# บทที่ 3

# การแทนข้อมูลในคอมพิวเตอร์

## วัตถุประสงค์การเรียนรู้

อธิบายแนวคิดของการเก็บข้อมูลในภาษาโปรแกรม การแทนข้อมูลในคอมพิวเตอร์ และความแตกต่างระหว่างข้อมูลชนิดต่างๆ

1. อธิบายเหตุผลที่ต้องกำหนดชนิดของข้อมูลเมื่อทำงานกับคอมพิวเตอร์ได้ และเลือกใช้ชนิดของข้อมูลได้เหมาะสมกับข้อมูลและรูปแบบการใช้งาน
2. อธิบายการใช้พื้นที่ในหน่วยความจำสำหรับการเก็บข้อมูลได้ เรียกใช้ใช้ข้อมูลโดยผ่านการอ้างอิงถึงที่อยู่ได้ถูกต้อง

ใช้เวลารวม 3 คาบ

## 3.1 บิตและไบต์

ข้อมูลในคอมพิวเตอร์เก็บอยู่ในรูปของเลขฐานสองเนื่องมาจากเทคโนโลยีดิจิทัลที่ใช้สวิตช์สถานะปิดและเปิดในการเก็บข้อมูล สวิตช์หนึ่งตัวแทนข้อมูลได้ 2 รูปแบบ หากต้องการแทนข้อมูลที่มีรูปแบบมากกว่านี้ก็ต้องใช้สวิตช์หลายตัวประกอบกัน การแทนข้อมูลด้วยสถานะเปิดปิดนี้เทียบได้กับการแทนข้อมูลด้วยเลขฐานสองหนึ่งหลัก เราจึงมีหน่วยเรียกความจุข้อมูลนี้ว่าความจุข้อมูลขนาดหนึ่งบิต

ขนาดของหน่วยความจำที่ใช้ในการเก็บอักขระ 1 ตัวคือ 7 บิต แต่เรามักกำหนดให้ขนาดของข้อมูลเป็นกำลังของสอง จึงใช้ขนาด 8 บิต ($2^3$) แทนอักขระหนึ่งตัว เรียกว่า 1 ไบต์ (byte) เราจึงมักนับขนาดของหน่วยความจำเป็นไบต์ เพื่อให้เทียบเคียงกับการนับขนาดของข้อความเป็นจำนวนตัวอักษร เช่น ข้อความ 200 ตัวอักษร จะใช้พื้นที่ในการเก็บ 200 ไบต์ เป็นต้น

ความจุข้อมูลที่ใหญ่ขึ้น จะสามารถแทนรูปแบบของข้อมูลได้มากขึ้น ตัวอย่างจำนวนรูปแบบของข้อมูลที่สามารถแทนได้เมื่อมีหน่วยความจำขนาดต่างๆ ดังตารางที่ 3.1

ตารางที่ 3.1: จำนวนข้อมูลที่แทนได้เมื่อมีพื้นที่ในหน่วยความจำขนาดต่าง ๆ

| จำนวนบิต | จำนวนรูปแบบที่แทนได้ |
|---|---|
| 1 | $2^1$ = 2 |
| 2 | $2^2$ = 4 |
| 3 | $2^3$ = 8 |
| 4 | $2^4$ = 16 |
| 8 | $2^8$ = 256 |
| 16 | $2^{16}$ = 65,536 |
| 32 | $2^{32}$ = 4,294,967,296 |
| b | $2^b$ = $2^b$ |

หากเลขฐานสองมีขนาดยาวมาก เช่น $00010010_2$ = 18 การเขียนเลขฐานสองด้วยสัญลักษณ์ 0 (ศูนย์) และ 1 (หนึ่ง) จะทำให้อ่านยาก ในงานทางคอมพิวเตอร์จึงนิยมเขียนเลขฐานสองสี่หลักรวมกันเป็นเลขฐานสิบหกหนึ่งหลัก (ใช้สัญลักษณ์ 0-F - ศูนย์ถึงเอฟ) และใช้สัญลักษณ์ 0x (ศูนย์เอ็กซ์) นำหน้าจำนวนที่เป็นเลขฐานสิบหก เช่น $00010010_2 = 12_{16}$ = 0x12 เป็นต้น

โดยทั่วไปเราใช้คอมพิวเตอร์ทำงานกับข้อมูลพื้นฐานสองประเภท ได้แก่ จำนวน และ ข้อความ โดยจำนวนใช้แทนค่าซึ่งนำมาคำนวณทางคณิตศาสตร์ได้ และข้อความซึ่งอยู่ในรูปของสายอักขระ ส่วนข้อมูลลักษณะอื่นๆ เช่น ภาพ เสียง เป็นข้อมูลที่ซับซ้อนขึ้น และอยู่นอกเหนือขอบเขตของวิชานี้

## 3.2 ข้อมูลประเภทจำนวน

ข้อมูลประเภทจำนวนยังแบ่งย่อยออกเป็น 2 รูปแบบ ได้แก่ จำนวนเต็ม และ จำนวนจริง ความแตกต่างของการเก็บข้อมูลทั้งสองรูปแบบอยู่ที่วิธีการกำหนดรหัสเลขฐานสองเพื่อแทนค่าของจำนวนนั้น

### 3.2.1 ข้อมูลชนิดจำนวนเต็ม

การแทนข้อมูลชนิดจำนวนจะใช้เลขฐานสองหนึ่งค่า แทนจำนวนหนึ่งจำนวน การแทนจำนวนเต็ม หากมีพื้นที่ 4 ไบต์ (32 บิต) จะแทนข้อมูลได้ราวสี่ล้านรูปแบบ ซึ่งแบ่งรูปแบบการเก็บข้อมูลได้ดังนี้

- unsigned: เก็บค่าที่มากกว่าหรือเท่ากับศูนย์เท่านั้น จะเก็บค่าได้ตั้งแต่ 0 ถึง $2^{32}$
- signed: ใช้ 1 บิตในการเก็บเครื่องหมาย (บวกหรือลบ) และใช้พื้นที่ที่เหลือในการเก็บค่า หากให้เก็บทั้งค่าลบและค่าบวกเป็นจำนวนเท่าๆ กัน จะมีค่าในช่วง $[-2^{16},-1,],0,[1,2^{16}-1]$

ในการเขียนโปรแกรม หากให้ integer เป็นชนิดข้อมูลจำนวนเต็มปกติ จะมีชนิดของข้อมูลจำนวนเต็มความจุต่ำ (short) และจำนวนเต็มความจุสูง (long) ให้ด้วย ความจุของชนิดข้อมูลในแต่ละเครื่องอาจไม่

เหมือนกัน จึงควรตรวจสอบความจุของข้อมูลแต่ละชนิดในเครื่องที่ใช้งานทุกครั้ง โดยเฉพาะอย่างยิ่งเมื่อต้องการเก็บค่าที่เข้าใกล้ขอบเขตสูงสุดหรือต่ำสุดของข้อมูลแต่ละชนิด แต่การเรียงลำดับความจุจากน้อยไปมากจะเป็น short ≤ integer ≤ long เสมอ

### 3.2.2 ข้อมูลชนิดจำนวนจริง

การแทนจำนวนจริงก็ใช้เลขฐานสองหนึ่งค่าแทนจำนวนหนึ่งจำนวนเช่นเดียวกัน ในการเก็บค่าจำนวนจริงจะแบ่งพื้นที่ในการเก็บข้อมูลออกเป็นเครื่องหมาย เลขนัยสำคัญ และเลขชี้กำลัง ตามลำดับ ถึงแม้พื้นที่ในการเก็บเท่ากัน จำนวนข้อมูลที่เก็บได้เท่ากัน เนื่องจากมีการใช้เลขชี้กำลังช่วย การเก็บข้อมูลแบบจำนวนจริงจะสามารถเก็บค่าของจำนวนได้ช่วงกว้างกว่าจำนวนเต็ม แต่นัยสำคัญสูงสุดที่เก็บได้จะน้อยกว่าของจำนวนเต็ม

จำนวนเต็มนัยสำคัญปกติเรียกว่าชนิด float และจำนวนเต็มนัยสำคัญสูงเรียกว่าชนิด double (นัยสำคัญสองเท่าของปกติ - double precision) อย่างไรก็ดี ในบางเครื่อง พื้นที่เก็บข้อมูลและนัยสำคัญของ float และ double อาจไม่ต่างกัน แต่พื้นที่และนัยสำคัญของ float จะไม่เกินของ double เสมอ ก่อนการใช้งานจึงควรตรวจสอบขอบเขตของค่าที่ใช้ได้เสมอ เช่นเดียวกับการใช้จำนวนเต็มชนิดต่าง ๆ

**ลองคิด**

หากกำหนดพื้นที่ในการเก็บข้อมูลเต็มและจำนวนจริงที่ 4 ไบต์เท่ากัน มีค่าที่เก็บเป็นจำนวนเต็มได้ แต่เก็บเป็นจำนวนจริงไม่ได้หรือไม่ เพราะเหตุใด

## 3.3 ข้อมูลประเภทข้อความ

ข้อความประกอบด้วยอักขระต่างๆ เรียงต่อกัน การแทนข้อมูลประเภทนี้ในคอมพิวเตอร์จึงเป็นการเก็บอักขระแต่ละตัวเรียงต่อๆ กันไป เนื่องจากคอมพิวเตอร์รับรู้ข้อมูลในรูปแบบของเลขฐานสองเท่านั้น การแทนข้อมูลอักขระในคอมพิวเตอร์ก็ต้องแปลงให้อยู่ในรูปของเลขฐานสองเช่นเดียวกัน สมมติว่าต้องการแทนอักขระภาษาอังกฤษได้แก่ a-z มีสัญลักษณ์ของอักขระทั้งหมดที่เป็นไปได้ 26 แบบ การเก็บข้อมูลอักขระ 1 ตัวจะต้องใช้พื้นที่อย่างน้อย 5 บิต เพราะความจุ 5 บิตสามารถแทนข้อมูลได้ 32 รูปแบบ ซึ่งเพียงพอในการแทนข้อมูล 26 รูปแบบที่เราต้องการ กำหนดให้เลขฐานสอง 1 ค่าแทนสัญลักษณ์อักขระ 1 รูปแบบ เช่น $00000_2$=a, $00001_2$=b, $00010_2$=c, $11001_2$=z เป็นต้น เรียกวิธีการกำหนดค่าฐานสองแทนตัวเลขนี้ว่า รหัสอักขระ (character encoding)

หากข้อมูลประกอบด้วยทั้งตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก และต้องการแยกตัวอักษรทั้งสองประเภทออกจากกัน ก็จำเป็นจะต้องใช้รหัสเพิ่มเติมจาก 26 รูปแบบแรกที่มีอยู่ หากแจกแจงอักขระทั้งหมดที่ใช้เพื่อแทนข้อความในภาษาอังกฤษ จะได้ดังนี้

- ตัวพิมพ์เล็ก a-z รวม 26 รูปแบบ

- ตัวพิมพ์ใหญ่ A-Z รวม 26 รูปแบบ
- ตัวเลข 0-9 รวม 10 รูปแบบ
- สัญลักษณ์วรรคตอนต่างๆ เช่น เคาะวรรค จุลภาค (,) มหัพภาค (.) อัศเจรีย์ (!) และอื่น ๆ

ซึ่งรวมแล้วมากกว่า 64 รูปแบบแต่ไม่เกิน 128 รูปแบบ จึงใช้พื้นที่ 7 บิต ในการแทนข้อมูลได้ทั้งหมด

เราสามารถกำหนดรหัสแทนข้อมูลได้โดยอิสระ แต่หากต้องการใช้ข้อมูลร่วมกับผู้อื่น ก็ควรมีรหัสกลางที่เหมือนกัน เพื่อให้แลกเปลี่ยนข้อมูลกันได้ รหัสมาตรฐานมีหลากหลายรูปแบบ ที่นิยมใช้เช่น EBCDIC, ASCII

### 3.3.1 รหัสอักขระ

รหัสแอสกี (ASCII) ใช้พื้นที่ 7 บิตในการแทนข้อมูลอักขระภาษาอังกฤษ แต่การทำงานกับคอมพิวเตอร์เรามักใช้พื้นที่ขนาดเป็นจำนวนเท่าของ 8 เพื่อความสะดวกในการดำเนินการต่างๆ ของหน่วยประมวลผลกลาง อักขระในรหัสแอสกีหนึ่งตัวจึงใช้พื้นที่ 8 บิต โดยบิตแรกเป็น 0 เสมอ ดังช่วง 128 ตัวแรกในตารางที่ 3.2 โดย 32 ตัวแรกเป็นอักขระควบคุมที่ใช้ในการส่งข้อมูล

เนื่องจากรหัสแอสกีออกแบบมาสำหรับภาษาอังกฤษเท่านั้น เมื่อต้องการใช้แทนอักขระในภาษาอื่นจึงต้องมีการกำหนดรหัสอักขระเพิ่มเติม เรียกว่า รหัสแอสกีแบบขยาย (extended ASCII) ซึ่งจะให้บิตแรกเป็น 1 และจะใช้พื้นที่อีก 7 บิตที่เหลือในการกำหนดอักขระสำหรับภาษาอื่น ตัวอย่างของรหัสแอสกีแบบขยายที่ใช้กับภาษาไทย ได้แก่ Windows-874, TIS-620, ISO-8859-11 (ตาราง 3.2) เป็นต้น

ข้อเสียหลักของรหัสแอสกีแบบขยายคือ ใช้ได้ทีละคู่ภาษาเท่านั้น คือภาษาอังกฤษและอีกภาษาหนึ่ง ไม่สามารถใช้แทนข้อความที่ใช้อักขระมากกว่าสองชุดพร้อมกัน จึงมีการกำหนดมาตรฐานรหัสอักขระขึ้นมาใหม่ เรียกว่ารหัสยูนิโค้ด (unicode encoding) ใช้พื้นที่ตั้งแต่ 16-32 บิตในการแทนข้อมูล จึงทำให้แทนภาษาได้หลากหลายด้วยรหัสชุดเดียวกัน

รหัสยูนิโค้ดขนาด 16 บิตสำหรับภาษาไทยมี 8 บิตแรกเป็น 0x0E และเรียงลำดับตั้งแต่ ก ที่ 0x0E01 ไปเรื่อยๆ ตามลำดับของอักขระในรหัส ISO8859-11 ทุกประการ

การเปิดแฟ้มข้อความโดยใช้รหัสที่ไม่ถูกต้อง อาจทำให้ได้ข้อความที่อ่านไม่ออก ได้อักขระผิดพลาดไปหมด ปัจจุบันจึงนิยมใช้รหัสยูนิโค้ดกันมากกว่ารหัสแอสกีแบบขยาย เพื่อลดโอกาสที่จะเกิดความผิดพลาดในการใช้แฟ้มข้อมูล

### 3.3.2 สายอักขระ

สายอักขระคือการนำอักขระมาต่อกัน ดังนั้นจึงต้องใช้พื้นที่ในการเก็บข้อมูลอย่างน้อยเท่ากับจำนวนอักขระที่ต้องการเก็บ แต่ข้อมูลชนิดสายอักขระมักมีความยาวที่ไม่แน่นอน เช่น ชื่อคน อาจมีความยาวตั้งแต่สามตัวอักษรไปจนถึงมากกว่าสิบตัวอักษร การจองพื้นที่เพื่อเก็บข้อมูลชนิดสายอักขระจึงต้องเผื่อไว้สำหรับความยาวที่มากที่สุดที่เป็นไปได้

ปัญหาถัดมาในการจัดการกับสายอักขระซึ่งมีความยาวไม่แน่นอนคือ ไม่รู้ขอบเขตสิ้นสุดของข้อมูล หากจองพื้นที่ไว้สิบตัว แต่ข้อมูลมีแค่สามตัว คอมพิวเตอร์ไม่สามารถแยกแยะได้ว่าข้อมูลจะสิ้นสุดที่ใด จึง

ตารางที่ 3.2: ตารางรหัสแอสกีแบบขยาย ISO8859-11

| | _0 | _1 | _2 | _3 | _4 | _5 | _6 | _7 | _8 | _9 | _A | _B | _C | _D | _E | _F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0_ | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 1_ | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 2_ | SP | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 3_ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4_ | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5_ | P | Q | R | S | T | U | V | W | X | Y | Z | [ | / | ] | ^ | _ |
| 6_ | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7_ | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | DEL |
| 8_ | | | | | | | | | | | | | | | | |
| 9_ | | | | | | | | | | | | | | | | |
| A_ | NBSP | ก | ข | ฃ | ค | ฅ | ฆ | ง | จ | ฉ | ช | ซ | ฌ | ญ | ฎ | ฏ |
| B_ | ฐ | ฑ | ฒ | ณ | ด | ต | ถ | ท | ธ | น | บ | ป | ผ | ฝ | พ | ฟ |
| C_ | ภ | ม | ย | ร | ฤ | ล | ฦ | ว | ศ | ษ | ส | ห | ฬ | อ | ฮ | ฯ |
| D_ | ะ | ั | า | ำ | ิ | ี | ึ | ื | ุ | ู | ฺ | ฿ | | | | |
| E_ | เ | แ | โ | ใ | ไ | ๅ | ๆ | ็ | ่ | ้ | ๊ | ๋ | ์ | ํ | ๎ | ๏ |
| F_ | ๐ | ๑ | ๒ | ๓ | ๔ | ๕ | ๖ | ๗ | ๘ | ๙ | ๚ | ๛ | | | | |

จำเป็นต้องมีอักขระว่างเป็นอักขระพิเศษเพื่อบอกขอบเขตของข้อมูลด้วย ดังนั้น การจองพื้นที่ในการเก็บข้อมูลชนิดสายอักขระ จึงต้องเผื่อสำหรับเก็บอักขระปิดท้ายข้อมูลด้วย อักขระว่าง (null character - NUL) มีในรหัสอักขระหลายรูปแบบ ในรหัสแอสกีและรหัสยูนิโค้ด คือ 0x0000 และ 0x00000000 ตามลำดับ

ตัวอย่างเช่น การเก็บข้อความ “ภาษาไทย” ด้วยรหัสอักขระ ISO8859-11 จะใช้พื้นที่ 7+1=8 ไบต์

การแทนข้อมูลประเภทข้อความที่เป็นตัวเลขต่างกับการแทนข้อมูลประเภทจำนวน การแทนข้อมูลแบบสายอักขระจะใช้รหัสเลขฐานสองหนึ่งค่าต่อตัวเลขหนึ่งหลัก และต้องใช้อักขระปิดท้ายด้วย ในขณะที่การแทนข้อมูลแบบจำนวนจะใช้รหัสเลขฐานสองหนึ่งค่าต่อจำนวนนั้นทั้งจำนวน

**ลองคิด**

แฟ้มข้อมูลที่เก็บข้อความภาษาอังกฤษและบันทึกด้วยรหัสแอสกี จะสามารถเปิดอ่านแบบรหัสยูนิโค้ด 16 บิตได้ถูกต้องหรือไม่ เพราะเหตุใด

## 3.4 ข้อมูลชนิดตรรกะ

ข้อมูลชนิดตรรกะมีค่าที่เป็นไปได้เพียงสองค่า คือ จริง และ เท็จ จึงใช้พื้นที่เก็บข้อมูลเพียงหนึ่งบิตก็เพียงพอ แต่การใช้พื้นที่เพียงหนึ่งบิตอาจทำให้การประมวลผลโดยหน่วยประมวลผลกลางไม่สะดวก โดยมากจึงมักใช้พื้นที่เท่ากับจำนวนเต็มปกติ บางภาษาโปรแกรมจะใช้จำนวนเต็มแทนข้อมูลชนิดตรรกะ โดยกำหนดให้ 0 แทนค่าเท็จ และค่าอื่นๆ นอกจาก 0 แทนค่าจริง

ข้อมูลชนิดตรรกะพบมากในการทำงานแบบเงื่อนไข ซึ่งการตรวจสอบเงื่อนไขจะได้ค่าจริงหรือเท็จเพื่อเลือกเส้นทางการทำงานตามตรรกะที่ได้ต่อไป

นอกจากชนิดของข้อมูลพื้นฐานข้างต้นแล้ว ยังมีชนิดข้อมูลที่ซับซ้อนอีกหลายรูปแบบที่ใช้ในการเขียนโปรแกรม เช่น แถวลำดับ รายการ ระเบียน ออบเจกต์ ซึ่งอยู่นอกเหนือขอบเขตของวิชานี้

## 3.5 ตัวแปร

### 3.5.1 การอ้างถึงข้อมูลและการจองพื้นที่ในหน่วยความจำ

การดำเนินการใดๆ ของคอมพิวเตอร์จะเรียกใช้ข้อมูลจากหน่วยความจำเสมอ จึงต้องมีการจองพื้นที่ในหน่วยความจำเพื่อเก็บข้อมูล และกำหนดค่าให้พื้นที่นั้นก่อนนำไปใช้งานเสมอ การอ้างอิงถึงข้อมูลนั้นทำได้ด้วยอ้างถึงเลขที่อยู่ (address) ของหน่วยความจำ เพื่อนำค่า (value) ที่เก็บไว้ ณ ตำแหน่งนั้นมาใช้งาน เมื่อเริ่มโปรแกรม ระบบปฏิบัติการจะจองพื้นที่ในหน่วยความจำให้สำหรับโปรแกรมใช้งาน

ในขั้นตอนการเขียนโปรแกรมจะมีการระบุพื้นที่ที่ต้องการใช้ การอ้างอิงถึงพื้นที่เหล่านี้ทำโดยประกาศตัวแปรขึ้นแทนการเก็บข้อมูลต่างๆ ชื่อตัวแปรมักเป็นภาษาอังกฤษความยาวไม่มาก และสื่อถึงข้อมูลที่เก็บอยู่ในหน่วยความจำนั้น

| | intPtr | | | | number | | | |
|---|---|---|---|---|---|---|---|---|
| เลขที่อยู่ | 0x70 | 0x71 | 0x72 | 0x73 | 0x74 | 0x75 | 0x76 | 0x77 |
| ค่า | 0x74 | 0x75 | 0x37 | 0x30 | 0x00 | 0x00 | 0x00 | 0x64 |
| เลขที่อยู่ | 0x78 | 0x79 | 0x7A | 0x7B | 0x7C | 0x7D | 0x7E | 0x7F |
| ค่า | 0x00 | 0x46 | 0x00 | 0x00 | 0x00 | 0x30 | 0x30 | 0x00 |
| | | | | | | c | | |

ตารางที่ 3.3: ตัวอย่างการเก็บข้อมูลในหน่วยความจำ

ข้อกำหนดในการตั้งชื่อตัวแปรโดยทั่วไปคือ ไม่เป็นคำสงวน (reserved word) ในภาษาโปรแกรม มีความยาวไม่มากเกินกำหนด ไม่ใช้อักขระพิเศษซึ่งใช้เป็นตัวดำเนินการ ขึ้นต้นด้วยตัวอักษร เป็นต้น

การใช้ตัวแปรมีสองลักษณะ ได้แก่ การกำหนดค่าให้ตัวแปร (set) ซึ่งเป็นการนำข้อมูลไปเก็บไว้ ณ ตำแหน่งที่ตัวแปรอ้างถึง และการเรียกใช้ค่าในตัวแปร (get) ซึ่งเป็นการนำค่าที่เก็บไว้ ณ ตำแหน่งที่ตัวแปรอ้างถึงมาใช้งาน

### 3.5.2 ข้อมูลชนิดตัวชี้

การใช้ตัวแปรโดยทั่วไปนั้นเราต้องการค่าที่ถูกเก็บไว้ในหน่วยความจำ แต่ในบางกรณี เราจะเป็นต้องใช้เลขที่อยู่ในหน่วยความจำ เช่น การสั่งให้คอมพิวเตอร์อ่านข้อมูลไปเก็บไว้ในหน่วยความจำ เราต้องให้เลขที่อยู่สำหรับการเก็บข้อมูลในคำสั่งอ่าน

ในอีกทางหนึ่ง หากเราเก็บเลขที่อยู่ไว้ในหน่วยความจำ เราก็สามารถดึงข้อมูลที่เก็บอยู่ ณ เลขที่อยู่นั้นได้เช่นกัน การจองพื้นที่เพื่อเก็บเลขที่อยู่นี้ใช้ผ่านการประกาศตัวแปรชนิดตัวชี้ โดยต้องกำหนดชนิดของตัวชี้ตามชนิดของข้อมูลปลายทางด้วย

จากตารางที่ 3.3 หากให้ข้อมูลชนิดตัวชี้มีขนาด 1 ไบต์ และจำนวนเต็มมีขนาด 4 ไบต์ intPtr เป็นข้อมูลชนิดตัวชี้ของจำนวนเต็ม และคอมพิวเตอร์จองพื้นที่ ณ ตำแหน่ง 0x70 ให้สำหรับเก็บข้อมูล เมื่อกำหนดค่าให้ intPtr เป็น 0x74 การอ้างถึงข้อมูลปลายทางผ่านตัวชี้ intPtr จะได้ค่าจำนวนเต็มขนาด 4 ไบต์ที่เก็บอยู่ ณ ตำแหน่ง 0x74 - 0x77 ซึ่งมีค่าเป็น 0x00000064 เป็นต้น

### 3.5.3 ตัวแปรในภาษาซี

เมื่อโปรแกรมเริ่มทำงานและมีการประกาศใช้ตัวแปร ระบบปฏิบัติการจะจองพื้นที่ในหน่วยความจำให้เท่ากับขนาดของตัวแปรนั้น การอ้างอิงถึงพื้นที่ในหน่วยความจำในส่วนรหัสต้นฉบับจะทำผ่านตัวแปรโดยไม่จำเป็นต้องรู้ว่าได้หน่วยความจำที่ตำแหน่งใดมา เช่น สมมติให้ตัวแปรชนิดจำนวนเต็มมีขนาด 4 ไบต์ หากกำหนดตัวแปร number เป็นชนิดจำนวนเต็ม เครื่องจะจองพื้นที่ว่างติดกันขนาด 4 ไบต์ให้ สมมติว่าได้เป็นตำแหน่ง 0x74 ถึง 0x77 เป็นต้น

เมื่อใช้คำสั่งกำหนดค่าให้ตัวแปร ระบบปฏิบัติการจะกำหนดค่า ณ ตำแหน่งที่อ้างถึง เช่น เมื่อกำหนดให้ number มีค่าเป็น 100 ระบบปฏิบัติการจะกำหนดค่าให้ตำแหน่ง 0x74 ถึง 0x77 เก็บค่าเลขฐานสองในรูปแบบจำนวนเต็มขนาด 4 ไบต์ซึ่งแทนจำนวน 100 (0x00000064)

การเรียกใช้ค่าในตัวแปรทำได้โดยเรียกใช้ชื่อตัวแปรนั้น และจะหมายถึงค่าที่เก็บไว้ ณ ตำแหน่งนั้น เช่น การเรียกใช้ number จะได้ค่าเป็นจำนวนเต็มขนาด 4 ไบต์ที่เก็บไว้ ณ ตำแหน่ง 0x74-0x77 เป็นต้น

ในทำนองเดียวกัน หากกำหนดตัวแปร c ให้เป็นอักขระในรหัสแอสกี ซึ่งใช้พื้นที่ 1 ไบต์ เครื่องจะจองพื้นที่ในหน่วยความจำให้ สมมติว่าได้เป็นตำแหน่ง 0x7D เมื่อกำหนดค่าให้ตัวแปร c เป็น `'0'` จะเป็นการใส่รหัสแอสกีของเลขศูนย์ (0x30) ลงในหน่วยความจำ ณ ตำแหน่ง 0x7D และการเรียกใช้ตัวแปร c ก็จะได้ค่ารหัสแอสกีขนาด 1 ไบต์ ณ ตำแหน่ง 0x7D ออกมาเช่นกัน

จากที่กล่าวไปแล้วว่าข้อมูลทั้งหมดในคอมพิวเตอร์ถูกเก็บอยู่ในรูปแบบเลขฐาน 2 ในตารางที่ 3.3 หากมองข้อมูลที่ตำแหน่ง 0x77 เป็นอักขระในรหัส ISO8859-11 จะเห็นข้อมูล ณ ตำแหน่งนี้เป็นอักขระ d (0X64) ในการจองพื้นที่เพื่อเก็บข้อมูล จึงต้องกำหนดชนิดของข้อมูลเพื่อให้นำไปใช้งานต่อได้ถูกต้อง

ในการเขียนโปรแกรมนั้นภาษาซี ต้องกำหนดชนิดของข้อมูลที่ตัวแปรนั้นแทนอยู่เสมอ และควรตั้งชื่อให้สื่อความถึงข้อมูลที่เก็บอยู่ด้วย เมื่อจบการทำงานของโปรแกรม โปรแกรมจะคืนพื้นที่ในหน่วยความจำที่จองไว้สำหรับใช้งานให้ระบบปฏิบัติการ เพื่อนำไปจัดสรรให้โปรแกรมอื่นๆ ใช้ต่อไป

