บทที่ 5

==========

แบบชนิดข้อมูล
(Data type)

วัตถุประสงค์

1) เพื่อให้ผู้เรียนเข้าใจถึงความสำคัญของข้อมูลที่มีต่อกระบวนการทำงานของโปรแกรมและสามารถอธิบายคุณสมบัติของข้อมูลได้
2) เพื่อให้ผู้เรียนเข้าใจโครงสร้างและการใช้งานชนิดข้อมูลแบบสเกลาร์
3) เพื่อให้ผู้เรียนเข้าใจโครงสร้างและการใช้งานชนิดข้อมูลแบบคอมโพสิต
4) เพื่อให้ผู้เรียนเข้าใจโครงสร้างและการใช้งานชนิดข้อมูลแบบโครงสร้าง
5) เพื่อให้ผู้เรียนเข้าใจในหลักการของชนิดข้อมูลนามธรรมและสามารถแยกความแตกต่างระหว่างชนิดข้อมูลนามธรรมและชนิดข้อมูลพื้นฐาน

การเขียนโปรแกรมเป็นการเขียนคำสั่งควบคุมการทำงาน เพื่อให้เกิดการเปลี่ยนแปลงกับข้อมูล ภาษาคอมพิวเตอร์ทั้งหลายจะมีโครงสร้างพื้นฐานที่เหมือนกันคือ การแทนข้อมูล, การปฏิบัติการกับข้อมูล, และการควบคุมลำดับการทำงานกับข้อมูล ข้อแตกต่างของแต่ละภาษาจะอยู่ที่วิธีการกำหนดรูปแบบการแทนข้อมูล การกำหนดชนิดข้อมูลพื้นฐาน วิธีการสั่งงานเพื่อเปลี่ยนแปลงข้อมูล การศึกษาเกี่ยวกับแบบชนิดข้อมูลจึงเป็นพื้นฐานสำคัญในการทำความเข้าใจเกี่ยวกับโครงสร้างของภาษาคอมพิวเตอร์ รวมถึงวิธีการทำโปรแกรมแบบต่างๆ

5.1 คุณสมบัติของข้อมูล

(Property of data)

เมื่อโปรแกรมประกาศใช้ข้อมูล คุณสมบัติหรือคุณลักษณะประกอบข้อมูลที่จะช่วยให้โปรแกรมใช้งานข้อมูลนั้นได้ ประกอบด้วย ชื่อของข้อมูล (name), ตำแหน่งในหน่วยความจำที่ใช้เก็บค่าของข้อมูลนั้น (location), ชนิดของข้อมูล (type), ค่าของข้อมูล (value), ช่วงอายุ (lifetime) และขอบเขต (scope) ของข้อมูล ดังตัวอย่างคำประกาศในภาษา C ต่อไปนี้

```
const int MAX = 30;
int n;
n = 27;
n = n+MAX;
```

ชื่อของข้อมูลคือ *MAX, n, "30", "27"* โดย *n* เป็นชื่อตัวแปร *MAX* เป็นชื่อของค่าคงที่ "*30*"และ "*27*" เป็นค่าคงที่ที่เรียกว่า literal ชื่อทั้งหมดนี้จะมีตำแหน่งที่อยู่ (address) ในหน่วยความจำซึ่งจะจัดการโดยโปรแกรมระบบ ชนิดของข้อมูลในตัวอย่างนี้เป็นชนิดเลขจำนวนเต็ม (integer) ค่าของข้อมูล *MAX* คือ 30, *n* คือ 57, "*30*" คือ 30 (หรือ 00000000 00011110 ในรูปแบบของเลขฐานสอง) และค่าของ "*27*" คือ 27 ช่วงอายุของข้อมูลจะคงอยู่ตลอดระยะการทำงานของฟังก์ชันของชุดคำสั่งนี้ ขอบเขตที่ข้อมูลเหล่านี้สามารถถูกเรียกใช้งาน จะอยู่ภายในขอบเขตของฟังก์ชันที่ชุดคำสั่งเหล่านี้บรรจุอยู่

จากคุณสมบัติต่างๆ ของข้อมูลดังกล่าวข้างต้น คุณสมบัติหรือคุณลักษณะที่มีความสำคัญและมีบทบาทมากในการตรวจสอบความถูกต้องของข้อความสั่ง คือชนิดของข้อมูล (type) เมื่อโปรแกรมเมอร์ประกาศชื่อข้อมูล และชนิดของข้อมูลเช่น int n; ชนิดข้อมูล int จะบอกถึงขนาดของเนื้อที่ในหน่วยความจำที่ต้องใช้ในการเก็บค่าของข้อมูล n (เช่น ใช้ขนาด 2 ไบต์ หรือ 16 บิต), บอกถึงช่วงของค่าที่เป็นไปได้ (เช่น ค่าของจำนวนเลขต้องไม่เกิน 65535), บอกถึงการกระทำที่สามารถกระทำกับตัวแปร n ได้ (เช่น สามารถใช้คำสั่งบวกค่า *n = n+MAX;*) ข้อกำหนดต่างๆ ที่เกี่ยวเนื่องกับชนิดข้อมูลนี้เรียกว่า ระบบชนิดข้อมูล (type system)

5.2 ระบบชนิดข้อมูล

(Type systems)

การกำหนดชนิดข้อมูล คือ การกำหนดเซตของค่าและการกำหนดปฏิบัติการที่สามารถกระทำกับค่าเหล่านั้น เช่น ชนิดข้อมูล int หมายถึงเซตของค่าที่เป็นเลขจำนวนเต็ม {..., -2, -1, 0, 1, 2, …} และเซตของปฏิบัติการ {+, -, *, /, … } ที่สามารถกระทำกับเลขจำนวนเต็ม, ชนิดข้อมูล boolean จะหมายถึงเซตของค่าตรรกะ {true, false} และการปฏิบัติการ {&&, ||, !}

ระบบชนิดข้อมูล คือ ระบบที่ทำหน้าที่เชื่อมโยงชนิดข้อมูลเข้ากับตัวแปรและข้อมูลอื่นๆ ในโปรแกรม รวมถึงการมีกลไกในการตรวจสอบความถูกต้องของชนิดข้อมูล ภาษาคอมพิวเตอร์โดยทั่วไปเช่น ภาษา C, Java จะเชื่อมโยงชนิดข้อมูลเพียงชนิดเดียวให้กับแต่ละตัวแปร เช่น ข้อความประกาศ int x, y; จะทำให้เกิดการเชื่อมโยงชนิดข้อมูล int ให้กับตัวแปร x และ ชนิดข้อมูล int ให้กับตัวแปร y โดยการเชื่อมโยงชนิดข้อมูลกับชื่อตัวแปรจะเกิดขึ้นตั้งแต่ช่วงการคอมไพล์โปรแกรม ภาษาคอมพิวเตอร์ที่มีลักษณะเช่นนี้จะเรียกว่า *ภาษาที่มีชนิดข้อมูลคงตัว* (statically typed language) แต่จะมีภาษาบางประเภท เช่นภาษา APL, LISP ที่อนุญาตให้ตัวแปรเปลี่ยนชนิดข้อมูลได้ ระหว่างการดำเนินงานของโปรแกรม ตัวอย่างเช่นคำสั่ง

```
LIST ← 10.3      1.8   0.0
```

เป็นการกำหนดตัวแปรชื่อ LIST เป็นอาร์เรย์มิติเดียว เก็บค่าตัวเลขทศนิยม สามค่า ถ้ามีคำสั่งต่อมาในโปรแกรม

```
LIST ←44
```

ตัวแปร LIST จะถูกเปลี่ยนชนิดข้อมูลเป็นตัวแปรเดี่ยวเก็บค่าเลขจำนวนเต็ม ภาษาที่มีลักษณะแบบนี้จะเรียกว่า *ภาษาที่มีชนิดข้อมูลพลวัต* (dynamically typed language)

ขณะที่โปรแกรมดำเนินงาน ข้อผิดพลาดอาจจะเกิดขึ้นได้เมื่อมีคำสั่งที่พยายามจะใช้ข้อมูลผิดชนิด เช่น ถ้าบางส่วนของภาษา C ปรากฏชุดคำสั่งดังนี้

```
union value {
        int a;
        double p;
} u;
double x=0.0;
u.a = 1;
x = x+u.p;
```

ตัวแปร u เป็นชนิดข้อมูลยูเนียน ที่ค่าเลือกเป็นได้ชนิดเดียวคือ int หรือ double ซึ่งในตัวอย่างกำหนดให้เก็บค่าเป็นชนิด int (ด้วยคำสั่ง *u.a = 1;*) แต่คำสั่งสุดท้ายในตัวอย่างเป็นการเรียกใช้ค่า u ที่เป็นชนิด double ซึ่งเป็นชนิดข้อมูลที่ผิด ข้อผิดพลาดที่เกี่ยวกับชนิดข้อมูลเช่นนี้เรียกว่า type error

ภาษาคอมพิวเตอร์ที่มีระบบชนิดข้อมูลที่สามารถตรวจพบข้อผิดพลาดทั้งหลาย ที่เกี่ยวข้องกับชนิดข้อมูลไม่ว่าจะเป็นการตรวจพบในช่วงคอมไพล์หรือในช่วงรันโปรแกรม จะเรียกภาษานั้นว่า*มีระบบชนิดข้อมูลที่เข้มแข็ง* (strongly typed) ตัวอย่างภาษาประเภทนี้ได้แก่ ภาษา Java, ภาษา Pascal การมีระบบชนิดข้อมูลที่เข้มแข็ง จะช่วยให้โปรแกรมมีความน่าเชื่อถือและมีความถูกต้องสูงขึ้น ภาษา C จัดว่ามีระบบชนิดข้อมูลที่เข้มแข็งน้อยกว่าภาษา Java (เรียกว่า*มีระบบชนิดข้อมูลอ่อนแอ* หรือ weakly typed) เนื่องจากการตรวจสอบเกี่ยวกับชนิดข้อมูลไม่เข้มงวด โปรแกรมเมอร์สามารถเขียนนิพจน์ x+1 ได้โดยไม่ต้องมีข้อบังคับว่า x ต้องเป็นชนิดข้อมูลประเภทจำนวนเลขได้ประเภทเดียวเท่านั้น (x อาจเป็นค่า boolean, pointer หรืออื่นๆ ได้)

โปรแกรมที่ปราศจากข้อผิดพลาดเกี่ยวกับชนิดข้อมูล จะเรียกว่า *มีชนิดข้อมูลที่ปลอดภัย* (type safe) ดังนั้นโดยนิยามแล้วภาษาที่มีระบบชนิดข้อมูลที่เข้มแข็ง (เช่น ภาษา Java) จะทำให้โปรแกรมที่เขียนในภาษานั้นมีชนิดข้อมูลที่ปลอดภัย เพราะข้อผิดพลาดทั้งหลายที่เกี่ยวเนื่องกับชนิดข้อมูลจะถูกตรวจพบได้โดยคอมไพเลอร์และระบบประมวลผล

