การเขียนโปรแกรมคอมพิวเตอร์ 1

Computer Programming I

ตัวดำเนินการ (operators), การรับข้อมูลเข้า และการแสดงผลลัพธ์

ภิญโญ แท้ประสาทสิทธิ์

Emails : pinyotae+111 at gmail dot com, pinyo at su.ac.th

Web : http://www.cs.su.ac.th/~pinyotae/compro1/

Facebook Group : ComputerProgramming@CPSU

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

สัปดาห์ที่สี่

หัวข้อเนื้อหา

- พื้นฐานเกี่ยวกับตัวดำเนินการ
- ตัวดำเนินการทางคณิตศาสตร์ (math operator)
  - ตัวดำเนินการพื้นฐาน
  - ลำดับความสำคัญของตัวดำเนินการ
- การแปลงชนิดข้อมูล (casting)
  - การแปลงโดยปริยาย
  - การแปลงโดยชัดแจ้ง
- ตัวดำเนินการทางตรรกะ (logical operator)

(เนื้อหาตามบทที่ 6 ของหนังสือเรียน)

พื้นฐานเกี่ยวกับตัวดำเนินการ

- ตัวดำเนินการ (หรือที่นิยมเรียกทับศัพท์ว่า Operator) คือ สัญลักษณ์พิเศษต่าง ๆ ที่ทำให้เกิดการดำเนินการทางคณิตศาสตร์ ทางตรรกศาสตร์ หรือ อื่น ๆ
- ตัวอย่างโอเปอเรเตอร์ : + - * / && || > <
- โอเปอแรนด์ (operand) คือ ตัวถูกดำเนินการ ซึ่งก็คือข้อมูลที่ใช้คู่กับตัวดำเนินการต่าง ๆ ซึ่งอาจจะเป็นค่าคงที่ ตัวแปร *นิพจน์* หรือ *ฟังก์ชัน* ก็ได้
- ตัวอย่าง : y + 1
  - เครื่องหมาย + เป็นโอเปอเรเตอร์
  - y เป็นโอเปอแรนด์ที่เป็นตัวแปร
  - เลข 1 เป็นโอเปอแรนด์ที่เป็นค่าคงที่

นิพจน์

คือการนำเอาโอเปอเรเตอร์และโอเปอแรนด์หลาย ๆ ตัวมารวมกันเพื่อพิจารณาเป็นประโยคเดียวหรือค่าข้อมูลตัวเดียว เช่น

5 + 7 หากเรามองโอเปอแรนด์และโอเปอเรเตอร์สามอย่างนี้รวมกันเป็นค่าข้อมูลเดียว เรากล่าวว่า 5 + 7 เป็นนิพจน์

y + 3 หากเรามองโอเปอแรนด์และโอเปอเรเตอร์สามอย่างนี้รวมกันเป็นค่าข้อมูลเดียว เรากล่าวว่า y + 3 เป็นนิพจน์

x + y หากเรามองโอเปอแรนด์และโอเปอเรเตอร์สามอย่างนี้รวมกันเป็นค่าข้อมูลเดียว เรากล่าวว่า x + y เป็นนิพจน์

x + y - z + 2 หากเรามองโอเปอแรนด์และโอเปอเรเตอร์ 7 อย่างนี้รวมกันเป็นค่าข้อมูลเดียว เรากล่าวว่า x + y - z + 2 เป็นนิพจน์

เรื่องน่ารู้เกี่ยวกับนิพจน์

- นิพจน์มีชนิดของข้อมูลกำกับเสมอ และทำตัวเหมือนค่าคงที่หรือตัวแปร
  - นิพจน์ 5 + 7 ผลของมันคือ 12 ซึ่งก็คือจำนวนเต็ม
  - นิพจน์ y + 7 ถ้า y เป็นจำนวนเต็ม ผลบวกก็เป็นจำนวนเต็ม ดังนั้นนิพจน์นี้จึงทำตัวเหมือนจำนวนเต็ม
  - นิพจน์ x + y – z + 2 ผลของการดำเนินการบวกลบไม่ว่าจะมีสักกี่ครั้ง ย่อมให้ค่าออกมาเป็นแบบใดแบบหนึ่ง นิพจน์จึงมีชนิดข้อมูลกำกับเสมอ และทำตัวเหมือนค่าคงที่หรือว่าตัวแปร
- เราสามารถใช้วงเล็บเพื่อกำหนดลำดับการคำนวณของนิพจน์ได้ เช่น จากนิพจน์ x + y – z + 2 โดยปรกติเราจะบวกลบตามลำดับ แต่ถ้าเราอยากเปลี่ยนลำดับก็ทำได้ เช่น x + y – (z + 2)

นิพจน์ในนิพจน์ และ นิพจน์ที่ถูกต้อง

พิจารณานิพจน์ 5 * (a * b – (3 +c))

เราสามารถมองทุกอย่างรวมเป็นนิพจน์เดียวได้ และ เราก็สามารถมอง คู่หรือกลุ่มของการคำนวณ**ที่สมบูรณ์**ว่าเป็นนิพจน์ได้ นั่นคือ ในนิพจน์ก็มีนิพจน์ย่อยร่วมอยู่ได้ เช่น

มอง a * b เป็นนิพจน์, (3 + c) เป็นนิพจน์ และมองสองนิพจน์นี้รวมกันเป็น (a * b – (3 + c)) ว่าเป็นนิพจน์

แต่เราไม่สามารถมองกลุ่มของการคำนวณที่ไม่สมบูรณ์ว่าเป็นนิพจน์ได้ เช่น

a * แบบนี้ก็ไม่ได้ เพราะเราไม่สามารถคำนวณค่าของมันได้

b – (3 ก็ไม่ได้เพราะเราคำนวณค่ามันไม่ได้ แต่ถ้าเป็น

b – (3 + c) จะถือว่าเป็นนิพจน์เพราะเราคำนวณค่าของมันได้แล้ว

ดังนั้นนิพจน์ที่ถูกต้องจะต้องทำให้เราคำนวณค่าของมันออกมาได้

ตัวดำเนินการทางคณิตศาสตร์

- การดำเนินการทางคณิตศาสตร์เป็นพื้นฐานของโปรแกรมจำนวนมาก
- เป็นการนำโอเปอแรนด์มากระทำเพื่อหาค่าหาผลลัพธ์

  ย้ำอีกที โอเปอแรนด์คือตัวเลขและตัวแปร ต่าง ๆ
- มีอยู่ 7 ตัวที่ใช้บ่อย แบ่งได้เป็นสองกลุ่ม
  - กลุ่มที่ใช้กับตัวเลขหรือตัวแปรหรือนิพจน์ก็ได้ คือ

    บวก +, ลบ -, คูณ *, หาร / และ หาเศษ % (นิยมเรียกว่า mod)
  - กลุ่มที่ใช้ได้กลับตัวแปรเท่านั้น คือ

    เพิ่มหนึ่ง ++ (เครื่องหมายบวกติดกันสองอัน, นิยมเรียกว่า บวกบวก)

    ลดหนึ่ง -- (เครื่องหมายลบติดกันสองอัน, นิยมเรียกว่า ลบลบ)

    (ดูหนังสือหน้า 72 ประกอบ)

เครื่องหมายบวกลบคูณหาร

- ทำตัวเหมือนกับเครื่องหมายบวกลบคูณหารในคณิตศาสตร์ทั่วไป
  - ตัวอย่าง 9 – 3 ให้ผลลัพธ์เท่ากับ 6 และ 9 * 3 ให้ผลลัพธ์เท่ากับ 27
  - x + 4 ให้ผลเท่ากับ 4 + x (การบวกสลับตำแหน่งได้)
- ส่วนการหาร ผลลัพธ์จะขึ้นอยู่กับชนิดข้อมูลด้วย
  - ถ้าทั้งตัวตั้งและตัวหารเป็นจำนวนเต็ม ผลลัพธ์ก็จะเป็นจำนวนเต็มด้วย โดยเศษถูกปัดทิ้ง เช่น 8 / 3 ได้ผลลัพธ์เท่ากับ 2 (แม้เศษจากการหารจะเกิน 0.5 ก็ต้องโดนปัดทิ้ง)
  - แต่ถ้าเป็นเลขทศนิยม ผลหารจะเป็นไปตามปรกติ แต่ความเที่ยงของผลลัพธ์จะขึ้นอยู่กับว่าเราใช้ float หรือ double ในการคำนวณ