### ลองคิด

หากประกาศตัวแปรโดยไม่ได้กำหนดค่าให้ตัวแปร และเรียกใช้ตัวแปรนั้น ผลที่ได้น่าจะเป็นอย่างไร

## แบบฝึกหัด

กำหนดชนิดของข้อมูลและพื้นที่ในการเก็บในภาษาซีดังต่อไปนี้

| **ชนิดข้อมูล** | **พื้นที่** (ไบต์) | **ชนิดข้อมูล** | **พื้นที่** (ไบต์) |
|---|---|---|---|
| `char` | 1 | | |
| `short` | 2 | `int` | 4 |
| `unsigned` | 4 | `long` | 8 |
| `float` | 4 | `double` | 8 |

1. จงระบุพื้นที่น้อยที่สุดที่ใช้ในการเก็บข้อมูลต่อไปนี้
    a) สายอักขระค่า “100”
    b) จำนวนเต็มค่า 100
    c) จำนวนจริงค่า 100
    d) อักขระเลข ‘0’
    e) สายอักขระเลข “0”

2. จงระบุจำนวนตัวแปรที่ต้องใช้และชนิดของข้อมูลที่เหมาะสมในการเก็บข้อมูลต่อไปนี้
   a) เกรดประจำรายวิชา 2301170 ของนิสิต 1 คน
   b) หน่วยกิตสะสมตลอดหลักสูตรสำหรับนิสิต 1 คน
   c) ขนาดอะตอมในหน่วยนาโนเมตรนัยสำคัญ 3 ตำแหน่ง
   d) ค่าเฉลี่ยของเลข 10 จำนวน
   e) เลขปีคริสต์ศักราช
   f) เลขประจำตัวประชาชน 13 หลัก
3. checksum คือการเก็บข้อมูลเพิ่มหนึ่งค่าเพื่อตรวจสอบความถูกต้องของข้อมูลที่มีความยาวมาก ในเลขประจำตัวนิสิต 10 หลักนั้น สองหลักแรกคือปีที่เข้าศึกษา หลักที่สามคือระดับการศึกษา หลักที่ 4-7 เป็นลำดับนิสิต หลักที่ 8 คือค่า checksum หลักที่ 9-10 คือรหัสคณะ ต้องเก็บข้อมูลเลขประจำตัวนิสิตอย่างไรจึงจะนำมาคำนวณค่า checksum ได้ ให้อธิบายตัวแปรที่ใช้ ระบุชนิดและวิธีการใช้งานคร่าว ๆ
4. ต้องการคำนวณจำนวนชั่วโมงการใช้งานอินเทอร์เน็ตเพื่อนำไปคิดค่าบริการ โดยให้ป้อนข้อมูลเข้าเป็นเวลาเริ่มใช้งาน และเวลาสิ้นสุดการใช้งาน หากคิดค่าบริการเป็นนาที ควรเก็บข้อมูลอย่างไร ให้อธิบายตัวแปรต่างๆ ที่จำเป็นต้องใช้ ระบุชนิดของตัวแปรด้วย

# บทที่ 4

# การทำงานแบบลำดับ

## วัตถุประสงค์การเรียนรู้

อธิบายการทำงานของโปรแกรมในรูปการทำงานแบบลำดับ

1. ใช้ตัวดำเนินการต่างๆ เพื่อกำหนดการทำงานตามที่ต้องการได้ รวมถึงใช้สัญลักษณ์ของผังงานได้ถูกต้อง
2. อ่านรหัสเทียมหรือผังงานที่ใช้การทำงานแบบลำดับได้ ดัดแปลงหรือแก้ไขผังงานที่มีอยู่แล้วให้ทำงานได้ถูกต้องตามต้องการ
3. เขียนขั้นตอนการแก้ปัญหาที่ใช้การทำงานแบบลำดับได้
4. เขียนคำสั่งภาษาซีได้

ใช้เวลา 3 คาบ

## 4.1 ตัวดำเนินการ

โปรแกรมคอมพิวเตอร์เป็นชุดคำสั่งซึ่งให้คอมพิวเตอร์ทำงานตามที่กำหนด การเขียนโปรแกรมเพื่อแก้ปัญหาต่างๆ นั้น ต้องรู้ก่อนว่าคอมพิวเตอร์สามารถทำอะไรได้บ้าง และจะสั่งการให้คอมพิวเตอร์ทำงานเหล่านั้นได้อย่างไร จากนั้นจึงสร้างชุดคำสั่งซึ่งมีลำดับการทำงานที่ถูกต้องเพื่อการแก้ปัญหาที่ต้องการ

การสั่งให้คอมพิวเตอร์ทำงานงานพื้นฐานจำนวนหนึ่งใช้การสั่งผ่านตัวดำเนินการ (operator) โดยให้ดำเนินการกับตัวถูกดำเนินการ (operand) ซึ่งเป็นข้อมูลที่เก็บอยู่ในหน่วยความจำโดยอ้างถึงผ่านตัวแปรก็ได้ หรือเป็นค่าคงที่ก็ได้ เช่น x + 3 มีตัวดำเนินการคือเครื่องหมาย + และตัวถูกดำเนินการคือค่าในตัวแปร x และค่าคงที่ 2 ตามลำดับ

นอกจากตัวดำเนินการซึ่งใช้สำหรับทำงานพื้นฐานแล้ว ยังมีฟังก์ชันซึ่งเป็นชุดคำสั่งที่รวมขั้นตอนการทำงานพื้นฐานเอาไว้เพื่อทำงานบางอย่างอีกด้วย การเรียกใช้ฟังก์ชันจะกล่าวถึงในบทที่ 8

ตารางที่ 4.1: ตัวดำเนินการพีชคณิต

| ลำดับ | สัญลักษณ์ | ตัวดำเนินการในภาษาซี | การใช้งาน | ความหมาย |
|---|---|---|---|---|
| 1 | ^ หรือ ** | `pow(b,e)` | b ^e | การยกกำลัง ($b^e$) |
| 2 | * | * | A * 3 | การคูณ |
| 2 | / | / | A / 3 | การหารจำนวนจริง |
| 2 | DIV | / | A DIV 3 | การหารปัดเศษทิ้ง |
| 2 | MOD | % | A MOD 3 | การหารเอาเฉพาะเศษเหลือ |
| 3 | + | + | A + 3 | การบวก |
| 3 | - | – | A - 3 | การลบ |

นิพจน์ หมายถึง การใช้สัญลักษณ์ต่างๆ ไม่ว่าจะเป็นค่าคงที่ ตัวแปร หรือฟังก์ชันซึ่งคืนค่าออกมา ประกอบกับตัวดำเนินการต่างๆ ตามข้อกำหนดของตัวดำเนินการ และคำนวณผลลัพธ์ออกมาเป็นค่าค่าหนึ่งได้ เมื่อมีฟังก์ชันและตัวดำเนินการหลายๆ ตัว โดยทั่วไปลำดับความสำคัญในการทำงานจะเป็นดังนี้

1. ฟังก์ชัน
2. ตัวดำเนินการซึ่งมีตัวถูกดำเนินการเพียงตัวเดียว ที่ใช้บ่อยได้แก่ NOT (!), increment (++), decrement (-), วงเล็บ เป็นต้น
3. ตัวดำเนินการซึ่งมีตัวถูกดำเนินการสองตัว เช่น ตัวดำเนินการพีชคณิต ตัวดำเนินการเปรียบเทียบ ตัวดำเนินการตรรกะ การกำหนดค่า เป็นต้น

ในแต่ละหมวดอาจมีลำดับความสำคัญย่อยภายในหมวดด้วย หากมีตัวดำเนินการที่มีลำดับความสำคัญเท่ากัน คอมพิวเตอร์จะทำงานตามกฎการจัดหมู่ (associativity) ของตัวดำเนินการเหล่านั้น ซึ่งส่วนมาก จะเริ่มทำจากตัวดำเนินการทางซ้ายก่อน

### 4.1.1 ตัวดำเนินการพีชคณิต

ตัวดำเนินการพีชคณิตคือคำสั่งการคำนวณทางพีชคณิตเบื้องต้นต่างๆ ลำดับความสำคัญในการทำงานเป็นดังตารางที่ 4.1 เริ่มจากการยกกำลัง ซึ่งบางภาษาโปรแกรมเช่นภาษาซีสร้างเป็นฟังก์ชันมาให้ใช้งาน จึงมีลำดับความสำคัญของฟังก์ชันสูงจะกว่าการคำนวณเบื้องต้นต่างๆ อยู่แล้ว การคูณและหารทุกรูปแบบ ตามด้วยบวกและลบ

หากมีตัวดำเนินการยกกำลังแยกต่างหาก ไม่เป็นฟังก์ชัน กฎการจัดหมู่ของการยกกำลังคือทำทางขวาก่อน ส่วนตัวดำเนินการบวก ลบ คูณ และหาร ทำทางซ้ายก่อนเสมอ

ตารางที่ 4.2: ตัวดำเนินการเปรียบเทียบ

| ลำดับ | สัญลักษณ์ | ตัวดำเนินการในภาษาซี | การใช้งาน | ความหมาย |
|---|---|---|---|---|
| 1 | > | > | A > 2 | มากกว่า |
| 1 | ≥ | >= | A ≥ 2 | มากกว่าหรือเท่ากับ |
| 1 | < | < | A < 2 | น้อยกว่า |
| 1 | ≤ | <= | A ≤ 2 | น้อยกว่าหรือเท่ากับ |
| 2 | = | == | A = 2 | เปรียบเทียบการเท่ากัน |
| 2 | ≠ | != | A ≠ 2 | เปรียบเทียบการไม่เท่ากัน |

ตารางที่ 4.3: ตัวดำเนินการตรรกะ

| ลำดับ | สัญลักษณ์ | ตัวดำเนินการในภาษาซี | การใช้งาน | ความหมาย |
|---|---|---|---|---|
| 1 | NOT | ! | NOT(A) | ให้ค่าความจริงตรงกันข้ามกับค่าเดิม |
| 2 | AND | && | A AND B | ให้ค่าจริงเมื่อ A และ B เป็นจริง |
| 3 | OR | \|\| | A OR B | ให้ค่าจริงเมื่อ A หรือ B เป็นจริง |

### 4.1.2 ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบเป็นคำสั่งเปรียบเทียบค่าระหว่างตัวถูกดำเนินการทั้งสองข้างของเครื่องหมาย ให้ผลลัพธ์เป็นค่าจริงหรือเท็จ

ลำดับความสำคัญในการทำงานคือ เปรียบเทียบแบบมากกว่าหรือน้อยกว่าก่อน แล้วจึงเปรียบเทียบการเท่ากัน ดังตารางที่ 4.2 แต่บางภาษา เช่น Python กำหนดลำดับความสำคัญของตัวดำเนินการเปรียบเทียบทุกรูปแบบไว้เท่ากัน ก่อนเขียนโปรแกรมจึงควรตรวจสอบข้อกำหนดของแต่ละภาษาอีกครั้ง

เครื่องหมาย = นั้นมีสองความหมาย นัยหนึ่งหมายถึงการเปรียบเทียบการเท่ากัน แต่เครื่องหมาย = ก็ใช้เป็นตัวดำเนินการกำหนดค่าในหลายๆ ภาษาโปรแกรมด้วย จึงนิยมเขียน == เหมือนในภาษาโปรแกรมเพื่อแสดงการเปรียบเทียบมากกว่า

### 4.1.3 ตัวดำเนินการตรรกะ

ตัวดำเนินการตรรกะเป็นคำสั่งการคำนวณทางตรรกะของตัวถูกดำเนินการ ให้ผลลัพธ์เป็นค่าจริงหรือเท็จ

เนื่องจาก NOT เป็นตัวดำเนินการซึ่งมีตัวถูกดำเนินการเพียงตัวเดียว จึงมีลำดับความสำคัญสูงที่สุด ถัดมาเป็นคำสั่ง AND และคำสั่ง OR ตามลำดับ ดังตารางที่ 4.3

### 4.1.4 การกำหนดค่า

การกำหนดค่าในภาษาโปรแกรมและผังงานส่วนมากใช้ตัวดำเนินการ = หรือ := และมีตัวถูกดำเนินการสองตัว ซึ่งหมายถึง การนำ**ค่า**ของตัวถูกดำเนินการ**ทางขวา** ในรหัสคำสั่งที่ 4.1 คือ source ไปเก็บใน**ที่อยู่**ของตัวดำเนินการ**ทางซ้าย** ในที่นี้คือ target ตัวถูกดำเนินการทางขวาอาจจะเป็นตัวแปร ค่าคงที่ หรือนิพจน์ก็ได้

```
target = source
```

รหัสคำสั่งที่ 4.1: การกำหนดค่า

การกำหนดค่ามีลำดับความสำคัญต่ำที่สุดในบรรดาคำสั่งและตัวดำเนินการทั้งหมดที่กล่าวมาแล้ว ซึ่งทำให้การกำหนดค่าต้องทำนิพจน์ทางขวาให้เป็นผลลัพธ์ก่อน แล้วจึงดำเนินการกำหนดค่าให้พื้นที่ที่จองไว้ซึ่งระบุด้วยตัวแปรทางซ้ายของตัวดำเนินการ

### 4.1.5 การอ่านเขียนข้อมูลเบื้องต้น

จากที่กล่าวไปแล้วในบทที่ 1 และ 3 การทำงานกับข้อมูลของคอมพิวเตอร์นั้นจะเก็บข้อมูลในหน่วยความจำเสมอ ซึ่งสามารถอ้างถึงหน่วยความจำนั้นได้ผ่านตัวแปร การอ่านหรือรับข้อมูลเข้ามาเก็บอยู่ในคอมพิวเตอร์ จึงเป็นการรับข้อมูลจากอุปกรณ์ภายนอกแล้วนำมาเก็บในหน่วยความจำ ณ ตำแหน่งที่กำหนดด้วยชื่อตัวแปร ในขณะที่การเขียนข้อมูล คือการนำค่าที่เก็บอยู่ในหน่วยความจำซึ่งอ้างถึงด้วยชื่อตัวแปรออกมาแสดงผลผ่านอุปกรณ์ที่กำหนด โดยปกติแล้ว วิธีการอ่านเขียนข้อมูลจะเป็นฟังก์ชันอยู่ในคลังโปรแกรม

การอ่านข้อมูลครั้งหนึ่งสามารถอ่านค่าได้มากกว่าหนึ่งค่า และต้องใช้ตัวแปรจำนวนเท่ากันในการรับค่าที่อ่านเข้ามาด้วย ในทำนองเดียวกัน การเขียนข้อมูลก็เขียนได้มากกว่าหนึ่งค่าในหนึ่งครั้ง และต้องระบุค่าที่ต้องการให้เขียนทุกค่าลงไป สำหรับการเขียนข้อมูลนั้น นอกจากจะเขียนค่าที่เก็บอยู่ในตัวแปรโดยตรงได้แล้ว ยังสามารถเขียนค่าคงที่ และเป็นนิพจน์ต่างๆ ซึ่งถูกทำให้เป็นผลสำเร็จและกลายเป็นค่าคงที่ได้อีกด้วย

#### ลองคิด

จะเห็นว่าตัวดำเนินการต่างๆ โดยเฉพาะการคำนวณใช้กับข้อมูลชนิดจำนวนเป็นหลัก เราสามารถใช้ตัวดำเนินการเพื่อการคำนวณกับข้อมูลชนิดอักขระได้หรือไม่ จงให้เหตุผลประกอบ

## 4.2 การออกแบบโปรแกรม

การแก้ปัญหาโดยคอมพิวเตอร์คือการเขียนลำดับการทำงานอย่างเป็นขั้นเป็นตอนเพื่อให้คอมพิวเตอร์ทำตาม เมื่อสามารถแทนข้อมูลเพื่อใช้ในการทำงานของคอมพิวเตอร์ได้แล้ว ลำดับต่อไปคือการนำการดำเนินการต่างๆ มาเรียงเป็นขั้นตอนเพื่อแก้ปัญหาที่กำหนด

### 4.2.1 ขั้นตอนวิธี

การออกแบบโปรแกรมคือการออกแบบขั้นตอนวิธีเพื่อแก้ปัญหาที่กำหนด โดยทั่วไปมักเริ่มจากกำหนดรูปแบบของข้อมูลเข้าและข้อมูลออกก่อน จากนั้นจึงแบ่งงานที่ต้องทำในการแก้ปัญหาออกเป็นส่วนย่อยๆ แล้วเขียนลำดับคำสั่งเพื่อแก้ปัญหาย่อยเหล่านั้น วิธีการออกแบบโปรแกรมแบบนี้เรียกว่า การออกแบบแบบบนลงล่าง (top-down approach)

**ตัวอย่าง 4.1** (การบวกเลขสองจำนวน). การออกแบบการทำงานของเครื่องบวกเลข จะเริ่มจากการกำหนดลักษณะข้อมูลเข้าและข้อมูลออกก่อน ให้เครื่องบวกเลขรับจำนวนสองจำนวน แล้วคืนผลลัพธ์เป็นผลบวกของสองจำนวนนั้น ลักษณะของข้อมูลเข้าจะเป็นจำนวน 2 ค่า และข้อมูลออกจะเป็นจำนวน 1 ค่า จากนั้นออกแบบขั้นตอนการทำงาน เขียนเป็นลำดับการทำงานได้ดังนี้

1. รับข้อมูลเข้า 2 จำนวน
2. หาผลบวก เก็บผลลัพธ์เอาไว้
3. ส่งผลลัพธ์คืนเป็นข้อมูลออก

จากนั้นจึงเปลี่ยนลำดับการทำงานนี้เป็นคำสั่งสำหรับคอมพิวเตอร์ต่อไป □

**ตัวอย่าง 4.2** (การหาค่าเฉลี่ยของเลขสามจำนวน). การออกแบบโปรแกรมนี้ก็คล้ายกับตัวอย่าง 4.1 คือมีข้อมูลเข้าเป็นจำนวน 3 ค่า และข้อมูลออกเป็นจำนวน 1 ค่า และเขียนลำดับการทำงานได้ในทำนองเดียวกัน ดังนี้

1. รับข้อมูลเข้า 3 จำนวน
2. หาค่าเฉลี่ย เก็บผลลัพธ์เอาไว้
3. ส่งผลลัพธ์คืนเป็นข้อมูลออก

และแปลงลำดับการทำงานนี้เป็นคำสั่งสำหรับคอมพิวเตอร์

หรืออีกวิธีหนึ่ง เราอาจแยกขั้นตอนที่ 2 ออกเป็น 2 ขั้นตอนย่อยเพื่อหาค่าเฉลี่ยได้ดังนี้

1. รับข้อมูลเข้า 3 จำนวน
2. หาค่าเฉลี่ย เก็บผลลัพธ์เอาไว้ โดยมีขั้นตอนดังนี้
    a) หาผลบวกของสามจำนวน เก็บผลลัพธ์เอาไว้
    b) นำผลลัพธ์ข้างต้นหารด้วย 3 เก็บไว้เป็นผลลัพธ์สุดท้าย
3. ส่งผลลัพธ์คืนเป็นข้อมูลออก

□

ข้อสังเกตสำคัญจากตัวอย่าง 4.2 คือในขั้นตอนที่ 2 ของแบบแรกนั้น เราให้คอมพิวเตอร์หาค่าเฉลี่ย แสดงว่าคอมพิวเตอร์ต้องมีคำสั่งสำหรับหาค่าเฉลี่ยของจำนวน 3 จำนวนอยู่แล้ว หากไม่มี การเขียนขั้นตอนการทำงานแบบนี้จะนำไปแปลงเป็นคำสั่งในคอมพิวเตอร์ไม่ได้ ในการออกแบบขั้นตอนการทำงาน

ผู้ออกแบบจึงจำเป็นต้องรู้ความสามารถของคอมพิวเตอร์ด้วย และการออกแบบโปรแกรมนั้นจึงเป็นการแตกการทำงานออกมาเป็นขั้นตอนพื้นฐานย่อยๆ ที่คอมพิวเตอร์สามารถทำงานได้

### 4.2.2 รหัสเทียม (pseudocode) และ ผังงาน (flowchart)

รหัสเทียม คือชุดคำสั่งคอมพิวเตอร์ซึ่งใช้แสดงขั้นตอนการทำงานต่างๆ รหัสเทียมมีลักษณะคล้ายรหัสคำสั่งภาษาต่างๆ แต่ไม่มีข้อกำหนดเรื่องไวยากรณ์มากเท่ารหัสคำสั่ง ข้อดีของการใช้รหัสเทียมคือสามารถแปลงไปเป็นรหัสคำสั่งตามภาษาที่ต้องการได้สะดวก เนื่องจากโครงสร้างคำสั่งคล้ายกันมาก แต่หากขั้นตอนวิธีนั้นซับซ้อน รหัสเทียมจะเข้าใจได้ยากกว่าการใช้แผนภาพ

แผนภาพหนึ่งที่ใช้ในการออกแบบขั้นตอนการทำงานคือผังงาน ผังงานประกอบด้วยสัญลักษณ์กำหนดขั้นตอนการทำงาน และลูกศรแสดงลำดับขั้นตอนการทำงาน มีการกำหนดมาตรฐานความหมายของสัญลักษณ์ต่างๆ ในผังงานเพื่อให้เข้าใจได้ตรงกัน มาตรฐานที่ใช้กันมากคือ ISO5807:1985 สัญลักษณ์ที่ใช้กันทั่วไป เป็นดังตารางที่ 4.4

ลำดับการทำงานในผังงานกำหนดด้วยลูกศร การทำงานจะเริ่มจากสัญลักษณ์เริ่มต้น ทำงานตามลูกศรไปทีละขั้นตอน และไปสิ้นสุดที่สัญลักษณ์สิ้นสุดการทำงาน สัญลักษณ์การอ่านเขียนข้อมูล การทำงาน และการเรียกใช้ขั้นตอนการทำงานที่กำหนดไว้ล่วงหน้านั้น จะมีลูกศรเข้าหนึ่งทาง และลูกศรออกหนึ่งทางเสมอ ในขณะที่สัญลักษณ์เริ่มต้นจะไม่มีลูกศรเข้า มีเฉพาะลูกศรออก และสัญลักษณ์จบการทำงานจะมีเฉพาะลูกศรเข้า และไม่มีลูกศรออก ส่วนสัญลักษณ์จุดเชื่อมต่อข้ามหน้าใช้ลูกศรกำหนดทิศทางการทำงานเช่นเดียวกับสัญลักษณ์เริ่มต้นและสิ้นสุด

สัญลักษณ์เงื่อนไขมีลูกศรเข้าได้ทางเดียว แต่มีลูกศรออกได้หลายทาง การเลือกเส้นทางขึ้นกับผลของเงื่อนไขที่ระบุในสัญลักษณ์ ไม่ว่าจะแยกไปทำงานตามทางเลือกใดก็ตาม เมื่อทำงานจบแล้วก็จะกลับมาทำงานตามขั้นตอนต่อไปเสมอ จึงใช้สัญลักษณ์จุดเชื่อมต่อในการรวมเส้นทางต่างๆ เข้าด้วยกัน แล้วกำหนดให้มีลูกศรออกจากจุดเชื่อมต่อเพียงเส้นทางเดียว เพื่อไปทำงานตามขั้นตอนต่อไป

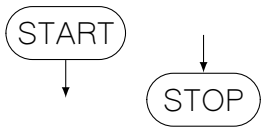
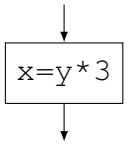
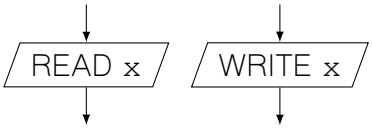
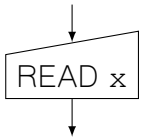
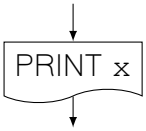
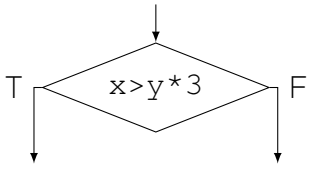
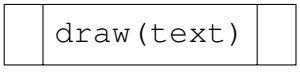
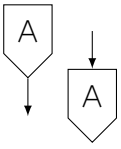
ตัวอย่าง 4.1 เมื่อนำมาเขียนเป็นผังงาน จะได้ผังงานดังรูปที่ 4.1 ส่วนตัวอย่าง 4.2 เมื่อนำมาเขียนเป็นผังงาน จะได้ผังงานดังรูปที่ 4.2

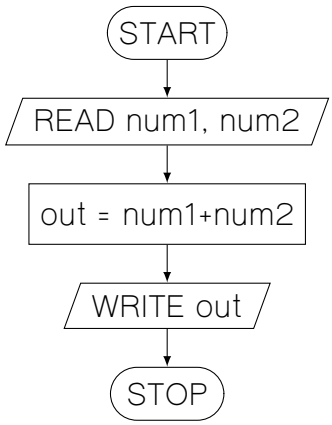
### 4.2.3 แผนภาพอื่นๆ ในการออกแบบ

ผังงานมักใช้ในการออกแบบขั้นตอนการทำงาน แต่ในการออกแบบโปรแกรมขนาดใหญ่จำเป็นต้องมีการระบุองค์ประกอบต่างๆ ในโปรแกรม และแผนภาพอธิบายการทำงานร่วมกันระหว่างส่วนต่างๆ แผนภาพสำหรับการออกแบบอื่นๆ ที่นิยมใช้กันเช่น UML, Data Flow Diagram เป็นต้น

ไม่ว่าจะใช้แผนภาพรูปแบบใดก็ตาม สิ่งสำคัญคือต้องใช้สัญลักษณ์ที่เป็นมาตรฐานเพื่อให้เข้าใจได้ตรงกัน วางโครงให้สะอาดสะอ้าน อ่านง่าย ระบุชื่อผู้เขียนและวันที่ปรับปรุงแก้ไขไว้เสมอเพื่อการติดตามในอนาคต

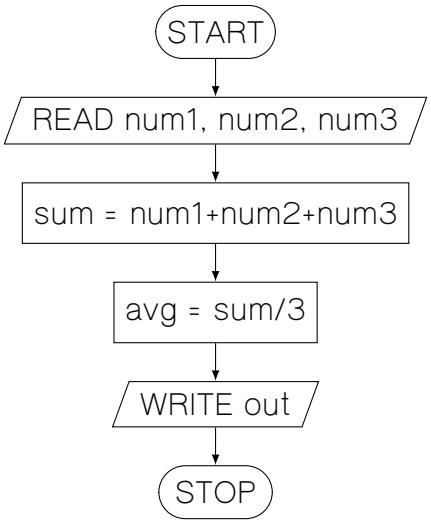
ตารางที่ 4.4: สัญลักษณ์ของผังงานและความหมาย

| สัญลักษณ์ | ความหมาย |
|---|---|
|  | การเริ่มต้นหรือสิ้นสุดการทำงาน โดยเขียนกำกับภายในสัญลักษณ์ด้วยข้อความระบุการเริ่มต้น/สิ้นสุด เช่น START/STOP หรือ BEGIN/END เสมอ |
|  | การทำงานหรือการกำหนดค่า โดยเขียนกำกับภายในสัญลักษณ์สำหรับวิชานี้ใช้กับการคำนวณต่างๆ โดยใช้ตัวดำเนินการที่กล่าวไปแล้วข้างต้น ร่วมกับการกำหนดค่าให้ตัวแปร |
|  | การอ่าน/เขียนข้อมูลโดยไม่ระบุสื่อ โดยเขียนคำสั่งระบุการอ่านหรือเขียน เช่น READ/INPUT พร้อมที่อยู่สำหรับเก็บค่าที่รับเข้า หรือ WRITE/OUTPUT/PRINT พร้อมค่าที่ต้องการแสดงผลกำกับในสัญลักษณ์เสมอ |
|  | การรับข้อมูลผ่านแป้นพิมพ์ โดยเขียนคำสั่งระบุการรับข้อมูล เช่น INPUT/READ พร้อมที่อยู่สำหรับเก็บค่าที่รับเข้ากำกับในสัญลักษณ์เสมอ |
|  | การแสดงผลข้อมูลผ่านเครื่องพิมพ์ โดยเขียนคำสั่งระบุการพิมพ์ เช่น PRINT พร้อมค่าที่ต้องการแสดงผลกำกับในสัญลักษณ์เสมอ |
|  | ระบุเงื่อนไขในสัญลักษณ์เพื่อกำหนดทางเลือกในการทำงาน ลำดับการทำงานต่อไปมีได้หลายรูปแบบขึ้นกับผลการดำเนินการตามเงื่อนไข |
|  | ขั้นตอนการทำงานที่กำหนดไว้ล่วงหน้า เช่นการเรียกใช้ฟังก์ชันหรือโปรแกรมย่อยที่กำหนดเอง |
|  | จุดเชื่อมต่อภายในหน้า ใช้รวมทางแยกการทำงานให้เป็นทางเดียว ใช้ในการทำงานแบบทางเลือกและการวนซ้ำ |
|  | จุดเชื่อมต่อข้ามหน้า ต้องเขียนหมายเลขอ้างอิงการเชื่อมต่อไว้ภายในสัญลักษณ์เสมอ ใช้แทนการสิ้นสุดการทำงานในหน้าต้นทาง และใช้แทนการเริ่มต้นการทำงานในหน้าปลายทาง และต้องใช้หมายเลขอ้างอิงเดียวกันทั้งต้นทางและปลายทาง |