<u>การตรวจสอบชนิดข้อมูล</u>

ระบบชนิดข้อมูลสร้างกลไกในการตรวจสอบชนิดข้อมูล (type checking) เพื่อเป็นหลักประกันว่าโปรแกรมที่ผ่านการตรวจสอบแล้วจะไม่มีข้อผิดพลาดใดๆ เกี่ยวกับชนิดข้อมูล ขั้นตอนการตรวจสอบชนิดข้อมูลแสดงได้ดังตัวอย่างต่อไปนี้

สมมุติให้ภาษาคอมพิวเตอร์ที่ใช้เขียนโปรแกรม มีรูปแบบคำสั่งที่คล้ายภาษา Java แต่ชนิดข้อมูลมีเพียงสองชนิดคือ int และ boolean โปรแกรมที่เขียนขึ้นเพื่อคำนวณค่าแฟคตอเรียล มีชุดคำสั่งดังนี้

```
// compute result = the factorial of integer n
void main ( ) {
      int n, i, result;
      n = 8;
      i = 1;
      result = 1;
      while (i<=n)  {
            i = i+1;
            result = result * i;
      }
}
```

ขั้นตอนการตรวจสอบชนิดข้อมูลมีดังนี้

1) ตรวจสอบว่าตัวแปรที่ประกาศใช้ทุกตัวต้องมีชื่อที่ไม่ซ้ำกัน
   ในโปรแกรมแฟคตอเรียลข้างต้น มีการประกาศตัวแปร n, i, result ซึ่งแต่ละตัวแปรใช้ชื่อที่ไม่ซ้ำกัน จึงผ่านการตรวจสอบในขั้นตอนนี้

2) ตรวจสอบชนิดข้อมูลว่าตรงตามไวยากรณ์ของภาษา

ภาษาที่ใช้กำหนดชนิดข้อมูลของตัวแปรว่าต้องเป็น int หรือ boolean ตัวแปร n, i, result ประกาศว่าเป็นชนิด int จึงผ่านการตรวจสอบในขั้นตอนนี้

3) ตัวแปรที่มีการอ้างถึงในนิพจน์ทุกนิพจน์ต้องมีการประกาศชื่อและชนิดมาก่อน
นิพจน์ในโปรแกรมได้แก่ 8, 1, i<=n, i+1, result*i มีการใช้ตัวแปร i, n, result ซึ่งมีการประกาศมาแล้ว จึงผ่านการตรวจสอบในขั้นตอนนี้

4) ชนิดข้อมูลที่เป็นผลลัพธ์จากการประมวลผลนิพจน์ต้องได้รับการตรวจสอบดังนี้
   4.1) ถ้านิพจน์เป็นตัวแปรเดี่ยวหรือค่าเดี่ยว ชนิดข้อมูลผลลัพธ์ของนิพจน์ก็จะเป็นชนิดเดียวกับตัวแปรหรือค่านั้นๆ
   4.2) ถ้านิพจน์มีการใช้เครื่องหมายคำนวณทางคณิตศาสตร์ (+, -, *, /) เทอมทุกเทอมต้องเป็นชนิด int และผลลัพธ์จากการคำนวณจะเป็นชนิด int (ภาษาสมมุตินี้มีชนิดข้อมูลเพียงสองชนิดคือ int และ boolean)
   4.3) ถ้านิพจน์มีการใช้เครื่องหมายเปรียบเทียบค่า (<, <=, >, >=, = =, !=) เทอมทุกเทอมต้องเป็นชนิด int และผลลัพธ์ของนิพจน์เปรียบเทียบเป็นชนิด boolean
   4.4) ถ้านิพจน์มีการใช้เครื่องหมายตรรกะ (&&, ||, !) เทอมทุกเทอมต้องเป็นชนิด boolean และผลลัพธ์จะเป็นชนิด boolean

   นิพจน์เดี่ยวในโปรแกรมตัวอย่างประกอบด้วยตัวเลข 8, 1 เป็นชนิด int, นิพจน์ i<=n ทั้ง i และ n เป็นชนิด int, นิพจน์ i+1 ทั้ง i และ 1 เป็นชนิด int, และนิพจน์ result*i ทั้ง result และ i เป็นชนิด int ดังนั้นชนิดข้อมูลทั้งหมดถูกต้อง

5) ข้อความสั่งกำหนดค่าต้องมีการตรวจสอบชนิดข้อมูลของตัวแปรปลายทาง (ตัวแปรทางซ้ายมือของเครื่องหมาย =) ว่าตรงกันกับชนิดข้อมูลของนิพจน์ต้นทาง (นิพจน์ทางขวามือของเครื่องหมาย =)

   ข้อความสั่ง *n = 8;* ชนิดข้อมูลของตัวแปรและของนิพจน์เป็น int ตรงกัน
   ข้อความสั่ง *i = 1;* ชนิดข้อมูลเป็น int ตรงกัน
   ข้อความสั่ง *result = 1;* ชนิดข้อมูลเป็น int ตรงกัน
   ข้อความสั่ง *i = i+1;* ชนิดข้อมูลเป็น int ตรงกัน
   ข้อความสั่ง *result = result * int;* ชนิดข้อมูลเป็น int ตรงกัน
   ดังนั้นสรุปผลการตรวจสอบข้อความสั่งกำหนดค่าได้ว่ามีชนิดข้อมูลที่ถูกต้อง

6) ข้อความสั่งทำงานแบบมีเงื่อนไขและสั่งวนรอบ นิพจน์ในคำสั่งต้องมีการตรวจสอบว่าเป็นชนิดข้อมูล boolean

   นิพจน์ i<=n ในคำสั่ง while ให้ผลลัพธ์เป็นชนิด boolean ดังนั้นคำสั่ง while มีชนิดข้อมูลที่ถูกต้อง

เมื่อโปรแกรมผ่านขั้นตอนการตรวจสอบทั้งหมด จึงสรุปได้ว่าโปรแกรมแฟคตอเรียลไม่มีข้อผิดพลาดใดเกี่ยวกับชนิดข้อมูลสามารถรับการประมวลผลในขั้นต่อไปได้ ถ้าหากโปรแกรมมีการใช้คำสั่ง

ผิดพลาด เช่นประกาศตัวแปร n เป็นชนิด boolean ระบบชนิดข้อมูลจะตรวจพบข้อผิดพลาดนี้ และแจ้งข้อความเตือนโปรแกรมเมอร์ให้มีการแก้ไขโปรแกรม

การตรวจสอบชนิดข้อมูลนี้อาจจะเกิดขึ้นในช่วงการคอมไพล์โปรแกรม (เรียกว่า static type checking) หรือ อาจจะเกิดขึ้นในช่วงรันโปรแกรม (เรียกว่า dynamic type checking)

ภาษาคอมพิวเตอร์ที่ไม่บังคับให้ต้องมีการประกาศชนิดของตัวแปรล่วงหน้า เช่น ภาษา APL, LISP, PROLOG การตรวจสอบชนิดข้อมูลจะเป็นแบบ dynamic type checking เนื่องจากจะทราบชนิดของข้อมูลได้ก็ต่อเมื่อมีการรันโปรแกรมและผู้ใช้ป้อนข้อมูลเข้ามา ซึ่งผลเสียของวิธี dynamic type checking ก็คือโปรแกรมจะทำงานได้ช้าลงเพราะเสียเวลาส่วนหนึ่งไปกับการตรวจสอบชนิดข้อมูล (ต่างจากภาษาที่บังคับให้ประกาศชนิดข้อมูลของตัวแปรก่อนการใช้งาน เช่น ภาษา C, Java, Pascal การตรวจสอบชนิดข้อมูลจะทำได้ตั้งแต่ช่วงคอมไพล์โปรแกรม ทำให้ช่วงรันโปรแกรมทำงานได้เร็ว) แต่ข้อดีของการไม่ต้องประกาศชนิดของตัวแปร ก็คือ สามารถใช้งานตัวแปรได้อย่างยืดหยุ่น เพราะตัวแปรจะเก็บค่าประเภทใดก็ได้ขึ้นอยู่กับการป้อนข้อมูลเข้ามาขณะรันโปรแกรม

<u>ความเข้ากันได้ของชนิดข้อมูล</u>

ภาษาคอมพิวเตอร์โดยทั่วไปมักจะไม่เข้มงวดกับชนิดของตัวแปรมากจนเกินไปเพราะจะช่วยให้การเขียนโปรแกรมทำได้ง่ายขึ้น (นั่นคือมีคุณสมบัติ writability แต่จะไปลดคุณสมบัติ reliability) ตัวแปรที่มีชนิดคล้ายกันมักจะอนุโลมให้ใช้งานในคำสั่งร่วมกันได้ ในกรณีที่มีการยืดหยุ่นเช่นนี้ ระบบชนิดข้อมูลจะมีกลไกการตรวจสอบเพิ่มขึ้น นั่นคือ ต้องตรวจสอบความเข้ากันได้ของชนิดข้อมูล (type compatibility) หรือบางครั้งเรียกว่า ความเท่าเทียมกันของชนิดข้อมูล (type equivalence)

ความเข้ากันได้ของชนิดข้อมูล จำแนกย่อยออกเป็นสองประเภท คือ *ความเข้ากันได้โดยชื่อ* (name compatibility) และ*ความเข้ากันได้โดยโครงสร้าง* (structural compatibility)

ความเข้ากันได้โดยชื่อ หมายถึง ชนิดข้อมูลที่มีชื่อตรงกัน เช่น *int x, y;* *x* และ *y* ถือเป็นชนิดข้อมูลที่เข้ากันได้โดยชื่อเพราะมีชนิดข้อมูลเป็น int เหมือนกัน **ความเข้ากันได้โดยโครงสร้าง** หมายถึง ชนิดข้อมูลที่มีชื่อตรงกันหรือมีโครงสร้างที่เหมือนกัน ตัวอย่างต่อไปนี้แสดงลักษณะความเข้ากันได้ทั้งสองประเภท

```
type t1 is struct {
      int y;
      int w;
};
type t2 is struct {
      int y;
      int w;
};
type t3 is struct {
      int y;
};
t3 func (t1 z)
{…
   }
…
t1  a,x;
t2  b;
t3  c;
int d;
…
a = b;           /* (1) */
x = a;           /* (2) */
c = func (b);    /* (3) */
d = func (a);    /* (4) */
...
```

จากตัวอย่างโปรแกรมในภาษาสมมุติข้างต้น ถ้าภาษานี้ใช้ระบบชนิดข้อมูลที่อนุญาตเฉพาะชนิดข้อมูลที่<u>เข้ากันได้โดยชื่อ</u>

