศชนิดแจงนับ (enumerated type) ข้างล่างนี้ไม่ถูกต้องเพราะเหตุใด ให้อธิบาย

```
type Prime = (2, 3, 5, 7, 9, 11, 13);
```

10. จงพิจารณาการประกาศชนิดแจงนับต่อไปนี้

```
type Class = (Frosh, Soph, Jr, Sr);
```

จงบอกค่าของนิพจน์แต่ละชุดข้างล่างนี้

a) Ord (Succ (Pred (Soph)))

b) Pred (Pred (Jr))

คำถามทบทวน (Review Questions)

1. จงบอกข้อดีของแบบชนิดข้อมูล Integer ซึ่งมีเหนือกว่าแบบชนิดข้อมูล Real

2. จงแสดงรายการและอธิบายข้อผิดพลาดการคำนวณสามชนิดซึ่งอาจเกิดขึ้นในนิพจน์ชนิด Real

3. จงเขียนการประกาศชนิดแจงนับ (enumerated) สำหรับ Fiscal ซึ่งเป็นเดือนจาก July ถึง June และประกาศพิสัยย่อย (subrange) ชื่อ Winter เป็น December ถึง February

4. จงเขียนกระบวนงานสำหรับการอ่านและการเขียนค่าต่างๆ สำหรับตัวแปร Season ชนิดแจงนับ

```
type Season = (Winter, Spring, Summer, Fall);
```

5. จงเขียน while ลูป ซึ่งสมนัยกับ for ลูปข้างล่างนี้ สมมติว่า ตัวแปร Ch เป็นชนิด Char

```
for Ch := 'A' to 'Z' do
    Write (Ch)
```

6. a) จงเขียนข้อความสั่ง case ซึ่งทดสอบว่า Today เป็นวันทำงานหรือไม่ พิมพ์ข้อความ 'Workday' หรือ 'Weekend' สมมติว่า Today เป็นชนิด Day ซึ่งเป็นชนิดแจงนับที่มีวันของสัปดาห์เป็นค่าของมัน

b) จำเป็นต้องมีการเฝ้าระวัง (guard) ข้อความสั่ง case นี้ หรือไม่

7. จงเขียนฟังก์ชันแบบบูลซึ่งกลับคืน True หรือ False โดยขึ้นอยู่กับเงื่อนไขที่เกี่ยวข้องกับอาร์กิวเมนต์ของมันดังนี้

either Flag is True or Color is Red, or both Money is Plenty and Time is Up

จงแสดงการประกาศแต่ละชนิดซึ่งอาจจำเป็นต้องมีก่อนประกาศฟังก์ชัน

8. จงเขียนฟังก์ชันแบบบูล เรียก OvertimeDue ซึ่งกลับคืนค่า True ก็ต่อเมื่อ Hours ชั่วโมงทำงานต่อสัปดาห์ของพนักงานมีค่ามากกว่า 40

9. จงเขียนการประกาศชนิด enumerated สำหรับวันของสัปดาห์ และชนิด sub-range สำหรับวันทำงาน (weekdays) และวันหยุด (weekend days) บทนิยามของชนิด enumerated ถูกบังคับโดยข้อกำหนดของ subrange อย่างไร

10. จงเขียนข้อความสั่ง Pascal เท่าที่จำเป็นเพื่อใส่เลขจำนวนเต็มระหว่าง 0 และ 9 นับทั้งต้นและท้าย และเปลี่ยนเลขให้เป็นค่าอักขระที่สมมูลกัน (เช่น 0 เป็น '0', 1 เป็น '1') แล้วเก็บในตัวแปรอักขระ Num

11. จงเขียนฟังก์ชัน Pascal ที่มีอาร์กิวเมนต์ N ชนิด Integer และอาร์กิวเมนต์ X ชนิด Real จากนั้นกลับคืนค่าเป็น N เทอมแรกของอนุกรม

$$X + \frac{1}{2} X^2 + \frac{1}{3} X^3 + \frac{1}{4} X^3 + \ldots + \frac{1}{N} X^N$$

เขียนโปรแกรม

1. เลขจำนวนเต็ม N หารด้วย 9 ลงตัว ถ้าผลบวกของเลขโดดของมันหารด้วย 9 ลงตัว จงพัฒนาโปรแกรมเพื่อตรวจสอบว่าเลขต่อไปนี้หารด้วย 9 ลงตัวหรือไม่ โดยใช้เทคนิคนี้ให้ประกาศ N เป็นชนิด LongInt

N = 154368

N = 621594

N = 123456

2. จงเขียนโปรแกรมข้อ 1 ใหม่ โดยการอ่านเลขโดดแต่ละตัวของเลข ซึ่งต้องการทดสอบไว้ในตัวแปร Digit ชนิด Char จากนั้นคำนวณผลบวกของค่าตัวเลขของเลขโดด

(**ข้อแนะนำ** ค่าตัวเลขของ Digit (ชนิด Char) คือ

Ord (Digit) - Ord ('0')

3. การจ่ายดอกเบี้ยของบัญชีออมทรัพย์ของธนาคารกระทำทุกวัน (compounded daily) สิ่งนี้หมายความถ้าเราเริ่มต้นด้วยเงิน StartBal (เป็นดอลลาร์) ในธนาคาร ตอนจบของวันแรก เราจะมียอดเงินคงเหลือในบัญชีเท่ากับ

$$\text{StartBal} \times \left(1 + \frac{\text{Rate}}{365}\right)$$

เมื่อ Rate คือ อัตราดอกเบี้ยต่อปี (0.01 ถ้าอัตราดอกเบี้ยต่อปีเท่ากับ 10%) ณ ตอนจบของวันที่สอง เงินในบัญชีเท่ากับ

$$\text{StartBal} \times \left(1 + \frac{\text{Rate}}{365}\right) \times \left(1 + \frac{\text{Rate}}{365}\right)$$

ณ ตอนจบของวันที่ N เงินในบัญชีเท่ากับ

$$\text{StartBal} \times \left(1 + \frac{\text{Rate}}{365}\right) \quad \text{ดอลลาร์}$$

จงเขียนโปรแกรมประมวลผลเซตของระเบียนข้อมูล ซึ่งระเบียนแต่ละชุดประกอบด้วยค่า StartBal, Rate และ N จากนั้นให้คำนวณยอดเงินคงเหลือสุดท้ายในบัญชี