เครื่องหมายลบ

- เครื่องหมายลบมีอยู่สองหน้าที่ในภาษาซี
  - เครื่องหมายลบที่เป็นการลบค่ากันธรรมดา (เช่น 8 – 3)
  - เครื่องหมายลบที่เป็นการสลับค่าบวกลบ เช่น -8 เป็นการเปลี่ยนค่าจาก +8 ให้เป็น -8
- การใช้เครื่องหมาย – ผสมกับการดำเนินการอย่างอื่นอาจทำให้ดูสับสน เช่น 8 * -3 (แปด คูณ ลบสาม)
- เพื่อป้องกันความสับสนควรใส่วงเล็บให้เรียบร้อย เช่น 8 * (-3)
- Tip : ตอนที่เราเขียนโปรแกรมเอง อย่าพยายามทำอะไรที่ชวนสับสน ควรเลือกทางที่ปลอดภัย เช่น ถ้ากลัวว่า 8 * -3 มันจะมีความหมายเป็นอย่างอื่น ก็อย่าลังเลที่จะใส่วงเล็บให้มันอยู่ในรูป 8 * (-3) แทน

เครื่องหมายหาเศษ (mod)

- เขียนแทนด้วยเครื่องหมาย %
- เป็น operator แสนสะดวก และมีประโยชน์ในการคำนวณหาจำนวนที่ขาดหรือเกิน (ดูตัวอย่างการใช้จากโจทย์ผลิตปลากระป๋อง)
- **ตัวอย่าง** 9 % 4 ได้ผลลัพธ์เท่ากับ 1
  8 % 3 ได้ผลลัพธ์เท่ากับ 2
  9 % 3 ได้ผลลัพธ์เท่ากับ 0

คำถาม ปรกติการหารจะปัดเศษทิ้ง แล้วเราสามารถใช้ % ช่วยในการปัดเศษขึ้นได้หรือเปล่า ?

คำตอบ ได้ แต่จะทำยังไงให้เอาไปคิดเป็นการบ้าน (ออกสอบด้วย จริง ๆ นะ)

เครื่องหมาย ++

ทำให้ค่าตัวแปรเพิ่มขึ้นหนึ่ง เป็นตัวดำเนินการที่ใช้ได้กับตัวแปรเท่านั้น

- ตัวอย่าง

  int x = 5;

  x++; แบบนี้ใช้ได้ ทำเสร็จแล้ว x จะเพิ่มขึ้นหนึ่งเป็น 6

  ++x; แบบนี้ก็ใช้ได้ ทำเสร็จแล้ว x จะเพิ่มขึ้นอีกหนึ่ง

  (จะเอาเครื่องหมายไว้ข้างหน้าหรือข้างหลังก็เพิ่มหนึ่ง)

- แต่แบบนี้ไม่ได้

  5++; → ไม่ได้

  ++5; → ก็ไม่ได้ อย่าเอาไปใช้กับค่าคงที่

ลำดับเครื่องหมาย ++ กับผลที่เกิดขึ้น

- การใช้เครื่องหมาย ++
  - ไม่ว่าจะใส่ไว้ด้านหน้าหรือหลังตัวแปร ก็ทำให้ค่าตัวแปรเพิ่มขึ้นอีกหนึ่งเหมือนกัน
  - แต่ถ้าเราเขียนในทำนองที่ว่า y = ++x; กับ y = x++; ผลลัพธ์จะไม่เหมือนกัน

```
x = 5;
y = ++x;
```

แบบนี้ y = 6

```
x = 5;
y = x++;
```

แบบนี้ y = 5

คำถาม รู้สึกชีวิตวุ่นวายเหลือเกินกับลำดับเครื่องหมาย ++ ทำไงดีครับ ?

ลดปัญหาความสับสนของลำดับเครื่องหมาย ++

คำตอบ ก็อย่าไปใช้แบบนั้นสิ แยกเป็นสองประโยคก็ได้ จะได้ปลอดภัยไม่มึนงง

ถ้าอยากให้ y เพิ่มตาม x แยกเป็นสองประโยคแบบนี้ ยังไงก็ไม่พลาด

```
x = 5;
++x;    ทำแล้ว x เพิ่มเป็น 6
y = x;  ได้ y เป็น 6
```

```
x = 5;
x++;    ทำแล้ว x เพิ่มเป็น 6
y = x;  ได้ y เป็น 6 เหมือนกัน
```

ถ้า***ไม่***อยากให้ y เพิ่มตาม x แยกเป็นสองประโยคแบบนี้ ยังไงก็ไม่พลาด

```
x = 5;
y = x;  ชิงลงมือก่อน x เพิ่ม
++x; อยากเพิ่มก็เพิ่มไป ไม่เกี่ยว
อะไรกับ y แล้ว
```

```
x = 5;
y = x;  ชิงลงมือก่อน x เพิ่ม
x++; อยากเพิ่มก็เพิ่มไป ไม่เกี่ยว
อะไรกับ y แล้ว
```

เครื่องหมาย --

ทำให้ค่าตัวแปรลดลงหนึ่ง เป็นตัวดำเนินการที่ใช้ได้กับตัวแปรเท่านั้น

- ตัวอย่าง int x = 5;

  x--; แบบนี้ใช้ได้ ทำเสร็จแล้ว x จะลดลงหนึ่งเป็น 4

  --x; แบบนี้ก็ใช้ได้ ทำเสร็จแล้ว x จะลดลงอีกหนึ่ง

  (จะเอาเครื่องหมายไว้ข้างหน้าหรือข้างหลังก็ลดลงหนึ่ง)

- แต่แบบนี้ไม่ได้

  5--; → ไม่ได้

  --5; → ก็ไม่ได้ อย่าเอาไปใช้กับค่าคงที่

- ลำดับเครื่องหมาย -- กับการรับค่าของตัวแปรให้ เช่น y = x--; และ

  y = --x; ให้ผลเหมือนกับ ++ ถ้าแยกเป็นสองประโยคช่วยลดความสับสน

ตัวอย่าง

```
void main() {
    int x = 19;
    int d = 5;

    printf("x + d = %d\n", x + d);
    printf("x - d = %d\n", x - d);
    printf("x * d = %d\n", x * d);
    printf("x / d = %d\n", x / d);
}
```

ผลที่พิมพ์ออกมาทางจอภาพ

```
x + d = 24
x - d = 14
x * d = 95
x / d = 3
```

ผลลัพธ์ของการคำนวณนิพจน์พวกนี้เป็นจำนวนเต็ม

เราจึงใช้ %d เพื่อแสดงผล

ตัวอย่าง 2

```
void main() {
    int x = 19;
    int d = 5;

    printf("x mod d = %d\n", x % d);
    ++x;
    printf("++x = %d\n", x);

    printf("--d = %d\n", --d);
}
```

```
x mod d = 4

++x = 20

--d = 4
```

ตัวอย่างชวนงงกับลำดับของ ++ และ --

```
void main() {
    int y = 10;
    int z;

    z = y--;
    printf("z = y--; y = %d, z = %d\n", y, z);

    y = 10;
    z = --y;
    printf("z = --y; y = %d, z = %d\n", y, z);
}
```

z = y--; y = 9, z = 10

z = --y; y = 9, z = 9

ลำดับการทำงานของโอเปอเรเตอร์

คำถาม 8 + 7 * 6 ได้เท่าไหร่

คำตอบ ในโลกของคณิตศาสตร์ เราควรจะใส่วงเล็บเพื่อกำหนดลำดับการคำนวณ เพราะ 8 + (7 * 6) ไม่เท่ากับ (8 + 7) * 6