- คำสั่งในบรรทัด (1) ถือว่าผิดพลาดเกี่ยวกับชนิดข้อมูล (type error) เพราะ a เป็นชนิด t1 แต่ b เป็นชนิด t2 (ถึงแม้ t1 และ t2 จะมีโครงสร้างเหมือนกันก็ตาม)
- คำสั่งในบรรทัด (2) ถูกต้องเพราะ x และ a เป็นชนิด t1 เหมือนกัน
- คำสั่งในบรรทัด (3) ผิดเพราะฟังก์ชัน func ประกาศว่ารับพารามิเตอร์ชนิด t1 แต่คำสั่งนี้ส่งข้อมูล b ที่เป็นชนิด t2
- คำสั่งในบรรทัด (4) ผิดเพราะฟังก์ชัน func ส่งค่ากลับเป็นชนิด t3 แต่คำสั่งนี้ใช้ตัวแปร d ที่เป็นชนิด int รับค่าจากฟังก์ชัน

ดังนั้นถ้าใช้หลักการความเข้ากันได้โดยชื่อ จะมีเฉพาะคำสั่งในบรรทัด (2) เท่านั้นที่ชนิดข้อมูลถูกต้อง

ถ้าภาษามีความยืดหยุ่นมากขึ้นด้วยการใช้ลักษณะความ<u>เข้ากันได้โดยโครงสร้าง</u> ในตัวอย่างโปรแกรมข้างต้น คำสั่งในบรรทัด (1), (2), (3) เขียนได้ถูกต้องเพราะ t1 และ t2 ถือว่าเป็นชนิดข้อมูลที่เข้ากันได้เพราะมีโครงสร้างภายในเป็น `int y; int w;` เหมือนกัน แต่คำสั่งในบรรทัด (4) มีความผิดพลาดเกี่ยวกับชนิดข้อมูลเพราะฟังก์ชัน func ส่งค่ากลับเป็นชนิด t3 แต่ d เป็น int ซึ่งถือว่าโครงสร้างแตกต่างกัน

ภาษา Pascal กำหนดความเข้ากันได้เป็นแบบเข้ากันได้โดยชื่อ ภาษา C ใช้ลักษณะความเข้ากันได้โดยโครงสร้าง(แต่ยกเว้นข้อมูลชนิด struct ที่ภาษา C ใช้ลักษณะความเข้ากันได้โดยชื่อ)

<u>การเปลี่ยนชนิดข้อมูล</u>

ในบางครั้งรูปแบบคำสั่งต้องการชนิดข้อมูล T1 แต่ข้อมูลที่เกิดขึ้นจริงเป็นชนิด T2 ถ้า T1และ T2 เป็นชนิดข้อมูลที่เข้ากันได้โปรแกรมก็สามารถทำงานได้ แต่ถ้า T1และ T2 เป็นชนิดข้อมูลที่ต่างกัน อาจต้องใช้การเปลี่ยนชนิดข้อมูล (type conversion)

การเปลี่ยนชนิดข้อมูลมีได้สองรูปแบบ รูปแบบแรกเรียกว่า*โคเอิร์ชชัน* (coercion) เป็นการเปลี่ยนชนิดข้อมูลให้โดยอัตโนมัติโดยคอมไพเลอร์ (เรียกว่า implicit conversion) ตัวอย่างเช่น คำสั่งภาษา C ต่อไปนี้

```
int x;
float z;
x = x+z;
```

ตัวแปร x และ z มีชนิดข้อมูลที่ต่างกัน คอมไพเลอร์จะเปลี่ยนชนิดข้อมูล x ให้เป็น float และเมื่อบวกค่าเสร็จ ผลลัพธ์ที่เกิดขึ้นจะถูกเปลี่ยนกลับเป็น int ดังนั้นคำสั่ง *x = x+z;* ข้างต้น จะถูกคอมไพเลอร์แปลงเป็น

x = (int) ((float)x + z);

การเปลี่ยนชนิดข้อมูลรูปแบบที่สองเรียกว่า *คาสติ้ง* (casting) เป็นการบังคับเปลี่ยนชนิดข้อมูลโดยโปรแกรมเมอร์ (เรียกว่า explicit conversion) จากตัวอย่างคำสั่ง *x = x+z;* ข้างต้น โปรแกรมเมอร์อาจบังคับเปลี่ยนชนิดข้อมูลของ z แทนที่จะเป็นการเปลี่ยนชนิดข้อมูลของ x ได้ดังนี้

x = x + (int) z;

การเปลี่ยนชนิดข้อมูลจากขนาดเล็กไปสู่ขนาดที่ใหญ่ขึ้น เช่น จาก int ไปเป็น float หรือจาก short int เป็น long int เรียกว่า *การทำให้กว้างขึ้น* (widening) จะไม่มีผลกระทบต่อความถูกต้องของค่าข้อมูล ส่วนการเปลี่ยนชนิดข้อมูลจากขนาดใหญ่สู่ขนาดเล็ก เช่นจาก float ไปเป็น int จะเรียกว่า *การทำให้แคบลง* (narrowing) อาจจะมีผลกระทบต่อค่าของข้อมูล เช่น 1.5 อาจถูกเปลี่ยนค่าเป็น 1 หรือ 2

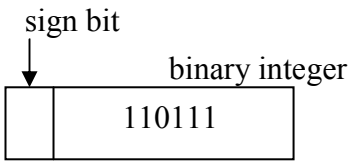
5.3 ชนิดข้อมูลแบบสเกลาร์
(Scalar data type)

ชนิดข้อมูลแบบสเกลาร์เป็นชนิดข้อมูลพื้นฐาน (primitive data type) ที่เก็บค่าเดี่ยว เช่น จำนวนเลข, อักขระ, ค่าตรรกะ ชนิดข้อมูลพื้นฐานแบบนี้ภาษาคอมพิวเตอร์ส่วนมากจะออกแบบเตรียมไว้ให้ผู้ใช้เรียกใช้ได้ทันที (เรียกว่าเป็น built-in) โดยผู้ใช้ไม่ต้องเขียนคำสั่งสร้างขึ้นใหม่ เช่น ชนิดข้อมูล int, float, char ในภาษา C

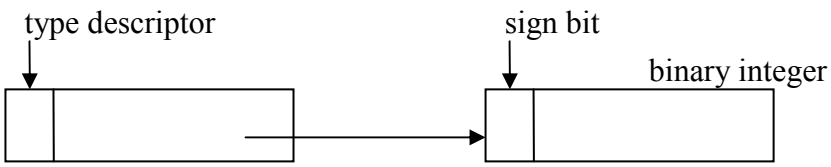
<u>ชนิดข้อมูลเลขจำนวนเต็ม</u> (integer)

เป็นชนิดข้อมูลที่มีค่าเรียงลำดับจากค่าจำนวนเต็มลบไปถึงค่าจำนวนเต็มบวก การปฏิบัติการที่สามารถกระทำกับข้อมูลประกอบด้วย การคำนวณค่าทางคณิตศาสตร์ได้แก่ การบวก, ลบ, คูณ, หาร, มอดูโล, นิเสธ การเปรียบเทียบค่าได้แก่ น้อยกว่า, มากกว่า, น้อยกว่าหรือเท่ากับ, มากกว่าหรือเท่ากับ, เท่ากับ, ไม่เท่ากับ

การเก็บข้อมูลเลขจำนวนเต็มในคอมพิวเตอร์ นิยมใช้วิธีการสองรูปแบบ (รูปที่ 5.1) รูปแบบแรก (รูปที่ 5.1 a) เป็นการเก็บค่าโดยไม่ต้องมีคำอธิบายเกี่ยวกับชนิดของค่า เป็นวิธีที่ใช้ทั่วไปในภาษาที่บังคับให้ต้องประกาศชนิดของตัวแปรล่วงหน้าและใช้วิธีการตรวจสอบชนิดข้อมูลแบบ static type checking เช่น ภาษา C, Java รูปแบบที่สอง (รูปที่ 5.1 b) มีการเก็บคำอธิบายเกี่ยวกับชนิดของค่าแยกจากการเก็บค่าของข้อมูล เป็นวิธีการที่ใช้ในภาษาที่เป็น dynamic type checking เช่น ภาษา LISP



(a) No descriptor



(b) Descriptor stored in separate word

รูปที่ 5.1 รูปแบบการจัดเก็บข้อมูลเลขจำนวนเต็มในคอมพิวเตอร์

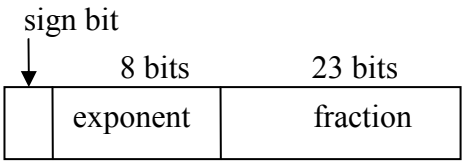
ในภาษาคอมพิวเตอร์บางภาษา เช่น Pascal, Ada อนุญาตให้ประกาศชนิดข้อมูลเลขจำนวนเต็มที่มีขอบเขตของค่าอยู่ภายในช่วงแคบๆ เช่น 1 ถึง 10 หรือ -5 ถึง 50 ชนิดข้อมูลที่เป็นช่วงแคบๆ นี้เรียกว่า subrange หรือ subtype ซึ่งมีรูปแบบคำประกาศ (ภาษา Pascal) ดังนี้

```
type natural = 0 .. maxint;
digit = 0 .. 9;
small = -9 .. 9;
days_of_month = 1..31;
```

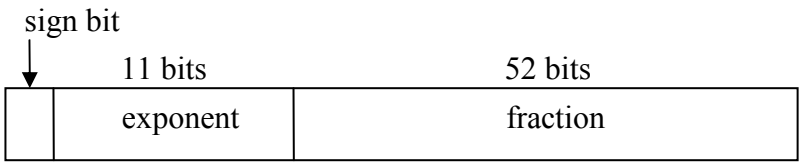
การใช้ชนิดข้อมูลที่กำหนดช่วงของค่าเช่นนี้ จะช่วยให้การตรวจสอบความถูกต้องของโปรแกรมทำได้ดียิ่งขึ้น เช่น ถ้าค่าปัจจุบันของ d ที่เป็นชนิด days_of_month มีค่า 31 และในโปรแกรมมีการใช้คำสั่ง `d := d+1;` คอมไพเลอร์สามารถแสดงข้อความเตือนได้ว่าค่าของ d เกินขอบเขตของช่วงค่าที่เป็นไปได้

<u>ชนิดข้อมูลเลขจำนวนจริง</u> (floating – point)

การจัดเก็บข้อมูลที่เป็นชนิดเลขจำนวนจริงมีมาตรฐานตามที่กำหนดโดย IEEE แยกเป็นสองกลุ่ม คือ กลุ่มที่มีความเที่ยงตรงต่ำ หรือเรียกว่า single precision (รูปที่ 5.2a) และกลุ่มที่มีความเที่ยงตรงสูง หรือเรียกว่า double precision (รูปที่ 5.2b) ซึ่งในภาษา C จะเป็นชนิดข้อมูลที่เรียกว่า float และ double ตามลำดับ