1. รับข้อมูลเข้า 2 จำนวน
2. หาผลบวก เก็บผลลัพธ์เอาไว้
3. ส่งผลลัพธ์คืนเป็นข้อมูลออก

รูปที่ 4.1: ผังงานและขั้นตอนวิธีสำหรับตัวอย่าง 4.1




1. รับข้อมูลเข้า 3 จำนวน
2. หาผลบวก เก็บผลลัพธ์เอาไว้
3. นำผลลัพธ์ข้างต้นหารด้วย 3 เก็บไว้เป็นผลลัพธ์สุดท้าย
4. ส่งผลลัพธ์คืนเป็นข้อมูลออก

รูปที่ 4.2: ผังงานและขั้นตอนวิธีสำหรับตัวอย่าง 4.2

**ลองคิด**

หากกำหนดโจทย์ให้ มีกรณีใดบ้างหรือไม่ ที่สามารถแก้ปัญหาจริงได้ แต่ไม่สามารถเขียนขั้นตอนการแก้ปัญหาออกมาเป็นผังงานได้ ให้อธิบายเหตุผลประกอบ

## 4.3 การเขียนและตรวจสอบโปรแกรม

เมื่อออกแบบขั้นตอนวิธีเสร็จแล้ว ควรตรวจสอบความถูกต้องของการออกแบบด้วยการสมมติชุดข้อมูลเข้าและออก แล้วทดลองทำงานตามขั้นตอนที่ออกแบบไว้ หากได้ผลลัพธ์ตรงตามที่ต้องการ แสดงว่าขั้นตอนวิธีที่ออกแบบไว้นั้นถูกต้อง อย่างไรก็ตาม การตรวจสอบขั้นตอนวิธีนี้เป็นเพียงการทดสอบเบื้องต้นเท่านั้น ในการใช้งานจริงนั้นโปรแกรมยังอาจมีความผิดพลาดเกิดขึ้นได้อีกหลายรูปแบบ หลังจากทดสอบ

เบื้องต้นแล้ว จึงควรเขียนโปรแกรมและทดสอบการทำงานจริงของโปรแกรมด้วย

### 4.3.1 การเขียนโปรแกรม

ขั้นตอนวิธีที่ออกแบบนั้นเขียนในรูปแบบที่เข้าใจง่าย เป็นภาษามนุษย์ แต่คอมพิวเตอร์ทำงานโดยใช้ภาษาโปรแกรม จึงต้องแปลงขั้นตอนวิธีที่ออกแบบให้เป็นภาษาโปรแกรมก่อน หากออกแบบไว้ได้ดีพอ การแปลงขั้นตอนวิธีไปเป็นภาษาโปรแกรมจะตรงไปตรงมา ทำได้ง่าย

ผู้เขียนโปรแกรมสามารถเลือกภาษาโปรแกรมได้หลากหลายตามความถนัดและสภาพแวดล้อมที่กำหนด ผู้เขียนต้องศึกษารูปแบบคำสั่งในภาษาต่างๆ แล้วจึงแปลงการออกแบบให้เป็นรหัสต้นฉบับในภาษานั้นๆ ปัจจุบันมีอุปกรณ์ช่วยเหลือในการแปลงการออกแบบให้เป็นรหัสต้นฉบับอยู่พอสมควร โดยเฉพาะอย่างยิ่งการออกแบบสมัยใหม่อย่าง UML มี IDE เช่น Eclipse, Microsoft Visual Studio สนับสนุนการแปลงการออกแบบเป็นรหัสต้นฉบับโดยอัตโนมัติ

หลังจากได้รหัสต้นฉบับแล้ว จะอาศัยโปรแกรมแปลภาษาดังที่กล่าวแล้วในบทที่ 1 เพื่อแปลรหัสต้นฉบับในภาษาโปรแกรมให้กลายเป็นภาษาเครื่อง แล้วอาศัยโปรแกรมเชื่อมโยงในการเชื่อมโยงออบเจกต์โค้ดที่เกี่ยวข้องเพื่อสร้างเป็นโปรแกรมที่ทำงานได้ต่อไป

### 4.3.2 ข้อผิดพลาดรูปแบบต่าง ๆ

โปรแกรมที่ออกแบบหรือเขียนขึ้นอาจมีข้อผิดพลาดต่างๆ ได้ 3 รูปแบบ ดังนี้

1. ข้อผิดพลาดทางตรรกะ (logical error) คือการได้ผลลัพธ์ไม่ตรงตามที่ควรจะเป็น ข้อผิดพลาดนี้อาจเกิดจากการออกแบบที่ไม่ถูกต้อง หรือการแปลงการออกแบบเป็นรหัสต้นฉบับที่ไม่ถูกต้อง
2. ข้อผิดพลาดทางโครงสร้างภาษา (syntax error) คือการเขียนรหัสต้นฉบับไม่ถูกต้องตามข้อกำหนดของภาษาโปรแกรมนั้นๆ ทำให้ไม่สามารถแปลรหัสต้นฉบับออกมาเป็นภาษาเครื่องได้ โดยปกติโปรแกรมแปลภาษาจะบอกข้อผิดพลาดนี้ในการแปล
3. ข้อผิดพลาดขณะทำงาน (runtime error) แม้การออกแบบจะถูกต้อง เขียนรหัสต้นฉบับได้ถูกต้อง ก็อาจมีข้อผิดพลาดที่เกิดขึ้นจากข้อมูลเข้าที่ป้อน หรือข้อจำกัดต่างๆ ของสภาพแวดล้อมที่โปรแกรมไม่สามารถรู้ล่วงหน้าได้

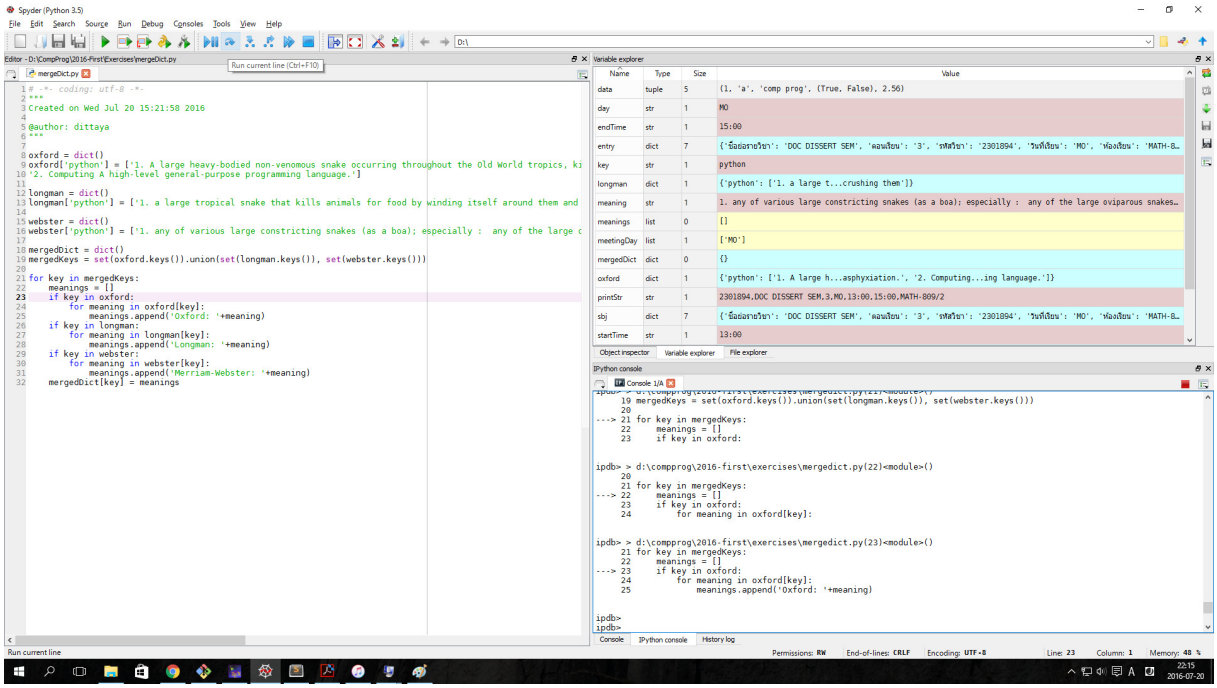
### 4.3.3 การตรวจสอบโปรแกรม

จากข้อผิดพลาดทั้งสามรูปแบบนั้น ข้อผิดพลาดทางโครงสร้างภาษาเป็นข้อผิดพลาดที่ตรวจพบได้ง่ายที่สุด เพราะโปรแกรมแปลภาษาจะแจ้งให้ทราบทันที แต่ข้อผิดพลาดทางตรรกะและข้อผิดพลาดขณะทำงานนั้นต้องอาศัยข้อมูลทดสอบจึงจะตรวจสอบได้

หากชุดทดสอบนั้นครอบคลุมข้อมูลทุกรูปแบบที่เป็นไปได้ เราจะสามารถยืนยันได้ว่าโปรแกรมทำงานถูกต้องแน่นอน แต่ส่วนใหญ่ข้อมูลที่เป็นไปได้จะมีไม่จำกัด การเลือกตัวอย่างข้อมูลเพื่อมาทดสอบจึงควร

ตารางที่ 4.5: การตรวจสอบค่าตัวแปร สำหรับตัวอย่างที่ 4.1

| คำสั่ง | num1 | num2 | out |
|---|---|---|---|
| READ num1, num2 | | | |
| out = num1 + num2 | | | |
| WRITE out | | | |



รูปที่ 4.3: ตัวอย่างการทำงานของโปรแกรมตรวจแก้จุดบกพร่อง

เลือกให้ครอบคลุมกรณีต่างๆ อย่างครบถ้วน เช่น ขอบของค่าต่างๆ กรณีที่เป็น 0 ซึ่งมักมีปัญหากับการหาร กรณีที่ข้อมูลเป็นลบ กรณีที่ข้อมูลเต็มหรือเกินจำนวนที่น่าจะเก็บได้

หลังจากเลือกชุดข้อมูลทดสอบได้แล้ว จะเป็นการทดลองให้โปรแกรมทำงานด้วยข้อมูลทดสอบ หากเป็นผังงาน เราอาจจำลองการทำงานได้โดยตารางตรวจสอบค่าตัวแปร ให้แต่ละบรรทัดในตารางแทนคำสั่งต่างๆ และคอลัมน์ในตารางแทนตัวแปร ค่าในช่องของตารางจะเป็นค่าของตัวแปรนั้นเมื่อทำคำสั่งเสร็จ ตารางที่ 4.5 เป็นตารางตรวจสอบค่าตัวแปรสำหรับตัวอย่างที่ 4.1

โปรแกรมตรวจแก้จุดบกพร่อง (debugger) เป็นโปรแกรมที่ทำงานแบบเดียวกับตารางตรวจสอบค่าตัวแปร โดยโปรแกรมจะทำงานไปทีละคำสั่ง และแสดงค่าในตัวแปรแต่ละตัวให้เห็น ซอฟต์แวร์เพื่อพัฒนาโปรแกรมในปัจจุบันมักจะรวมโปรแกรมตรวจแก้จุดบกพร่องนี้เข้าไปด้วย รูปที่ 4.3 เป็นซอฟต์แวร์เพื่อพัฒนาโปรแกรมของภาษาไพธอน ซึ่งกำลังตรวจแก้จุดบกพร่องอยู่ ฝั่งซ้ายของจอภาพแสดงรหัสคำสั่ง และสามารถสั่งทำงานทีละบรรทัดได้ ฝั่งล่างขวาแสดงส่วนของรหัสคำสั่งที่กำลังทำงาน ส่วนฝั่งขวาบนแสดงรายการตัวแปรและค่าของตัวแปรขณะกำลังทำงาน

**ลองคิด**

จากตัวอย่างที่ 4.1 จงหาชุดข้อมูลเข้าเพื่อทดสอบโปรแกรมที่ครอบคลุมกรณีต่างๆ ได้ครบถ้วนเพียงพอ

## 4.4 คำสั่งพื้นฐานในภาษาซี

คำสั่งที่พบบ่อยในภาษาซีที่จะกล่าวถึงในบทนี้ได้แก่ การประกาศตัวแปร การกำหนดค่า และการอ่านเขียนข้อมูลเบื้องต้น

### 4.4.1 การประกาศตัวแปรและการกำหนดค่า

การประกาศตัวแปรคือการจองพื้นที่ในหน่วยความจำสำหรับข้อมูลชนิดที่ต้องการ ในภาษาซี การประกาศตัวแปรทำได้ดังรหัสคำสั่งที่ 4.2

```
<data type> <name>;
```

รหัสคำสั่งที่ 4.2: การประกาศตัวแปรในภาษาซี

โดยที่ <type> คือชนิดของตัวแปร และ <name> คือชื่อของตัวแปร และปิดท้ายด้วย ;[1]

วิธีหนึ่งในการกำหนดค่าให้ตัวแปรคือใช้ตัวดำเนินการ = ซึ่งมีตัวถูกดำเนินการสองตัว อยู่ทางซ้ายและขวาของเครื่องหมาย ดังรหัสคำสั่งที่ 4.3

```
<var name> = <value>;
```

รหัสคำสั่งที่ 4.3: การกำหนดค่าให้ตัวแปรในภาษาซี

หมายถึง การนำค่า <value> ที่อยู่ทางขวา ไปเก็บไว้ในที่อยู่ <var> ทางซ้าย ตัว <value> จะเป็นตัวแปรหรือนิพจน์ก็ได้ หากเป็นนิพจน์ เครื่องจะคำนวณให้เป็นผลสำเร็จก่อนจะนำค่าผลสำเร็จไปใช้

การประกาศตัวแปรและการกำหนดค่าสามารถทำรวมกันในคำสั่งเดียวกันได้ ดังรหัสคำสั่งที่ 4.4

```
<data type> <var name> = <value>;
```

รหัสคำสั่งที่ 4.4: การประกาศตัวแปรพร้อมกับกำหนดค่าในภาษาซี

ชนิดของตัวแปรในภาษาซีที่ใช้บ่อย เป็นดังตารางที่ 4.6 ส่วนตัวแปรชนิดสายอักขระในภาษาซีกำหนดชนิดเป็น `char*` หรือตัวชี้ของอักขระ เนื่องจากภาษาซีเก็บสายอักขระด้วยแถวลำดับของอักขระ การประกาศตัวแปรจึงใช้หลักการเดียวกับการประกาศตัวแปรของแถวลำดับ นอกจากนี้ ยังสามารถประกาศตัวแปรสายอักขระไปพร้อมกับการกำหนดค่าเริ่มต้นด้วยการใช้ค่าคงที่ได้อีกด้วย ส่วนการกำหนดค่าให้ตัวแปรชนิดสายอักขระภายหลังการประกาศตัวแปรต้องทำผ่านฟังก์ชันการจัดการสายอักขระเท่านั้น

[1]โครงสร้างคำสั่งในภาษาซีจะปิดท้ายคำสั่งด้วย ; เสมอ

ตารางที่ 4.6: ชนิดของข้อมูลที่ใช้บ่อยในภาษาซี

| ชนิดตัวแปร | ค่าที่เก็บได้ |
|---|---|
| int | จำนวนเต็ม |
| float | จำนวนจริงนัยสำคัญปกติ |
| double | จำนวนจริงนัยสำคัญสูง |
| char | อักขระ |

ตารางที่ 4.7: รูปแบบข้อมูลเข้าและวิธีระบุที่อยู่สำหรับคำสั่ง scanf

| รูปแบบข้อมูลเข้า | การระบุที่อยู่ |
|---|---|
| %d | &intVar |
| %f | &floatVar |
| %c | &charVar |
| %s | strVar |

### 4.4.2 การอ่านข้อมูลเข้า

การรับข้อมูลเข้าอย่างง่ายคือการรับข้อมูลเข้าผ่านแป้นพิมพ์ ในภาษาซี มีฟังก์ชัน scanf ไว้เพื่ออ่านข้อมูลเข้าและเก็บข้อมูลเป็นชนิดที่กำหนดได้ โครงสร้างคำสั่ง scanf เป็นดังรหัสคำสั่งที่ 4.5

```
scanf("<format>",<addlist>);
```

รหัสคำสั่งที่ 4.5: โครงสร้างคำสั่ง scanf

โดยที่ format คือสายอักขระบ่งบอกรูปแบบของข้อมูลเข้า และ addlist คือรายการเลขที่อยู่สำหรับเก็บข้อมูล ซึ่งเป็นตัวแปรสำหรับเก็บข้อมูล แต่ละตัวคั่นด้วยจุลภาค (,)

รูปแบบของข้อมูลเข้าที่พบบ่อยเป็นดังตารางที่ 4.7 โดยที่ intVar, floatVar, charVar และ strVar เป็นตัวแปรชนิดจำนวนเต็ม จำนวนจริง อักขระ และสายอักขระตามลำดับ

การใส่ & นำหน้าชื่อตัวแปร เป็นการอ้างถึงเลขที่อยู่ของตัวแปรนั้น ในกรณีของสายอักขระ ซึ่งตัวแปรเก็บเลขที่อยู่ตำแหน่งแรกของสายอักขระอยู่แล้ว จึงไม่จำเป็นต้องใส่ & นำหน้าอีก

### 4.4.3 การแสดงผลทางหน้าจอ

การแสดงผลทางหน้าจอในภาษาซีใช้คำสั่ง printf ซึ่งมีโครงสร้างคล้าย scanf ดังรหัสคำสั่งที่ 4.6

```
printf("<format>",<varlist>);
```

รหัสคำสั่งที่ 4.6: โครงสร้างคำสั่ง printf

โดยที่ `format` คือรูปแบบของข้อมูลเข้า ใช้รูปแบบเดียวกับคำสั่ง `scanf` ในตารางที่ 4.7 และ `varlist` คือรายการตัวแปรที่เก็บข้อมูล หากมีมากกว่าหนึ่งตัวให้คั่นด้วยจุลภาค (,) เช่นเดียวกัน

รหัสคำสั่งที่ 4.7 คือคำสั่งสำหรับการทำงานในตัวอย่างที่ 4.1 บรรทัดที่ 1 สั่งให้รับข้อมูลเข้าเป็นจำนวนเต็ม 2 จำนวน และบรรทัดที่ 3 สั่งให้แสดงค่าที่อยู่ในตัวแปร `out` ซึ่งเป็นจำนวนเต็มออกทางหน้าจอ

```
1 scanf("%d %d", &num1, &num2);
2 out = num1+num2;
3 printf("%d", out);
```

รหัสคำสั่งที่ 4.7: รหัสคำสั่งสำหรับตัวอย่างที่ 4.1

## แบบฝึกหัด

1. จงเขียนผังงานแสดงการคำนวณจำนวนคนสูงสุดที่ลิฟท์รับน้ำหนักได้ เมื่อให้ข้อมูลเข้าเป็นน้ำหนักที่ลิฟท์รับได้ และน้ำหนักเฉลี่ยของคน กำหนดให้น้ำหนักทั้งสองค่าเป็นจำนวนเต็ม
2. ตามตำราการดูดวงมักจะให้นำตัวเลขมาบวกลบคูณหารกัน ตำราหนึ่งกล่าวไว้ว่า ให้นำอายุคูณกับ 12 แล้วหารด้วย 7 เหลือเศษเท่าใดให้ดูคำทำนายของเศษนั้น ให้รับข้อมูลเข้าเป็นอายุ คำนวณหาเศษเหลือ แล้วแสดงผลเศษเหลือให้ผู้ใช้ทางจอภาพ
3. กำหนดให้ค่าคงที่ $\pi$ มีค่า 3.14 จงเขียนผังงานแสดงการคำนวณหาปริมาตรของทรงกลมซึ่งผู้ใช้ป้อนข้อมูลรัศมีเข้ามาให้
4. จงเขียนผังงานแสดงการคำนวณหาพื้นที่ของสามเหลี่ยมเมื่อผู้ใช้ป้อนข้อมูลความยาวของด้านทั้งสามเข้ามาให้
5. หากให้ผู้ใช้ป้อนข้อมูลเป็นขนาดกว้าง ยาว สูง ของกล่องทรงสี่เหลี่ยมมุมฉาก จงเขียนผังงานเพื่อแสดงการคำนวณหาขนาดกระดาษสี่เหลี่ยมมุมฉากที่เล็กที่สุดที่จะห่อกล่องนี้ได้ทุกด้านพอดีโดยไม่ต้องตัดกระดาษ และแสดงค่าขนาดกว้างยาวของกระดาษที่คำนวณได้
6. เขียนผังงานและโปรแกรมภาษาซีเพื่อรับข้อมูลเข้าเป็นจำนวนเต็ม 2 ค่าเก็บในตัวแปร a, b ตามลำดับ แล้วแสดงค่าที่เก็บอยู่ในตัวแปร a, b จากนั้นสลับค่าที่เก็บอยู่ในตัวแปร a ไปเก็บไว้ในตัวแปร b และให้ค่าที่อยู่ในตัวแปร b มาเก็บไว้ในตัวแปร a จากนั้นแสดงผลค่าที่อยู่ในตัวแปร a, b อีกครั้ง
7. มีสิทธิสมัครทุนรัฐบาลญี่ปุ่นระดับปริญญาตรีต้องมีเกรดเฉลี่ยสะสมระดับมัธยมปลายไม่ต่ำกว่า 3.8 แต่หากผู้สมัครมีความรู้ภาษาญี่ปุ่นจะได้รับสิทธิในการสมัครแม้เกรดเฉลี่ยจะไม่ถึง 3.8 โดยหากมีความรู้ภาษาญี่ปุ่นระดับ N1 หรือ N2 ต้องมีเกรดเฉลี่ยสะสมไม่ต่ำกว่า 3.3 หรือหากมีความรู้ภาษาญี่ปุ่นระดับ N3 หรือ N4 ต้องมีเกรดเฉลี่ยสะสมไม่ต่ำกว่า 3.5 จึงจะมีสิทธิ์สมัคร จงเขียนโปรแกรมภาษาซีเพื่อรับข้อมูลคุณสมบัติผู้สมัคร แล้วเขียนนิพจน์เพื่อตรวจสอบว่าผู้สมัครรายนี้มีสิทธิสมัครสอบหรือไม่ ให้ออกแบบการรับข้อมูลและตัวแปรเองทั้งหมด

8. วิชาหนึ่งมีคะแนนเต็ม 100 คะแนน แบ่งเป็นคะแนนเก็บ 20 คะแนน คะแนนสอบกลางภาค 40 คะแนน และคะแนนสอบปลายภาคอีก 40 คะแนน เงื่อนไขการผ่านวิชานี้คือต้องมีคะแนนรวมไม่น้อยกว่า 50 คะแนน โดยต้องได้คะแนนสอบกลางภาคไม่น้อยกว่าครึ่งหนึ่ง และคะแนนสอบปลายภาคไม่น้อยกว่าครึ่งหนึ่ง จงเขียนนิพจน์ที่ให้ค่าจริงเมื่อคะแนนของนิสิตเข้าเงื่อนไขการผ่านวิชานี้ ให้ออกแบบการรับข้อมูลและตัวแปรสำหรับเก็บข้อมูลต่าง ๆ เองทั้งหมด
9. เดือนกุมภาพันธ์ในปีอธิกสุรทินจะมี 29 วัน ปีอธิกสุริทินคือปีคริสต์ศักราชที่หารด้วย 4 ลงตัวแต่หารด้วย 100 ไม่ลงตัวหรือเป็นปีที่หารด้วย 400 ลงตัว หากให้ตัวแปร year เก็บค่าปีคริสต์ศักราชเป็นจำนวนเต็มบวก จงเขียนเงื่อนไขที่ให้ค่าจริงเมื่อ year เป็นปีอธิกสุรทิน และให้ค่าเท็จเมื่อ year เป็นปีทั่วไป และเขียนชุดข้อมูลทดสอบเพื่อตรวจสอบว่านิพจน์ที่เขียนถูกต้องหรือไม่

# บทที่ 5

# การทำงานแบบทางเลือก

## วัตถุประสงค์การเรียนรู้

อธิบายการทำงานของโปรแกรมในรูปการทำงานแบบทางเลือก

1. แก้ปัญหาที่ต้องใช้การทำงานแบบทางเลือกได้ เขียนอธิบายขั้นตอนด้วยผังงานหรือรหัสเทียมได้
2. อ่านส่วนของโปรแกรมภาษาซีเพื่อทำงานแบบทางเลือกเข้าใจ และสามารถบอกผลของการทำงานได้ถูกต้อง
3. เขียนโปรแกรมภาษาซีเพื่อแก้ปัญหาที่ต้องใช้การทำงานแบบทางเลือกได้

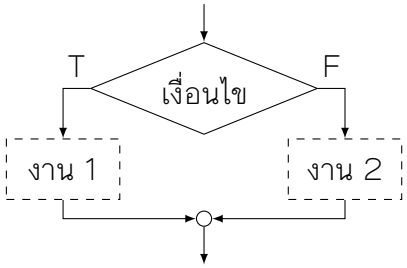
ใช้เวลารวม 6 คาบ

## 5.1 โครงสร้างการทำงานแบบเงื่อนไขสองทางเลือก

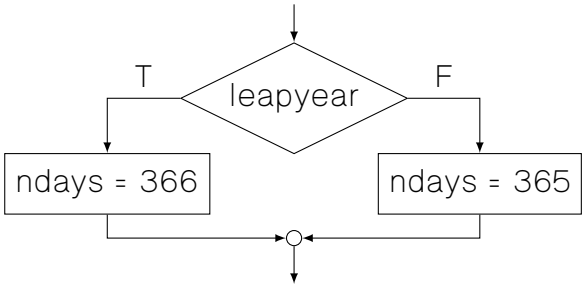
การทำงานแบบเงื่อนไขสองทางเลือก คือการทำงานแบบมีทางเลือกสองทาง และใช้เงื่อนไขซึ่งมีค่าที่เป็นไปได้สองรูปแบบในการเลือกการทำงาน เช่น

- หากได้คะแนนอย่างน้อยร้อยละ 90 จะได้เกรด A และจะไม่ได้เกรด A หากได้คะแนนน้อยกว่านี้ ทางเลือกรูปแบบนี้คือการเลือกกำหนดค่าผลลัพธ์โดยขึ้นกับเงื่อนไขคะแนนที่ได้
- ในระบบสูบน้ำเข้าถังเก็บน้ำ ปั๊มจะหยุดสูบน้ำเข้าโดยอัตโนมัติหากระดับน้ำในถึงถึงขีดที่กำหนด ทางเลือกรูปแบบนี้คือเลือกที่จะทำงาน (สูบน้ำ) หรือไม่ทำงาน ขึ้นกับเงื่อนไขค่าจากตัววัดระดับน้ำ

โครงสร้างการทำงานแบบเงื่อนไขจะประกอบด้วยเงื่อนไข และทางเลือกในการทำงาน โครงสร้างการทำงานแบบเงื่อนไขอย่างง่ายจะประกอบด้วยเงื่อนไข 1 เงื่อนไข ซึ่งเป็นนิพจน์ตรรกศาสตร์ มีค่าที่เป็นไปได้ 2 ค่า คือ จริง หรือ เท็จ และทางเลือกในการทำงาน 2 ทาง ทางแรกทำเมื่อเงื่อนไขเป็นจริง และทางที่สองทำเมื่อเงื่อนไขเป็นเท็จ เขียนเป็นผังงานได้ดังรูปที่ 5.1




รูปที่ 5.1: โครงสร้างการทำงานแบบเงื่อนไขสองทางเลือก




รูปที่ 5.2: ส่วนของผังงานสำหรับตัวอย่างที่ 5.1

โดย งาน 1 คืองานที่ทำเมื่อเงื่อนไขเป็นจริง และ งาน 2 คืองานที่ทำเมื่อเงื่อนไขเป็นเท็จ งาน 1 หรือ งาน 2 อาจเป็นโครงสร้างการทำงานแบบลำดับ ซึ่งประกอบด้วยการทำงานหลายอย่างเรียงกัน หรืออาจจะไม่มีการทำงานเลยก็ได้