ส่วนในโลกของภาษาซี ถ้าเราไม่ใส่วงเล็บกำกับการคำนวณให้ มันจะทำเหมือนมีการใส่วงเล็บให้เรา โดยมีลำดับการใส่ที่เป็นกฎแน่นอนตายตัว ดังนี้

1. ++ และ **--** จะถูกใส่วงเล็บอัตโนมัติเข้าไปก่อนใครเพื่อน เช่น
   x++ * 9 จะถูกพิจารณาว่าเป็น (x++) * 9 และ
   9 * x++ จะถูกพิจารณาว่าเป็น 9 * (x++) เช่นกัน

   แต่ถ้ามีทั้ง **--** และ ++ อยู่ด้วยกัน การใส่วงเล็บอัตโนมัติจะเป็นไปตามลำดับการปรากฏ (แต่จุดนี้มักจะไม่มีผลอะไรในทางปฏิบัติ)

ตัวอย่างลำดับการทำงานของโอเปอเรเตอร์

```
void main() {
  int x;
  x = 9;
  printf("x++ * 9 = %d\n", x++ * 9);
  x = 9;
  printf("9 * x++ = %d\n", 9 * x++);
  x = 9;
  printf("10 -  -x++ = %d\n", 10 -  -x++);
}
```

x++ * 9 = 81

9 * x++ = 81

10 - -x++ = 19

เครื่องหมายลบตรงนี้คือ การสลับค่าบวกลบ เช่น จาก 7 เป็น -7

ลำดับการทำงานของโอเปอเรเตอร์ (2)

2. ต่อจากเครื่องหมาย ++ และ -- เครื่องหมายที่มีลำดับการคำนวณรองลงมาก็คือเครื่องหมาย – ที่ทำหน้าที่กลับเครื่องหมาย
3. อันดับสามคือ *, / และ % ถ้าหากในนิพจน์มีทั้ง * / และ % อยู่ด้วยกัน ลำดับการคำนวณจะคิดตามลำดับปรากฏ เช่น
   - จากนิพจน์ 3 / 2 * 5 % 7 ลำดับการคิดที่ได้คือ (((3 / 2) * 5) % 7)
   - จากนิพจน์ 3 % 2 * 5 / 7 ลำดับการคิดที่ได้คือ (((3 % 2) * 5) / 7)
   - จากนิพจน์ 3 * 2 / 5 % 7 ลำดับการคิดที่ได้คือ (((3 * 2) / 5) % 7)
4. โอเปอเรเตอร์ที่มีความลำดับการคิดหลังสุดคือ + และ – เช่นเดียวกับชุดที่แล้ว ถ้ามี + และ – อยู่ด้วยกัน ลำดับการคิดจะเป็นไปตามลำดับการปรากฏจากซ้ายไปขวา

ตัวอย่างลำดับการทำงานของโอเปอเรเตอร์ (2)

- นิพจน์ 5 + 3 * 7 - 4 / 2 มีโอเปอเรเตอร์อยู่สองกลุ่ม คือ กลุ่มคูณหาร และ กลุ่มบวกลบ จัดตามลำดับกลุ่มและการปรากฏ ได้ผลดังนี้
  - กลุ่มคูณหารต้องมาก่อน ดังนั้นเมื่อคิดตามลำดับการปรากฏจะได้เป็น 5 + (3 * 7) – (4 / 2)
  - เมื่อเหลือแต่กลุ่มบวกลบก็คิดตามลำดับการปรากฏอีก (5 + (3 * 7)) – (4 / 2)
- นิพจน์ 5 + 3 * 7 / 4 - 2
  - คิดกลุ่มคูณหาร ได้เป็น 5 + ((3 * 7) / 4) – 2
  - คิดกลุ่มบวกลบ ได้เป็น (5 + ((3 * 7) / 4)) – 2

การผสมกันของสองกลุ่มนี้พบได้บ่อยที่สุด ถ้ากลัวงง ให้ใส่วงเล็บไว้ตั้งแต่แรก

หัวข้อเนื้อหา

- พื้นฐานเกี่ยวกับตัวดำเนินการ
- ตัวดำเนินการทางคณิตศาสตร์ (math operator)
  - ตัวดำเนินการพื้นฐาน
  - ลำดับความสำคัญของตัวดำเนินการ
- **การแปลงชนิดข้อมูล (casting)**
  - **การแปลงโดยปริยาย**
  - **การแปลงโดยชัดแจ้ง**
- ตัวดำเนินการทางตรรกะ (logical operator)

ความสำคัญของการแปลงชนิดข้อมูล

เพราะคอมพิวเตอร์ต้องเก็บผลลัพธ์เป็นชนิดใดชนิดหนึ่ง เมื่อการดำเนินการมีชนิดข้อมูลปนกันอยู่ โปรแกรมจึงต้องเลือกว่าควรคำนวณในรูปแบบใด

เช่น ถ้าจำนวนเต็ม ลบกับ เลขทศนิยม ควรเลือกคำนวณด้วยชนิดข้อมูลใด ?

- การเลือกอาจจะถูกกำหนดโดยคนเขียนโปรแกรมโดยตรงก็ได้ ซึ่งคนเขียนจะต้องระบุชนิดลงไปโดยชัดแจ้ง การเปลี่ยนชนิดข้อมูลโดยชัดแจ้งนี้มีชื่อภาษาอังกฤษว่า Explicit Type Conversion
- หากไม่มีการระบุชนิดข้อมูลจากคนเขียนโปรแกรม ภาษาซีจะเลือกชนิดของข้อมูลให้ตามกฎการเปลี่ยนชนิดข้อมูล การเปลี่ยนโดยนัยของกฎแบบนี้มีชื่อภาษาอังกฤษว่า Implicit Type Conversion

Implicit Type Conversion

คือ การที่คอมไพเลอร์เลือกเปลี่ยนชนิดข้อมูลให้อัตโนมัติ โดยมีหลักการดังนี้

1. การแปลงชนิดข้อมูลจะเกิดขึ้นก่อนการดำเนินการกับโอเปอเรเตอร์
2. การแปลงชนิดข้อมูลจะแปลงไปสู่ชนิดข้อมูลที่มีนัยสำคัญมากกว่า
3. ลำดับนัยสำคัญของข้อมูลเป็นไปตามแสดงในสไลด์ถัดไป

ลำดับนัยสำคัญของชนิดข้อมูล

ลำดับนัยสำคัญ	ชนิดข้อมูล
1 (มีนัยสำคัญสูงสุด)	double
2	float
3	unsigned int
4	int
5	short
6 (มีนัยสำคัญต่ำสุด)	char

- เช่น ถ้าเราเอา int + float ตัวเลข, ตัวแปร, หรือ นิพจน์ที่แทน int จะถูกเปลี่ยนชนิดข้อมูลเป็น float โดยอัตโนมัติ <u>การบวกจะเกิดขึ้นทีหลัง</u>

ตัวอย่าง Implicit Type Conversion

นิพจน์	การแปลงชนิดข้อมูล
char + int	แปลง char ไปเป็น int
float + double	แปลง float ไปเป็น double
int / double	แปลง int ไปเป็น double
(short + int) / float	การแปลงเกิดขึ้นสองครั้ง 1. แปลง short ไปเป็น int 2. แปลง int (ผลที่ได้จากวงเล็บ) ไปเป็น float
double / (short * float)	การแปลงเกิดขึ้นสองครั้ง 1. แปลง short ไปเป็น float 2. แปลง float (ผลที่ได้จากวงเล็บ) เป็น double

โค้ดตัวอย่าง Implicit Type Conversion

```
int  i =  10;  float f = 3.2;
```

คำถาม ถ้าเราจะหาค่าของ i / f เราควรจะนำตัวแปรชนิด int หรือ float มาเก็บผลลัพธ์นี้ ?