(a) single precision



(b) double precision

รูปที่ 5.2 การจัดเก็บชนิดข้อมูลเลขจำนวนจริงตามมาตรฐาน IEEE 754

ภาษาคอมพิวเตอร์ที่ใช้ในงานธุรกิจ เช่น ภาษา COBOL, PL/I มีการใช้ชนิดข้อมูลเลขจำนวนจริงประเภทที่เป็นเลขทศนิยม เรียกว่า decimal หรือ fixed – points เช่น ตัวอย่างในภาษา COBOL

```
X     PICTURE     999V99.
```

เป็นการประกาศค่าตัวแปร X ให้สามารถเก็บค่าตัวเลขทศนิยมที่มีตัวเลขหน้าจุดทศนิยมสามหลัก และตัวเลขหลังจุดทศนิยมสองหลัก (เช่น X = 115.34)

<u>ชนิดข้อมูลตรรกะ</u> (Boolean)

ชนิดข้อมูลตรรกะมีค่าเป็นไปได้สองค่าคือ จริง (true) และ เท็จ (false) และตัวแปรที่เป็นชนิดตรรกะ สามารถนำมาปฏิบัติการทางตรรกะได้ ซึ่งได้แก่ การกระทำและ (and), หรือ (or), นิเสธ (not)

การแทนข้อมูลตรรกะในคอมพิวเตอร์ โดยหลักการแล้วใช้เนื้อที่เพียงหนึ่งบิตเพื่อแทนค่า true/false ก็เพียงพอ แต่การอ้างอิงหน่วยความจำของระบบคอมพิวเตอร์จะอ้างอิงคราวละอย่างต่ำหนึ่งไบต์ จึงนิยมแทนข้อมูลตรรกะในสองลักษณะคือ ใช้เพียงส่วน sign bit แทนข้อมูลตรรกะโดยไม่ต้องพิจารณาส่วนที่เหลือของไบต์, หรือใช้เนื้อที่ทั้งไบต์โดยให้ค่าศูนย์แทน false และค่าอื่นที่ไม่ใช่ศูนย์แทน true (ลักษณะนี้ใช้ในภาษา C)

<u>ชนิดข้อมูลอักขระ</u> (character)

ภาษาคอมพิวเตอร์โดยทั่วไปแทนข้อมูลอักขระด้วยรหัสแอสกี (ASCII) ที่ใช้เนื้อที่หน่วยความจำหนึ่งไบต์ (ภาษา Java ใช้รหัส Unicode ใช้เนื้อที่สองไบต์ครอบคลุมรหัสแอสกีทั้งหมด ผนวกด้วยรหัสที่แทนภาษานานาชาติเพิ่มเติม) รหัสแอสกีใช้เป็นตัวเลขค่า 0 ถึง 127 เพื่อแทนอักขระต่างๆ 128 อักขระ ดังนั้นข้อมูลอักขระจึงสามารถนำมาเปรียบเทียบลำดับก่อน-หลังได้ เช่น ‘A’ < ‘B’

<u>ชนิดข้อมูลแจกแจงค่า</u> (enumeration)

ชนิดข้อมูลแจกแจงค่าเป็นชนิดข้อมูลที่เป็นค่าเดี่ยวกำหนดได้โดยโปรแกรมเมอร์ด้วยการระบุชื่อชนิดข้อมูล และค่าต่างๆ ที่เป็นไปได้ ตัวอย่างเช่นในภาษา C คำสั่ง

enum Class {Freshman, Sophomore, Junior, Senior};
enum Class studentClass;

เป็นการระบุชนิดข้อมูลแจกแจงค่าชื่อ Class โดยมีค่าเรียงตามลำดับจาก Freshman ไปถึง Senior จึงทำให้ใช้คำสั่งเปรียบเทียบค่าได้ เช่น Junior < Senior การกำหนดชนิดข้อมูลขึ้นใช้ตามความต้องการของโปรแกรมเมอร์เช่นนี้ จะช่วยให้การเขียนคำสั่งในโปรแกรมสื่อความหมายได้ชัดเจนขึ้น เช่นการใช้คำสั่ง

if studentClass = = Junior then ...

มีความหมายที่ชัดเจนกว่าการใช้คำสั่ง

if studentClass = = 3 then ...

นอกจากภาษา C แล้ว ยังมีภาษาคอมพิวเตอร์อื่นๆ เช่น ภาษา Ada, Pascal ที่อนุญาตให้โปรแกรมเมอร์สร้างชนิดข้อมูลแจกแจงค่าขึ้นใช้งาน ดังตัวอย่าง Class ข้างต้น ถ้าเขียนอยู่ในรูปแบบของภาษา Pascal จะเป็นดังนี้

```
type Class = (Freshman, Sophomore, Junior, Senior);
var studentClass : Class;
```

5.4 ชนิดข้อมูลแบบคอมโพสิต
(Composite data type)

ชนิดข้อมูลคอมโพสิตเป็นชนิดข้อมูลที่มีค่าเป็นกลุ่ม (compound or aggregated data) ได้แก่ชนิดข้อมูลแถวลำดับหรืออาร์เรย์ (array), ชนิดข้อมูลข้อความหรือสตริง (string), ชนิดข้อมูลยูเนียน (union), ชนิดข้อมูลเรคคอร์ดหรือโครงสร้าง (structure), ชนิดข้อมูลพ็อยเตอร์ (pointer)

ชนิดข้อมูลเหล่านี้เรียกว่าชนิดข้อมูลคอมโพสิต เพราะสร้างขึ้นจากชนิดข้อมูลพื้นฐานที่เป็นค่าเดี่ยวแล้วนำมาประกอบกันให้เกิดเป็นค่ากลุ่ม เช่น อาร์เรย์ของเลขจำนวนเต็ม สร้างขึ้นจากเลขจำนวนเต็ม (int) หลายจำนวนรวบรวมไว้เป็นกลุ่มเดียวกัน

<u>ชนิดข้อมูลอาร์เรย์</u>

อาร์เรย์หรือแถวลำดับเป็นชนิดข้อมูลคอมโพสิตที่เก็บข้อมูลประเภทเดียวกัน และมีจำนวนข้อมูลคงที่ เช่น คำประกาศในภาษา C

int A[10];

เป็นการประกาศตัวแปร A ว่าเป็นชนิดข้อมูลอาร์เรย์ ซึ่งแสดงด้วยสัญลักษณ์ [ ] จำนวนข้อมูลภายในอาร์เรย์มีจำนวน 10 ข้อมูล และข้อมูลทุกตัวเป็นค่าเลขจำนวนเต็ม แต่ถ้าเขียนคำประกาศนี้ในภาษา Pascal จะเป็นดังต่อไปนี้

```
A : array [ 1 .. 10 ] of integer;
```

สัญลักษณ์[1..10]ในภาษา Pascal เป็นการระบุหมายเลขอินเด็กซ์ (index or subscript) ให้มีค่า 1, 2, 3, ..., 10 ผู้ใช้สามารถกำหนดให้หมายเลขอินเด็กซ์เริ่มที่เท่าไรก็ได้ เช่น [-5..4] หมายถึง -5, -4, -3, ...,4 ซึ่งต่างจากภาษา C หรือ Java ที่อินเด็กซ์ต้องเริ่มต้นที่ 0 เท่านั้น เพราะการคำนวณตำแหน่งในอาร์เรย์จะทำได้รวดเร็วกว่า ภาษา Pascal อนุญาตให้กำหนดอินเด็กซ์ด้วยค่าประเภทอื่นนอกเหนือจากเลขจำนวนเต็มได้ ดังตัวอย่างต่อไปนี้

```
type Computer_Manufacturer = (IBM, DEC, HP, SUN, APPLE);
var profit : array [Computer_ Manufacturer] of integer;
```

`Computer_Manufacturer` เป็นชนิดข้อมูลแจกแจงค่าเพื่อแทนชื่อบริษัทผู้ผลิตเครื่องคอมพิวเตอร์ และตัวแปร `profit` เป็นชนิดอาร์เรย์เก็บค่าเลขจำนวนเต็ม ที่เป็นผลกำไรของบริษัทคอมพิวเตอร์เหล่านี้ ดังนั้นการใช้คำสั่ง `profit[IBM]` จึงเป็นการระบุถึงผลกำไรของบริษัท IBM จะเห็นได้ว่าการใช้ข้อความเป็นอินเด็กซ์แทนที่ จะเป็นตัวเลขช่วยให้การเขียนคำสั่งสื่อความหมายได้ดียิ่งขึ้น

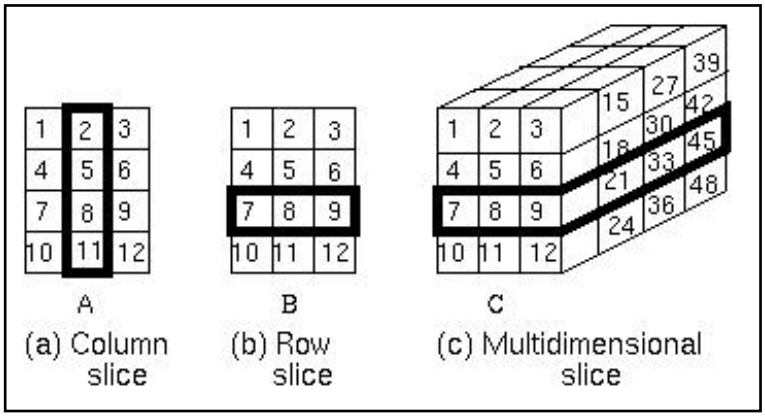
ข้อมูลแต่ละจำนวนในอาร์เรย์ (หรือสมาชิกของอาร์เรย์) สามารถเป็นอาร์เรย์ได้ ดังนั้นการกำหนดชนิดข้อมูลที่เป็นอาร์เรย์หลายมิติ (multi-dimensional array) จึงสามารถทำได้ ดังตัวอย่างภาษา C ต่อไปนี้

int x[10][20];

เป็นการกำหนดค่าตัวแปร x ให้เป็นอาร์เรย์สองมิติที่มี 10 แถว และ 20 คอลัมน์ ซึ่งถ้าจะประกาศตัวแปรโครงสร้างแบบเดียวกันนี้ในภาษา Pascal จะเป็นดังนี้

```
var x : array [1..10, 1..20] of integer;
```