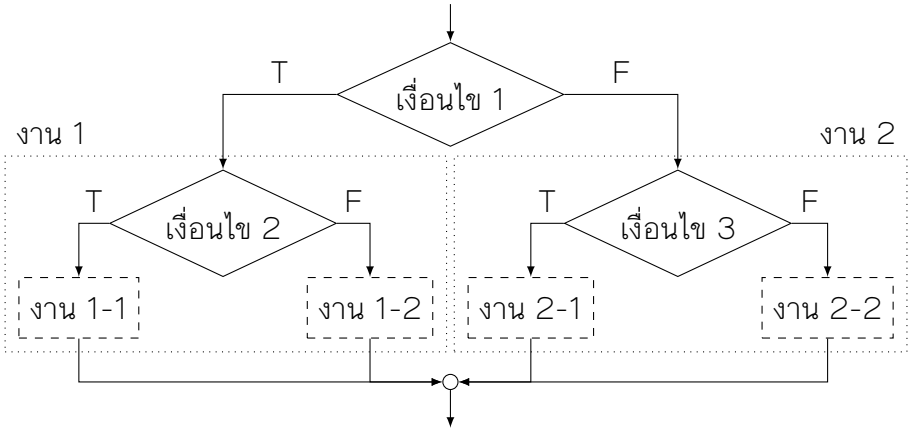
**ตัวอย่าง 5.1** (การหาจำนวนวันในหนึ่งปี). ให้ตัวแปร leapyear เป็นตัวแปรเก็บค่าตรรกะ ซึ่งมีค่าเป็นจริงเมื่อปีนั้นเป็นปีอธิกสุรทิน และมีค่าเป็นเท็จในกรณีอื่น กำหนดให้ตัวแปร ndays เป็นตัวแปรสำหรับเก็บค่าจำนวนวันในหนึ่งปี ปีนั้นจะมี 366 วันหากเป็นปีอธิกสุรทิน และมี 365 วันหากเป็นปีปกติ

จึงเขียนผังงานเพื่อกำหนดจำนวนวันในหนึ่งปี โดยขึ้นกับว่าปีนั้นเป็นปีอธิกสุรทินหรือไม่ ได้ผลลัพธ์เป็นดังรูปที่ 5.2 มีเงื่อนไขคือค่าในตัวแปร leapyear และงานที่ต้องทำเป็นการกำหนดจำนวนวันให้ตัวแปร ndays

□

## ลองคิด

ผลลัพธ์ของเงื่อนไขในตัวอย่าง 5.1 เป็นค่าตรรกะจริงหรือเท็จ เราสามารถใช้การทำงานแบบสองทางเลือกกับผลลัพธ์รูปแบบอื่นที่ไม่ใช่ค่าจริงหรือเท็จได้หรือไม่ ให้เหตุผลประกอบ




รูปที่ 5.3: โครงสร้างการทำงานแบบเงื่อนไขซ้อนเงื่อนไข

## 5.2 เงื่อนไขซ้อนและการทำงานแบบหลายทางเลือก

จากรูปที่ 5.1 งานที่ทำเมื่อเงื่อนไขเป็นจริง (งาน 1) หรืองานที่ทำเมื่อเงื่อนไขเป็นเท็จ (งาน 2) นอกจากจะเป็นคำสั่งพื้นฐานหรือโครงสร้างการทำงานแบบลำดับได้แล้ว ยังสามารถเป็นโครงสร้างการทำงานแบบเงื่อนไขได้เช่นกัน เมื่อซ้อนเงื่อนไขเพิ่มเข้าไป จะทำให้มีทางเลือกที่เป็นไปได้มากขึ้นอีก

### 5.2.1 เงื่อนไขซ้อน

การซ้อนเงื่อนไขทำได้ในทุกทางเลือก กรณีที่เงื่อนไขหลักเป็นเงื่อนไขแบบสองทางเลือกค่าจริง/เท็จ การซ้อนเงื่อนไขก็ทำได้ทั้งในฝั่งที่เงื่อนไขหลักเป็นจริงและฝั่งที่เงื่อนไขหลักเป็นเท็จ ดังรูปที่ 5.3

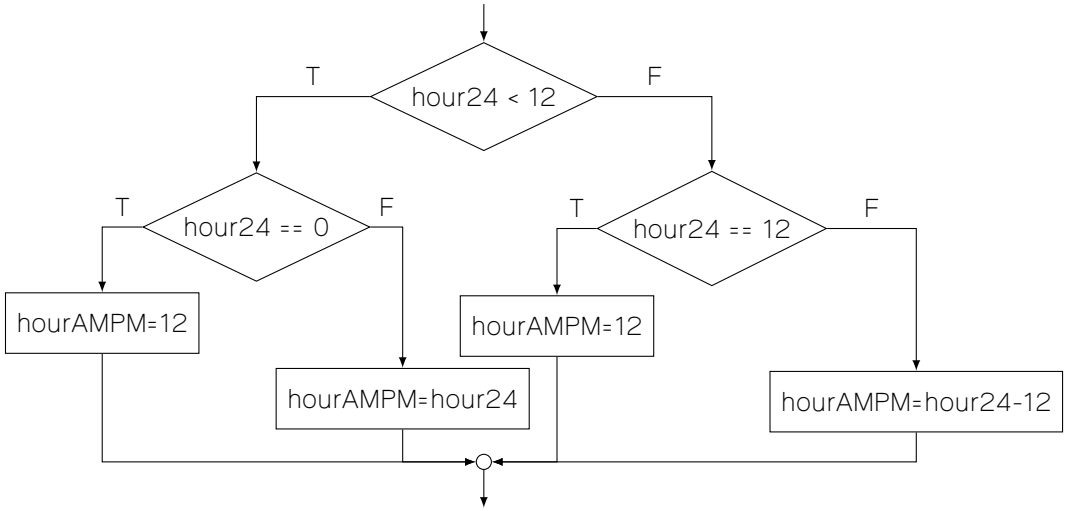
ในกรอบงาน 1 เป็นโครงสร้างแบบเงื่อนไข ซึ่งจะทำงาน 1-1 เมื่อเงื่อนไข 2 เป็นจริง และทำงาน 1-2 เมื่อเงื่อนไข 2 เป็นเท็จ ส่วนกรอบงาน 2 ก็เป็นโครงสร้างแบบเงื่อนไข ประกอบด้วยเงื่อนไข 3 งาน 2-1 และงาน 2-2 ในทำนองเดียวกัน

**ตัวอย่าง 5.2** (การแปลงเวลาจากระบบ 24 ชั่วโมงไปเป็นระบบ AM/PM). เวลาในระบบ 24 ชั่วโมงจะเริ่มต้นที่ 0 นาฬิกา ไปจนถึง 23 นาฬิกา แต่เวลาในระบบ AM/PM จะแบ่งครึ่งเช้าเป็น AM และครึ่งบ่ายเป็น PM โดยที่ 0 นาฬิกาจะเป็น 12AM และ 12 นาฬิกาจะเป็น 12PM เวลาอื่น ๆ เป็น 1-11 ตามลำดับ

การแปลงเวลาในระบบ 24 ชั่วโมงไปเป็นระบบ AM/PM อาจทำได้โดยแบ่งครึ่งเช้าและบ่ายก่อน (เงื่อนไข 1) จากนั้นจึงแยกแสดงผล 0 นาฬิกา กับ 1-11 นาฬิกา (เงื่อนไข 2) และครึ่งบ่ายก็แยกแสดงผล 12 นาฬิกา กับ 13-23 นาฬิกา (เงื่อนไข 3) เช่นเดียวกัน

ให้ hour24 เป็นตัวแปรเก็บจำนวนเต็มเลขชั่วโมงในระบบ 24 ชั่วโมง และ hourAMPM เป็นผลลัพธ์ เก็บจำนวนเต็มในระบบ AM/PM จะเขียนส่วนของผังงานเพื่อกำหนดค่าให้ hourAMPM ได้ดังรูปที่ 5.4

□




รูปที่ 5.4: ผังงานสำหรับตัวอย่างที่ 5.2

### 5.2.2 การทำงานแบบหลายทางเลือก

การใช้เงื่อนไขที่ให้ค่าจริงหรือเท็จนั้น มีทางเลือกที่เป็นไปได้เพียงสองทาง แต่ในบางกรณี เราอาจต้องการเลือกทำงานตามเงื่อนไขที่กำหนดเป็นชุดของค่าคงที่ต่าง ๆ เช่น

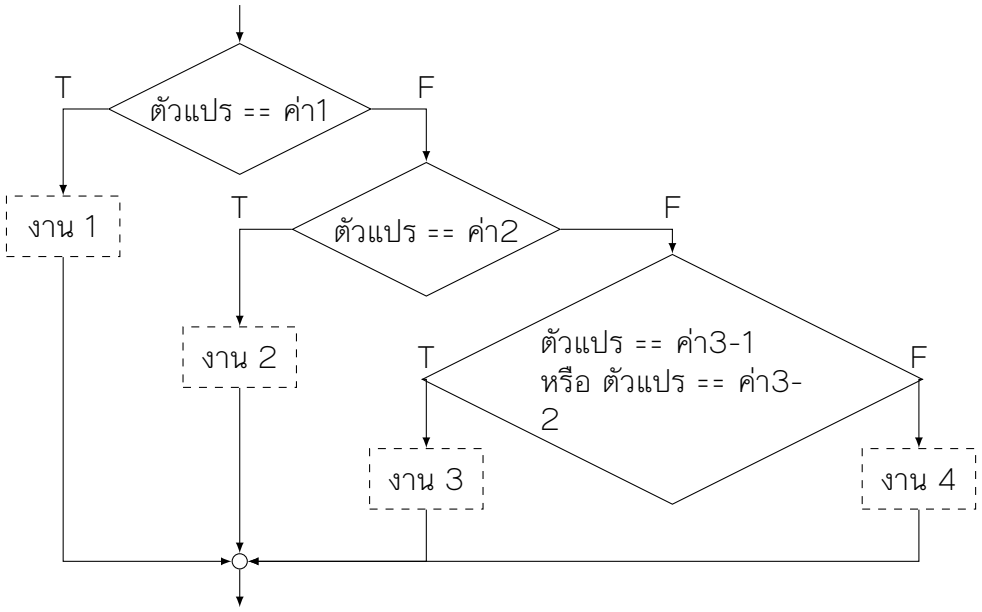
- ค่า 1 กำหนดทางเลือกที่ 1
- ค่า 2 กำหนดทางเลือกที่ 2
- ค่า 3-1 และ 3-2 กำหนดทางเลือกที่ 3
- ค่าอื่นๆ เป็นทางเลือกที่ 4

วิธีหนึ่งที่ทำได้คือเขียนนิพจน์เพื่อตรวจสอบค่านี้ทีละค่า หากเป็นค่าใด (นิพจน์ให้ค่าจริง) ก็ไปตามทางเลือกนั้น ดังรูปที่ 5.5 และรูปที่ 5.6

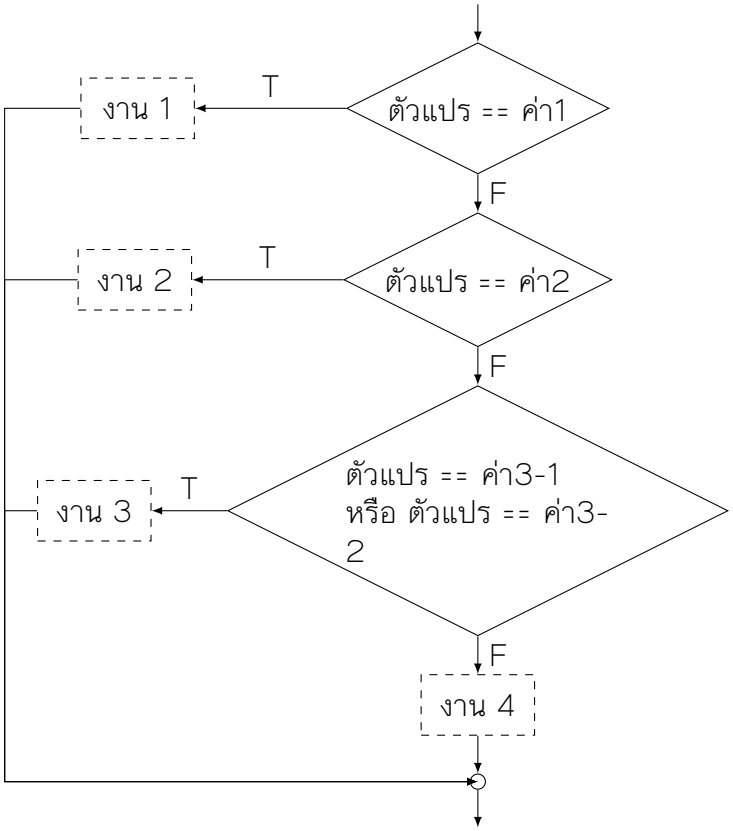
การทำงานแบบหลายเงื่อนไขดังรูปที่ 5.5 และรูปที่ 5.6 นั้นก็เป็นการทำงานแบบเงื่อนไขซ้อนเช่นเดียวกัน โดยรูปที่ 5.6 เป็นการจัดรูปเพื่อให้เข้าใจง่ายว่าต้องการตรวจสอบเงื่อนไขทางเลือกทีละเงื่อนไข หากเป็นค่าตามเงื่อนไขในทางเลือกใด ก็ให้ไปทำงานตามทางเลือกนั้น

การทำงานแบบตรวจสอบค่าตามเงื่อนไขไปทีละค่านี้พบเป็นประจำ แทนที่จะต้องเขียนให้อยู่ในรูปของนิพจน์ตรรกศาสตร์ทีละเงื่อนไข เราสามารถเขียนผังงานโดยระบุให้ค่าที่เข้ามาเป็นตัวเลือกเส้นทาง แล้วกำกับเส้นทางด้วยค่ากำหนดเส้นทางได้เลย หากทางเลือกใดมีค่ากำหนดเส้นทางที่มากกว่าหนึ่งค่า ให้ใช้จุลภาค (,) คั่นระหว่างค่าเหล่านั้น และหากค่าที่เป็นไปได้เป็นลำดับค่าที่แน่นอน เช่น 1-10 ก็ใช้ 1...10 แทนได้

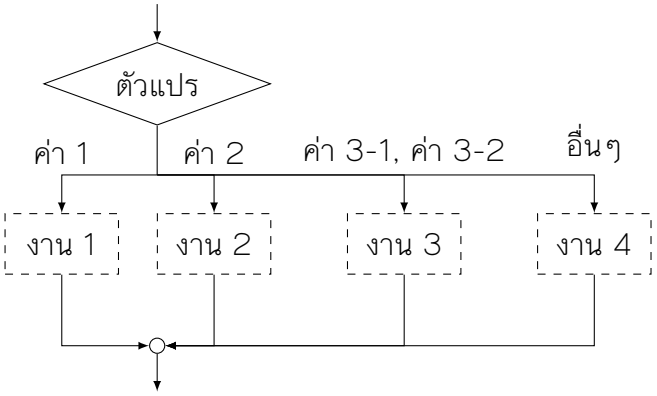
ส่วนของผังงานในรูปที่ 5.5 และรูปที่ 5.6 จึงสามารถเขียนใหม่ได้ดังรูปที่ 5.7



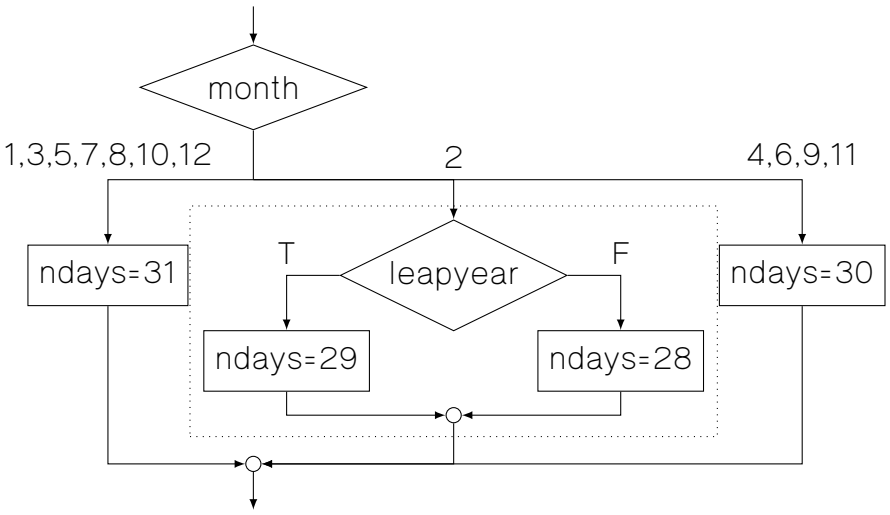

รูปที่ 5.5: โครงสร้างการทำงานแบบเงื่อนไขต่อเนื่องเพื่อตรวจสอบค่า (1)




รูปที่ 5.6: โครงสร้างการทำงานแบบเงื่อนไขต่อเนื่องเพื่อตรวจสอบค่า (2)




รูปที่ 5.7: โครงสร้างการทำงานแบบเงื่อนไขการตรวจสอบทางเลือก




รูปที่ 5.8: ผังงานแสดงการหาจำนวนวันในหนึ่งเดือน

**ตัวอย่าง 5.3** (การหาจำนวนวันในหนึ่งเดือนของปี). หากให้ month เป็นตัวแปรเก็บค่าจำนวนเต็ม มีค่าได้ตั้งแต่ 1-12 แทนเลขเดือน และ leapyear เป็นตัวแปรตรรกะมีค่าจริงก็ต่อเมื่อปีนั้นเป็นปีอธิกสุรทิน จำนวนวันในเดือนจะขึ้นอยู่กับค่าในตัวแปร month นี้ ดังนี้

- month มีค่าเป็น 1,3,5,7,8,10,12 จะมีจำนวนวันเป็น 31 วัน
- month มีค่าเป็น 2 จำนวนวันเป็น 29 หรือ 28 ขึ้นกับว่าปีนั้นเป็นปีอธิกสุรทินหรือไม่
- month มีค่าเป็น 4,6,9,11 จะมีจำนวนวันเป็น 30 วัน

นำมาเขียนเป็นส่วนของผังงานได้ดังรูปที่ 5.8

จะเห็นได้ว่า งานที่ต้องทำในทางเลือกนั้นอาจจะเป็นการทำงานพื้นฐาน หรือการทำงานรูปแบบอื่นก็ได้ ดังเช่นทางเลือกเมื่อค่า month เป็น 2 งานที่ต้องทำเป็นการทำงานแบบเงื่อนไข เป็นต้น □

**ลองคิด**

ผังงานในรูปที่ 5.3 นั้น สามารถแปลงมาเขียนเป็นผังงานแบบรูปที่ 5.5 หรือ 5.6 ได้หรือไม่ หากได้ให้ลองเขียนดู หากไม่ได้ ลองให้เหตุผลประกอบ

## 5.3 โครงสร้าง if ในภาษาซี

โครงสร้าง `if` ในภาษาซีประกอบด้วย เงื่อนไข และสิ่งที่ต้องทำเมื่อเงื่อนไขเป็นจริง โดยอาจมีสิ่งที่ต้องทำเมื่อเงื่อนไขเป็นเท็จด้วยก็ได้

ส่วนของผังงานในรูปที่ 5.1 เขียนเป็นคำสั่งภาษาซีได้รหัสคำสั่งที่ 5.1 และ 5.2

```
1 if (cond)
2     job1
3 else
4     job2
```

รหัสคำสั่งที่ 5.1: โครงสร้างคำสั่ง `if` (1)

```
1 if (cond) job1
2 else job2
```

รหัสคำสั่งที่ 5.2: โครงสร้างคำสั่ง `if` (2)

โดย `job1` และ `job2` เทียบได้กับ งาน 1 และ งาน 2 ตามลำดับ ในภาษาซีนั้น การใช้ช่องว่าง ไม่ว่าจะเป็นเว้นวรรค ย่อหน้า หรือขึ้นบรรทัดใหม่ ไม่ว่าจะเป็นกี่ตัวก็ตามในรหัสต้นฉบับ จะมีผลเท่ากัน เรานิยมย่อหน้าจากคำสั่งหลักเพิ่มมาหนึ่งย่อหน้า เพื่อเน้นให้เห็นชัดว่าเป็นทางเลือกย่อยที่แตกออกมาจากลำดับการทำงานหลัก และทำให้อ่านง่าย

จากตัวอย่างข้างต้น คำสั่งภาษาซีทั้งในรหัสคำสั่งที่ 5.1 และ 5.2 ให้ผลเหมือนกันทุกประการ แบบ 5.2 นั้นนิยมใช้หากต้องการประหยัดจำนวนบรรทัดในการพิมพ์ เมื่องานที่ต้องทำเป็นคำสั่งเดียวและไม่ยาวมาก จึงอาจเขียนงานที่ต้องทำหลังเงื่อนไขไว้ในบรรทัดเดียวกัน

โครงสร้าง `if` ก็นับเป็นหนึ่งคำสั่งเช่นเดียวกัน ดังนั้น ส่วนของผังงานในรูปที่ 5.3 จึงสามารถเขียนเป็นส่วนของโปรแกรมภาษาซีได้ดังรหัสคำสั่งที่ 5.3

```
1 if (cond1)
2     if (cond2) job1_1
3     else job1_2
4 else
5     if (cond3) job2_1
6     else job2-2
```

รหัสคำสั่งที่ 5.3: โครงสร้างคำสั่ง `if` แบบเงื่อนไขซ้อน

ในกรณีที่เป็นการทดสอบเงื่อนไขต่อเนื่อง เช่นรูปที่ 5.5 นั้น อาจจัดรูปแบบคำสั่งภาษาซีได้ดังรหัสคำสั่งที่ 5.4 และ 5.5

```
 1 if (cond1)
 2 {
 3     job1
 4 }
 5 else
 6 {
 7     if (cond2)
 8     {
 9         job2
10     }
11     else
12     {
13         if (cond3)
14         {
15             job3
16         }
17         else
18         {
19             job4
20         }
21     }
22 }
```

รหัสคำสั่งที่ 5.4: โครงสร้าง `if` แบบเงื่อนไขต่อเนื่อง (1)

```
1 if (cond1)
2     job1
3 else if (cond2)
4     job2
5 else if (cond3)
6     job3
7 else
8     job4
```

รหัสคำสั่งที่ 5.5: โครงสร้าง `if` แบบเงื่อนไขต่อเนื่อง (2)

รหัสคำสั่งที่ 5.4 เป็นการจัดรูปและใช้โครงสร้างบล็อกเพื่อให้เห็นขอบเขตของคำสั่ง `if` ในแต่ละระดับ สามารถจัดใหม่ให้ประหยัดเนื้อที่ได้ตาม 5.5 ซึ่งตรงกับส่วนของผังงานในรูปที่ 5.6 โดยสามารถอ่านได้ทีละบรรทัดว่าเป็นการตรวจสอบค่าทีละค่าไล่ไปตามลำดับ

งานที่ต้องทำนั้นอาจเป็นคำสั่งเดียวหรือหลายคำสั่งก็ได้ หากเป็นกรณีที่มีหลายคำสั่ง ในภาษาซีต้องรวมคำสั่งทั้งหมดให้เป็นโครงสร้างบล็อกดังเช่นรหัสคำสั่งที่ 5.6 และ 5.7

การเขียนคำสั่งในบล็อกนั้น นิยมย่อหน้าคำสั่งภายในบล็อกเพิ่มหนึ่งย่อหน้าเช่นเดียวกับเมื่อเขียนคำสั่ง `if` เพื่อให้เห็นว่าคำสั่งทั้งหมดอยู่ในบล็อกเดียวกัน รูปแบบการเขียนวงเล็บปีกกาครอบบล็อกกับคำสั่ง `if` ที่นิยมกันมีสองรูปแบบ แบบ 5.6 เน้นให้เห็นว่าวงเล็บปีกกาเปิดและวงเล็บปีกกาปิดที่อยู่คู่กันนั้นจะอยู่ตรงกัน แต่ข้อเสียคือจะสิ้นเปลืองจำนวนบรรทัดมาก และหากรหัสต้นฉบับยาวมากจนปีกกาเปิดและปิดไม่อยู่ในหน้าเดียวกันก็จะทำให้จับคู่ได้ลำบากอยู่ดี แบบ 5.7 จะเปิดวงเล็บปีกกาข้างหลังเงื่อนไข และ

ให้ปิดวงเล็บตรงกับคำสั่งเงื่อนไขแทน ซึ่งยังคงเพิ่มย่อหน้าให้คำสั่งในบล็อกเช่นกัน

```
1  if (cond1)
2  {
3      st1_1;
4      st1_2;
5  }
6  else
7  {
8      st2_1;
9      st2_2;
10 }
```

รหัสคำสั่งที่ 5.6: โครงสร้าง `if` แบบใช้บล็อก (1)

```
1  if (cond1) {
2      st1_1;
3      st1_2;
4  } else {
5      st2_1;
6      st2_2;
7  }
```

รหัสคำสั่งที่ 5.7: โครงสร้าง `if` แบบใช้บล็อก (2)

ถึงแม้ว่าการใช้บล็อกจะจำเป็นเฉพาะในกรณีที่มีงานที่ต้องทำมากกว่าหนึ่งคำสั่ง กรณีที่มีคำสั่งเดียวแต่เป็นคำสั่งที่ซับซ้อน เช่นคำสั่ง `if` ในรหัสคำสั่งที่ 5.3 การใช้บล็อกช่วยจะทำให้รหัสต้นฉบับอ่านง่าย สะดวกในการปรับปรุงแก้ไขในอนาคต จึงควรใช้โครงสร้างบล็อกในกรณีเหล่านี้ด้วยเช่นกัน

**ลองคิด**

โครงสร้าง `if` ในภาษาซีนั้น บังคับว่าต้องมีสิ่งที่ทำเมื่อเงื่อนไขเป็นจริง ในขณะที่ผังงานไม่มีข้อบังคับนี้ หากต้องเขียนเขียนคำสั่งภาษาซี โดยมีสิ่งที่ต้องทำเมื่อเงื่อนไขเป็นเท็จเท่านั้น ต้องทำอย่างไร

## 5.4 โครงสร้าง switch ในภาษาซี

ในภาษาซี การทำงานแบบทางเลือกโดยการตรวจสอบค่าในตัวแปรใช้คำสั่ง `switch` โครงสร้างของคำสั่ง `switch` ประกอบด้วยตัวแปรซึ่งเก็บค่าทางเลือก และทางเลือกต่าง ๆ

โครงสร้างในรูปที่ 5.7 เขียนเป็นคำสั่ง `switch` ได้ดังรหัสคำสั่งที่ 5.8

`var` เป็นตัวแปรเก็บค่าทางเลือก ในรหัสคำสั่งที่ 5.8 มีทางเลือกทั้งหมด 4 ทาง ได้แก่

- เมื่อ `var` มีค่าเป็น `val1` ทำ`job1`
- เมื่อ `var` มีค่าเป็น `val2` ทำ `job2`
- เมื่อ `var` มีค่าเป็น `val3-1` หรือ `val3-2` ทำ `job3`
- เมื่อ `var` มีค่าอื่น ๆ ทำ`job4`

```
 1 switch (var)
 2 {
 3     case val1:
 4         job1
 5         break;
 6     case val2:
 7         job2
 8         break;
 9     case val3-1
10     case val3-2:
11         job3
12         break;
13     default:
14         job4
15 }
```

รหัสคำสั่งที่ 5.8: โครงสร้างคำสั่ง switch

การระบุค่าประจำทางเลือกจะเขียนไว้หลังตัวระบุ case แล้วตามด้วยทวิภาค (:) เครื่องจะเปรียบเทียบค่าในแต่ละทางเลือกทีละทางเลือก เรียงลำดับจากบรรทัดบนลงล่าง หากค่าในตัวแปรตรงกับค่าในทางเลือก จะทำทุกคำสั่งที่อยู่หลังทวิภาค จนกว่าจะหมดหรือพบคำสั่ง break ตัวระบุ default ใช้สำหรับกรณีที่ค่าในตัวแปรไม่ตรงกับทางเลือกใดเลย ซึ่งอาจจะมีหรือไม่มีตัวระบุนี้ก็ได้ จะเห็นว่า สำหรับทางเลือก default นั้น ไม่จำเป็นต้องมีคำสั่ง break เนื่องจากเมื่อจบการทำงานในทางเลือกนี้ ก็จะสิ้นสุดการทำงานของคำสั่ง switch อยู่แล้ว

แนวทางในการการย่อหน้าเพิ่มสำหรับคำสั่งในทางเลือกนั้นเหมือนกับในคำสั่ง if และหากมีคำสั่งเดียวที่ไม่ซับซ้อน บางครั้งก็นิยมเขียนทั้งคำสั่งที่ต้องทำและคำสั่ง break รวมไว้ในบรรทัดเดียวกับตัวระบุทางเลือกด้วย

งานที่ต้องทำในแต่ละทางเลือกนั้นมีได้มากกว่า 1 คำสั่ง และไม่จำเป็นต้องใช้โครงสร้างบล็อก เพราะใช้คำสั่งเบรกในการตัดการทำงานแทนอยู่แล้ว