คำตอบ ถ้าต้องการเก็บค่าการหารที่มีความเที่ยงสูงสุดไว้ ควรใช้ตัวแปรแบบ float มาเก็บ ไม่เช่นนั้นความเที่ยงของข้อมูลจะลดลง

```
float result_float = i / f;
int result_int = i / f;
printf("%f\n", result_float);
printf("%d\n", result_int);
```

```
3.125000
3
```

โค้ดตัวอย่าง Implicit Type Conversion (2)

- ระวังการแปลงข้อมูลกับ printf เพราะเราจะต้องระบุชนิดข้อมูลที่จะแสดงใน printf ให้เหมือนกับชนิดที่คอมไพเลอร์แปลงให้
- จากโค้ดข้างล่าง เรารู้ว่าผลลัพธ์ของ i / f เป็นชนิด float

คำถาม จะเกิดอะไรขึ้น ถ้าเราบอก printf ให้พิมพ์ค่าจำนวนเต็มออกมา

```
int  i =  10;  float f = 3.2;
printf("%f\n", i / f);
printf("%d\n", i / f);
```

```
3.125000
-104857598
```

คำตอบ ผลลัพธ์ผิดไปคนละทาง ดังนั้นความรู้เกี่ยวกับการแปลงชนิดข้อมูลจึงเป็นสิ่งที่จำเป็น แม้แต่กับเรื่องแสดงผลลัพธ์ให้ถูกต้อง

Explicit Type Conversion (Casting)

- การแปลงชนิดข้อมูลแบบนี้มีชื่อเรียกสั้น ๆ ว่า casting หรือ cast
- วิธีใช้ก็คือ ให้เอาชนิดข้อมูลที่ต้องการใส่ไว้ในวงเล็บ แล้วเอาไปไว้ด้านหน้าของค่าคงที่ ตัวแปร หรือ นิพจน์ที่เราต้องการ เช่น
  - (int) 2.5 เป็นการแปลงเลข 2.5 ให้กลายเป็นจำนวนเต็ม
    นิพจน์ผลลัพธ์ที่ได้นั้นจะตัดเลขทศนิยมทิ้ง
  - float x = 2.5;
    (int) x;
    นิพจน์ผลลัพธ์ที่ได้นั้นจะตัดเลขทศนิยมทิ้ง **แต่ตัวแปร x ยังมีค่าเท่าเดิม**
- **การ cast เป็นการเปลี่ยนค่านิพจน์ ไม่ใช่การเปลี่ยนค่าตัวแปร**
  ตัวแปรที่เรา cast นั้น ถ้าหากไม่ถูกเปลี่ยนค่าด้วย = แสดงว่าค่าเหมือนเดิม
- ส่วนชนิดข้อมูล ยังไงก็เปลี่ยนไม่ได้ ประกาศไว้เป็นอะไร ก็ต้องเป็นแบบนั้น

ตัวอย่างการทำ Casting ง่าย ๆ

```
printf("7 / (int) 2.5 = %d\n", 7 / (int)2.5);
```

```
7 / (int) 2.5 = 3
```

```
printf("(int)(7 / 2.5) = %d\n", (int)(7 / 2.5));
```

```
(int)(7 / 2.5) = 2
```

```
printf("(int) 7 / 2.5 = %f (use %%f)\n", (int)7 / 2.5);
```

```
(int) 7 / 2.5 = 2.800000 (use %f)
```

```
printf("(int) 7 / 2.5 = %d (use %%d)\n", (int)7 / 2.5);
```

```
(int) 7 / 2.5 = 1717986918 (use %d)
```

- Casting จะทำกับค่า, ตัวแปร หรือ นิพจน์ที่ตามมันมาทันที
  - ไม่มีความเปลี่ยนแปลงในตัวอย่างสาม เพราะ (int) 7 ก็คือ 7 อยู่แล้ว
  - Implicit type conversion ถูกใช้งานตามปรกติ ถ้าชนิดข้อมูลไม่ตรงกัน

เรามักใช้ Casting กับตัวแปรมากกว่าค่าคงที่

- เรามัก cast จำนวนเต็มให้กลายเป็นเลขทศนิยมเพื่อหาผลหารที่เที่ยงตรง

```
int x = 5;  int y = 2;
float f1 = x / y;

float f2 = x / (float) y;

printf("%f\n", f1);

printf("%f\n", f2);
```

ทั้ง x และ y เป็น int ทั้งคู่ การแปลงชนิดข้อมูลจึงไม่เกิดขึ้น แบบนี้ต้องบังคับการ cast

```
2.000000

2.500000
```

- อย่าคิดว่าการแปลงชนิดข้อมูลเป็นไปตามชนิดตัวแปรที่เราใช้เก็บผลลัพธ์
- แท้จริงแล้ว **การแปลงชนิดข้อมูลเกิดขึ้นก่อนการคำนวณผลลัพธ์ต่างหาก**

แปลง float เป็น int เพื่อปัดเศษทิ้ง

- หนึ่งในการใช้งานที่นิยมที่สุดในการ cast ก็คือการปัดเศษทิ้ง

```
int x = 5;  int y = 2;
float f2 = x / (float) y;
printf("%f\n", f2);

float rf2 = (int) f2;

printf("%f\n", rf2);
```

เราบังคับการปัดเศษทิ้งด้วยวิธีนี้ก็ได้

```
2.000000
```

หัวข้อเนื้อหา

- พื้นฐานเกี่ยวกับตัวดำเนินการ
- ตัวดำเนินการทางคณิตศาสตร์ (math operator)
  - ตัวดำเนินการพื้นฐาน
  - ลำดับความสำคัญของตัวดำเนินการ
- **การแปลงชนิดข้อมูล (casting)**
  - **การแปลงโดยปริยาย**
  - **การแปลงโดยชัดแจ้ง**
- **ตัวดำเนินการทางตรรกะ (logical operator)**

ตัวดำเนินการทางตรรกะ (logical operator)

- นอกจากการดำเนินการทางเลขคณิตแล้ว เรายังสังเกตเห็นการดำเนินการบางอย่าง เช่น การเปรียบเทียบจำนวน ที่ทำเพื่อทดสอบว่าตัวเลขมันมากกว่าหรือน้อยกว่าค่าที่กำหนดไว้จริงหรือไม่
- การดำเนินการเพื่อตรวจว่าเหตุการณ์เป็นจริงหรือเป็นเท็จ เรียกว่าการดำเนินการทางตรรกะ หรือ logical operator มีอยู่สองกลุ่ม
  - กลุ่มเปรียบเทียบค่า ได้แก่ >, <, >=, <=, ==, และ !=
  - กลุ่มพิจารณาค่าตรรกะ ได้แก่ &&, ||, !
- ตัวดำเนินการเหล่านี้มีพื้นฐานมาจากคณิตศาสตร์มัธยมที่เราได้พบมาแล้ว

Logical Operator กลุ่มเปรียบเทียบค่า

โอเปอเรเตอร์ในภาษาซี	ความหมายและโอเปอเรเตอร์คณิตศาสตร์
<	น้อยกว่า ในคณิตศาสตร์ใช้ < ซึ่งเป็นตัวเดียวกัน
>	มากกว่า ในคณิตศาสตร์ใช้ > ซึ่งเป็นตัวเดียวกัน
<=	น้อยกว่าหรือเท่ากับ ในคณิตศาสตร์ใช้ ≤
>=	มากกว่าหรือเท่ากับ ในคณิตศาสตร์ใช้ ≥
==	เท่ากับ ในคณิตศาสตร์ใช้ = (เพราะคณิตศาสตร์ไม่มีการกำหนดค่าตัวแปร จึงไม่มีความกำกวม)
!=	ไม่เท่ากับ ในคณิตศาสตร์ใช้ ≠