ภาษาคอมพิวเตอร์บางภาษา เช่น ภาษา FORTRAN, APL, Algol 68, Ada อนุญาตให้ระบุอินเด็กซ์เป็นช่วงได้ เช่น ในภาษา Ada การระบุคำสั่ง x(3..5) จะหมายถึงสมาชิก 3 ตัวในอาร์เรย์ x (ภาษา Ada ใช้เครื่องหมาย ( ) เพื่อแทนอาร์เรย์แทนที่จะใช้ [ ] เหมือนในภาษา C) ซึ่งเทียบเท่ากับการใช้สามคำสั่ง x(3), x(4) และ x(5) การระบุอินเด็กซ์เป็นช่วงอย่างนี้เรียกว่า การทำ slicing (ลักษณะเช่นนี้ไม่มีในภาษา C) รูปที่ 5.3 แสดงการ slice อาร์เรย์สองมิติ และสามมิติในภาษา FORTRAN



รูปที่ 5.3 การระบุช่วงข้อมูลบนอาร์เรย์ด้วยคำสั่งภาษาฟอร์แทรน A(1:4,2), B(3,1:3) และ C(3,1:3,1:4)

ในภาษา SNOBOL ซึ่งจัดเป็นภาษาที่มีชนิดข้อมูลแบบพลวัต (dynamically typed language) จะเรียกอาร์เรย์ว่า เทเบิล (TABLE) และสมาชิกภายในอาร์เรย์ มีชนิดแตกต่างกันได้ ดังตัวอย่างต่อไปนี้

```
T = TABLE ( )
T<'RED'> = 'WAR'
T<b> = 25
T<4.6> = 'PEACE'
```

T เป็นเทเบิลหรืออาร์เรย์ที่เก็บข้อมูล 3 จำนวน ข้อมูลจำนวนแรกอยู่ที่ตำแหน่ง T<'RED'> เก็บค่าข้อความ 'WAR' ข้อมูลตัวที่สองอยู่ที่ตำแหน่ง T<b> เก็บค่าตัวเลข 25 และข้อมูลตัวที่สามอยู่ที่ตำแหน่ง T<4.6> เก็บค่าข้อความ 'PEACE' จะเห็นได้ว่าค่าที่เก็บภายในอาร์เรย์มีชนิดข้อมูลแตกต่างกัน และการระบุตำแหน่งอาร์เรย์แต่ละช่วงจะระบุเป็นชื่อหรือตัวเลข ก็ได้ โดยภาษา SNOBOL จะใช้วิธีสัมพันธ์ (associate) ชื่อช่องกับค่าที่เก็บอยู่ในช่องนั้น เช่น T<b> จะสัมพันธ์กับค่า 25 อาร์เรย์แบบนี้จะเรียกว่า associative array

ภาษา Perl มีการใช้ associative array เช่นเดียวกัน โดยใช้เครื่องหมายพิเศษ % นำหน้าชื่อตัวแปร เพื่อระบุประเภทว่าตัวแปรนั้นเป็นชนิด associative array เช่นคำสั่งต่อไปนี้ทำหน้าที่สร้าง associative array

```
%ClassList = ("Michelle", 'A', "Doris", 'B', "Mike", 'D');
```

ระบุว่า ClassList เป็นตัวแปรประเภท associative array เก็บข้อมูล 3 จำนวน ข้อมูลแต่ละจำนวนประกอบด้วยคู่ลำดับชื่อช่อง (เรียกว่า key) และค่าที่เก็บในช่องนั้น (เรียกว่า value) เช่นช่องชื่อ Michelle เก็บค่า A การใช้งานอาร์เรย์แสดงตัวอย่างได้ดังนี้

```
$ClassList {"Michelle"}; #has the value "A"
@y = % ClassList;        #make array y a 6-element enumeration array
for ($i = 0; $i < 6; $i++)
     {print "I = ,$i, $y[$i]\n";};
```

ผลลัพธ์ที่ได้จะเป็นค่า key และ value โดย key จะเรียงลำดับตามตัวอักษร

```
I = , 0, Doris
I = , 1, B
I = , 2, Mike
I = , 3, D
I = , 4, Michelle
I = , 5, A
```

<u>ชนิดข้อมูลสตริง</u> (string)

ข้อความหรือสตริง (string) ประกอบขึ้นจากอักขระหลายอักขระเรียงติดต่อกัน การเก็บข้อมูลสตริงจึงมักจะเก็บอยู่ในลักษณะของอาร์เรย์ของอักขระ เช่นตัวอย่างในภาษา C

char name [10] = { 'B', 'i', 'l', 'l', '\0'};

เป็นการประกาศตัวแปรชื่อ name ในลักษณะอาร์เรย์ของอักขระมีขนาด 10 พร้อมทั้งกำหนดค่าเริ่มต้นเป็นข้อความ Bill ซึ่งมีความยาวไม่ถึง 10 อักขระ ดังนั้นส่วนที่เหลือของอาร์เรย์จะไม่ได้ถูกใช้งาน สตริงในภาษา C จะลงท้ายด้วยอักขระว่าง (null character or '\0') ดังนั้นตัวแปร name จะเก็บข้อมูลยาว 5 อักขระ การประกาศตัวแปร name อาจทำได้อีกลักษณะดังนี้

char name [10] = "Bill";

การกำหนดค่าเริ่มต้นในลักษณะนี้ ไม่ต้องระบุอักขระว่าง แต่การเก็บค่าของตัวแปร name จะเป็นเช่นเดิมคือเก็บข้อมูลยาว 5 อักขระ โดยอักขระตัวที่ 5 เป็นอักขระว่าง

ในภาษา Java สตริงจะถูกเก็บในลักษณะของอ็อบเจ็คของคลาสสตริง การประกาศตัวแปรจึงเป็นการประกาศอ็อบเจ็คดังนี้

String name = "Bill";

การปฏิบัติการกับสตริงมีได้หลายรูปแบบแตกต่างกันไปในแต่ละภาษา ปฏิบัติการที่สำคัญมีดังนี้

(1) การเชื่อมต่อสตริง (concatenation)

ภาษา Java ใช้เครื่องหมาย “+” ในการเชื่อมต่อสตริง เช่น

```
String fullname, firstname, lastname;
firstname = “Bill”;
lastname = “Gates”;
fullname = firstname + lastname;
            // fullname is now “BillGates”
```

ภาษา C ใช้ฟังก์ชัน strcat ในการเชื่อมต่อสตริง เช่น

```
char firstname [15] = “Bill”;
char lastname [6] = “Gates”;
strcat (firstname, lastname);
        /* result of concatenation is in firstname */
```

(2) การเปรียบเทียบสตริง

การเปรียบเทียบข้อความจะเป็นการเปรียบเทียบตามลำดับก่อน-หลัง ของตัวอักษรในข้อความ ภาษา C ใช้ฟังก์ชัน strcmp เพื่อเปรียบเทียบสตริง โดยมีรูปแบบฟังก์ชันดังนี้

*int strcmp (char *str1, char *str2)*

ถ้าค่าที่ส่งกลับเป็นศูนย์แสดงว่า str1 และ str2 เท่ากัน ถ้าค่าที่ส่งกลับมีค่าน้อยกว่าศูนย์แสดงว่า str1 น้อยกว่า str2 หรือ str1 มีลำดับก่อน str2 และถ้าค่าที่ส่งกลับมีค่ามากกว่าศูนย์ แสดงว่า str1 มากกว่า str2

3) การคัดเลือกส่วนย่อยของสตริง

การคัดเลือกสตริงย่อย (substring selection) จะเป็นการระบุตำแหน่งของอักขระภายในสตริง เช่น ในภาษา FORTRAN คำสั่ง

```
NEXT = STR (6:10)
```

เป็นการคัดเลือกอักขระตำแหน่งที่ 6 ถึง 10 จากสตริง STR เก็บไว้ในตัวแปรชื่อ NEXT

4) การจับคู่แพทเทิร์น (pattern matching)

แพทเทิร์นเป็นรูปแบบของคำที่ปรากฏในข้อความ ภาษา SNOBOL และภาษา Perl เป็นภาษาที่ใช้ลักษณะการจับคู่แพทเทิร์นมากกว่าภาษาคอมพิวเตอร์อื่นๆ การระบุแพทเทิร์นในภาษา Perl จะใช้รูปแบบของนิพจน์ปรกติ (regular expression) เช่น

```
/ [A-Z a-z] [A-Z a-z \d]+/
```

เป็นการระบุแพทเทิร์นของข้อความว่าขึ้นต้นด้วยตัวอักษรจากนั้นตามด้วยตัวอักษรหรือตัวเลข การจับคู่แพทเทิร์นแสดงได้ดังตัวอย่างต่อไปนี้

```
$_ = <STDIN>;                    #Read in input to argument $_
if (/Λa+b+$/)
  then  {print "yes\n";}
  else  {print "no \n";}         #Match $_ with a+b+
```

ตัวอย่างข้างต้นเป็นคำสั่งรับข้อความ จากนั้นตรวจสอบข้อความตั้งแต่ต้นบรรทัด (เครื่องหมาย Λ) จนถึงท้ายบรรทัด (เครื่องหมาย $) เพื่อตรวจสอบว่ามีอักขระ a และ b ปรากฏอยู่หรือไม่ (ระบุแพทเทิร์น /a+b+/ ซึ่งเทียบเท่ากับนิพจน์ปรกติ a^+b^+) ถ้ามีปรากฏอยู่ให้พิมพ์คำว่า yes ถ้าไม่มีพิมพ์คำว่า no

<u>ชนิดข้อมูลยูเนียน</u> (union)

ยูเนียน (union) หมายถึง OR หรือการเลือกค่าใดค่าหนึ่งเพียงหนึ่งค่า ชนิดข้อมูลยูเนียนจึงเป็นชนิดข้อมูลที่มีการประกาศรวมได้หลายชนิด แต่การเก็บข้อมูลจริง ณ ช่วงเวลาหนึ่งๆ จะเลือกเก็บค่าได้เพียงประเภทเดียวเท่านั้น ตัวอย่างเช่น ในภาษา C

```
union shared_tag {
        char c;
        int i;
        long l;
        float f;
        double d;
} shared;
```

เป็นการกำหนดชนิดข้อมูลยูเนียนชื่อ shared_tag ให้สามารถเก็บข้อมูลเป็นชนิด char หรือ int หรือ long หรือ float หรือ double แต่เลือกได้เพียงแบบเดียว ณ ขณะเวลาหนึ่งเท่านั้น เช่น การใช้คำสั่ง

shared.c = '$';

เป็นการเลือกเก็บอักขระ $ แต่ต่อมาถ้ามีการใช้คำสั่ง

shared.d = 123456789.8765;

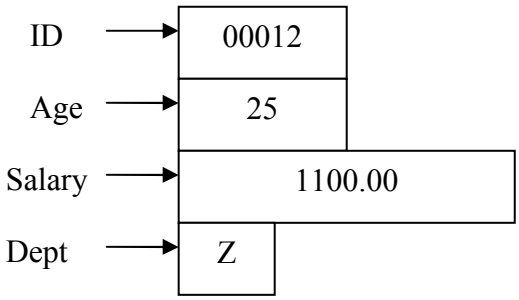
เป็นการเปลี่ยนการเก็บข้อมูลจากชนิดอักขระเป็นชนิดเลขจำนวนจริงแบบ double