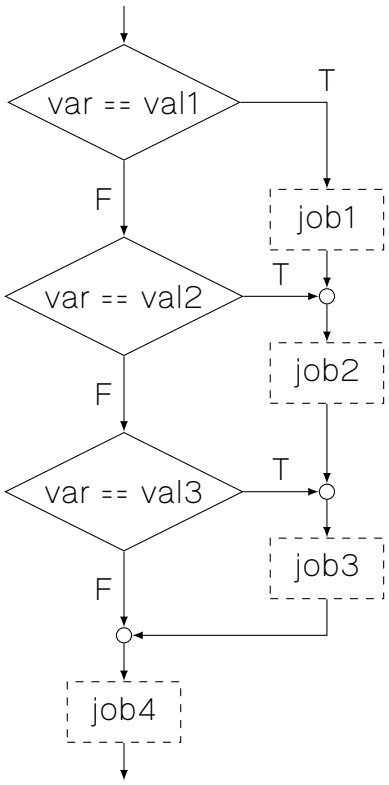
ส่วนของผังงานในรูปที่ 5.7 นั้น เส้นทางเลือกต่างๆ สามารถเขียนสลับตำแหน่งกันได้ และดูเหมือนว่าเครื่องจะเลือกเส้นทางที่ถูกต้องได้เสมอ แต่ในการทำงานจริงนั้นเครื่องทำงานเป็นลำดับเสมอ เมื่อมีทางเลือกหลายทาง เครื่องจะทยอยเปรียบเทียบไปทีละทางตามลำดับ แม้ในผังงานอาจจะดูเหมือนไม่มีลำดับ แต่หากเขียนเป็นคำสั่งในภาษาโปรแกรมแล้ว การเปรียบเทียบจะเป็นไปตามลำดับเสมอ

เมื่อเปลี่ยนคำสั่ง switch ในรหัสคำสั่งที่ 5.8 ให้เป็นผังงานที่แสดงขั้นตอนการทำงานจริง จะได้ดังรูปที่ 5.5 นั่นคือเปรียบเทียบค่าในตัวแปรทีละค่า เรียงจากทางเลือกแรกไปจนกว่าจะได้ทางเลือกที่ต้องการ

จากโครงสร้าง `switch` ที่ผ่านมานั้น จำเป็นต้องใช้คำสั่ง `break` หลังจบการทำงานตามทางเลือกเสมอ หากไม่มีคำสั่ง `break` เลย เช่นในรหัสคำสั่งที่ 5.9 การทำงานโดยคำสั่ง `switch` จะเป็นดังรูปที่ 5.9

```
1 switch (var)
2 {
3     case val1 : job1
4     case val1 : job2
5     case val1 : job3
6     default : job4
7 }
```

รหัสคำสั่งที่ 5.9: โครงสร้าง `switch` แบบไม่มี `break`




รูปที่ 5.9: ผังงานแสดงลำดับการทำงานของคำสั่ง `switch` แบบไม่มี `break`

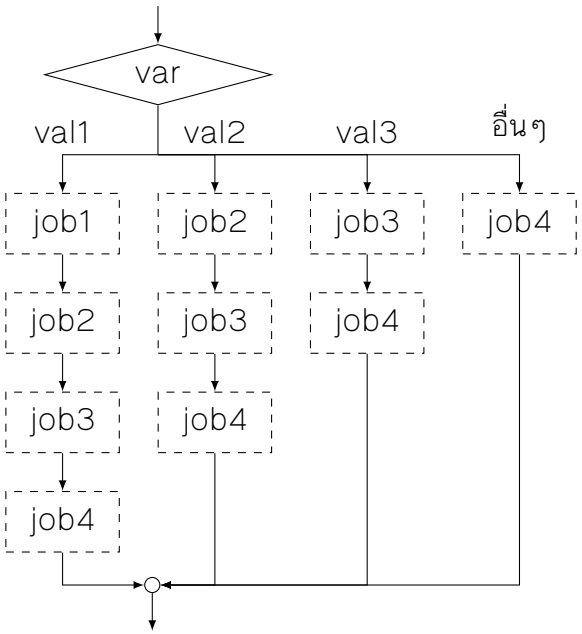
อย่างไรก็ตาม โครงสร้างลักษณะเช่นนี้เป็นโครงสร้างที่อาจทำให้สับสนได้ง่าย ควรใช้โครงสร้างแบบทางเลือกหลายทางดังรูปที่ 5.7 มากกว่า ซึ่งจะเขียนเป็นส่วนของผังงานใหม่ได้ดังรูปที่ 5.10

### ลองคิด

โครงสร้าง `if` และโครงสร้าง `switch` ในภาษาซี สามารถเขียนแทนกันได้เสมอหรือไม่ เพราะเหตุใด

## แบบฝึกหัด

1. เมื่อรับข้อมูลเข้าเป็นวันแรกของเดือนและเลขเดือน เดือนนั้นจะมีวันจันทร์ อังคาร พุธ พฤหัสบดี ศุกร์ เสาร์ และอาทิตย์ อย่างละกี่วัน จงเขียนผังงานเพื่อแสดงวิธีการคำนวณ
   กำหนดให้
   - 0 แทนวันอาทิตย์




รูปที่ 5.10: ผังงานของคำสั่ง `switch` แบบไม่มี `break`

- 1 แทนวันจันทร์
- 2 แทนวันอังคาร
- ...

2. ให้ 1 2 3 4 5 6 7 เป็นรหัสตัวเลขแทนวันจันทร์ถึงวันอาทิตย์ ถึง ตามลำดับ จงเขียนผังงานและโปรแกรมภาษาซีเพื่อแปลงรหัสตัวเลขนี้ให้เป็นชื่อวันแล้วแสดงผล เช่น Sun Mon
3. ให้ 1 - 12 เป็นตัวเลขของเดือน จงแปลงตัวเลขนี้ให้เป็นชื่อเดือน เช่น Jan Feb
4. ในบางประเทศ การอ้างอิงถึงเวลาช่วง 0.00 - 2.00 น. จะใช้ตัวเลข24.00 - 26.00 แทน เพื่อให้เข้าใจตรงกันว่าเป็นช่วงกลางคืนที่ต่อเนื่องจากวันก่อนหน้า ให้รับข้อมูลเข้าเป็นวัน (เลข 1 - 7 แทนวันจันทร์ถึงวันอาทิตย์) วันที่ เดือน และเวลา (0:00 - 23:59) แล้วแสดงผลในรูป วัน วันที่ เดือน และ เวลา (0:00 - 26:00)
5. หากต้องการทำแฟ้มภาพนิสิตโดยวางรูปถ่ายขนาด 1 x 1.5 นิ้ว ในกระดาษ A4 (8.3 x 11.7 นิ้ว) ให้ได้มากที่สุดเท่าที่จะมากได้ โดยรูปทุกรูปต้องวางตั้งทั้งหมด ควรวางกระดาษ A4 ในแนวนอนหรือแนวตั้ง จึงจะวางรูปได้มากที่สุด จงเขียนผังงานแสดงการคำนวณ
6. จงเขียนผังงานสำหรับโปรแกรมเพื่อการคำนวณปริมาตรหรือพื้นที่ผิวของรูปทรงต่างๆ เมื่อโปรแกรมเริ่มทำงานจะให้ผู้ใช้เลือกหมวดการคำนวณ ว่าจะคำนวณปริมาตร (กด 1) หรือพื้นที่รอบรูป (กด 2) จากนั้นให้ผู้ใช้เลือกรหัสรูปทรงที่ต้องการ และป้อนค่าที่จำเป็นในการคำนวณให้โปรแกรม จากนั้นโปรแกรมจะแสดงผลปริมาตรหรือพื้นที่รอบรูปให้แล้วแต่กรณี กำหนดค่า Pi = 3.14
7. จงเขียนผังงานเพื่อสร้างเครื่องคิดเลขที่ทำการบวก ลบ คูณ หาร ยกกำลัง จำนวนไม่เป็นลบสอง

จำนวนได้ สมมติให้ข้อมูลเข้าอยู่ในรูปแบบ “ตัวเลข” “เครื่องหมาย” “ตัวเลข” เท่านั้น เช่น 2 ^ 2 หรือ 2 / 4

8. จงเขียนผังงานสำหรับโปรแกรมเพื่อการคำนวณปริมาตรหรือพื้นที่ผิวของรูปทรงต่างๆ เมื่อโปรแกรมเริ่มทำงานจะให้ผู้ใช้เลือกหมวดการคำนวณ ว่าจะคำนวณปริมาตร (กด 1) หรือพื้นที่รอบรูป (กด 2) จากนั้นให้ผู้ใช้เลือกรหัสรูปทรงที่ต้องการ และป้อนค่าที่จำเป็นในการคำนวณให้โปรแกรม จากนั้นโปรแกรมจะแสดงผลปริมาตรหรือพื้นที่รอบรูปให้แล้วแต่กรณี
กำหนดค่า $\pi$ = 3.14 และค่าต่างๆ ที่จำเป็นดังนี้

| | **ทรงกลม** | **กรวย** | **ทรงกระบอก** |
|---|---|---|---|
| **รหัสรูปทรง** | A | B | C |
| **ค่าที่จำเป็นในการคำนวณ** | 1. รัศมี R | 1. รัศมี R | 1. รัศมี R |
| | | 2. ความสูง h | 2. ความสูง h |

9. เขียนผังงานเพื่อรับข้อมูลเข้าเป็นเกรดของนิสิต ได้แก่ A, B+, B, C+, C, D+, D, F แล้วแสดงผลเป็นค่าประจำเกรดนั้น ตั้งแต่ 4.0 ถึง 0.0
10. Zeller's congruence เป็นอัลกอริทึมเพื่อหาวันในสัปดาห์ของวันที่ใดๆ ในปฏิทิน เมื่อกำหนดให้

    - $h$ เป็นวันในสัปดาห์ 0 = เสาร์, 1 = อาทิตย์, ...
    - $q$ เป็นวันที่
    - $m$ เป็นเดือน โดย 13 = มกราคมของปีก่อนหน้า, 14 = กุมภาพันธ์ของปีก่อนหน้า, 3 = มีนาคม, 4 = เมษายน, ... , 12 = ธันวาคม
    - $K$ คือปีในคริศต์ศตวรรษ (สองตัวหลังของปีคริสต์ศักราช)
    - $J$ คือเลขศตวรรษ (สองตัวหน้าของปีคริสต์ศักราช)

    เช่น

    - 1 มกราคม 2016 จะได้ $q$=1, $m$=13, $K$=15, $j$=20

    ในระบบปฏิทินเกรกอเรียน (ปฏิทินปัจจุบัน) จะคำนวณวันในสัปดาห์ได้ดังสมการ (5.1)

$$h = (q + \lfloor(13(m+1))/5\rfloor + K + \lfloor K/4\rfloor + \lfloor J/4\rfloor + 5J) \mod 7 \qquad (5.1)$$

    จงเขียนผังงานและโปรแกรมภาษาซีเพื่อแสดงผลเป็น Saturday, Sunday, ... เมื่อวันในสัปดาห์เป็น เสาร์, อาทิตย์, ... ตามลำดับ กำหนดให้ข้อมูลเข้าเป็น

    - วันที่ (ตัวเลข)
    - เดือน (ตัวเลข 1-12) และ
    - ปีคริสต์ศักราช (ตัวเลข 4 หลัก)

11. มีกล่องทรงสี่เหลี่ยมมุมฉากขนาดกว้าง a ยาว b และสูง c ต้องการห่อกล่องนี้ด้วยกระดาษสี่เหลี่ยมมุมฉาก จงเขียนผังงานเพื่อตัดสินใจว่าจะต้องใช้กระดาษขนาดเล็กที่สุดเท่าไรจึงจะห่อกล่องนี้ได้พอดีโดยไม่ต้องตัด

# บทที่ 6

# การทำงานแบบวนซ้ำ

## วัตถุประสงค์การเรียนรู้

อธิบายการทำงานของโปรแกรมในรูปการทำงานแบบวนซ้ำ

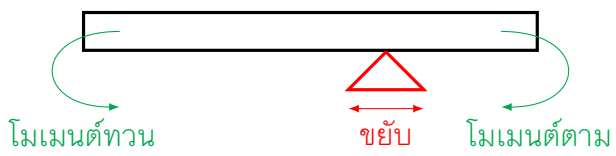
1. บอกผลของการทำงานแบบวนซ้ำได้ และประยุกต์ใช้การทำงานแบบวนซ้ำเพื่อแก้ปัญหาต่างๆ ที่พบได้
2. อธิบายลำดับการทำงานเมื่อใช้โครงสร้าง `while-do`, `do-while` และ `for` ในภาษาซีได้
3. ใช้โครงสร้าง `while-do`, `do-while` และ `for` ในภาษาซีเพื่อแทนการทำงานแบบวนซ้ำได้

ใช้เวลารวม 6 คาบ

## 6.1 ลักษณะและองค์ประกอบของการทำงานแบบวนซ้ำ

การทำงานแบบวนซ้ำคือการทำงานบางอย่างซ้ำหลายๆ ครั้ง เช่น

1. การหาตำแหน่งรองคาน (จุดหมุนของคาน) ที่สมดุล ด้วยการ**ขยับตำแหน่งซ้ายขวา**ไปเรื่อยๆ หากคานยังไม่สมดุล ดังรูปที่ 6.1
2. เครื่องหยดสาร**ค่อยๆ เติมสารเคมี**ลงไปทีละ 1 มิลลิลิตรจนกว่าจะเกิดปฏิกิริยาเคมี
3. **รับข้อมูล**เกรดของนิสิต 20 รายวิชาเพื่อคำนวณเกรดเฉลี่ย




รูปที่ 6.1: การหาจุดสมดุลของคาน

ตารางที่ 6.1: ตัวอย่างการทำงานแบบวนซ้ำ

| ข้อ | สิ่งที่ต้องทำซ้ำ | เงื่อนไขในการทำซ้ำ/เลิกทำซ้ำ |
|---|---|---|
| 1 | การขยับตำแหน่งซ้าย/ขวา | **ทำต่อ**เมื่อคานยังไม่สมดุล ณ จุดปัจจุบัน |
| 2 | การหยดสาร | **เลิกทำ**เมื่อเกิดปฏิกิริยาเคมี |
| 3 | การรับข้อมูลเกรด | **ทำต่อ**เมื่อข้อมูลยังไม่ครบ 20 รายวิชา |

การทำงานแบบวนซ้ำจะประกอบด้วย สิ่งที่ต้องทำซ้ำ และเงื่อนไขในการทำซ้ำหรือเลิกทำซ้ำ จากสามตัวอย่างข้างต้น อาจเขียนสิ่งที่ต้องทำซ้ำ และเงื่อนไขในการทำซ้ำหรือเลิกทำซ้ำได้ดังตารางที่ 6.1

จากองค์ประกอบนี้ จะเห็นว่าการทำงานแบบวนซ้ำต้องมีเงื่อนไขประกอบเสมอ โดยเงื่อนไขจะเป็นตัวกำหนดว่าต้องทำซ้ำต่อหรือเลิกทำซ้ำ โครงสร้างผังงานสำหรับการทำงานแบบวนซ้ำจึงประกอบด้วยการทำงานแบบลำดับซึ่งมีการทำงานแบบเงื่อนไขอย่างน้อยหนึ่งเงื่อนไขเป็นองค์ประกอบ และเมื่อทำงานครบทุกลำดับแล้วจะย้อนกลับไปเริ่มทำที่ลำดับแรกอีกครั้ง ตัวอย่างเช่น ลำดับขั้นตอนอย่างง่ายของการทำงานในข้อแรกของตารางที่ 6.1 (รูปที่ 5.3) เขียนได้ดังนี้

1. เปรียบเทียบโมเมนต์ฝั่งซ้ายและฝั่งขวา ตรวจสอบความสมดุลของคาน หากสมดุล จบการทำงาน หากไม่สมดุล ทำข้อ 2
2. ขยับตำแหน่งรองไปฝั่งที่มีโมเมนต์มากกว่า

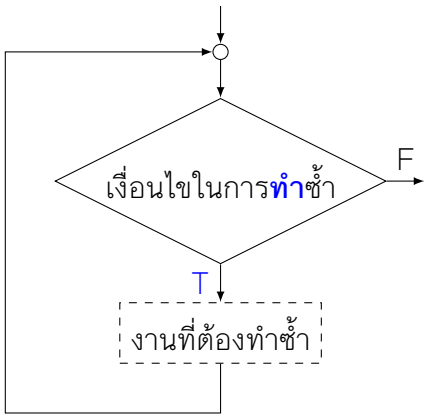
ถึงแม้การเขียนผังงานจะสามารถเขียนเงื่อนไขไว้ส่วนใดในลำดับก็ได้ก็ตาม เพื่อให้ผังงานเป็นระเบียบเรียบร้อย และเชื่อมโยงกับการเขียนโปรแกรม เรามักตรวจสอบเงื่อนไขก่อนเริ่มต้นการทำซ้ำ และเรียกโครงสร้างลักษณะนี้ว่า การทำซ้ำแบบทดสอบก่อนทำ หรือตรวจสอบเงื่อนไขเป็นงานสุดท้ายก่อนกลับไปเริ่มทำซ้ำรอบใหม่ และเรียกโครงสร้างลักษณะนี้ว่า การทำซ้ำแบบทดสอบหลังทำ ดังรายละเอียดในหัวข้อต่อไป

### ลองคิด

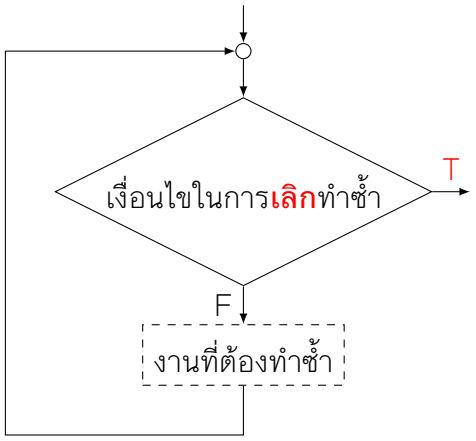
การทำงานใดๆ นั้นจะมีจุดเริ่มต้นและจุดสิ้นสุดเสมอ ปัญหาหนึ่งที่มักพบเมื่อมีโครงสร้างแบบวนซ้ำ คือ เกิดการวนซ้ำไม่รู้จบ สาเหตุของการวนซ้ำไม่รู้จบคือผลลัพธ์ของเงื่อนไขคงเดิมเสมอในทุกรอบของการวนซ้ำ เราจะป้องกันปัญหานี้ได้อย่างไร

## 6.2 การทำซ้ำแบบทดสอบก่อนทำและทดสอบหลังทำ

โครงสร้างการทำซ้ำในภาษาโปรแกรมแบ่งตามตำแหน่งการวางเงื่อนไขทดสอบได้สองรูปแบบ คือทดสอบก่อนเริ่มทำซ้ำ และทดสอบหลังจากทำซ้ำครบรอบ




(ก) โครงสร้างการวนซ้ำโดยใช้เงื่อนไขให้ทำซ้ำต่อ




(ข) โครงสร้างการวนซ้ำโดยใช้เงื่อนไขให้เลิกทำซ้ำ

รูปที่ 6.2: โครงสร้างการทำซ้ำแบบทดสอบก่อนทำ

### 6.2.1 การทำซ้ำแบบทดสอบก่อนทำ

โครงสร้างการทำงานแบบทดสอบก่อนทำจะเริ่มจากการตรวจสอบเงื่อนไขก่อน แล้วจึงเริ่มงานที่ต้องทำซ้ำ แบ่งเงื่อนไขออกเป็น 2 กรณี ดังนี้

1. เงื่อนไขให้**ทำ**ซ้ำต่อ
   - เมื่อเป็นจริง จะ**ทำ**งานที่ต้องทำซ้ำ
   - เมื่อเป็นเท็จ จะเลิกการทำซ้ำ
2. เงื่อนไขให้**เลิก**ทำซ้ำ
   - เมื่อเป็นจริง จะ**เลิก**การทำซ้ำ
   - เมื่อเป็นเท็จ จะทำงานที่ต้องทำซ้ำต่อ

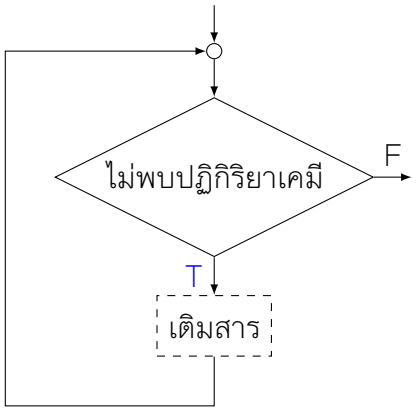
เมื่อเขียนเป็นผังงาน จะได้ดังรูปที่ 6.2(ก) และ 6.2(ข) ตามลำดับ

เงื่อนไขทั้งสองกรณีเป็นนิเสธของกันและกัน เราสามารถเลือกใช้เงื่อนไขแบบใดก็ได้ ในการเขียนผังงานนั้น ควรเลือกเงื่อนไขแบบที่เข้าใจง่าย เช่น จากตัวอย่างการเติมสารจนกว่าจะเกิดปฏิกิริยาเคมี จะเขียนเงื่อนไขได้สองรูปแบบดังรูปที่ 6.3

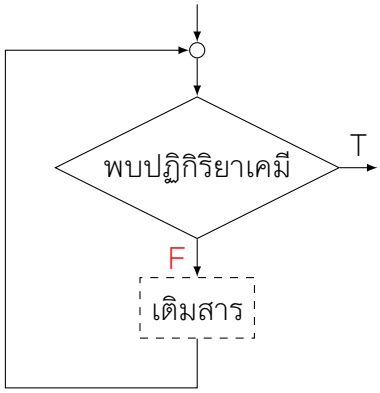
เงื่อนไขคือการตรวจสอบการเกิดปฏิกิริยาเคมี แบบ 6.3(ก) นั้นจะเป็นการทำซ้ำเมื่อเงื่อนไขเป็นเท็จ ส่วนแบบ 6.3(ข) เป็นการเลิกทำซ้ำเมื่อเงื่อนไขเป็นจริง

### 6.2.2 การทำซ้ำแบบทดสอบหลังทำ

อีกรูปแบบหนึ่งในการทำซ้ำคือ ทำสิ่งที่ต้องทำให้ครบก่อน แล้วจึงตรวจสอบเงื่อนไขว่าจะทำต่อหรือเลิกทำ ซึ่งเขียนเงื่อนไขได้ทั้งแบบทำต่อเมื่อเงื่อนไขเป็นจริง และเลิกทำเมื่อเงื่อนไขเป็นจริง ดังรูปที่ 6.4(ก)

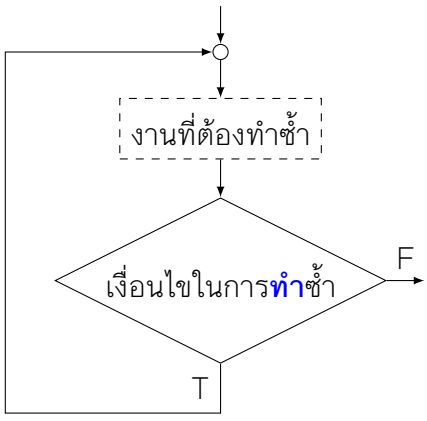



(ก) เมื่อไม่มีปฏิกิริยาเคมีให้เติมสาร

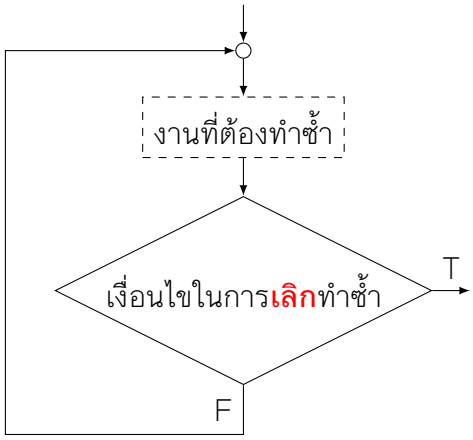



(ข) เมื่อมีปฏิกิริยาเคมีให้หยุดเติมสาร

รูปที่ 6.3: ส่วนของผังงานแสดงการเติมสารเคมีจนกว่าจะเกิดปฏิกิริยา




(ก) โครงสร้างการวนซ้ำโดยใช้เงื่อนไขให้ทำซ้ำต่อ




(ข) โครงสร้างการวนซ้ำโดยใช้เงื่อนไขให้เลิกทำซ้ำ

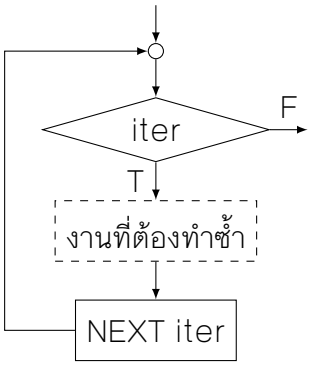
รูปที่ 6.4: โครงสร้างการทำซ้ำแบบทดสอบหลังทำ

และ 6.4(ข) ตามลำดับ

### 6.2.3 เปรียบเทียบการทำซ้ำแบบทดสอบก่อนทำและทดสอบหลังทำ

ในการออกแบบการทำงานนั้น จะเลือกการทำงานแบบทดสอบก่อนหรือหลังทำก็ได้ ข้อสังเกตหนึ่งคือ การทำซ้ำแบบทดสอบหลังทำ จะมีการทำซ้ำอย่างน้อย 1 รอบเสมอ ในขณะที่การทำซ้ำแบบทดสอบก่อนทำจะตรวจสอบเงื่อนไขก่อน จึงอาจจะไม่เกิดการทำซ้ำเลยก็ได้ หากต้องการทำซ้ำอย่างน้อย 1 รอบ ไม่ว่าจะเลือกโครงสร้างแบบใดก็ให้ผลลัพธ์ที่เหมือนกัน

โครงสร้างการทำซ้ำแบบทดสอบก่อนทำที่มักพบในภาษาโปรแกรมคือโครงสร้าง `while-do` ซึ่งหมายถึงในขณะที่เงื่อนไขยังคงเป็นจริงอยู่ ให้ทำซ้ำไปเรื่อยๆ ส่วนโครงสร้างการทำซ้ำแบบทดสอบหลัง




รูปที่ 6.5: โครงสร้างการทำซ้ำแบบใช้ตัวแจงนับ

ทำมีทั้งแบบ `do-while` คือทำซ้ำในขณะที่เงื่อนไขยังเป็นจริง และ `repeat-until` ซึ่งเป็นการทำซ้ำจนกว่าเงื่อนไขเป็นจริงจึงเลิกทำ

### ลองคิด

เขียนผังงานแบบทดสอบก่อนทำที่ทำงาน**เหมือนกับ**การทำงานแบบทดสอบหลังทำทุกประการ และลองเขียนในทางกลับกันด้วย

## 6.3 ตัวแจงนับ

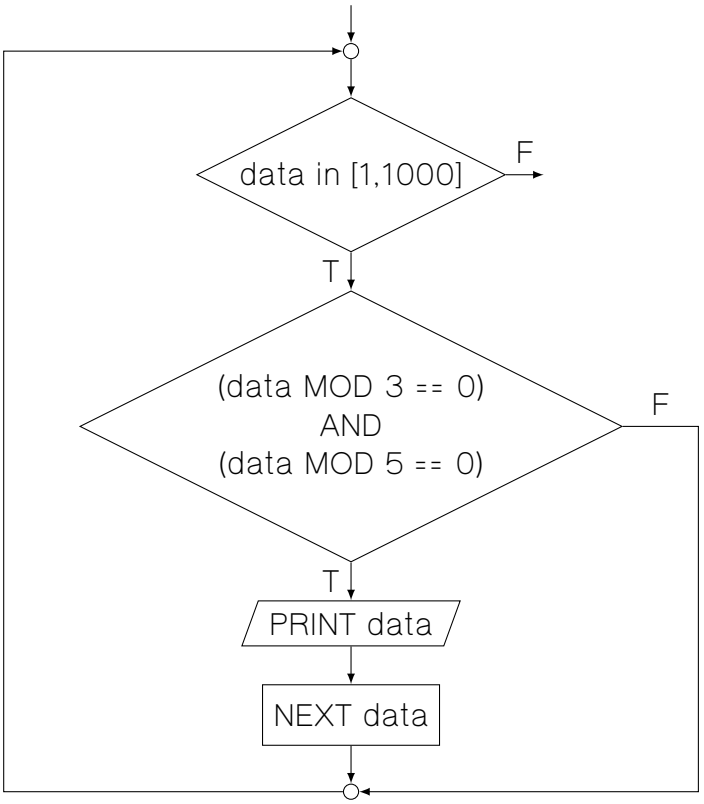
ตัวแจงนับ (iterator) เป็นวัตถุหนึ่งในภาษาโปรแกรม ซึ่งสามารถแจกแจงข้อมูลออกมาตามลำดับได้ การแจกแจงค่าโดยตัวแจงนับนั้นเป็นเพียงการสำเนาค่าหรือสร้างตัวชี้ไปยังข้อมูลในลำดับเท่านั้น โดยตัวแจงนับจะคืนค่าข้อมูลถัดไปในลำดับออกมาให้ และให้ผลลัพธ์เป็นเท็จเมื่อไม่มีข้อมูลเหลือในตัวลำดับอีกแล้ว จึงมักจะถูกใช้ร่วมกับโครงสร้างการทำซ้ำ โดยเป็นการแจกแจงค่าหรือข้อมูลทีละตัวเพื่อทำซ้ำ และยุติการทำซ้ำเมื่อตัวแจงนับไม่สามารถเลื่อนไปที่ข้อมูลตัวถัดไปได้อีก