Tip : คนจำนวนมากสับสนว่าจะใช้ < หรือ <= ดี และสับสนว่าจะใช้ > หรือ >= ดี ในกรณีที่งง ให้ลองแทนตัวเลขลงไปในโปรแกรมแล้วคิดตามว่า อันไหนคือตัวที่ถูก

ผลของการเปรียบเทียบค่า

- หากเราใช้ logical operator ผลที่ได้จะออกมาเป็น จริง หรือ เท็จ เท่านั้น
  - เช่น ถ้าเราเขียนว่า 1 > 0 ผลลัพธ์ของการทำงานคือ จริง
  - เช่น ถ้าเราเขียนว่า 1 < 0 ผลลัพธ์ของการทำงานคือ เท็จ
- ใช้ logical operator กับตัวแปรหรือตัวเลขก็ได้ เช่น

  จาก int x = 1; int y = 0;
  - x > y; ผลลัพธ์ที่ได้จากการเปรียบเทียบคือ จริง
  - x < y; ผลลัพธ์ที่ได้จากการเปรียบเทียบคือ เท็จ
  - x > 0; ผลลัพธ์ที่ได้จากการเปรียบเทียบคือ จริง
  - 1 > y; ผลลัพธ์ที่ได้จากการเปรียบเทียบคือ จริง
  - x < 0; ผลลัพธ์ที่ได้จากการเปรียบเทียบคือ เท็จ
  - 1 < y; ผลลัพธ์ที่ได้จากการเปรียบเทียบคือ เท็จ

Logical Operator กลุ่มพิจารณาค่าตรรกะ

โอเปอเรเตอร์ในภาษาซี	ความหมายและโอเปอเรเตอร์คณิตศาสตร์
&&	และ (AND) ในคณิตศาสตร์ใช้ ^ เช่น p ^ q (ทบทวนได้ในเรื่องตรรกศาสตร์)
||	หรือ (OR) ในคณิตศาสตร์ใช้ v เช่น p v q
!	นิเสธ (NOT) ในคณิตศาสตร์ใช้ ~ เช่น ~p ซึ่งเป็นการสลับค่าความจริง

ผลของการใช้ตัวดำเนินการกลุ่มพิจารณาค่าตรรกะ (1)

1. ตัวดำเนินการ &&
   - ถ้า A และ B เป็นจริงทั้งคู่
     (เช่น ถ้า A คือ 1 > 0 และ B คือ 5 > 3 ซึ่งเป็นจริงทั้งคู่)
     นิพจน์ A && B ให้ผลลัพธ์เป็น จริง
   - ถ้า A และ B มีอย่างน้อยหนึ่งตัวที่เป็นเท็จ (คือมีความเท็จปนอยู่)
     นิพจน์ A && B ให้ผลลัพธ์เป็น เท็จ
2. ตัวดำเนินการ ||
   - ถ้า A และ B มีอยู่อย่างน้อยหนึ่งตัวที่เป็นจริง
     นิพจน์ A || B ให้ผลลัพธ์เป็นจริง
   - ถ้า A และ B เป็นเท็จทั้งคู่
     นิพจน์ A || B ให้ผลลัพธ์เป็นเท็จ

ผลของการใช้ตัวดำเนินการกลุ่มพิจารณาค่าตรรกะ (2)

3. ตัวดำเนินการ !
   (ตัวดำเนินการนี้ใช้สลับค่าความจริง ต้องการโอเปอแรนด์แค่ตัวเดียว)
   - ถ้า A เป็นจริง !A ให้ผลลัพธ์เป็น เท็จ (ค่าสลับจากจริงเป็นเท็จ)
   - ถ้า A เป็นเท็จ !A ให้ผลลัพธ์เป็น จริง (ค่าสลับจากเท็จเป็นจริง)

เราสรุป Logical Operator กลุ่มนี้ได้ว่า

1. สำหรับ && โอกาสที่ผลลัพธ์จะเป็นค่าจริงมีน้อยกว่า || เพราะต้องให้ A และ B เป็นจริงทั้งคู่ ในขณะที่ || เป็นจริงแค่ตัวเดียวก็พอแล้ว
2. && และ || ต้องมีนิพจน์ค่าความจริงสองตัว ส่วน ! ใช้กับนิพจน์ค่าความจริงทีละนิพจน์เท่านั้น

เรื่องของค่าจริงค่าเท็จในภาษาซี

เรื่องนี้ค่อนข้างจะลึกลับอยู่ไม่น้อย ต้องตั้งสติแยกแยะให้ดี

เรื่องมันมีอยู่ว่า

1. โดยปรกติเครื่องคอมพิวเตอร์เก็บได้แต่ตัวเลข ไม่ได้มีของพิเศษที่เป็น ค่าจริงและค่าเท็จ ทำให้เราต้องเอาตัวเลขมาแทนค่าจริงค่าเท็จ
2. ภาษาซีมีข้อกำหนดว่า ผลของการใช้ logical operator จะให้ผลลัพธ์เป็นเลข 1 ถ้านิพจน์เป็นจริง และ เป็นเลข 0 ถ้านิพจน์เป็นเท็จ
3. แต่คำว่า จริง ในภาษาซีนั้น มันคือตัวเลขทุกตัวที่ไม่เป็น 0
   ส่วนคำว่า เท็จ มีอยู่แค่ค่าเดียว คือเลข 0 เท่านั้น
   เพียงแต่ logical operator เลือกคืนเฉพาะเลข 0 กับ 1 มาให้

(ดูหนังสือเรียนหน้า 83 – 85 เพื่อตัวอย่างเพิ่มเติมด้วย จะได้ไม่พลาด)

ตัวอย่างผลการใช้ Logical Operator (1)

นิพจน์	ค่าตรรกะ (ตัวเลขในวงเล็บคือค่าที่เป็นผลลัพธ์)
30 > 20	จริง (1)
20 >= 30	เท็จ (0)
5 == 5	จริง (1)
5 != 5	เท็จ (0)
5 != 3	จริง (1)
สมมติให้ int x = 3; x == 3	จริง (1)
สมมติให้ int x = 3; x != 3	เท็จ (0)

สำหรับตัวแปร ผลจะเป็นจริงหรือเท็จ ต้องดูที่ค่าของตัวแปรขณะทำการเปรียบเทียบ

ตัวอย่างผลการใช้ Logical Operator (2)

นิพจน์	ค่าตรรกะ (ตัวเลขในวงเล็บคือค่าที่เป็นผลลัพธ์)
(10 > 1) && (1 >= 1)	จริง (1)
(10 > 1) || (1 >= 1)	จริง (1)
!(10 > 1)	เท็จ (0) (10 > 1 เป็นจริง แต่โดน ! สลับค่า)
!(3 > 5)	จริง (1) (3 > 5 เป็นเท็จ แต่โดน ! สลับค่า)
(10 > 1) && (1 > 2)	เท็จ (0)
(10 > 1) || (1 > 2)	จริง (1) (ขอแค่มีอย่างน้อยตัวหนึ่งที่เป็นจริงก็พอ)
10 && 2	จริง (1) (โอเปอเรเตอร์คืนได้แค่ 1 กับ 0)
10 && 0	เท็จ (0)

ตัวอย่างการใช้ logical operator ในโปรแกรม (1)

*** หากเราจะพิมพ์ผลลัพธ์ของ logical operator ออกมา ให้ใช้ %d ตลอด เพราะผลลัพธ์ออกมาในรูปจำนวนเต็ม 1 หรือ 0 เท่านั้น

*** ถ้าจะเอาตัวแปรมาเก็บผลของ logical operator ก็ให้ใช้ int มาเก็บ

```
printf("30 > 20 is %d\n", 30 > 20);
```
```
30 > 20 is 1
```
```
printf("10 == 10 is %d\n", 10 == 10);
```
```
10 == 10 is 1
```
```
printf("10 == 1 is %d\n", 10 == 1);
```
```
10 == 1 is 0
```
```
printf("10 != 1 is %d\n", 10 != 1);
```
```
10 != 1 is 1
```