<u>ชนิดข้อมูลโครงสร้าง</u> (structure or record)

ชนิดข้อมูลเรคคอร์ดหรือโครงสร้าง (structure) เป็นการเก็บข้อมูลหลายจำนวนที่มีประเภทแตกต่างกัน เช่น ในภาษา C

```
struct EmployeeType {
        int ID;
        int Age;
        float Salary;
        char Dept;
} Employee;
```

เป็นการประกาศโครงสร้างของข้อมูล Employee ว่าประกอบด้วยข้อมูลสี่ส่วน คือ ID, Age, Salary และ Dept ซึ่งแสดงโครงสร้างได้ดังรูปที่ 5.4



รูปที่ 5.4 โครงสร้างการจัดเก็บข้อมูลของ struct Employee

การประกาศโครงสร้างเพื่อเก็บข้อมูลของ Employee จำนวน 500 คน สามารถทำได้ด้วยการสร้างอาร์เรย์ของ struct ดังนี้

```
struct EmployeeType {
        int ID;
        int Age;
        float Salary;
        char Dept;
} Employee[500];
```

การระบุข้อมูลเงินเดือนของพนักงานคนที่ 4 สามารถทำได้ด้วยคำสั่งต่อไปนี้

Employee [3].Salary = 13000.00;

ในภาษา Pascal จะเรียกข้อมูลโครงสร้างว่าเรคคอร์ด และสามารถกำหนดฟิลด์ในเรคคอร์ดให้มีค่าแปรผันได้ (เรียกว่า variant record) ตัวอย่างเช่น เงินเดือนของพนักงานอาจจะแยกออกเป็นสองประเภทคือ พนักงานที่ได้รับเงินรายเดือน (จะมีข้อมูลอัตราเงินเดือน และวันที่เริ่มทำงาน) และพนักงานที่ได้รับเงินเป็น

รายชั่วโมง (จะมีข้อมูลอัตราค่าจ้างรายชั่วโมง, จำนวนชั่วโมงทำงานปกติ, จำนวนชั่วโมงทำงานล่วงเวลา) การประกาศเรคคอร์ดแบบแปรผันในภาษา Pascal แสดงตัวอย่างได้ดังนี้

```
type  PayType = (Salaried, Hourly);
      var Employee: record
            ID: integer;
            Age: integer;
            Dept: array [1..2] of char;
            case  PayClass: PayType of
                  Salaried: (MonthlyRate: real;
                              StartDate: integer);
                  Hourly: (HourRate : real;
                           Regular : integer;
                           Overtime : integer)
            end
```

โครงสร้างการจัดเก็บข้อมูลของ Employee ที่เรคคอร์ดมีขนาดแปรผันได้ แสดงดังรูปที่ 5.5

ID	
Age	
Dept	
MonthlyRate	HourRate
StartDate	Regular
	Overtime

รูปที่ 5.5 โครงสร้างข้อมูลพนักงานที่เรคคอร์ดมีความยาวแปรผัน

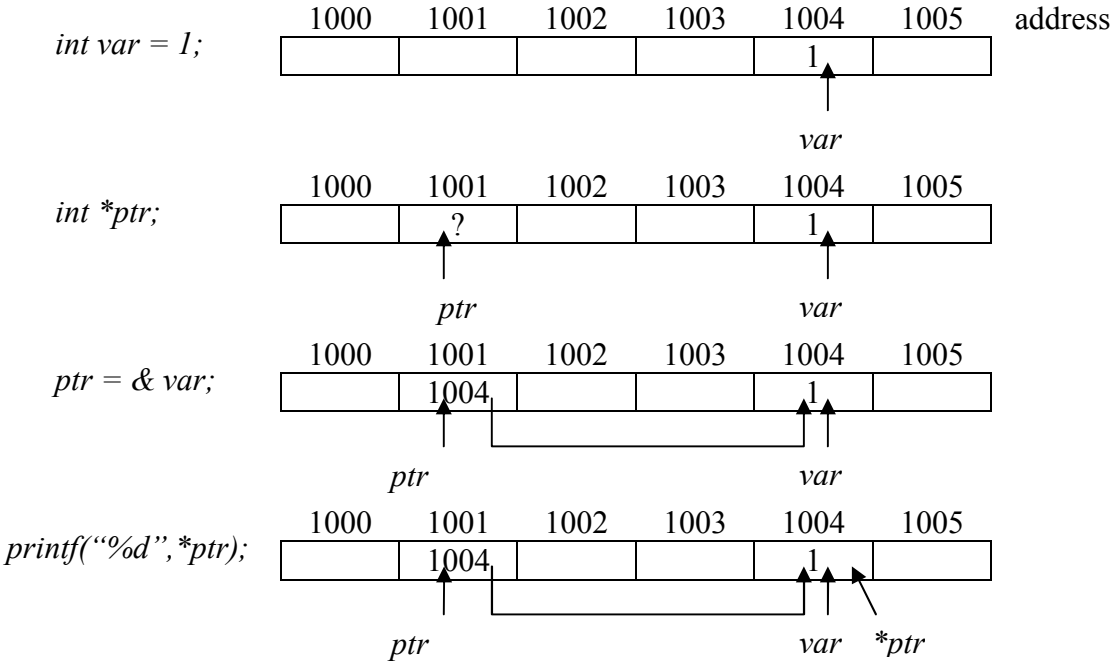
<u>ชนิดข้อมูลพ็อยเตอร์</u> (pointer)

ตัวแปรที่ได้รับการประกาศเป็นชนิดข้อมูลพ็อยเตอร์ (ด้วยการใช้เครื่องหมาย '*' ในภาษา C และ C++, เครื่องหมาย '^' ในภาษา Pascal, และใช้ข้อความ 'access' ในภาษา Ada) จะทำหน้าที่เก็บตัวเลขที่ระบุตำแหน่งของหน่วยความจำ หรือเก็บเป็นค่า null หรือ nil เมื่อยังไม่มีการอ้างอิงตำแหน่งในหน่วยความจำ

พ็อยเตอร์ถูกสร้างขึ้นด้วยวัตถุประสงค์สองประการ คือ เพื่อการเรียกใช้ค่าของข้อมูลโดยอ้อม (indirect addressing) และเพื่อการใช้เนื้อที่หน่วยความจำแบบพลวัต (dynamic memory allocation)

ตัวอย่างภาษา C ต่อไปนี้ แสดงการเรียกใช้ค่าของข้อมูลโดยตรง (direct access) ด้วยการเรียกใช้ตัวแปรตามปกติ เปรียบเทียบกับการเรียกใช้ค่าของข้อมูลโดยอ้อม (indirect access) ผ่านตัวแปรพ็อยเตอร์ และแสดงภาพที่เกิดขึ้นในหน่วยความจำได้ดังรูปที่ 5.6

```
int var = 1;
int *ptr;
ptr = & var;
printf ("%d",var);     /* direct access */
printf ("%d",*ptr);     /* indirect access */
```

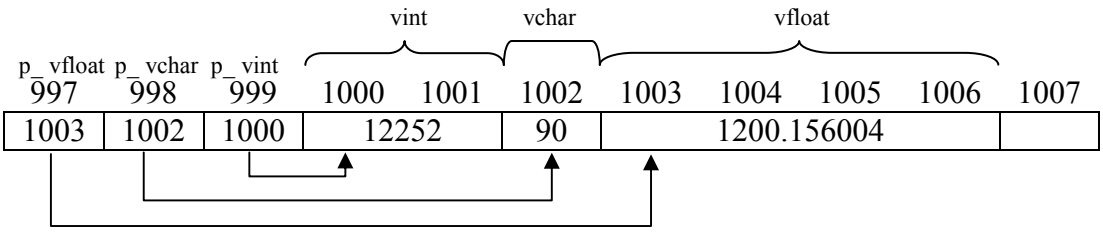


รูปที่ 5.6 โครงสร้างภายในหน่วยความจำเมื่อมีการใช้ตัวแปรชนิดพ็อยเตอร์

จากตัวอย่างข้างต้นจะเห็นว่า ptr และ &var เป็นการระบุถึงตำแหน่งในหน่วยความจำในขณะที่ *ptr และ var เป็นการระบุถึงค่าที่เก็บอยู่ในหน่วยความจำ การระบุค่าด้วยคำสั่ง *ptr เรียกว่า การอ้างอิงค่า (dereferencing)

การประกาศตัวแปรพ็อยเตอร์ต้องระบุว่าชี้ไปที่ค่าประเภทใด (เช่น int, char, float) เพื่อให้คอมไพเลอร์รู้ว่าตำแหน่งที่อ้างอิงไปนั้นยาวกี่ไบต์ (ดังแสดงด้วยตัวอย่างในรูปที่ 5.7) โดยค่าที่เก็บในตัวแปรพ็อยเตอร์จะเป็นตำแหน่งเริ่มต้นของค่าที่ถูกอ้างอิง

```
int vint = 12252;
char vchar = 90;
float vfloat = 1200.156004;
int *p_vint = & vint;
int *p_ vchar = & vchar;
int *p_ vfloat = & vfloat;
```



รูปที่ 5.7 การอ้างอิงค่าของตัวแปรพ็อยเตอร์ไปยังค่าประเภท int, char และ float

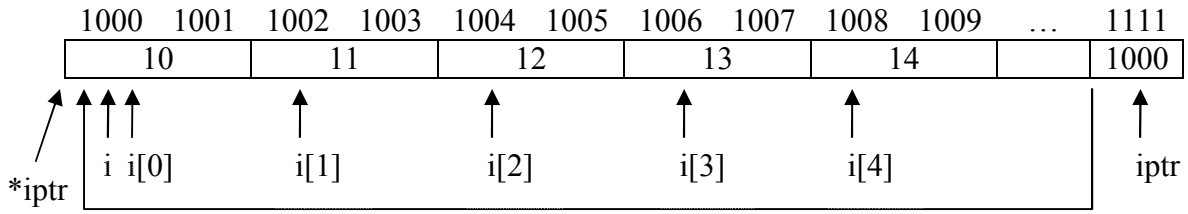
การรู้ขนาดของค่าที่ถูกอ้างอิงว่ายาวกี่ไบต์มีความจำเป็นต่อการทำงานของคอมไพเลอร์ โดยเฉพาะในกรณีที่ค่าที่ถูกอ้างอิงเป็นข้อมูลกลุ่มเช่นอาร์เรย์ ทั้งนี้เพื่อให้การเลื่อนการชี้ตำแหน่งของพ็อยเตอร์ทำได้อย่างถูกต้อง ดังตัวอย่างต่อไปนี้ (แสดงเป็นภาพได้ดังรูปที่ 5.8)

```
int i[5] = {10,11,12,13,14};
float f[3] = {0.0,0.1,0.2};
int *iptr;
float *fptr;
iptr = i;
fptr = f;
printf("%d", *iptr++);
printf("%d", *fptr++);
```