โครงสร้างการทำซ้ำแบบใช้ตัวแจงนับสามารถเขียนเป็นผังงานได้ดังรูปที่ 6.5

**ตัวอย่าง 6.1** (การหาจำนวนทั้งหมดที่หารด้วย 3 และ 5 ลงตัวในช่วง 1-1000). ลักษณะของจำนวนที่หารด้วย 3 และ 5 ลงตัวคือ

- จำนวนที่หารด้วย 3 แล้วเหลือเศษ 0 และ
- จำนวนที่หารด้วย 5 แล้วเหลือเศษ 0

จำนวนทั้งหมดตั้งแต่ 1 ถึง 1000 นั้น เราสามารถใช้ตัวแจงนับในการแจกแจงทีละตัวได้ การหาจำนวนทั้งหมดที่หารด้วย 3 และ 5 ลงตัวจึงเป็นการแจกแจงทุกจำนวนในช่วงที่กำหนด แล้วทดสอบการหารลงตัว สรุปงานที่ต้องทำซ้ำ และเงื่อนไขในการทำซ้ำได้ดังนี้




รูปที่ 6.6: ส่วนของผังงานแสดงการหาจำนวนที่หารด้วย 3 และ 5 ลงตัวในช่วง [1,1000]

1. งานที่ต้องทำซ้ำ คือ ทดสอบการหารด้วย 3 และ 5 ลงตัว
2. เงื่อนไขในการทำซ้ำ คือ ตัวแจงนับที่มีค่าเรียงลำดับจาก 1 ถึง 1000

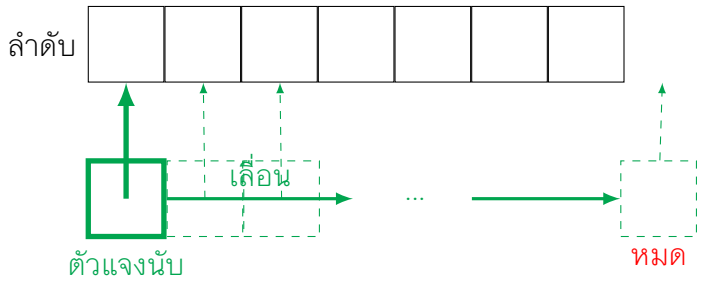
เขียนเป็นผังงานได้ดังรูปที่ 6.6 □

## 6.4 การออกแบบการทำงานแบบวนซ้ำ

งานบางชนิดนั้นเห็นได้ชัดตั้งแต่เริ่มว่าการทำซ้ำคืออะไร เช่น การเติมสารเคมีจนกว่าจะเกิดปฏิกิริยา แต่บางงานก็อาจจะไม่ตรงไปตรงมานัก เช่น การหาจุดสมดุลการหมุนของคานในรูปที่ 6.1 ซึ่งเมื่อมองปัญหานี้เป็นปัญหาการทดสอบจุดสมดุลและขยับจุดหมุนของคานไปเรื่อยๆ จนกว่าจะพบจุดที่สมดุล ก็จะเห็นว่าปัญหานี้เป็นการทำซ้ำรูปแบบหนึ่งเช่นกัน

การออกแบบการทำงานแบบวนซ้ำต้องกำหนดสิ่งที่ต้องทำซ้ำและเงื่อนไขในการทำซ้ำหรือเลิกทำซ้ำออกมาให้ได้ก่อน วิธีตรวจสอบว่างานที่ต้องทำซ้ำคืออะไร อาจจะลองทำด้วยมือไปสักสองสามรอบจนกว่าจะเห็นรูปแบบที่เกิดซ้ำขึ้น จากนั้นจึงไปดูว่าเขียนเงื่อนไขกำกับอย่างไร จึงจะทำให้รอบของการทำซ้ำเป็นไปตามที่ต้องการ

รูปแบบการทำซ้ำที่พบบ่อยแบ่งตามรูปแบบของเงื่อนไขในการทำซ้ำได้ดังนี้




รูปที่ 6.7: การใช้ตัวแจงนับแจกแจงสมาชิกในลำดับ

1. นับรอบ เงื่อนไขในการทำซ้ำคือ ทำยังไม่ครบจำนวนรอบที่กำหนด วิธีการนับรอบจะใช้ตัวแปรหนึ่งตัวในการนับ การนับเป็นได้ทั้งแบบนับก่อนเริ่มรอบ โดยใช้ตัวนับเริ่มต้นที่ 1 หรือนับรอบที่ทำไปแล้ว โดยตัวนับรอบจะเริ่มต้นที่ 0 (ยังไม่ได้ทำ) และทำไปจนกว่าจะครบจำนวนรอบ ในแต่ละรอบเมื่อทำงานครบรอบแล้ว จะมีการเพิ่มค่าตัวนับไปทีละหนึ่ง
2. แจกแจงสมาชิกในลำดับ เงื่อนไขในการทำซ้ำคือ ยังมีข้อมูลเหลืออยู่ในลำดับ หากมีตัวแจงนับให้ก็สามารถใช้ค่าจากตัวแจงนับเป็นเงื่อนไขได้ เช่นในรูปที่ 6.7 แต่หากไม่ใช้ตัวแจงนับ การทำงานแบบแจกแจงสมาชิกในลำดับจะเป็นการเลื่อนตัวชี้ของข้อมูลในลำดับไปทีละตัว เริ่มจากตัวแรกไปจนถึงตัวสุดท้าย เงื่อนไขในการทำซ้ำจึงเป็นตัวชี้ยังไม่เกินตัวสุดท้าย
3. แบบอื่นๆ เงื่อนไขในการทำซ้ำไม่มีรูปแบบตายตัว

**ตัวอย่าง 6.2** (การตั้งเวลาถ่ายภาพแบบ timelapse ทุก 1 นาทีเป็นเวลา 24 ชั่วโมง). การถ่ายภาพแบบ timelapse คือการตั้งเวลาถ่ายภาพทุกๆ ช่วงที่กำหนด แล้วนำภาพมาเรียงต่อกันเป็นภาพเคลื่อนไหว

งานที่ต้องทำซ้ำคือ การกดชัตเตอร์ หรือสั่งถ่ายภาพ ส่วนเงื่อนไขในการทำซ้ำคือถ่ายรูปไปจนครบ 24 ชั่วโมง ซึ่งอาจแปลงเป็นจำนวนครั้งที่ต้องถ่ายภาพ แล้วนับครั้งแทนได้

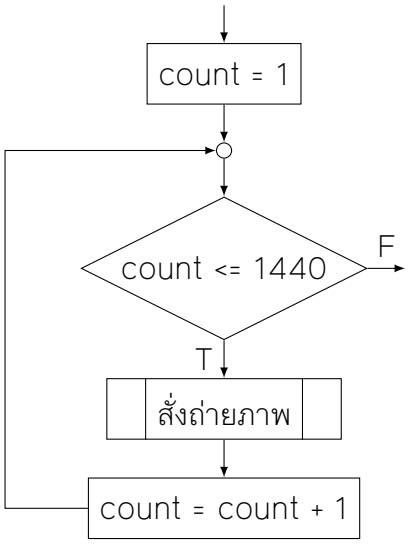
- ต้องถ่ายภาพทั้งหมด $60 \times 24 = 1440$ ครั้ง

ดังนั้น จึงสามารถเขียนเป็นผังงาน โดยกำหนดรอบของการวนซ้ำทั้งหมด 1440 รอบ และการทำงานแต่ละรอบคือการสั่งถ่ายภาพ

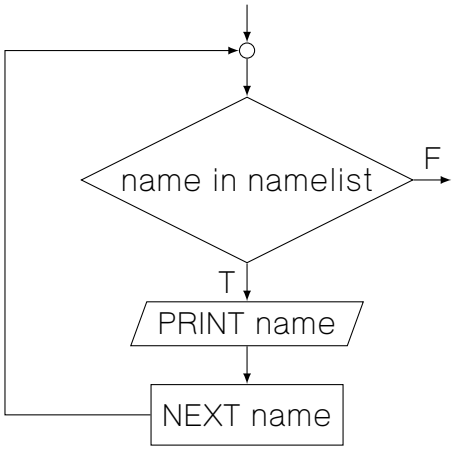
ให้ตัวแปร count เป็นตัวนับการเริ่มต้นของรอบ จึงเริ่มทำตั้งแต่รอบที่ 1 ถึงรอบที่ 1440 ในการนับรอบนั้นนับเพิ่มทีละ 1 จึงเขียนเป็นส่วนของผังงานได้ดังรูปที่ 6.8

□

**ตัวอย่าง 6.3** (การแสดงรายชื่อนักเรียนทุกคนในลำดับ). กำหนดให้มีข้อมูลแบบลำดับ ซึ่งมีตัวแจงนับสำหรับแจกแจงข้อมูลแต่ละตัวในลำดับได้อยู่ การแสดงรายชื่อนักเรียนทุกคนในลำดับจึงเป็นการสร้างตัวแจงนับสำหรับลำดับขึ้นมา แล้วแสดงข้อมูลแต่ละค่าที่ตัวแจงนับนั้นอ้างถึง ไปจนกว่าจะครบทุกข้อมูลในลำดับ




รูปที่ 6.8: ผังงานแสดงการตั้งเวลาถ่ายรูปทุก 1 นาที เป็นเวลา 24 ชั่วโมง



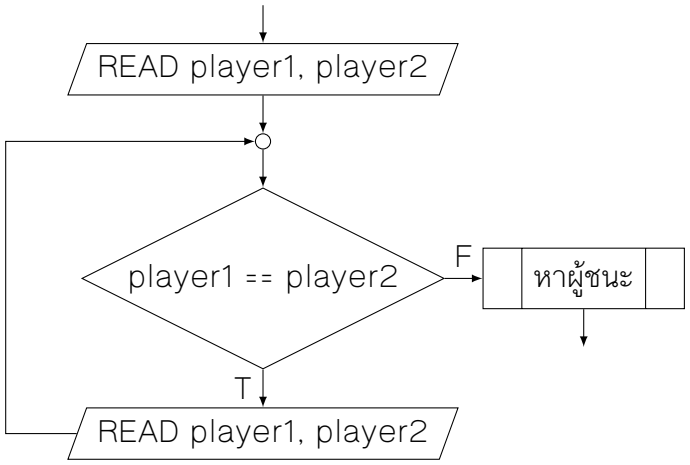

รูปที่ 6.9: ผังงานการแสดงรายชื่อนักเรียนทุกคนในลำดับ

เมื่อแจงนับครบทุกตัวแล้ว ตัวแจงนับจะชี้ไปยังข้อมูลนอกลำดับ หรือสถานะข้อมูลหมด ซึ่งจะทำให้ผลลัพธ์ของเงื่อนไขการเป็นสมาชิกในลำดับเป็นเท็จและจบการทำซ้ำโดยอัตโนมัติ เขียนงานที่ต้องทำซ้ำและเงื่อนไขได้ดังนี้

1. งานที่ต้องทำซ้ำ คือ แสดงข้อมูลชื่อ
2. เงื่อนไขในการทำซ้ำ คือ ตัวแจงนับยังชี้ไปที่ข้อมูลในลำดับ

กำหนดตัวแปร name เก็บชื่อที่ได้จากตัวแจงนับ จะเขียนเป็นส่วนของผังงานได้ตามรูปที่ 6.9 □

**ตัวอย่าง 6.4** (การหาผู้ชนะเกมเป่ายิ้งฉุบ)**.** การเล่นเป่ายิ้งฉุบมีผู้เล่น 2 คน แต่ละคนสามารถออก ค้อน หรือ กรรไกร หรือ กระดาษ ได้ การแพ้ชนะเป็นดังนี้




รูปที่ 6.10: ผังงานการหาผู้ชนะเกมเป่ายิ้งฉุบ

- ค้อน ชนะ กรรไกร
- กรรไกร ชนะ กระดาษ
- กระดาษ ชนะ ค้อน

หากผู้เล่นทั้งสองฝ่ายออกเหมือนกันจะถือว่าเสมอ และเริ่มต้นเล่นไหม

การหาผู้ชนะของเกมเป่ายิ้งฉุบจึงเป็นการเล่นวนซ้ำไปเรื่อยๆ จนกว่าจะมีฝ่ายใดฝ่ายหนึ่งชนะ สรุปได้ดังนี้

1. งานที่ต้องทำซ้ำ คือ การออกของผู้เล่นทั้งสองฝ่าย
2. เงื่อนไขในการทำซ้ำ คือ ผู้เล่นทั้งสองฝ่ายออกเหมือนกัน (เสมอ)

หลังจากการวนซ้ำ จะหาผู้ชนะระหว่างผู้เล่นทั้งสองฝ่ายได้ กำหนดตัวแปร player1 และ player2 เก็บข้อมูลการออกของผู้เล่นแต่ละฝ่าย เขียนส่วนของผังงานได้ดังรูปที่ 6.10

□

ข้อสังเกตหนึ่งจากตัวอย่าง 6.4 คือ ตัวแปรที่ใช้ในการตรวจสอบค่า ต้องถูกกำหนดค่า หรือมีค่า ก่อนจะใช้ในการตรวจสอบ จึงต้องมีการอ่านข้อมูลครั้งแรกนอกการวนซ้ำก่อน

## 6.5 ภาษาโปรแกรมที่ใช้ในการวนซ้ำ

### 6.5.1 โครงสร้าง `while-do`

โครงสร้าง `while-do` มีลักษณะเป็นการวนซ้ำแบบทดสอบก่อนทำ และทำซ้ำเมื่อเงื่อนไขเป็นจริง เขียนเป็นรหัสเทียมได้ดังรหัสคำสั่งที่ 6.1

```
1 WHILE cond
2 DO
3     job
4 ENDWHILE
```

รหัสคำสั่งที่ 6.1: โครงสร้าง `while-do`

`cond` เป็นนิพจน์ที่ให้ค่าจริงหรือเท็จ ส่วน `job` แทนงานที่ต้องทำ ซึ่งอาจมีมากกว่าหนึ่งงานก็ได้ รหัสคำสั่งที่ 6.1 เทียบได้กับผังงานในรูปที่ 6.2(ก)

### 6.5.2 โครงสร้าง **do-while**

โครงสร้าง `do-while` มีลักษณะเป็นการวนซ้ำแบบทดสอบหลังทำ และทำซ้ำเมื่อเงื่อนไขเป็นจริง เขียนเป็นรหัสเทียมได้ดังรหัสคำสั่งที่ 6.2

```
1 DO
2     job
3 WHILE cond
```

รหัสคำสั่งที่ 6.2: โครงสร้าง `do-while`

รหัสคำสั่งที่ 6.2 นี้เทียบได้กับผังงานในรูปที่ 6.4(ก) ส่วน `cond` และ `job` เป็นเงื่อนไขในการทำซ้ำและงานที่ต้องทำซ้ำ เช่นเดียวกับในรหัสคำสั่งที่ 6.1

### 6.5.3 โครงสร้าง **repeat-until**

โครงสร้าง `repeat-until` มีลักษณะเป็นการวนซ้ำแบบทดสอบหลังทำ และเลิกทำซ้ำเมื่อเงื่อนไขเป็นจริง เขียนเป็นคำสั่งได้ดังนี้

```
1 REPEAT
2     job
3 UNTIL cond
```

รหัสคำสั่งที่ 6.3: โครงสร้าง `repeat-until`

รหัสคำสั่งที่ 6.3 เทียบได้กับผังงานในรูปที่ 6.4(ข) ส่วน `cond` และ `job` เป็นเงื่อนไขในการทำซ้ำ และงานที่ต้องทำซ้ำ เช่นเดียวกับในรหัสคำสั่งที่ 6.1 และ 6.2

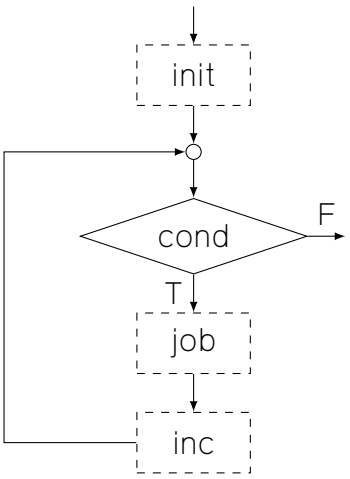
### 6.5.4 โครงสร้าง `for`

โครงสร้าง `for` มีลักษณะเป็นการวนซ้ำแบบทดสอบก่อนทำ และทำซ้ำเมื่อเงื่อนไขเป็นจริงเช่นเดียวกับโครงสร้าง `while-do` แต่โครงสร้าง `for` แบ่งออกเป็นสองรูปแบบ ได้แก่

1. โครงสร้าง `for` แบบดั้งเดิม ซึ่งมีการกำหนดงานที่ต้องทำก่อนทำซ้ำ เงื่อนไข และงานที่ต้องทำก่อนครบรอบการทำซ้ำ เขียนเป็นรหัสเทียมได้ดังรหัสคำสั่งที่ 6.4 และเขียนส่วนของผังงานได้ดังรูปที่ 6.11
2. โครงสร้าง `for` แบบใช้กับตัวแจงนับ มักใช้ในการทำซ้ำกับข้อมูลทั้งหมดในรายการ โดยอาศัยตัวแจงนับเป็นตัวแจกแจงข้อมูล เขียนเป็นรหัสเทียมได้ดังรหัสคำสั่งที่ 6.5 และเขียนส่วนของผังงานได้ดังรูปที่ 6.12 ซึ่งเหมือนกับผังงานในรูปที่ 6.5

```
1 FOR (init; cond; inc)
2     job
3 ENDFOR
```

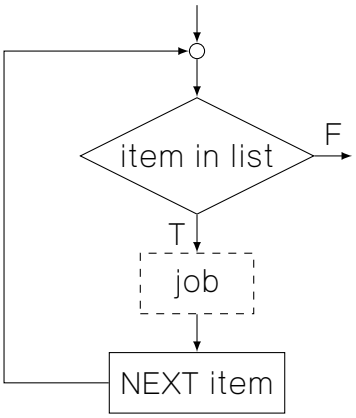
รหัสคำสั่งที่ 6.4: โครงสร้าง `for` แบบดั้งเดิม




รูปที่ 6.11: โครงสร้าง `for` แบบดั้งเดิม

```
1 FOR item IN list
2     job
3 ENDFOR
```

รหัสคำสั่งที่ 6.5: โครงสร้าง `for` แบบใช้ตัวแจงนับ




รูปที่ 6.12: โครงสร้าง `for` แบบใช้ตัวแจงนับ

สำหรับแบบดั้งเดิม init คืองานที่ต้องทำก่อนเริ่มทำซ้ำ cond คือเงื่อนไขในการทำซ้ำเมื่อจริง และ inc คืองานสุดท้ายที่ต้องทำในรอบของการทำซ้ำ และ job คือคำสั่งต่างๆ ที่ต้องทำซ้ำ

ส่วนแบบตัวแจงนับนั้น list เป็นรายการข้อมูลซึ่งมีตัวแจงนับ item เป็นตัวแปรซึ่งรับค่าจากตัวแจงนับซึ่งจะชี้ไปที่ข้อมูลในรายการทีละตัว เมื่อเวียนไปครบทุกข้อมูลแล้ว ก็จบการทำซ้ำได้

## 6.6 โครงสร้างการทำซ้ำในภาษาซี

ภาษาซีมีโครงสร้างการทำซ้ำ 3 รูปแบบ ได้แก่

1. โครงสร้าง `while-do`

```
while (cond)
    stmt
```

รหัสคำสั่งที่ 6.6: โครงสร้าง **while-do** ในภาษาซี

โดยที่ cond เป็นตัวแปรหรือนิพจน์ซึ่งมีค่าเป็นจำนวนเต็ม และทำซ้ำในกรณีที่ cond ไม่เป็น 0 ส่วน stmt คืองานที่ต้องทำซ้ำ ซึ่งอยู่ในรูปแบบคำสั่งโดดหรือบล็อก

2. โครงสร้าง `do-while`

```
do
{
    stmts
} while (cond);
```

รหัสคำสั่งที่ 6.7: โครงสร้าง **do-while** ในภาษาซี

โดยที่ cond เป็นตัวแปรหรือนิพจน์ซึ่งมีค่าเป็นจำนวนเต็มเช่นเดียวกับ `while-do` และย้อนกลับไปทำซ้ำในกรณีที่ cond ไม่เป็น 0 เช่นกัน และ stmts คือชุดของคำสั่งที่ทำซ้ำ จะมีคำสั่งเดียวหรือมากกว่าก็ได้

3. โครงสร้าง `for` แบบดั้งเดิม

```
for (init; cond; inc)
    stmt
```

รหัสคำสั่งที่ 6.8: โครงสร้าง **for** ในภาษาซี

โดยที่ init คือชุดของคำสั่งที่ทำก่อนเริ่มทำซ้ำ หากมีมากกว่าหนึ่งคำสั่งให้คั่นแต่ละคำสั่งด้วยจุลภาค (,) cond คือตัวแปรหรือนิพจน์ซึ่งมีค่าเป็นจำนวนเต็ม และทำซ้ำในกรณีที่ cond ไม่เป็น 0 ส่วน inc คือชุดของคำสั่งที่ต้องทำก่อนจะจบรอบการวนซ้ำ และ stmt คืองานที่ต้องทำซ้ำ ซึ่งอยู่ในรูปแบบคำสั่งโดดหรือบล็อก ในภาษาซีไม่มีโครงสร้าง for แบบใช้ตัวแจงจับ

### ลองคิด

โครงสร้างการทำซ้ำในภาษาซีนั้นเป็นแบบทำซ้ำเมื่อเงื่อนไขเป็นจริงทั้งหมด หากต้องการใช้โครงสร้างการทำซ้ำเมื่อเงื่อนไขเป็นเท็จ (เลิกทำเมื่อเงื่อนไขเป็นจริง) จะเขียนเป็นภาษาซีได้อย่างไร

## แบบฝึกหัด

1. เมื่อรับจำนวนเข้าเป็นจำนวนเต็มบวกหนึ่งจำนวน จงเขียนผังงานและโปรแกรมภาษาซีเพื่อหาว่าจำนวนที่รับเข้ามานี้เป็นจำนวนเฉพาะหรือไม่

   จำนวนเฉพาะคือจำนวนที่มีเฉพาะ 1 และตัวมันเองเท่านั้นที่หารลงตัว

2. จงเขียนผังงานและโปรแกรมภาษาซีให้ผู้ใช้ป้อนจำนวนบรรทัดที่ต้องการ แล้วพิมพ์อักขระ * ออกมาโดยมีลักษณะดังนี้

```
6
*
**
***
****
*****
******
```

   บรรทัดที่ 1 มี * 1 ตัว บรรทัดที่ 2 มี * 2 ตัว ...
   (บรรทัดที่ i จะมี * i ตัว โดยที่ i เป็นจำนวนเต็มบวก)

3. เมื่อให้ผู้ใช้ป้อนจำนวนบรรทัดเข้ามา จงเขียนโปรแกรมภาษาซีเพื่อพิมพ์ * ให้เป็นรูปสามเหลี่ยมหน้าจั่วซึ่งมีลักษณะดังนี้

```
5
     *
    ***
   *****
  *******
*********
```

4. ระบบนับถอยหลังจะรับข้อมูลเป็นจำนวนเต็มบวกหนึ่งจำนวน แล้วนับถอยหลังโดยแสดงค่าตั้งแต่จำนวนที่รับเข้า ลดลงทีละหนึ่ง จนถึง 1 แล้วจบการทำงาน จงเขียนผังงานและโปรแกรมภาษาซีของระบบนับถอยหลังนี้
5. ในการแข่งขันยิมนาสติก จะมีกรรมการทั้งหมด 6 คน แต่ละคนให้คะแนนได้ตั้งแต่ 0.0 - 10.0 การคิดคะแนนของผู้เข้าแข่งขันจะตัดคะแนนสูงสุดและต่ำสุดออก แล้วหารเฉลี่ยคะแนนของ

กรรมการทั้ง 4 คนที่เหลือ หากมีผู้เข้าแข่งขันทั้งหมด 10 คน จงเขียนโปรแกรมเพื่อหาผู้ชนะเมื่อผู้ดูแลการแข่งขันกรอกคะแนนจากกรรมการทั้ง 6 คนของผู้เข้าแข่งขันแต่ละคนให้ระบบ

6. เกมทายจำนวนใช้ผู้เล่นสองคน ผู้เล่นคนแรกจะป้อนข้อมูลตัวเลขในใจของตนเองให้กับระบบ จากนั้นผู้เล่นคนที่สองจะทายจำนวนที่ผู้เล่นคนแรกป้อนให้ระบบ โดยระบบจะนับจำนวนครั้งที่ทายจนกว่าจะทายถูก หากทายไม่ถูก ระบบจะช่วยบอกใบ้ด้วยการบอกว่าค่าที่ทายนั้น มากกว่า หรือน้อยกว่า ค่าที่ผู้คนแรกกำหนดไว้ หากทายถูก ระบบจะแสดงจำนวนครั้งที่ทายแล้วจบการทำงาน จงเขียนผังงานและโปรแกรมภาษาซีของระบบเกมทายจำนวนนี้

# บทที่ 7

# ตัวแปรแถวลำดับ

## วัตถุประสงค์การเรียนรู้

อธิบายการเก็บข้อมูลแบบแถวลำดับ การเก็บข้อมูลสายอักขระ และการใช้ตัวแปรแถวลำดับ

1. กำหนดค่าให้ตัวแปรแถวลำดับ และเรียกใช้ค่าในตัวแปรแถวลำดับได้
2. ประกาศตัวแปรแบบแถวลำดับได้
3. อธิบายการเข้าถึงอักขระต่างๆ ในตัวแปรแบบสายอักขระได้
4. อธิบายโครงสร้างการเก็บข้อมูลแบบแถวลำดับในหน่วยความจำ
5. ใช้แถวลำดับขนาดมากกว่าหนึ่งมิติได้

ใช้เวลารวม 6 คาบ

## 7.1 การเข้าถึงและการเรียกใช้ตัวแปรแถวลำดับ

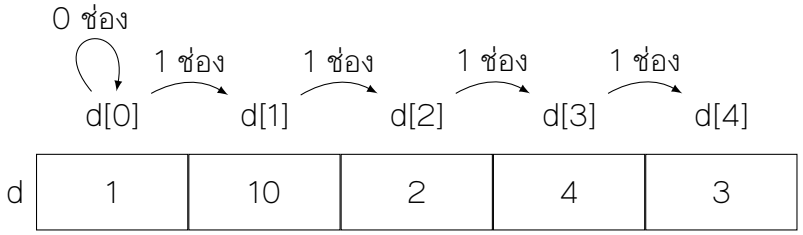
ตัวแปรโดยปกติทั่วไปนั้น จะเก็บค่าได้หนึ่งค่าต่อตัวแปรหนึ่งตัว แต่หากข้อมูลมีลักษณะเป็นชุดซึ่งมีหลายค่า เราจำเป็นต้องประกาศตัวแปรเท่ากับจำนวนข้อมูลที่มี ซึ่งไม่สะดวกในการจัดการข้อมูลซึ่งเป็นชุดในลักษณะนี้ จึงมีตัวแปรชนิดแถวลำดับ (array) ขึ้นมาช่วยอำนวยความสะดวกในการจัดการตัวแปรให้เป็นชุดๆ

ข้อมูลแต่ละตัวในตัวแปรชนิดแถวลำดับจะมีลำดับกำกับ เรียกว่า ดัชนี (index) การเข้าถึงข้อมูลในแถวลำดับทำได้ด้วยการอ้างอิงชื่อตัวแปรประกอบกับเลขดัชนีของข้อมูลนั้น เขียนได้ดังรหัสคำสั่งที่ 7.1

```
varname[index]
```

รหัสคำสั่งที่ 7.1: การอ้างถึงตัวแปรแถวลำดับ

โดยที่ `varname` คือชื่อของตัวแปร และ `index` เป็นจำนวนเต็มซึ่งเป็นเลขดัชนีของข้อมูล




รูปที่ 7.1: เลขดัชนีของตัวแปรแถวลำดับ

ภาษาโปรแกรมส่วนมาก เช่น ซี จาวา ไพธอน จะมีดัชนีเริ่มต้นที่ 0 หมายถึง ระยะห่างศูนย์ช่อง จากช่องแรกที่เก็บข้อมูล แต่บางภาษาโปรแกรม เช่น Matlab ก็ให้มีดัชนีเริ่มต้นที่ 1 ซึ่งหมายถึงช่องที่หนึ่ง ก่อนจะอ้างถึงตัวแปรแถวลำดับจึงควรตรวจสอบข้อกำหนดของดัชนีในแต่ละภาษาเสียก่อน ในเอกสารฉบับนี้จะให้ดัชนีเริ่มต้นที่ศูนย์