ตัวอย่างการใช้ logical operator ในโปรแกรม (2)

```
printf("(10 == 1) && (10 != 1) is %d\n",
        (10 == 1) && (10 != 1));
```

```
(10 == 1) && (10 != 1) is 0
```

```
printf("(10 == 1) || (10 != 1) is %d\n",
        (10 == 1) || (10 != 1));
```

```
(10 == 1) || (10 != 1) is 1
```

```
printf("!(10 == 1) is %d\n", !(10 == 1));
```

```
!(10 == 1) is 1
```

การจะไล่ค่าผลลัพธ์ได้อย่างถูกต้อง เราจะต้องรู้ค่าความจริงแต่ละส่วนให้ได้เป็นอย่างดี (ตอนสอบควรไล่ค่าและบันทึกค่าจริงเท็จแต่ละส่วนไว้ด้วย)

ตัวอย่างการใช้ logical operator ในโปรแกรม (3)

- เราสามารถใช้เครื่องหมายนิเสธซ้อนกันได้
- เราสามารถใช้เครื่องหมายนิเสธกับนิพจน์ที่มาจากหลายส่วนรวมกันได้

```
printf("!!(10 == 1) is %d\n", !!(10 == 1));
```

```
!!(10 == 1) is 0
```

```
printf("!((10 == 1) || (10 != 1)) is %d\n",
       !((10 == 1) || (10 != 1)));
```

```
!((10 == 1) || (10 != 1)) is 0
```

ตัวอย่างการใช้ logical operator ในโปรแกรม (4)

ตัวแปรจะดำเนินการเลขคณิตด้วยก็ได้

นี่เป็นสิ่งที่เราทำบ่อย ๆ กับเรื่องการวนทำซ้ำ

```
int x = 5;  int y = 3;  int z = 3;
printf("y == z is %d\n", y == z);
```

```
y == z is 1
```

```
printf("y == z + 1 is %d\n", y == z + 1);
```

```
y == z + 1 is 0
```

```
printf("y + 2 >= x is %d\n", y + 2 >= x);
```

```
y + 2 >= x is 1
```

```
printf("!(y + z <= x*x) is %d\n",
        !(y + z <= x*x));
```

```
!(y + z <= x*x) is 0
```

สรุปเรื่องโอเปอเรเตอร์เลขคณิต

- มีสองกลุ่มคือ กลุ่มที่ใช้กับตัวเลขและตัวแปรได้ (+, -, *, / และ %) และกลุ่มที่ใช้ได้กับตัวแปรอย่างเดียว (++ และ --)
- เครื่องหมายลบมีสองความหมาย : (1) ถ้าใช้กับค่าสองค่ามันเหมือนกับลบเลขทั่วไป และ (2) ถ้าใช้กับค่าค่าเดียวมันหมายถึงการสลับเครื่องหมาย
- อย่าลืมว่าเรามี % ซึ่งใช้ในการหาเศษด้วย

  → ตัวนี้แหละที่ช่วยให้เราหาว่าเลขเป็นคู่หรือคี่ได้ง่าย ๆ
- ลำดับการทำงานของโอเปอเรเตอร์มีกฎควบคุมตายตัว

  → แต่วงเล็บมีความสำคัญสูงสุดเสมอ ถ้าไม่แน่ใจให้ใส่วงเล็บช่วย
- เครื่องหมาย ++ และ -- มีเรื่องชวนงงตอนที่เราจะให้ตัวแปรมาเก็บผลไว้

  → อย่าเสี่ยงทำเรื่องชวนงง ให้เราแยกบรรทัดออกมาเลยจะดีกว่า

สรุปเรื่องการแปลงชนิดข้อมูล

- ความสอดคล้องของชนิดข้อมูลเป็นสิ่งที่สำคัญ
  - ถ้าชนิดข้อมูลไม่ตรงกัน เครื่องจะคำนวณไม่ได้ (จึงต้องมีการทำ Type conversion)
  - ถ้าเราไม่เปลี่ยนชนิดข้อมูลให้ตรงกัน คอมไพเลอร์จะเปลี่ยนให้
- การเปลี่ยนชนิดข้อมูลแบบอัตโนมัติจะแปลงไปหากลุ่มที่มีนัยสำคัญสูงกว่า
- บางครั้งการเปลี่ยนชนิดข้อมูลด้วยตัวเราเอง (casting) เป็นสิ่งที่จำเป็น
  - เช่น เราต้องการบังคับให้การหารมีผลเป็นเลขทศนิยม
  - เช่น เราต้องการให้การปัดเศษทิ้ง
- ข้อมูลแบบทศนิยมมีนัยสำคัญสูงกว่าข้อมูลแบบจำนวนเต็ม

สรุปเรื่องโอเปอเรเตอร์ตรรกะ

- มีสองกลุ่มคือ กลุ่มที่ใช้ในการเปรียบเทียบค่า และ กลุ่มที่ใช้พิจารณาค่าความจริง
  - ทั้งสองกลุ่มให้ผลลัพธ์ออกมาเป็น จริง (1) และ เท็จ (0) เท่านั้น
- เลขศูนย์ แปลว่า เท็จ ส่วนจำนวนอื่น ๆ แปลว่า จริง
- ผลของการดำเนินการนี้มีชนิดข้อมูลเป็นจำนวนเต็ม
- การเปรียบเทียบค่าตัวแปร ต้องดูค่าที่บรรทัดนั้น ๆ
  - อย่าไปคิดดูค่าตอนต้นอย่างเดียว เราต้องไล่โปรแกรมเพื่อหาค่าตัวแปรในขณะทำการเปรียบเทียบให้ได้
- เครื่องหมาย == ใช้ในการเปรียบเทียบ และเครื่องหมาย = ใช้ในการกำหนดค่าตัวแปร

แบบฝึกหัด

ข้อ 1, 2 และ 3 หน้า 91 และ 92

ฝึกไล่ผลลัพธ์ในตัวอย่างที่ให้ไป (อย่าแอบดูคำตอบ ให้คิดก่อนแล้วค่อยดู)

บทที่ 7 การรับและแสดงผลข้อมูล

จากเดิมที่ผ่านมา เราทำการแสดงผลออกมาหลายอย่างแล้ว แต่ยังไม่มีการรับข้อมูลเข้ามาจากผู้ใช้ (เราใช้วิธีกำหนดค่าต่าง ๆ เข้าไปในตัวแปรโดยตรง)

มาถึงตอนนี้เราพร้อมที่จะรับข้อมูลจากผู้ใช้ พร้อมทั้งทำให้การแสดงผลอยู่ในรูปแบบที่ตรงกับความต้องการมากขึ้น

(อย่าลืมทบทวนเนื้อหาที่ให้ไว้ช่วงท้ายสัปดาห์ที่สาม เราจะข้ามตรงนั้นไปแล้วเข้าประเด็นส่วนที่ซับซ้อนขึ้นเลย)

ทบทวนเรื่องการแสดงผล

- เราแสดงผลด้วยการใช้คำสั่ง printf ซึ่งสามารถเรียกใช้ได้หาก เราได้ระบุ #include <stdio.h> ไว้ที่ต้นไฟล์
- printf ช่วยให้เราแสดงข้อความที่ต้องการบนหน้าจอได้
  - ข้อความจะเป็นแบบกำหนดไว้ตายตัว เช่น printf(“Hello World”);
  - เป็นค่าจากตัวตัวแปร เช่น printf(“%d”, x);
  - หรือจะเป็นข้อความกำหนดตายตัวผสมกับตัวแปรก็ได้ เช่น printf(“The value of x is %d”, x);
- จุดที่เป็นตัวแปรจะต้องเขียนโดยใช้เครื่องหมาย % ตามด้วยชนิดข้อมูลที่ตรงกับตัวแปรที่ต้องการ
- ลำดับการปรากฏของเครื่องหมาย % กับตัวแปรจะต้องสอดคล้องกัน