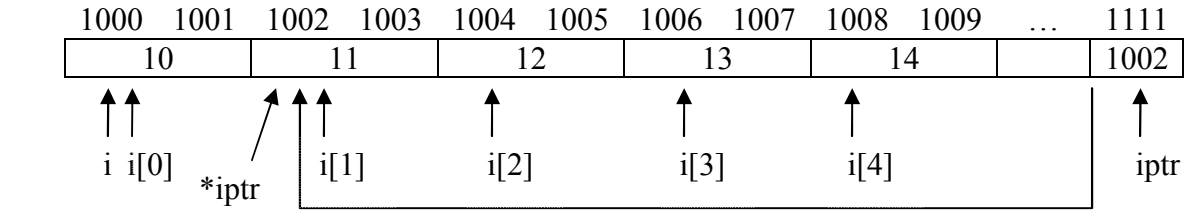
จากรูปที่ 5.8 จะสังเกตได้ว่าการระบุชื่ออาร์เรย์ i และ f โดยไม่มีวงเล็บระบุอินเด็กซ์ของอาร์เรย์ จะหมายถึงตำแหน่งแรกของอาร์เรย์ นั่นคือ 1000 และ 1250 ตามลำดับ ดังนั้นการเปรียบเทียบค่าต่อไปนี้เป็นจริง

i = = 1000	*f = = 1250*
&i [0] = = 1000	*&f [0] = = 1250*
&i [1] = = 1002	*&f [1] = = 1254*
**iptr = = i [0]*	**fptr = = f [0]*
**(iptr+1) = = i [1]*	**(fptr+1) = = f [1]*

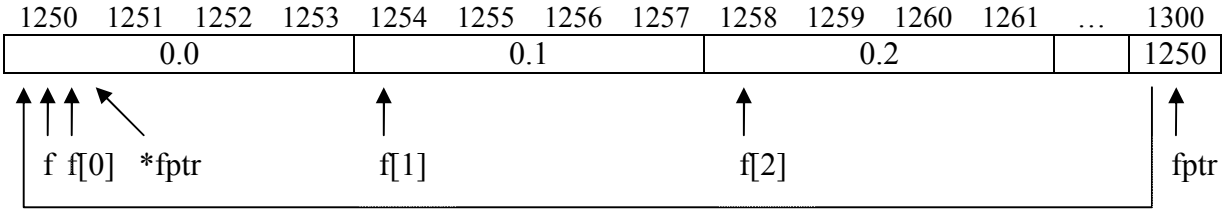
*int i[5] = {10,11,12,13,14}; int *iptr; iptr = i;*



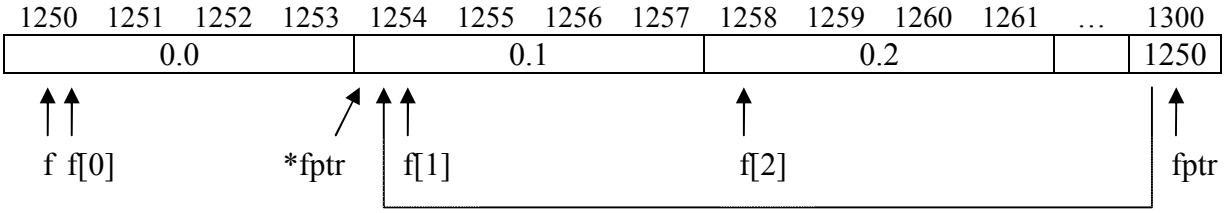
iptr ++;



*float f[3] = {0.0,0.1,0.2}; float *fptr; fptr = f;*



fptr ++;



รูปที่ 5.8 การเพิ่มค่าตัวแปรพ็อยเตอร์เพื่อชี้ค่า int และ float

ในกรณีของการส่งข้อมูลอาร์เรย์ระหว่างฟังก์ชัน ตามข้อกำหนดของภาษา C วิธีการส่งพารามิเตอร์จะเป็นแบบโดยการอ้างอิง (pass-by-reference) นั่นคือจะใช้การส่งตำแหน่งเริ่มต้นของอาร์เรย์ไปให้ฟังก์ชัน ดังตัวอย่างโปรแกรมต่อไปนี้

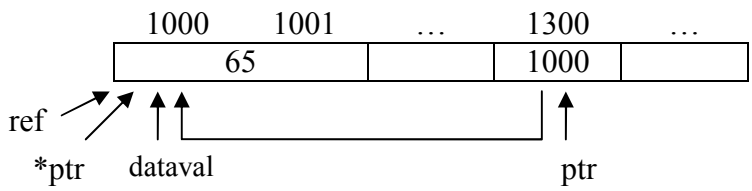
```
1:      /* Passing an array to a function */
2:      #include <stdio.h>
3:      #define MAX 10
4:      int array [MAX], count;
5:
6:      int largest (int x[ ], int y);
7:
8:      int main( )
9:      {       /* input array value from the keyboard */
10:             for (count = 0; count < MAX; count ++)
11:                     {
12:                             printf (“Enter an integer value:  ” );
13:                             scanf (“%d”, &array[count]);
14:                     }
15:             printf (“\n Largest value = %d \n”, largest (array, MAX));
16:             return 0;
17:     }
18:
19:     int largest (int x[ ], int y)
20:     {
21:             int count, biggest = -12000;
22:             for (count = 0; count <y; count++)
23:                     {
24:                             if (x[count] > biggest)
25:                                     biggest = x[count];
26:                     }
27:             return biggest;
28:     }
```

ในโปรแกรมตัวอย่างข้างต้น ฟังก์ชัน largest รับพารามิเตอร์เป็นอาร์เรย์ และเลขจำนวนเต็มเพื่อบอกขนาดของอาร์เรย์ การรับพารามิเตอร์อาร์เรย์เป็นค่าตำแหน่งเริ่มต้นของอาร์เรย์จึงมีผลเทียบเท่ากับการใช้พ็อยเตอร์ ดังต่อไปนี้

```
int largest (int *x, int y)
{
        int count, biggest = -12000 ;
        for (count = 0 ; count <y ; count ++)
        {
                if (*(x+count) > biggest)
                        biggest = *(x+count);
        }
}
```

ภาษา C++ นอกจากมีตัวแปรชนิดพ็อยเตอร์ เพื่ออ้างอิงค่าโดยอ้อมแล้ว ยังมีการกำหนดตัวแปรชนิดใหม่ขึ้นใช้งานเรียกว่า ตัวแปรชนิดอ้างอิง (reference variable) ที่มีการทำงานคล้ายพ็อยเตอร์ เพียงแต่การอ้างอิงค่า (dereferencing) จะมีรูปแบบคำสั่งที่ง่ายกว่า และมีการอ้างอิงให้โดยอัตโนมัติ ตัวอย่างคำสั่งและรูปภาพโครงสร้างหน่วยความจำ (รูปที่ 5.9) ต่อไปนี้ แสดงความแตกต่างระหว่างตัวแปรชนิดพ็อยเตอร์และตัวแปรชนิดอ้างอิง

```
int dataval = 6.5;        // regular variable
int *ptr = & dataval;     // pointer variable to dataval
int & ref = dataval;      // reference variable to dataval
*ptr++;                   // increment the value to which ptr points
ref++;                    // increment reference variable
dataval++;                // increment regular variable
```



รูปที่ 5.9 เปรียบเทียบตัวแปรชนิด pointer และชนิด reference

จากรูปที่ 5.9 จะเห็นว่าตัวแปรชนิดพ็อยเตอร์แยกความแตกต่างระหว่าง ptr ที่เป็นชื่อตัวแปรพ็อยเตอร์ และ *ptr ที่เป็นค่าซึ่งถูกชี้โดยตัวแปร ptr แต่ตัวแปรชนิดอ้างอิงไม่มีการแยกความแตกต่างเช่นนี้ ทุกครั้งที่ระบุชื่อ ref จะหมายถึงค่าที่ถูกชี้เสมอ ซึ่งเป็นการอ้างอิงค่าให้โดยอัตโนมัติ ลักษณะการอ้างอิงให้โดยอัตโนมัติเช่นนี้ ถูกนำมาใช้ในภาษา Java โดยไม่มีการใช้ตัวแปรชนิดพ็อยเตอร์เพื่อลดโอกาสเกิดข้อผิดพลาดในการเขียนโปรแกรม

5.5 ชนิดข้อมูลนามธรรม

(Abstract data types)

ภาษาคอมพิวเตอร์ในปัจจุบันโดยทั่วไปมักจะสร้างชนิดข้อมูลพื้นฐาน เช่น int, char, float เตรียมไว้ให้โปรแกรมเมอร์สามารถเรียกใช้งานได้ทันที ชนิดข้อมูลพื้นฐานที่มีอยู่ในภาษาคอมพิวเตอร์เหล่านี้จะเรียกว่า built_in types นอกจากนี้ชนิดข้อมูลที่มีอยู่แล้วในตัวภาษา ยังมีชนิดข้อมูลที่ผู้ใช้หรือโปรแกรมเมอร์สร้างขึ้นเพิ่มเติม เรียกว่า user_defined types ชนิดข้อมูลที่ผู้ใช้สร้างเพิ่มเติม จะสร้างขึ้นจากชนิดข้อมูลพื้นฐาน เช่น ชนิดข้อมูลโครงสร้างในภาษา C

```
typedef struct {
        float real_part, imaginary_part;
} complex;
```

เป็นการสร้างชนิดข้อมูลขึ้นใหม่ชื่อ complex โดยสร้างจากชนิดข้อมูลพื้นฐาน float เมื่อมีชนิดข้อมูล complex ให้ใช้งานแล้ว โปรแกรมเมอร์สามารถประกาศตัวแปรให้เป็นชนิด complex ได้ดังนี้

```
complex a, b, c;
```

ภาษาคอมพิวเตอร์ที่อำนวยความสะดวกให้ผู้ใช้สามารถสร้างชนิดข้อมูลขึ้นใหม่ ให้เหมาะสมกับความต้องการใช้งานจะช่วยให้โปรแกรมอ่านง่ายและสื่อความหมายดีขึ้น โครงสร้างข้อมูลที่ต้องมีการใช้งานบ่อยเมื่อประกาศเป็นชนิดข้อมูลใหม่ จะช่วยลดความซ้ำซ้อนในการเขียนโครงสร้างเดิมซ้ำๆ กันหลายครั้ง ทำให้ลดโอกาสการพิมพ์คำสั่งผิดพลาด