จากรูปที่ 7.1 ตัวแปร d เป็นตัวแปรชนิดแถวลำดับขนาด 5 ช่อง โดยมีดัชนีเริ่มต้นที่ 0 ข้อมูลในช่องที่หนึ่ง (ดัชนีเป็น 0) มีค่า 1 อ้างอิงถึงโดยชื่อตัวแปรและดัชนีได้เป็น d[0]

การกำหนดค่าให้แต่ละช่องในตัวแปรแถวลำดับใช้การอ้างอิงถึงช่องนั้นเช่นเดียวกับการเข้าถึงข้อมูล การกำหนดค่าใช้หลักการเดียวกับการกำหนดค่าให้ตัวแปรทั่วไป ค่าที่กำหนดได้เป็นได้ทั้งค่าคงที่ นิพจน์ และตัวแปร เช่น d[0] หมายถึงช่องที่หนึ่ง หากกำหนดค่าด้วยคำสั่ง d[0] = d[1]+d[2] ค่าในช่องที่หนึ่งก็จะเปลี่ยนเป็น 12 ซึ่งเกิดจาก 10+2 เป็นต้น

ชนิดของข้อมูลแต่ละตัวในแถวลำดับขึ้นกับข้อกำหนดของภาษาโปรแกรม บางภาษา เช่น ภาษาซี จะกำหนดให้ข้อมูลทุกตัวในแถวลำดับต้องเป็นชนิดเดียวกัน แต่บางภาษา เช่น ไพธอน ก็ไม่มีข้อกำหนดเรื่องนี้[1]

**ลองคิด**

เลขดัชนีของแถวลำดับสามารถเป็นจำนวนจริงได้หรือไม่

## 7.2 แถวลำดับกับการวนซ้ำ

เนื่องจากการทำงานกับตัวแปรแถวลำดับนั้นมักต้องไล่เรียงค่าตามช่องต่างๆ ในแถวลำดับ การทำงานกับแถวลำดับจึงมักมาคู่กับการวนซ้ำแบบทดสอบก่อนทำแบบโครงสร้าง `for` โดยเปลี่ยนค่าดัชนีไปในแต่ละรอบของการวนซ้ำ หรือการใช้ตัวแจงนับเพื่อเข้าถึงข้อมูลแต่ละตัวในแถวลำดับ

โครงสร้าง `for` แบบดั้งเดิมนั้นออกแบบมาให้มีการตั้งค่าเริ่มต้น ซึ่งใช้เป็นการตั้งค่าเริ่มต้นให้ตัวแปรซึ่งเก็บค่าดัชนี และงานสุดท้ายที่ต้องทำก่อนจบรอบการทำซ้ำ ให้เป็นการเพิ่มค่าในตัวแปรดัชนีเพื่อให้ชี้

[1]ข้อมูลชนิดแถวลำดับในไพธอน คือ List แต่มีลักษณะเป็นแถวลำดับ (array) ซึ่งสามารถเข้าถึงตำแหน่งใดในลำดับก็ได้ด้วยการกำหนดดัชนี ข้อมูลชนิด List ของไพธอนไม่ใช่ข้อมูลชนิดรายการ (list) ซึ่งเข้าถึงได้ตามลำดับจากต้นรายการ แต่ไม่สามารถกำหนดดัชนีเพื่อเข้าถึงตำแหน่งที่ต้องการได้โดยตรง

ตารางที่ 7.1: รายการตัวแปรสำหรับการแปลงค่าให้อยู่ในช่วง [0,1]

| **ชื่อตัวแปร** | **ชนิด** | **คำอธิบาย** |
|---|---|---|
| x | float[100] | ข้อมูลเข้า |
| z | float[100] | ข้อมูลที่แปลงเป็นบรรทัดฐานแล้ว |
| min | float | ค่าต่ำสุด |
| max | float | ค่าสูงสุด |

ไปที่ช่องถัดไป ส่วนโครงสร้าง `for` แบบใช้ตัวแจงนับนั้นจะเป็นการแจกแจงสมาชิกแต่ละตัวในแถวลำดับทีละตัว

**ตัวอย่าง 7.1** (การแปลงค่าให้อยู่ใช้ช่วง 0-1 (normalization)). การนำข้อมูลต่างๆ ไปใช้วิเคราะห์ผลนั้น เราควรปรับค่าของข้อมูลในอยู่ในช่วงที่เข้าใจง่าย เช่น ค่าต่ำสุดเป็น 0 และค่าสูงสุดเป็น 1 วิธีแปลงข้อมูลเข้าอย่างง่ายคือใช้การแปลงเชิงเส้นดังสมการ (7.1)

$$z = \frac{x - min}{max - min} \tag{7.1}$$

เมื่อ $x$ คือข้อมูลเข้าแต่ละตัว $min$ และ $max$ คือค่าต่ำสุดและค่าสูงสุดจากข้อมูลเข้าทั้งหมด จะได้ $z$ เป็นข้อมูลที่ปรับให้อยู่ในช่วง [0,1] แล้ว

เมื่อรับข้อมูลจำนวนจริงเข้ามา 100 ค่า เก็บลงในตัวแปรแถวลำดับ จะเขียนผังงานเพื่อแปลงแถวลำดับของข้อมูลเข้านี้ให้เป็นแถวลำดับของค่าที่ปรับช่วงแล้วได้ดังรูปที่ 7.2

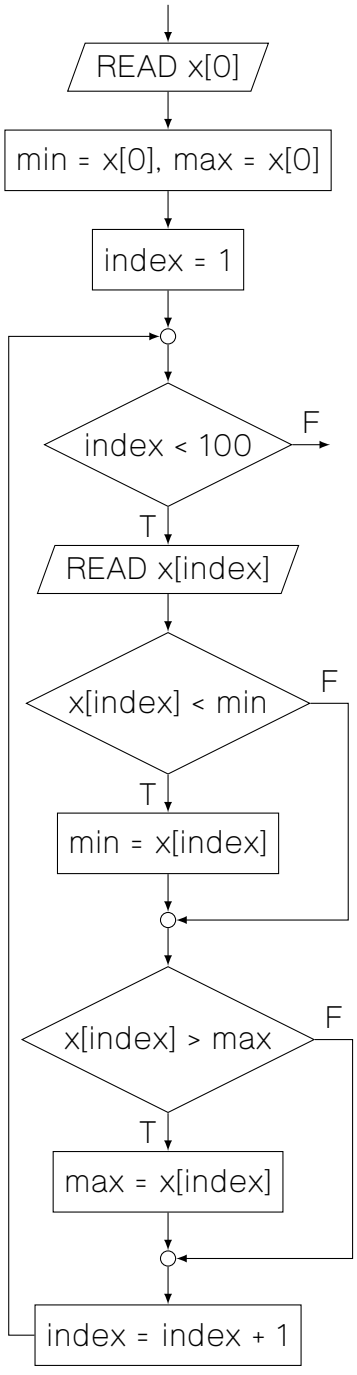
จากสมการ (7.1) การแปลงค่าให้อยู่ในช่วง [0,1] นั้นต้องหาค่าต่ำสุด ($min$) และค่าต่ำสุด ($max$) ก่อน จากนั้นจึงคำนวณค่า $z$ สำหรับข้อมูลแต่ละตัวในแถวลำดับ จึงแบ่งงานออกเป็น 2 ส่วน ดังนี้

1. หาค่าต่ำสุดและค่าสูงสุด ซึ่งใช้การวนซ้ำให้ครบจำนวนข้อมูลทั้งหมด 1 รอบ
2. แปลงข้อมูลให้อยู่ในช่วง [0,1] โดยใช้สมการ 7.1 และเก็บลงแถวลำดับใหม่ ซึ่งใช้การวนซ้ำคำนวณค่าข้อมูลทีละจำนวนจนครบข้อมูลทั้งหมด 1 รอบ

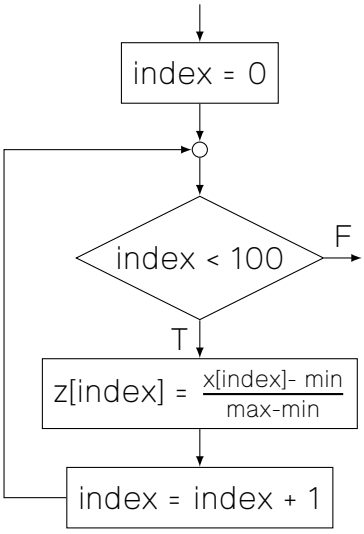
รายการตัวแปรที่ต้องใช้เป็นดังตารางที่ 7.1 และเขียนผังงานแต่ละส่วนได้ดังรูปที่ 7.2(ก) และ 7.2(ข) ตามลำดับ □

## 7.3 แถวลำดับสองมิติ

ตัวแปรแถวลำดับนั้นเป็นตัวชี้ไปยังข้อมูล หรืออีกนัยหนึ่งคือเป็นตัวแปรซึ่งเก็บที่อยู่ของข้อมูล โดยที่ข้อมูลจะมีชนิดใดๆ ก็ได้ตามแต่จะประกาศตัวแปรไว้ สมมติให้มีตารางขนาด $m$ แถว แถวละ $n$ ตัว เราสามารถจองพื้นที่สำหรับตัวแปรนี้เป็นให้เก็บข้อมูล $m$ ช่อง แต่ละช่องเป็นที่สำหรับเก็บแถวลำดับขนาด $n$ ตัวได้




(ก) การหาค่าสูงสุดและค่าต่ำสุด




(ข) การแปลงข้อมูลให้อยู่ในช่วง [0,1]

รูปที่ 7.2: ส่วนของผังงานสำหรับการแปลงข้อมูลให้อยู่ในช่วง [0,1]

ตารางที่ 7.2: การอ้างถึงแถวลำดับสองมิติ

| | [0] | [1] | [2] |
|---|---|---|---|
| แถว 0 | 1 | 2 | 3 |
| แถว 1 | 4 | 5 | 6 |

จะเห็นว่าการจองพื้นที่นี้เป็นการสร้างแถวลำดับซ้อนแถวลำดับ เรียกว่า แถวลำดับสองมิติ การเข้าถึงข้อมูลในแถวลำดับสองมิติทำได้โดยเลือกแถวลำดับที่ต้องการก่อนด้วยการระบุดัชนีของแถวลำดับนั้น จากนั้นจึงระบุตำแหน่งของข้อมูลด้วยดัชนีในแถวลำดับนั้นอีกทีหนึ่ง

**ตัวอย่าง 7.2** (แถวลำดับสองมิติ)**.** ตาราง ชื่อ `table` ขนาด 2 × 3 ซึ่งมีข้อมูลดังตารางที่ 7.2 การเข้าถึงข้อมูลต่างๆ จะได้ผลลัพธ์ดังนี้

- `table[1][0]` เก็บค่า 4 (ดัชนีแถว 1 ดัชนีข้อมูล 0)
- `table[0][1]` เก็บค่า 2 (ดัชนีแถว 0 ดัชนีข้อมูล 1)
- `table[1]` หมายถึงเลขที่อยู่ของแถวที่สอง ซึ่งเก็บค่า [4,5,6]
- `table` หมายถึงเลขที่อยู่ของตารางนี้

□

**ลองคิด**

เราสามารถจองพื้นที่สำหรับตัวแปรแถวลำดับมากกว่าสองมิติได้หรือไม่ หากได้ วิธีการอ้างถึงข้อมูลควรเป็นอย่างไร

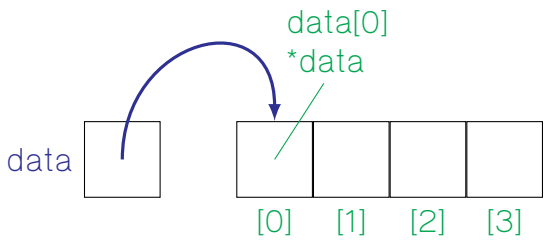
## 7.4 ตัวแปรแถวลำดับในภาษาซี

ภาษาซีกำหนดให้ข้อมูลทุกตัวในแถวลำดับต้องเป็นชนิดเดียวกัน การประกาศตัวแปรแถวลำดับจึงใช้การระบุชนิดของข้อมูลในแถวลำดับ และขนาดของแถวลำดับ ขนาดของแถวลำดับคือจำนวนข้อมูลที่อยู่ในแถวลำดับ

```
int data[10];
float score[30];
char name[40];
```

รหัสคำสั่งที่ 7.2: ตัวอย่างการประกาศตัวแปรแถวลำดับในภาษาซี

การประการตัวแปรแถวลำดับแบ่งออกเป็น 2 แบบ แยกตามวิธีกำหนดขนาดของแถวลำดับ ได้แก่ แบบกำหนดขนาดล่วงหน้าด้วยค่าคงที่ และแบบระบุขนาดเมื่อทำงาน การประกาศตัวแปรแถวลำดับแบบ




รูปที่ 7.3: แถวลำดับกับตัวแปรชนิดตัวชี้

กำหนดขนาดล่วงหน้าจะระบุชนิดของข้อมูลในแถวลำดับ และจำนวนช่องที่ต้องการเป็นค่าคงที่ จากรหัสคำสั่งที่ 7.2 ตัวแปร `data` เป็นแถวลำดับของจำนวนเต็มขนาด 10 ช่อง ตัวแปร `score` เป็นแถวลำดับของจำนวนจริงขนาด 30 ช่อง และตัวแปร `name` เป็นแถวลำดับของอักขระขนาด 40 ช่อง

การประกาศตัวแปรในรูปแบบนี้เรียกว่าเป็นการจองพื้นที่สำหรับตัวแปร และกำหนดขนาดคงที่ตั้งแต่ตอนคอมไพล์ (static allocation) เนื่องจากจำนวนช่องเป็นค่าคงที่ในรหัสต้นฉบับ คอมไพเลอร์จะรู้ขนาดของหน่วยความจำที่ต้องใช้ และกำหนดให้ระบบปฏิบัติการจองพื้นที่ได้ตามขนาดที่ต้องการ เมื่อใช้งานโปรแกรมเสร็จ เครื่องจะสามารถคืนพื้นที่ในหน่วยความจำให้ระบบปฏิบัติการได้โดยอัตโนมัติ

ในบางครั้งเราอาจไม่รู้ขนาดของแถวลำดับในขณะเขียนรหัสต้นฉบับ การประกาศตัวแปรเมื่อไม่รู้ขนาดล่วงหน้าจึงเป็นการจองพื้นที่สำหรับตัวชี้ของแถวลำดับ แล้วให้หน่วยความจำจองพื้นที่เพิ่มเติมเมื่อทำงาน (dynamic allocation) ทำผ่านคำสั่ง `malloc`, `calloc`

การจองพื้นที่ในรูปแบบนี้นั้น ระบบปฏิบัติการไม่สามารถรู้ขนาดพื้นที่ที่ต้องใช้ในหน่วยความจำได้ล่วงหน้า ทำให้เมื่อใช้งานเสร็จแล้ว ผู้เขียนโปรแกรมจำเป็นต้องระบุคำสั่งสำหรับคืนพื้นที่ในหน่วยความจำให้ระบบปฏิบัติการเอง การคืนพื้นที่ใช้คำสั่ง `free`

ในการเก็บข้อมูลชนิดแถวลำดับ เครื่องจะจองพื้นที่ติดกันในหน่วยความจำสำหรับแถวลำดับ และอ้างถึงพื้นที่ทั้งบริเวณด้วยตัวแปรชนิดตัวชี้ โดยข้อมูลในตัวแปรชนิดตัวชี้จะเป็นที่อยู่ของข้อมูลตำแหน่งแรกในแถวลำดับ ข้อมูลตัวที่สองจะอยู่ติดกับข้อมูลตัวแรก ซึ่งเป็นที่อยู่ในตัวชี้บวกกับขนาดของข้อมูลตัวแรกนั้น

เช่น ในรูปที่ 7.3 แถวลำดับของจำนวนเต็มขนาด 4 ช่อง อ้างถึงด้วยตัวแปรชื่อ `data` และจำนวนเต็ม 1 จำนวนใช้พื้นที่ 4 ไบต์ ข้อมูลที่อยู่ในตัวแปร data จะเป็นที่อยู่ของแถวลำดับ และเป็นที่อยู่เดียวกับข้อมูลตัวแรก หากให้การอ้างถึงข้อมูลด้วยตัวชี้ใช้สัญลักษณ์ * การอ้างถึง `*data` จะหมายถึงข้อมูลตัวแรก ซึ่งเทียบเท่ากับ `data[0]` สำหรับข้อมูลตัวถัดไป จะอยู่ ณ เลขที่อยู่ `data+4` เนื่องจากจำนวนเต็มหนึ่งตัวมีขนาด 4 ไบต์ จึงอ้างถึงผ่านตัวแปรชนิดตัวชี้ได้เป็น `*(data+4)` หรือ `data[1]` ดังนั้น การใช้เลขดัชนีช่วย จะช่วยให้อ้างถึงข้อมูลในแถวลำดับได้โดยไม่จำเป็นต้องบวกลบเลขที่อยู่เอง ช่วยอำนวยความสะดวกให้ผู้เขียนโปรแกรม

หากต้องการเพิ่มขนาดของแถวลำดับ ต้องจองพื้นที่ใหม่ที่มีขนาดตามต้องการ แล้วสำเนาค่าในแถวลำดับเดิมมายังที่ใหม่ ระบบปฏิบัติการอาจไม่สามารถใช้หน่วยความจำในตำแหน่งเดิมได้เนื่องจากพื้นที่ข้างเคียงบริเวณเดิมอาจจะมีข้อมูลอื่นอยู่

**ลองคิด**

หากใส่ดัชนีเป็นลบ หรือมีค่าเกินกว่าจำนวนช่องของข้อมูล จะเกิดผลอย่างไร

## 7.5 ตัวแปรชนิดสายอักขระในภาษาซี

### 7.5.1 การเก็บข้อมูลแบบสายอักขระ

ภาษาซีไม่มีชนิดข้อมูลโดยตรงสำหรับข้อความ การแทนข้อความจึงใช้ตัวแปรแบบแถวลำดับของอักขระแทน และเข้าถึงข้อมูลอักขระแต่ละตัวได้ในลักษณะเดียวกับตัวแปรแถวลำดับอื่นๆ แต่เนื่องจากข้อมูลที่เป็นข้อความนั้นมีความยาวไม่คงที่ เช่น ชื่อคน อาจยาว 3-10 ตัวอักษร การจองพื้นที่ให้พอดีกับข้อมูลเข้านั้นเป็นไปได้ยาก ส่วนการแสดงผลข้อความนั้น เครื่องจะวนซ้ำแสดงผลอักขระทีละตำแหน่งไปจนหมดข้อความ จึงจำเป็นต้องมีอักขระพิเศษเพื่อบ่งบอกขอบเขตของสายอักขระ ในภาษาซีใช้อักขระว่าง (null character, '\0' หรือค่า 0) แทนตัวระบุจุดสิ้นสุดของข้อความ

การจองพื้นที่สำหรับสายอักขระในภาษาซี จึงเป็นการจองพื้นที่สำหรับตัวแปรชนิดแถวลำดับของอักขระ โดยกำหนดจำนวนช่องเป็นจำนวนอักขระที่ต้องการ บวกหนึ่งช่องสำหรับอักขระระบุจุดสิ้นสุด

**ตัวอย่าง 7.3** (การแปลงข้อความทั้งหมดให้เป็นตัวพิมพ์ใหญ่). บางครั้งเราต้องการบันทึกข้อมูลเป็นตัวพิมพ์ใหญ่ทั้งหมด เช่น ชื่อบทความตีพิมพ์ แต่ข้อมูลที่ป้อนโดยผู้ใช้อาจเป็นทั้งตัวพิมพ์ใหญ่และตัวพิมพ์เล็กผสมกัน การบังคับให้ผู้ใช้ป้อนข้อมูลเข้าเป็นตัวพิมพ์ใหญ่ทั้งหมดนั้นอาจสร้างความลำบากให้กับผู้ใช้ ผู้ทำระบบรับข้อมูลจึงทำฟังก์ชันสำหรับแปลงข้อความให้เป็นตัวพิมพ์ใหญ่ทั้งหมดแทน

การเก็บข้อมูลอักขระต่างๆ นั้นเก็บเป็นรหัสอักขระ เช่น รหัสแอสกีในตารางที่ 3.2 (หน้า 47) ซึ่งเสมือนเป็นจำนวนเต็ม หากอักขระนั้นเป็นจำนวนในช่วง [97-122] (0x61 - 0x7A) แสดงว่าเป็นตัวพิมพ์เล็ก และจะเห็นได้ว่าตัวอักษร a-z ตัวพิมพ์เล็กนั้นอยู่เรียงกัน ส่วนตัวอักษร A-Z ตัวพิมพ์ใหญ่ก็อยู่เรียงกันเช่นกัน ระยะห่างระหว่างอักขระ a-A b-B ... z-Z นั้นคงที่เท่ากับ 32 การแปลงอักขระตัวพิมพ์เล็กเป็นตัวพิมพ์ใหญ่จึงทำได้โดยตรวจสอบว่าอักขระนั้นมีรหัสอยู่ในช่วง [97-122] หรือไม่ หากใช่ให้นำรหัสนั้นลบด้วย 32 แล้วเก็บลงไปที่เดิม ทำซ้ำกับทุกอักขระในข้อความ ก็จะเป็นการแปลงสายอักขระที่มีตัวพิมพ์เล็กให้กลายเป็นตัวพิมพ์ใหญ่ทั้งหมด เขียนเป็นส่วนของโปรแกรมภาษาซีได้ดังรหัสคำสั่งที่ 7.3

```
1 char input[100];
2 scanf("%s", input);
3 int i;
4 for (i = 0; input[i] != '\0'; i++)
5     if (input[i] >= 97 && input[i] <= 122)
6         input[i] -= 32;
7 printf("%s\n", input);
```

รหัสคำสั่งที่ 7.3: การแปลงข้อความทั้งหมดให้เป็นตัวพิมพ์ใหญ่

`input` เป็นตัวแปรชนิดแถวลำดับของอักขระขนาด 100 ช่อง สามารถเก็บข้อความได้ยาว 99 อักขระ และเผื่อที่สำหรับอักขระพิเศษปิดท้ายสายอักขระ 1 ช่อง เงื่อนไขในการวนซ้ำคืออักขระ ณ ดัชนีนั้นยังไม่เป็นอักขระระบุจุดสิ้นสุด สิ่งที่ต้องทำซ้ำประกอบด้วยการแปลงอักขระนั้นให้เป็นตัวพิมพ์ใหญ่ และการเพิ่มดัชนีไปอีกหนึ่ง ซึ่งหากใช้โครงสร้าง `for` แบบดั้งเดิม การเพิ่มดัชนีไปอีกหนึ่งจะเป็นงานสุดท้ายที่ต้องทำซ้ำก่อนเริ่มรอบใหม่

□

### 7.5.2 ฟังก์ชันต่างๆ เกี่ยวกับสายอักขระในภาษาซี

เนื่องจากข้อมูลชนิดข้อความนั้นมีการใช้บ่อยครั้ง และโดยมากจะเป็นการอ่านเขียนทั้งสายอักขระ ไม่จำเพาะเจาะจงที่อักขระใดอักขระหนึ่ง การเขียนโปรแกรมด้วยการวนซ้ำสำหรับอ่านเขียนทุกครั้งไปนั้นจึงทำให้รหัสต้นฉบับหรือผังงานยาวกว่าที่ควรจะเป็น จึงมีการสร้างฟังก์ชันเฉพาะสำหรับการทำงานกับข้อความขึ้น เก็บอยู่ในคลังโปรแกรม string การเรียกใช้ฟังก์ชันต่างๆ ในคลังโปรแกรมนี้ทำได้โดยระบุ `#include <string.h>` ในตัวชี้แนะ (directive) ของโปรแกรมตอนต้นแฟ้มรหัสต้นฉบับ เพื่อให้คอมไพเลอร์เรียกใช้ฟังก์ชันเหล่านี้ได้ถูกต้อง มีฟังก์ชันที่ใช้งานบ่อยครั้ง เช่น

1. `strlen` การหาความยาวของข้อความในตัวแปรสายอักขระ เครื่องจะหาค่าดัชนีของอักขระระบุจุดสิ้นสุด แล้วคืนค่าดัชนีนั้น ซึ่งก็จะเป็นความยาวของสายอักขระด้วย
2. `strcpy` การคัดลอกข้อความในตัวแปรสายอักขระ ต้องจองพื้นที่สำหรับตัวแปรปลายทางก่อนเรียกใช้ฟังก์ชัน เมื่อเรียกใช้ ฟังก์ชันนี้จะสำเนาค่าอักขระจากตัวแปรต้นทางไปยังตัวแปรปลายทางทีละตำแหน่งจากดัชนีเริ่มต้นไปยังตำแหน่งสุดท้ายของข้อความ และคืนค่าเป็นตัวชี้ไปยังอักขระระบุจุดสิ้นสุดของตัวแปรปลายทาง
3. `strcat` การคัดลอกข้อความจากสายอักขระต้นทางไปต่อท้ายสายอักขระปลายทาง ต้องจองพื้นที่สำหรับตัวแปรปลายทางไว้ก่อนเช่นเดียวกัน ฟังก์ชันจะสำเนาอักขระแรกของตัวแปรต้นทางไปยังตำแหน่งเดิมของอักขระปิดท้ายข้อความในตัวแปรปลายทาง และสำเนาทีละตำแหน่งไปจนถึงตำแหน่งสุดท้ายของข้อความต้นทาง จากนั้นคืนค่าเป็นตัวชี้ไปยังอักขระระบุจุดสิ้นสุดของตัวแปรปลายทางหลังสำเนาข้อมูลทั้งหมด
4. `strcmp` การเปรียบเทียบข้อความในตัวแปรสายอักขระ ฟังก์ชันจะเปรียบเทียบอักขระทีละตัวตามค่าของรหัสอักขระที่ใช้ โดยตัวเลขมาก่อนตัวอักษร และตัวพิมพ์ใหญ่ทั้งหมดจะมาก่อนตัวพิมพ์เล็ก เช่นลำดับในตารางที่ 3.2 (หน้า 47)

```
size_t strlen(const char *s);
char *stpcpy(char *dest, const char *src);
char *strcat(char *dest, const char *src);
int strcmp(const char *s1, const char *s2);
```

รหัสคำสั่งที่ 7.4: ฟังก์ชันต่างๆ เกี่ยวกับสายอักขระในภาษาซี

วิธีการเรียกใช้ฟังก์ชันเกี่ยวกับสตริงเป็นดังต้นแบบในรหัสคำสั่งที่ 7.4 ฟังก์ชันเหล่านี้หากมีผลลัพธ์เป็นสายอักขระ เช่น การต่อสายอักขระ ฟังก์ชันจะคืนค่าผลลัพธ์ที่เป็นสายอักขระผ่านพารามิเตอร์ตัวแรก (`dest`) ส่วนค่าที่รับคืนจากฟังก์ชันจะเป็นตัวชี้ไปยังตำแหน่งสุดท้ายของข้อความในผลลัพธ์

## แบบฝึกหัด

จงเขียนผังงานและโปรแกรมภาษาซีเพื่อทำงานดังต่อไปนี้

1. รับข้อมูลจำนวนเต็ม 10 จำนวน จากนั้นแสดงผลข้อมูลจากลำดับสุดท้ายไปยังลำดับแรก
2. ลำดับฟีโบนักชี (Fibonacci) มีนิยามความสัมพันธ์คือ จำนวนถัดไปเท่ากับผลบวกของสองจำนวนก่อนหน้า เขียนเป็นสัญลักษณ์ได้ดังนี้

$$F_n = F_{n-1} + F_{n-2}$$

   เมื่อ $F_n$ คือเลขในลำดับฟีโบนักชีตัวที่ $n$ โดยที่ $F_0 = 0$ และ $F_1 = 1$ ให้รับค่า $n$ ซึ่งเป็นจำนวนเต็มมีค่าไม่เกิน 100 แล้วแสดงผลลัพธ์ $F_n$
3. ต้องการสร้างตารางแจกแจงความถี่ของข้อมูลคะแนนสอบของนิสิตซึ่งอยู่ในช่วง [0,100] จำนวน 100 คน โดยแบ่งข้อมูลเป็น 5 ช่วง ตั้งแต่ [0,20), [20,40), [40,60), [60,80) และ [80,100] แล้วแสดงจำนวนนิสิตที่มีคะแนนอยู่ในช่วงต่างๆ
4. ให้ผู้ใช้ป้อนข้อมูลหมายเลขโทรศัพท์ของไทยเข้ามาซึ่งเป็นตัวเลขโดด 9-10 ตัว ซึ่งอาจจะมี - คั่นกลางระหว่างตัวเลขบางตัวได้ เช่น 02-2185148 หรือ 022185148 ให้แปลงหมายเลขที่ได้นี้ให้อยู่ในรูปแบบ [เลข2-3 ตัว]-[เลข 3 ตัว]-[เลข 4 ตัว] เช่น 02-218-5148
5. รับข้อมูลเป็นสายอักขระความยาวไม่เกิน 20 ตัวอักษร แล้วตรวจสอบว่าสายอักขระนั้นเป็น palindrome หรือไม่

   palindrome คือข้อความที่อ่านจากซ้ายไปขวาหรือขวาไปซ้ายได้เป็นคำเดียวกัน เช่น anna, otto, madam กำหนดให้ข้อความมีเฉพาะอักษร a-z เท่านั้น