รูปแบบการใช้งาน printf

printf มีรูปแบบการใช้งานดังแสดงข้างล่างนี้

```
printf(“string_format”, data_list);
```

string_format คือ ข้อความที่ต้องการแสดงผล ภายในสามารถประกอบไปด้วย ข้อความทั่วไป ตัวเลข อักขระพิเศษ รวมทั้งชนิดข้อมูลที่ต้องการแสดงผล

data_list คือ รายการของข้อมูลที่ต้องการแสดงผล ซึ่งก็คือ ค่าคงที่ ตัวแปร และนิพจน์ใด ๆ ที่คำนวณค่าออกมาได้
แต่ที่พบบ่อยจะเป็น ตัวแปร และ นิพจน์

ลำดับการแสดงผลและนิพจน์

คำถาม หากเราต้องการแสดงค่า float f = 0.5; และ int i = 2; โดยแสดงค่า f ก่อน ตามด้วยช่องว่าง แล้วจึงแสดงค่า i แบบนี้เราควรจะเขียนอย่างไร ?

คำตอบ ควรเขียนว่า `printf("%f %d", f, i);` จะสะดวกที่สุด

บรรทัดนี้หมายความว่า ให้แสดง %f ซึ่งเป็นข้อมูลแบบ float ก่อน จากนั้นตามด้วยช่องว่าง (เราต้องพิมพ์ space ลงไปด้วยจริง ๆ) จากนั้นจึงตบท้ายด้วย %d ซึ่งเป็นข้อมูลแบบ int

ลำดับการแสดงผลข้อมูลที่กำหนดไว้ใน string_format จะต้องตรงกับลำดับค่า (ในที่นี้คือตัวแปร) ที่ให้ไว้ใน data_list อย่าคิดว่าชนิดข้อมูลคือตัวกำหนด คือ**อย่า**เขียนว่า `printf("%f %d", i, f);` ภาษาซี**ไม่ได้**จัดลำดับใหม่ให้เรา

ตัวอย่างและผลลัพธ์ที่ได้

```
#include <stdio.h>
void main() {
    float f = 0.5;
    int i = 2;
    printf("%f %d", f, i);
}
```

```
0.500000 2
```

คำถาม ผลลัพธ์ที่ได้มีเลขทศนิยมตั้งหลายตัว เรากำหนดได้มั้ยว่าเอากี่ตัว ?

คำตอบ เรากำหนดได้โดยง่ายว่าให้แสดงตัวเลขหลังทศนิยมกี่ตัว แต่มันเป็นเรื่องยากที่เราจะบอกว่าให้ตัดเลขศูนย์ที่ตามมาทิ้งทั้งหมดโดยไม่ทราบล่วงหน้า ในคอร์สนี้เราจะจัดการเฉพาะกรณีแรกคือกำหนดจำนวนหลักหลังทศนิยมไว้แบบตายตัว

การกำหนดจำนวนหลักหลังเลขทศนิยม

เราทำได้ด้วยการปรับแต่ง %f

เช่นหากต้องการแสดงเลขทศนิยม 3 ตัว ให้ใช้

`%.3f` (เปอร์เซ็นต์ จุด สาม เอฟ)

ถ้าอยากได้หลักเดียวให้ใช้

`%.1f` (เปอร์เซ็นต์ จุด หนึ่ง เอฟ)

และถ้าไม่ต้องการให้แสดงเลขทศนิยมเลยให้ใช้

`%.0f` (เปอร์เซ็นต์ จุด หนึ่ง เอฟ)

ตัวอย่างการกำหนดจำนวนหลักเลขทศนิยม

```
#include <stdio.h>
void main() {
    printf("%.3f\n", 12.3456789);
    printf("%.1f\n", 12.3456789);
    printf("%.0f\n", 12.3456789);
}
```

```
12.346
12.3
12
```

นิพจน์ตรงนี้โดยมากเป็นตัวแปร แต่ที่จริงจะเป็นค่าคงที่หรืออะไรก็ได้ที่มีผลลัพธ์ออกมา

- การปัดเศษในการแสดงผลตรงนี้ ไม่ใช่การปัดทิ้ง แต่เป็นการปัดหาค่าที่ใกล้เคียง (คือปัดขึ้นหรือปัดลงก็ได้ทั้งนั้น)
- หากตัวเลขหลังการตัดเป็นเลข 5 หรือมากกว่าจะเป็นการปัดขึ้น
- หากตัวเลขหลังการตัดเป็นเลข 4 หรือน้อยกว่าจะเป็นการปัดลง

การรับข้อมูล (Input)

- ภาษาซีมีหลายคำสั่งที่ใช้รับข้อมูลจากคีย์บอร์ด
- คำสั่ง scanf มีความสามารถที่สอดคล้องกับสิ่งที่เราต้องการมากที่สุด
- การรับข้อมูลเข้า ใช้การกำหนดชนิดตัวแปรคล้ายกับ printf เช่น scanf(“%d”, &x); เป็นการรับข้อมูลเข้ามาเก็บไว้ที่ตัวแปร x
- รูปแบบการใช้งานคือ

```
scanf(“string_format”, address_list);
```

- string_format ของ scanf แต่มีความแตกต่างกันกับ printf พอสมควร

string_format ของ scanf

- ใช้เครื่องหมาย % เพื่อกำหนดชนิดข้อมูลที่จะรับเข้ามา (ตรงนี้เหมือน printf)
- แต่เนื่องจาก scanf ไม่ได้ใช้ในการแสดงผล อย่าใส่อะไรเข้าไปปนนอกจากการกำหนดชนิดข้อมูลและช่องว่างเท่านั้น
  - เช่น `scanf("%d %d", &x, &y);` แบบนี้ใช้ได้ เพราะมีแค่การกำหนดชนิดข้อมูลและช่องว่าง (ช่องว่างที่จริงไม่ต้องใส่ก็ได้)
  - **อย่า**เขียนว่า scanf("value of x is %d", &x); **แบบนี้ใช้ไม่ได้ เพราะว่ามีการใส่ข้อความแสดงผลลงไปด้วย**
  - **อย่า**ใช้อักขระพิเศษ เช่น \n เป็นต้น ตัวอย่าง scanf("%d\n", &x); **แบบนี้ใช้ไม่ได้ เพราะอักขระพิเศษใช้ในการแสดงผล ไม่ใช่การรับค่า**

address_list ใน scanf

- ใน printf นั้นส่วนด้านท้ายคือ data_list ซึ่งเป็นรายการข้อมูลที่จะนำมาใช้แสดงผล
- แต่ scanf ต้องการ address_list ซึ่งเป็นรายการ**ที่อยู่**ของตัวแปร
- data_list ของ printf จะเป็นค่าคงที่ ตัวแปร หรือ นิพจน์อะไรก็ได้ แต่ address_list จะเกี่ยวกับ**ที่อยู่ตัวแปรเท่านั้น**
- ที่อยู่ของตัวแปร คือ ที่ที่คอมพิวเตอร์ใช้เก็บค่าของตัวแปรเอาไว้
- เราส่งที่อยู่ของตัวแปรที่เราสนใจไปให้ scanf ได้ผ่านการใช้ตัวดำเนินการพิเศษ &
  - เช่น ถ้าต้องการส่งที่อยู่ของตัวแปร x ให้เขียนว่า &x เป็นต้น
  - ตัวแปรทั่วไปทุกตัวต้องมี & ไว้ทางด้านหน้า

ตัวอย่างโค้ด scanf

```
#include <stdio.h>
void main() {
    int x = 3;
    printf("Value of x is %d\n", x);
    scanf("%d", &x);
    printf("New value of x is %d\n", x);
}
```