การสร้างชนิดข้อมูลขึ้นใหม่ตามตัวอย่างข้างต้น เป็นเพียงการสร้างโครงสร้างข้อมูลขึ้นใหม่โดยไม่ได้ระบุว่าโครงสร้าง หรือชนิดข้อมูลชื่อ complex สามารถใช้งานกับคำสั่งหรือปฏิบัติการแบบใดได้บ้าง การสร้างชนิดข้อมูลใหม่ที่สมบูรณ์ ควรจะต้องมีการกำหนดโครงสร้างข้อมูลภายในของชนิดข้อมูลนั้นรวมถึงมีการกำหนดปฏิบัติการที่สามารถกระทำกับข้อมูลเหล่านั้นได้ การกำหนดที่สมบูรณ์นี้ ทำให้เกิดเป็นชนิดข้อมูลประเภทที่เรียกว่าชนิดข้อมูลนามธรรม

ชนิดข้อมูลนามธรรม หมายถึง ชนิดข้อมูลที่ผู้ใช้สร้างขึ้นใหม่ให้มีทั้งส่วนโครงสร้างข้อมูลภายใน (ที่มักจะซ่อนรายละเอียดไว้เป็นการภายใน ไม่อนุญาตให้ผู้ใช้ภายนอกเข้ามาเปลี่ยนแปลงแก้ไขโครงสร้าง) และส่วนปฏิบัติการกับข้อมูล (ที่อนุญาตให้ผู้ใช้ภายนอกเรียกใช้งานได้) ตัวอย่างในภาษา C++ ต่อไปนี้แสดงการกำหนดชนิดข้อมูลนามธรรมชื่อ point ที่สื่อความหมายถึงชนิดข้อมูลจุดที่ประกอบขึ้นจากพิกัด x และ y

```
class point {
public:
    point (int a, int b)    {x=a; y=b;}     // initializes coordinates of a new point
    void x_move (int a)     {x+=a;}         // moves the point horizontally
    void y_move (int b)     {y+=b;}         // moves the point vertically
    void reset ( )          {x=0; y=0;}     // moves the point to the origin
private:
        int x, y;
};
```

ชนิดข้อมูลนามธรรม point มีโครงสร้างภายในคือ พิกัด x และ y ที่เป็นค่าเลขจำนวนเต็ม และประกาศว่าเป็นข้อมูล private คือใช้เฉพาะภายในคลาสเท่านั้น โปรแกรมภายนอกสามารถเรียกใช้ชนิดข้อมูล point ได้ โดยมีปฏิบัติการที่สามารถใช้ได้ 4 ปฏิบัติการ คือ point() ใช้กำหนดพิกัดเริ่มต้นของจุด, x_move() ใช้เลื่อนจุดไปในแนวแกนนอน, y_move() ใช้เลื่อนจุดไปในแนวแกนตั้ง, reset() ใช้กำหนดจุดไปที่พิกัด (0,0) ตัวอย่างการเรียกใช้ชนิดข้อมูล point แสดงได้ดังนี้

```
point  p1(1,3);                 // instantiates p1 and initializes its value
point  p2(55,0);                // instantiates p2 and initializes its value
point *p3=new point (0,0)       // p3 points to the origin
p1.x_move (3);                  // moves p1 horizontally
p2.y_move (99);                 // moves p2 vertically
p1.reset ( );                   // positions p1 at the origin
```

5.6 สรุป

การทำงานของโปรแกรมคอมพิวเตอร์โดยทั่วไปมักจะสั่งการทำงานผ่านตัวแปร โดยกำหนดค่าเริ่มต้นให้กับตัวแปร ต่อจากนั้นจะสั่งเปลี่ยนแปลงค่าของตัวแปรไปตามลำดับจนจบคำสั่งสุดท้ายของโปรแกรม สถานะสุดท้ายซึ่งก็คือค่าของตัวแปรทุกตัวในขณะนั้นจะเป็นสิ่งบอกผลลัพธ์การทำงานของโปรแกรม ตัวแปรทุกตัวในโปรแกรมจะต้องมีชนิดข้อมูลกำกับว่าค่าที่เก็บภายในตัวแปรจะเป็นค่าใดได้บ้าง เพื่อให้การเก็บค่าและการเปลี่ยนแปลงค่าภายในตัวแปรเป็นไปอย่างถูกต้อง

การระบุชนิดข้อมูลให้กับตัวแปร อาจระบุอย่างชัดเจนก่อนการเรียกใช้ตัวแปร (เช่น ภาษา C, Java, Pascal ซึ่งจัดเป็น statically typed language) หรือไม่ต้องระบุชนิดข้อมูลล่วงหน้า แต่ใช้วิธีตีความจากค่าจริงของข้อมูลที่เกิดขึ้นขณะรันโปรแกรม (เช่น ภาษา Perl, APL, LISP ซึ่งจัดเป็น dynamically typed language)

การตรวจสอบความถูกต้องทั้งหลายที่เกี่ยวข้องกับชนิดข้อมูล (type checking) เป็นหน้าที่ของระบบชนิดข้อมูล (type system) ซึ่งเป็นระบบที่ประกอบด้วยกฎเกณฑ์จำนวนมากเพื่อการบังคับใช้ชนิดข้อมูลที่ถูกต้องด้วยวัตถุประสงค์ที่จะไม่ให้โปรแกรมเกิดข้อผิดพลาดที่เกี่ยวเนื่องกับชนิดข้อมูล (type error) โปรแกรมที่ปราศจากข้อผิดพลาดด้านชนิดข้อมูลจะเรียกว่า type safe หรือ type secure การตรวจสอบความถูกต้องของชนิดข้อมูลจำแนกได้เป็นสองประเภท คือ การตรวจสอบขณะคอมไพล์โปรแกรม (เรียกว่า static checking ใช้ได้กับภาษาที่เป็น statically typed language) และการตรวจสอบขณะรันโปรแกรม (เรียกว่า dynamic checking ใช้กับภาษาประเภท dynamically typed language)

ภาษาคอมพิวเตอร์โดยทั่วไป มักจะกำหนดรูปแบบชนิดข้อมูลพื้นฐานมาพร้อมกับตัวภาษา เรียกว่า built-in data type และเพื่อความสะดวกในการใช้งานก็มักจะเตรียมโครงสร้างเพื่อให้ผู้ใช้กำหนดชนิดข้อมูลใหม่ขึ้นใช้งานได้ เรียกว่า user-defined data type ภาษารุ่นใหม่โดยเฉพาะภาษาเชิงวัตถุ นอกจากอนุญาตการสร้างชนิดข้อมูลใหม่แล้วผู้ใช้ยังสามารถระบุฟังก์ชันการทำงานต่างๆ ที่สามารถกระทำกับชนิดข้อมูลนั้นได้ เรียกว่า ชนิดข้อมูลนามธรรม (abstract data type)

แบบฝึกหัดท้ายบทที่ 5

คำถามอัตนัย

1. เมื่อโปรแกรมประกาศตัวแปร เช่น int x

   คุณลักษณะ(property of data) ที่เกิดขึ้นจากการประกาศตัวแปรนี้มีอะไรบ้าง?

2. Type system คืออะไร? มีความสำคัญอย่างไร?
3. Type compatibility คืออะไร? ให้ยกตัวอย่างมาหนึ่งตัวอย่าง
4. Type casting ต่างจาก Type coercion อย่างไร?
5. Abstract data type คืออะไร? ให้ยกตัวอย่างมาหนึ่งตัวอย่าง

คำถามปรนัย: ให้เลือกคำตอบที่ถูกต้องที่สุด

1. พิจารณาส่วนของโปรแกรมภาษา C ต่อไปนี้

```
int  list[5] = {1, 3, 5, 7, 9};
printf("%d \n", list[3]);
```

ผลลัพธ์ที่ได้จากคำสั่ง printf( ) คืออะไร ?

ก. 1 ข. 3

ค. 5 ง. 7

2. ถ้า integer ใช้เนื้อที่ 2 ไบต์ในการเก็บข้อมูล ตัวแปร list ในข้อ 1 จะใช้เนื้อที่เก็บข้อมูลกี่ไบต์ ?

ก. 2 ไบต์ ข. 5 ไบต์

ค. 10 ไบต์ ง. 12 ไบต์

3. การกำหนดชนิดของข้อมูลสตริงดังต่อไปนี้ พบได้ในภาษาใด ?

```
String  s1 = "hello there";
String  s2 = new String("hello there");
```

ก. Perl ข. Java

ค. Postscript ง. Snobol

4. ถ้าอักขระ (character) ในภาษา C ใช้เนื้อที่ 1 ไบต์ในการเก็บ 1 อักขระ คำสั่งกำหนดข้อความต่อไปนี้ ใช้เนื้อที่กี่ไบต์ในการเก็บข้อมูล ?

```
char *s = "hello";
```

ก. 1 ไบต์ ข. 5 ไบต์

ค. 6 ไบต์ ง. 7 ไบต์

5. คำสั่งในภาษา Pascal ต่อไปนี้ เป็น data type ประเภทใด ?

```
type score = 0..100;
```

ก. ordinal ข. subrange

ค. enumeration ง. short integer

6. คำสั่งในภาษา C ต่อไปนี้ เป็น data type ประเภทใด ?

```
enum days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

ก. ordinal ข. subrange

ค. enumeration ง. array of string

7. จากข้อ 6 ถ้ากำหนดตัวแปร d เป็นชนิด days ดังนี้

```
days d = Mon;
```

คำสั่งในข้อใดที่ทำให้ d มีค่าเป็น Tue ?

ก. d = Tue; ข. d++;

ค. ++d; ง. ถูกทุกข้อ

8. พิจารณาส่วนของโปรแกรมภาษา C ต่อไปนี้

```
int list[5] = {1, 3, 5, 7,9};
int * ptr;
ptr = list;
printf("%d \n", *(ptr + 1));
```

ถ้าค่าแรกของตัวแปร list เก็บไว้ที่หน่วยความจำตำแหน่ง 1003 คำสั่ง printf() จะพิมพ์ค่าใด ?

ก. 1 ข. 3

ค. 1003 ง. 1004

9. พิจารณาคำประกาศ data type ในภาษา C ต่อไปนี้

```
typedef union {
        int  x;
        float  r;
    } int_or_real;
```

ถ้าข้อมูล integer ใช้เนื้อที่ 2 ไบต์ ข้อมูล float ใช้เนื้อที่ 4 ไบต์ ชนิดของข้อมูล int_or_real ใช้เนื้อที่กี่ไบต์ ?

ก. 2 ไบต์ ข. 4 ไบต์

ค. 6 ไบต์ ง. 8 ไบต์

10. ภาษา C และ C++ ใช้ฟังก์ชันใดในการทำงานกับสตริง ?

ก. strcpy, strcat, strcmp, strlen ข. concat, copy, buffer, string

ค. length, mid, str, pos ง. d+, d*, d-, d/