6. การหาค่าพื้นที่ปิดล้อมของฟังก์ชันไม่เป็นลบ $f(x)$ กับแกน $x$ ในช่วง $a, b]$ หาได้จาก $\int_a^b f(x)dx$ ซึ่งสามารถประมาณค่าได้โดยแบ่ง $[a, b]$ เป็นช่วงแคบๆ $n$ ช่วง แต่ละช่วงกว้าง $l = (b-a)/n$ แล้วคำนวณค่าจากสมการ (7.2)

$$\sum_1^n f(a + nl - \frac{l}{2}) \cdot l \qquad (7.2)$$

   กำหนดฟังก์ชันไม่เป็นลบ `float f ( float x )` มาให้ เมื่อรับข้อมูลขอบล่าง `a` และขอบบน `b` เป็นจำนวนจริง และรับจำนวนเต็ม `n` เป็นจำนวนช่วงที่ต้องการแบ่ง จงคำนวณ $\int_a^b f(x)dx$ โดยใช้วิธีการประมาณค่าข้างต้น

7. จากการประมาณในสมการ (7.2) นั้น $f(a + nl - \frac{l}{2}) \cdot l$ เป็นการคำนวณพื้นที่สี่เหลี่ยมมุมฉากที่สูงเท่ากับจุดกึ่งกลางของแต่ละช่วง เราสามารถประมาณค่าให้ใกล้เคียงขึ้นได้โดยใช้พื้นที่ของสี่เหลี่ยมด้านขนานที่มีความสูงสองจุดเท่ากับขอบล่างและขอบบนของช่วงตามลำดับ ลองคำนวณโดยปรับวิธีการประมาณค่าดู

# บทที่ 8

# การเรียกใช้ฟังก์ชัน

## วัตถุประสงค์การเรียนรู้

อธิบายการทำงานของโปรแกรมในรูปแบบการเรียกใช้ฟังก์ชัน

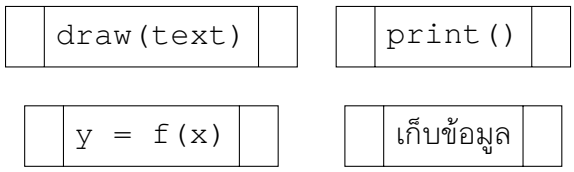
1. กำหนดพารามิเตอร์และรับค่าคืนจากฟังก์ชันได้เมื่อกำหนดต้นแบบของฟังก์ชันมาให้
2. แยกแยะขอบเขตการใช้งานของตัวแปรเมื่อมีฟังก์ชันมาให้ได้
3. เขียนฟังก์ชันเพื่อให้ทำงานตามที่ต้องการได้

## 8.1 นิยามและสัญลักษณ์

โปรแกรมย่อย (Pre-defined process หรือ subroutine) เป็นชุดคำสั่งซึ่งมีการกำหนดการทำงานไว้ล่วงหน้าแล้ว โดยมากเรามักแบ่งโปรแกรมขนาดใหญ่ออกเป็นโปรแกรมย่อยหลายๆ ส่วน เพื่อให้เข้าใจได้ง่าย บำรุงรักษาสะดวก นอกจากนี้ ส่วนของโปรแกรมใดที่ถูกเรียกใช้บ่อย ก็มักจะถูกเขียนแยกออกมาเป็นโปรแกรมย่อยด้วย การใช้สัญลักษณ์โปรแกรมย่อยแทนขั้นตอนการทำงานแบบละเอียดทำให้ผังงานกระชับ เข้าใจง่าย ในรหัสต้นฉบับเองก็เช่นกัน

โปรแกรมย่อยกับฟังก์ชันมีลักษณะเป็นชุดคำสั่งซึ่งทำงานจบในตัวเหมือนกัน แต่แตกต่างกันที่ฟังก์ชันนั้นจะสามารถคืนค่าได้เช่นเดียวกับฟังก์ชันทางคณิตศาสตร์ แต่โปรแกรมย่อยจะไม่มีการคืนค่า ภาษาโปรแกรมบางภาษา เช่น ภาษาซี ใช้โครงสร้างแบบฟังก์ชันทั้งหมด และเพิ่มลักษณะการคืนค่าชนิด `void` แทนการไม่คืนค่าเข้ามา ดังนั้นบางครั้งอาจใช้คำว่าฟังก์ชันและโปรแกรมย่อยแทนกันได้โดยไม่ผิดความหมาย

ผังงานใช้สัญลักษณ์สี่เหลี่ยมมุมฉากซึ่งมีขอบข้างเป็นเส้นขนานแทนการเรียกใช้โปรแกรมย่อย และมีชื่อโปรแกรมย่อยกำกับในสัญลักษณ์ ซึ่งจะอ้างอิงไปถึงการทำงานอย่างใดอย่างหนึ่งซึ่งกำหนดไว้ในผังงานอื่นหรือผังงานหน้าอื่น ดังตัวอย่างในรูปที่ 8.1




รูปที่ 8.1: สัญลักษณ์การเรียกใช้โปรแกรมย่อยหรือฟังก์ชัน

เราอาจระบุค่าต่างๆ ที่ต้องการส่งให้โปรแกรมย่อยหรือฟังก์ชัน และตัวแปรสำหรับรับค่าคืนลงในผังงานด้วยก็ได้ หรือไม่ระบุก็ได้ เพื่อให้ผังงานแสดงขั้นตอนการทำงานที่ชัดเจน ควรระบุค่าที่รับส่ง

สัญลักษณ์ของโปรแกรมย่อยในรหัสต้นฉบับขึ้นกับภาษาที่ใช้ โดยมากจะเป็นการประกาศหรือนิยามชื่อก่อนใช้งาน การอ้างอิงถึงโปรแกรมย่อยหรือฟังก์ชันทำได้ผ่านชื่อของโปรแกรมย่อยหรือฟังก์ชันนั้น ในกรณีที่มีการคืนค่า การเรียกใช้ฟังก์ชันนั้นสามารถมองเสมือนเป็นนิพจน์ซึ่งให้ผลลัพธ์เป็นค่าในชนิดที่กำหนดได้

## 8.2 โครงสร้างของฟังก์ชันและการเรียกใช้

ฟังก์ชันประกอบด้วยองค์ประกอบหลัก 4 ส่วน ได้แก่

1. ชื่อของฟังก์ชัน มีข้อกำหนดในการตั้งเช่นเดียวกับตัวแปร ในบางภาษาจะยอมให้ตั้งชื่อฟังก์ชันซ้ำกันได้หากมีพารามิเตอร์ที่ต่างกัน เครื่องจะเลือกฟังก์ชันที่ถูกต้องจากลักษณะของพารามิเตอร์ที่ป้อนเข้ามาให้เอง บางภาษาเช่นไพธอนยอมให้ตั้งชื่อซ้ำได้ โดยจะเป็นการนิยามชื่อใหม่ตามการประกาศล่าสุดที่มี อย่างไรก็ตาม ควรหลีกเลี่ยงการใช้ชื่อซ้ำเพื่อป้องกันความสับสนเมื่อเรียกใช้
2. พารามิเตอร์ คือตัวแปรสำหรับรับค่าที่ส่งเข้ามาให้ใช้ในฟังก์ชัน จะมีหรือไม่มีก็ได้
3. ชนิดของค่าที่ส่งคืนให้ตัวเรียก หรือไม่มีหากไม่ต้องการคืนค่าใดๆ
4. ชุดคำสั่งสำหรับขั้นตอนการทำงานของฟังก์ชัน และคำสั่งสำหรับคืนค่า ตัวชุดคำสั่งนี้คือสิ่งที่นำไปเขียนเป็นผังงานของฟังก์ชัน

หากเป็นโปรแกรมย่อย ก็จะตัดส่วนการคืนค่าออกไปได้ การประกาศฟังก์ชันโดยทั่วไปมีโครงสร้างดังรหัสคำสั่งที่ 8.1

```
ชนิดของตัวแปรคืนค่า ชื่อฟังก์ชัน ( ตัวแปรรับค่า,  ตัวแปรรับค่า, ...)
    ชุดคำสั่ง
    return ค่าส่งคืน
```

รหัสคำสั่งที่ 8.1: โครงสร้างของฟังก์ชัน

การเรียกใช้โปรแกรมย่อยหรือฟังก์ชันทำโดยการระบุชื่อโปรแกรมย่อยนั้นในโปรแกรมหลัก พร้อมทั้งส่งค่าต่างๆ ที่จำเป็นไปให้ ในกรณีที่เป็นฟังก์ชันและมีค่าคืนกลับมา หากต้องการเก็บค่าไว้ใช้งาน ก็สามารถตั้งตัวแปรมาแล้วกำหนดค่าให้เป็นผลลัพธ์ของฟังก์ชันได้

ต้นแบบ (signature, prototype) ของฟังก์ชัน ประกอบด้วย ชื่อฟังก์ชัน ชนิดของตัวแปรรับค่าต่างๆ ซึ่งเป็นพารามิเตอร์ของฟังก์ชัน และชนิดของตัวแปรคืนค่า การเรียกใช้ฟังก์ชันนั้นไม่จำเป็นต้องรู้ขั้นตอนการทำงานภายในฟังก์ชัน เพียงรู้ต้นแบบก็จะสามารถเรียกใช้ฟังก์ชันเหล่านั้นได้ถูกต้อง

**ตัวอย่าง 8.1** (การเรียกใช้ฟังก์ชันการหารากที่สอง). ฟังก์ชันการหารากที่สอง มีต้นแบบดังนี้

```
double sqrt(double x);
```

แสดงว่าฟังก์ชันหารากที่สองชื่อ `sqrt` รับค่าเป็นจำนวนจริงนัยสำคัญสูงหนึ่งค่า และคืนค่าผลลัพธ์เป็นจำนวนจริงนัยสำคัญสูงเช่นกัน เมื่อเขียนคำสั่ง

```
double x;
x = sqrt(10.5);
```

จะมีการทำงานดังนี้

1. ประกาศตัวแปร `x` ให้เป็นจำนวนจริงนัยสำคัญสูง
2. เรียกใช้ฟังก์ชัน `sqrt` โดยส่งจำนวน 10.5 ให้
3. ได้ผลลัพธ์กลับมา แล้วกำหนดค่าให้ `x` เท่ากับผลลัพธ์นั้น

หากสั่ง

```
printf("%f",sqrt(2.5));
```

จะเป็นการเรียกใช้ฟังก์ชัน `sqrt` และป้อนจำนวน 2.5 ให้ฟังก์ชัน และเมื่อได้ผลลัพธ์กลับมาแล้ว ใช้ผลลัพธ์นั้นเป็นข้อมูลเข้าส่งต่อให้ฟังก์ชัน `printf` แต่ไม่เก็บผลลัพธ์ไว้ใช้งานอีก □

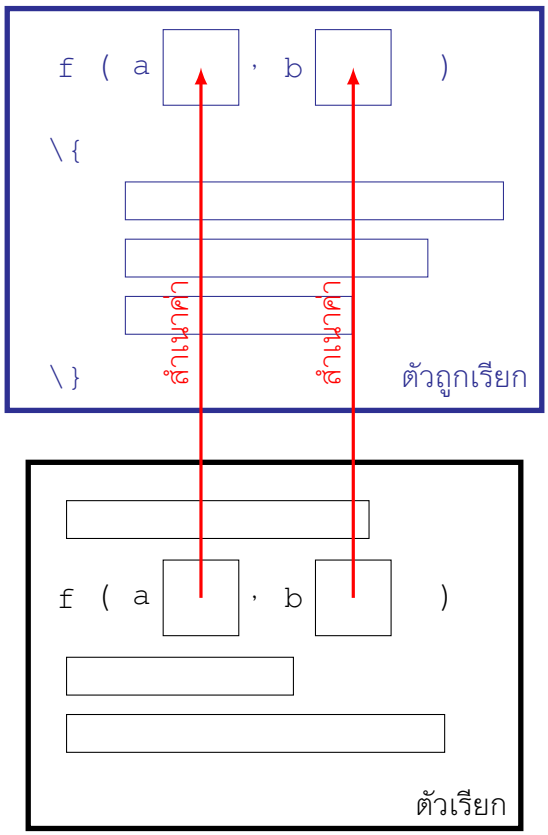
การเขียนโปรแกรมย่อยหรือฟังก์ชันนั้นเหมือนกับการออกแบบโปรแกรมปกติ คือ

1. ระบุข้อมูลเข้า ซึ่งอาจรับมาจากโปรแกรมหลักผ่านวิธีการส่งค่าแบบต่างๆ หรือรับจากผู้ใช้
2. ระบุข้อมูลออกถ้ามี อยู่ในรูปของการคืนค่าแบบต่างๆ
3. ออกแบบขั้นตอนการทำงานของโปรแกรม

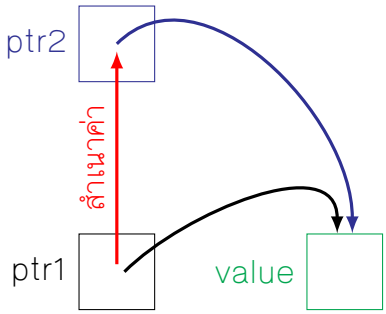
ดังนั้น เราจึงสามารถเขียนผังงานย่อยสำหรับโปรแกรมย่อยหรือฟังก์ชันได้เช่นเดียวกับโปรแกรมทั่วไป

## 8.3 การรับส่งค่า

การรับส่งค่าให้ฟังก์ชันหรือโปรแกรมย่อยมีสองรูปแบบ คือ ผ่านพารามิเตอร์ของฟังก์ชัน ซึ่งเป็นการสำเนาค่าจากตัวเรียก ส่งไปให้ตัวถูกเรียก และผ่านตัวแปรร่วมที่ประกาศไว้สำหรับทั้งโปรแกรม




(ก) การส่งค่าผ่านพารามิเตอร์




(ข) การส่งค่าผ่านตัวแปรชนิดตัวชี้

### 8.3.1 พารามิเตอร์

พารามิเตอร์ (formal parameter) เป็นตัวแปรรับค่าของฟังก์ชันหรือโปรแกรมย่อย ซึ่งตัวเรียกส่งให้ในการเรียกใช้แต่ละครั้ง การส่งค่าผ่านพารามิเตอร์เป็นการ**สำเนาค่า**จริงที่อยู่ในตัวเรียก (actual parameter) ไปให้ตัวแปรรับค่าในตัวถูกเรียก ดังรูปที่ 8.2(ก) การส่งค่าผ่านพารามิเตอร์ยังแบ่งออกเป็นสองรูปแบบ คือการสำเนาค่าทั่วๆไป เช่น จำนวนเต็ม จำนวนจริง และอักขระ กับการสำเนาค่าตัวชี้ซึ่งเป็นตำแหน่งที่อยู่ในหน่วยความจำส่งไปยังตัวถูกเรียก การสำเนาค่าทั่วๆ ไปไม่ทำให้ค่าที่มีอยู่ในตัวแปรของตัวเรียกเปลี่ยนแปลงไป แต่ในกรณีที่เป็นตัวแปรชนิดตัวชี้ เช่นตัวแปร ptr1 ในรูปที่ 8.2(ข) ซึ่งเก็บตำแหน่งที่อยู่ของค่าในตัวแปร value (ชี้ไปยัง value) เมื่อสำนาเค่าไปยังตัวแปร ptr2 ถึงแม้ค่าในตัวชี้จะถูกสำเนาไปเช่นเดียวกับการสำเนาค่าปกติ แต่ค่านั้นคือตำแหน่งที่อยู่ของข้อมูลจริง การเปลี่ยนแปลงค่าต่างๆ โดยอาศัยที่อยู่ที่กำหนดในตัวชี้ เช่น

```
*ptr2 = 10
```

จะทำให้ตัวชี้อื่นๆ (ในที่นี้คือ ptr1) เห็นค่าที่เปลี่ยนแปลงตามไปด้วย จึงสามารถใช้วิธีนี้ในการคืนค่าโดยไม่ต้องผ่านตัวแปรคืนค่า แต่อาศัยการกำหนดค่าใหม่ให้ตำแหน่งข้อมูลระบุโดยตัวแปรชนิดตัวชี้ในพารามิเตอร์แทน เช่นในกรณีคำสั่ง `scanf` ซึ่งไม่ได้คืนค่าข้อมูลที่อ่านผ่านตัวแปรคืนค่า แต่ใช้การนำข้อมูลที่อ่านได้ไปใส่ในตำแหน่งที่ระบุด้วยตัวแปรตัวชี้แต่ละตัวแทน

จำนวนของตัวแปรรับค่าขึ้นกับผู้เขียนโปรแกรมจะกำหนด ชนิดของตัวแปรแต่ละตัวไม่จำเป็นต้องเหมือนกัน การส่งค่าในตัวเรียกจะส่งเรียงลำดับตามที่กำหนดในฟังก์ชันหรือโปรแกรมย่อย คือ ตัวที่ 1 2 3 ... ตามลำดับ

### 8.3.2 การคืนค่า

ตัวแปรคืนค่าใช้สำหรับส่งค่าบางอย่างคืนให้ตัวเรียกนำกลับไปใช้ต่อ โดยมีชนิดของตัวแปรตามที่ประกาศไว้ในฟังก์ชัน นอกจากชนิดตัวแปรพื้นฐานแล้ว ส่วนของโปรแกรมย่อยอาจไม่คืนค่าใดๆเลยก็ได้ ในภาษาซีกำหนดชนิดไม่คืนค่านี้ว่า ชนิด `void`

คำสั่ง `return` ใช้สำหรับกำหนดค่าให้ตัวแปรส่งคืนค่า และออกจากการทำงานของฟังก์ชัน หากไม่มีการคืนค่า ในกรณีที่ไม่ต้องการคืนค่า อาจใช้คำสั่ง `return` โดยไม่ระบุค่าก็ได้เช่นกัน

## 8.4 ขอบเขตของตัวแปร

ตัวแปรที่ใช้มีสองประเภท แบบแรกคือตัวแปรซึ่งใช้ร่วมกันทั้งโปรแกรม ตัวแปรชนิดนี้สามารถเรียกใช้จากส่วนใดของโปรแกรมก็ได้ ไม่ว่าจะเป็นโปรแกรมหลักหรือโปรแกรมย่อย แบบที่สองคือตัวแปรเฉพาะที่ ตัวแปรชนิดนี้จะใช้ได้เฉพาะในส่วนที่ประกาศเท่านั้น ตัวแปรทั้งหมดที่ประกาศอยู่ในส่วนของโปรแกรมย่อย หากประกาศให้เป็นตัวแปรเฉพาะที่ แม้จะเป็นชื่อเดียวกับตัวแปรที่ประกาศที่อื่น ค่าต่างๆจะใช้ได้ในส่วนของโปรแกรมย่อยนี้เท่านั้น

หากต้องการส่งค่าจากส่วนอื่นๆของโปรแกรมเข้ามาให้ส่วนของโปรแกรมย่อย การส่งค่าทำได้ผ่านตัวแปรรับค่า ตัวแปรรับค่านี้จะใช้เฉพาะในส่วนของโปรแกรมย่อย และไม่มีการเปลี่ยนแปลงค่าของตัวแปรนอกส่วนของโปรแกรมย่อย หากต้องการนำค่าที่คำนวณได้ในส่วนของโปรแกรมย่อยออกไปใช้ที่ส่วนอื่นๆ ต้องประกาศตัวแปรนั้นให้เป็นตัวแปรร่วม หรือคืนค่าที่คำนวณได้ผ่านตัวแปรคืนค่าไป

## 8.5 ลำดับการทำงานของฟังก์ชัน

การทำงานของโปรแกรมจะเริ่มที่โปรแกรมหลัก และมีการเรียกใช้โปรแกรมย่อยต่างๆ โปรแกรมย่อยเองก็สามารถเรียกโปรแกรมย่อยอื่นๆ รวมถึงเรียกใช้ตัวโปรแกรมย่อยเองได้เช่นเดียวกัน ในการวิเคราะห์การเรียกใช้โปรแกรมย่อย เราแบ่งโปรแกรมย่อยตามลักษณะการทำงานเป็น 2 ส่วน คือ ตัวเรียก (caller) และตัวถูกเรียก (callee) ทั้งตัวเรียกและตัวถูกเรียกสามารถเป็นโปรแกรมย่อยใดๆ ก็ได้ เป็นรหัสต้นฉบับชุดเดียวกันก็ได้ แต่การทำงานจะเป็นไปตามลำดับ คือ เริ่มจากตัวเรียกก่อน เมื่อตัวเรียกนั้นเรียกใช้ตัวถูกเรียก การทำงานจะข้ามไปเริ่มต้นที่จุดเริ่มต้นของตัวถูกเรียก จนกว่าจะเสร็จสิ้นการทำงานในส่วนของตัวถูกเรียกแล้ว จึงจะกลับมาทำงานส่วนที่เหลือของตัวเรียกต่อ

ตัวแปรต่างๆ ที่ประกาศภายในฟังก์ชันนั้น จะใช้ได้ในขอบเขตของการทำงานในการเรียกใช้ครั้งหนึ่งๆ เท่านั้น การเรียกใช้โปรแกรมย่อยจะเสมือนกับมีการสร้างชุดของตัวแปรเฉพาะที่สำหรับโปรแกรมย่อยนั้น

ขึ้นใหม่ โดยไม่เกี่ยวข้องกับตัวแปรเฉพาะที่อื่นๆ ที่เคยประกาศมาก่อนล่วงหน้า และเมื่อการทำงานของตัวถูกเรียกนั้นสิ้นสุด ตัวแปรเฉพาะที่ต่างๆ ของตัวถูกเรียกก็จะหมดอายุตามไปด้วย หากต้องการอ้างถึงค่าต่างๆ ที่มีอยู่ในตัวถูกเรียก ต้องส่งค่านั้นคืนให้ตัวเรียกผ่านการคืนค่าในรูปแบบต่างๆ

**ตัวอย่าง 8.2** (ฟังก์ชันแฟกทอเรียลและขั้นตอนการทำงาน). โปรแกรมในรหัสต้นฉบับที่ 8.2 ประกอบด้วยฟังก์ชัน `main` และ `factorial`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int nRound;
 5
 6 int factorial(int n)
 7 {
 8     printf("round %d, n = %d\n", nRound, n);
 9     nRound++;
10
11     if (n < 0)
12         return -1;
13     else if (n <= 2)
14         return n;
15     else
16         return n * factorial(n-1);
17 }
18
19 int main()
20 {
21     int n;
22     printf("Enter n: ");
23     scanf("%d", &n);
24
25     nRound = 1;
26
27     printf("%d! = %d\n", n, factorial(n));
28
29     return 0;
30 }
```

รหัสคำสั่งที่ 8.2: โปรแกรมแฟกทอเรียล

ข้อสังเกตที่น่าสนใจในโปรแกรมนี้มี 4 จุด ได้แก่

1. ตัวแปร `nRound` เป็นตัวแปรร่วม ประกาศที่บรรทัดที่ 4 อยู่นอกทั้งสองฟังก์ชัน ทำให้ทั้งสองฟังก์ชันสามารถใช้งานตัวแปรนี้ได้ และการเปลี่ยนแปลงค่าใดๆ ในตัวแปรนี้ด้วยฟังก์ชันหนึ่ง จะส่งผลให้ฟังก์ชันอื่นๆ เห็นค่าที่เปลี่ยนไปตามไปด้วย
   เช่น ใน `main` มีการกำหนดค่า `nRound` ให้เป็น 1 ในบรรทัดที่ 25 มื่อเรียกฟังก์ชัน `factorial` ครั้งแรกในบรรทัดที่ 27 ในฟังก์ชัน `factorial` ก็จะเห็นว่า `nRound` มีค่า 1 ตามไปด้วย
   ส่วนในฟังก์ชัน `factorial` นั้นมีการเพิ่มค่า `nRound` ไปทีละหนึ่งในแต่ละครั้งของการเรียกฟังก์ชันด้วยคำสั่งในบรรทัดที่ 9 การเรียกฟังก์ชันในรอบหลังๆ ก็จะเห็นค่า `nRound` ที่เปลี่ยนไปแล้วเช่นกัน
2. ตัวแปร `n` ประกาศอยู่ในทั้ง `main` และ `factorial` แต่ทั้งสองตัวแปรเก็บแยกจากกันโดยเด็ดขาด ความเชื่อมโยงของทั้งสองตัวแปรมีเฉพาะตอนที่ `main` สำเนาค่าใน `n` ของ `main` ไปให้ `n` ของ `factorial` ผ่านการเรียกใช้ในครั้งแรกเท่านั้น
3. ฟังก์ชันใดๆ สามารถเรียกใช้ตัวมันเองได้ เช่นในฟังก์ชัน `factorial` ก็มีการเรียกใช้ฟังก์ชัน `factorial` ด้วยเช่นกัน แต่การเก็บตัวแปรเฉพาะที่ต่างๆ ของตัวเรียกและตัวถูกเรียกนั้นแยกจากกัน ไม่ว่าชื่อตัวแปรหรือฟังก์ชันจะซ้ำกันหรือไม่ก็ตาม
4. ในรหัสคำสั่งที่ 8.2 เป็นโปรแกรมภาษาซี ประกาศฟังก์ชัน `factorial` และการทำงานไว้ก่อนการเรียกใช้ในฟังก์ชัน `main` จึงไม่จำเป็นต้องเขียนต้นแบบของฟังก์ชันก่อนการเรียกใช้อีก แต่หากสลับตำแหน่งของรหัสต้นฉบับสำหรับฟังก์ชัน `factorial` และฟังก์ชัน `main` ก่อนประกาศฟังก์ชัน `main` ต้องระบุต้นแบบของฟังก์ชัน `factorial` เอาไว้ก่อนฟังก์ชัน `main` ด้วย

□

## 8.6 คลังโปรแกรม

โปรแกรมย่อยและฟังก์ชันต่างๆ ที่ใช้บ่อยมักจะถูกรวบรวมเก็บไว้ในคลังโปรแกรม (library) เพื่อจัดให้เป็นหมวดหมู่และให้ผู้ที่ต้องการใช้สามารถดึงมาใช้ได้โดยไม่ต้องเขียนใหม่อีก การเรียกใช้ฟังก์ชันในคลังโปรแกรมทำได้โดยระบุชื่อคลังโปรแกรมที่ใช้ เพื่อให้คอมไพเลอร์หรืออินเทอร์พรีเตอร์รู้จักฟังก์ชันที่ต้องการเรียกใช้ และให้ตัวเชื่อมโยงสามารถไปดึงออบเจกต์โค้ดของฟังก์ชันเหล่านั้นมารวมเป็นโปรแกรมที่ทำงานได้

คลังโปรแกรมจึงขึ้นกับภาษาโปรแกรมที่ใช้ รวมถึงขึ้นกับระบบปฏิบัติการที่ใช้ด้วย คลังโปรแกรมที่เป็นที่นิยมนั้นจะสนับสนุนหลายภาษาและระบบปฏิบัติการ เช่น OpenGL (**Open** **G**raphic **L**ibrary) เป็นคลังโปรแกรมสำหรับงานกราฟิก สนับสนุนทั้ง Windows, Linux, OS X และมีส่วนต่อประสานโปรแกรม (API) ในหลากหลายภาษา เช่น ซี จาวา

## แบบฝึกหัด

1. กำหนดฟังก์ชัน `longdiv` เป็นฟังก์ชันการหารยาวทศนิยม 1 ตำแหน่ง กำหนดให้ `dividend` เป็นตัวตั้ง และ `divisor` เป็นตัวหาร มีต้นแบบดังนี้

   ```
   float longdiv(float dividend, float divisor);
   ```

   เช่น 10/4 จะได้ผลลัพธ์เป็น 2.5
   หากต้องการหาผลลัพธ์ของ 20/3 จะต้อง**เรียกใช้**ฟังก์ชัน `longdiv` นี้อย่างไร
2. ต้องการเขียนฟังก์ชัน `contains` เพื่อตรวจสอบว่า อักขระอินพุตอยู่ในสายอักขระที่ป้อนเข้ามาหรือไม่ โดยให้ผลลัพธ์เป็นจำนวนเต็ม 1 หากพบ หรือ 0 หากไม่พบ เช่น ตรวจสอบว่าอักขระ 'a' อยู่ในสายอักขระ "contains" หรือไม่ ฟังก์ชันนี้ควรมีต้นแบบเป็นอย่างไร
3. จากแบบฝึกหัด Fibonacci ในบทที่ 7 จงเขียนฟังก์ชันการคำนวณเลขฟีโบนักชีลำดับที่ $n$ เมื่อฟังก์ชันการหาค่าในลำดับฟีโบนักชีคือ

   $$F_n = F_{n-1} + F_{n-2}$$

   กำหนด $F_0 = 0$ และ $F_1 = 1$