ผลลัพธ์ตรงนี้ เป็นไปตามตัวเลขที่ผู้ใช้ป้อนให้กับ `scanf`

- เมื่อโปรแกรมทำงานจนถึงคำสั่ง scanf โปรแกรมจะหยุดรอให้ผู้ใช้ใส่ค่าเข้าไป เราจะเห็น cursor เครื่องหมายขีดเส้นใต้กระพริบรอผู้ใช้
- ถ้าเราใส่ค่าผิดประเภทเข้าไป ค่า x อาจจะไม่เปลี่ยนแปลงหรือโปรแกรมจะล้มเหลว (crash) ขึ้นอยู่กับคอมไพเลอร์ที่ใช้

ความสำคัญของ & ใน address list

ถ้าเราลืมใส่ & หน้าตัวแปรผลลัพธ์จะผิดพลาด โดยที่ค่าของตัวแปรอาจจะไม่เปลี่ยน และบางครั้งก็ทำให้โปรแกรมแครชได้

```
#include <stdio.h>
void main() {
    int x = 3;
    printf("Value of x is %d\n", x);

    scanf("%d", x);

    printf("New value of x is %d\n", x);
}
```

ลืมใส่ & แบบนี้โปรแกรมจะผิดหรือแครชได้

การรับค่า input หลายตัวพร้อมกัน

- คำสั่ง scanf สามารถรับข้อมูลเข้าหลายตัวพร้อมกันได้ในคำสั่งเดียว เช่น scanf(“%d %f %d”, &x, &y, &z);
- ลำดับการป้อนข้อมูลเข้าของผู้ใช้จะเป็นไปตามลำดับของตัวแปรใน string format และ address list
- ผู้ใช้แยกค่าของตัวแปรแต่ละตัวออกมาได้ด้วยของสามอย่าง คือ
  - การขึ้นบรรทัดใหม่ (ปุ่ม Enter)
  - ช่องว่าง (ปุ่ม Space bar)
  - เลื่อนตำแหน่งกั้นหน้า (ปุ่ม Tab)

ตัวอย่างการป้อนข้อมูลให้ scanf ที่รับหลายค่าพร้อมกัน

```
void main() {
    int x;
    float y;
    int z;
    scanf("%d %f %d", &x, &y, &z);
    printf("Inputs are %d, %f, %d\n", x, y, z);
}
```

เราสามารถใส่ค่าเข้าไปได้หลายแบบที่ให้ผลเหมือนกัน คือ ไม่ว่าเราจะใช้การขึ้นบรรทัดใหม่ ช่องว่าง การกดแท็บ หรือผสมกัน ผลลัพธ์ก็เหมือนกันหมด ตราบเท่าที่ตัวเลขเป็นตัวเดียวกัน (แต่ตัวสุดท้ายต้องกด Enter เสมอ)

```
5 6 7
Inputs are 5, 6.000000, 7
```

ตัวอย่างการป้อนข้อมูลให้ scanf แบบต่าง ๆ (1)

- แบบใช้ช่องว่างคั่นข้อมูลทั้งหมด

```
5 6 7
Inputs are 5, 6.000000, 7
```

- แบบใช้การขึ้นบรรทัดใหม่ (แบบที่คนทั่วไปรู้จักดีที่สุด)

```
5
6
7
Inputs are 5, 6.000000, 7
```

- แบบใช้แท็บ

```
5       6       7
Inputs are 5, 6.000000, 7
```

ตัวอย่างการป้อนข้อมูลให้ scanf แบบต่าง ๆ (2)

แบบผสมตัวคั่นข้อมูลหลายแบบ

```
5       6
7
Inputs are 5, 6.000000, 7
```

```
5
6 7
Inputs are 5, 6.000000, 7
```

```
5 6
7
Inputs are 5, 6.000000, 7
```

ถ้าใส่ข้อมูลเกินล่ะ ?

มีความเป็นไปได้สองแบบ

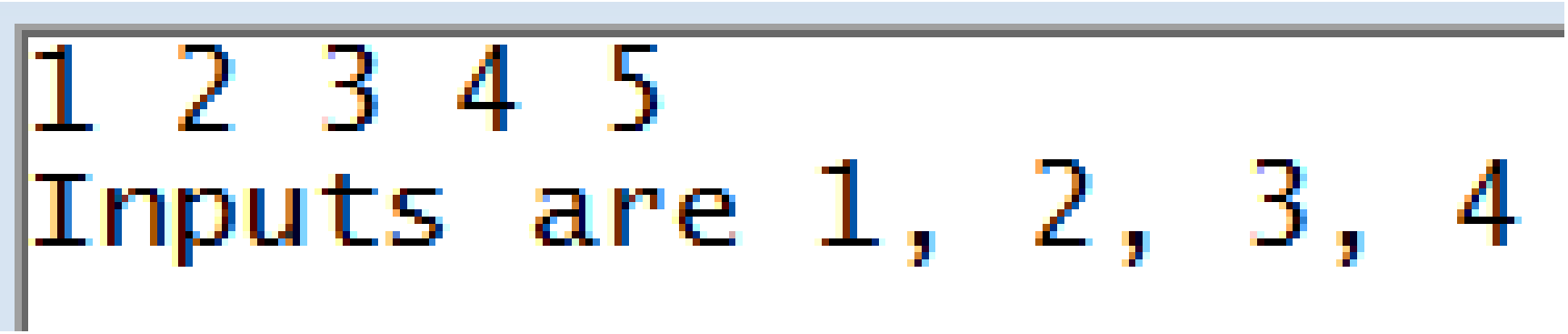
1. ถ้าโปรแกรมไม่ได้ทำคำสั่ง scanf อีกเลย ข้อมูลที่เกินมาจะถูกทิ้งไป
2. ถ้าโปรแกรมทำคำสั่ง scanf ในเวลาต่อมา ข้อมูลที่เกินมาจะถูกนำไปป้อนให้กับ scanf ที่ตามมา

ลองศึกษาจากตัวอย่างนี้

```
int w, x, y, z;
scanf("%d %d", &w, &x);
scanf("%d", &y);
scanf("%d", &z);
printf("Inputs are %d, %d, %d, %d\n", w, x, y, z);
```

scanf ตัวแรกรับค่าแค่สองตัว จะเกิดอะไรขึ้นถ้าเราใส่เลขเข้าไป 5 ตัวเลย

ทดสอบการใส่ข้อมูลเกิน



- ข้อมูลที่เกินมาถูกโอนต่อไปยัง scanf ที่ตามมาต่อเนื่องไปได้เรื่อย ๆ
- ถ้าไม่มี scanf มารับไว้แล้ว สุดท้ายข้อมูลก็ถูกทิ้งไป
  ไม่มีผลอะไรกับผลลัพธ์และโปรแกรมทั้งสิ้น

เรื่องลึกลับของการรับข้อมูลเข้ายังมีอยู่อีก
แต่เราจะเก็บไว้จนกว่าจะถึงเวลาอันสมควร

แบบฝึกหัดท้ายบท

ให้นักศึกษาทำแบบฝึกหัดท้ายบทที่ 7 ดังนี้

ข้อ 1 และ 2 เป็นการฝึกไล่ค่าในโปรแกรม (ข้อสอบก็ออกแบบนี้เหมือนกัน)

ข้อ 3 เป็นการแปลงโฟลวชาร์ตให้กลายเป็นโค้ด (ข้อสอบก็ออกแบบนี้อีกนั่นแหละ)

ข้อ 4 เป็นการเขียนโปรแกรม (ข้อสอบออกแน่ ๆ ยังไงก็หนีไม่พ้น)

ไม่ต้องทำข้อ 5 เพราะอยู่นอกเหนือสิ่งที่สอนในวันนี้ เราจะใช้วิธีอื่นที่ใช้ได้กว้างขวางกว่าในช่วงครึ่งหลังการสอบกลางภาค