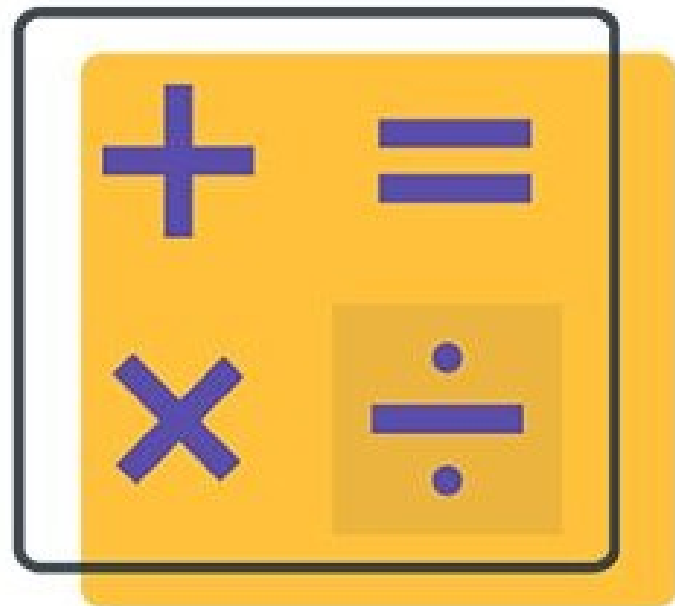




Computer Programming I: การเขียนโปรแกรมคอมพิวเตอร์ I

ตัวดำเนินการ (Operators) การรับข้อมูลเข้า และการแสดงผลลัพธ์



อ.ดร.ปัญญานต์ อ้นพงษ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

aonpong_p@su.ac.th

Outline



- พื้นฐานเกี่ยวกับตัวดำเนินการ
- ตัวดำเนินการทางคณิตศาสตร์ (Math operators)
 - ตัวดำเนินการพื้นฐาน
 - ลำดับความสำคัญของตัวดำเนินการ
- การแปลงชนิดข้อมูล (Casting)
 - การแปลงโดยปริยาย
 - การแปลงโดยชัดแจ้ง
- ตัวดำเนินการทางตรรกะ (Logical Operator)

พื้นฐานเกี่ยวกับตัวดำเนินการ



- ตัวดำเนินการหรือ Operator คือสัญลักษณ์ที่ใช้ดำเนินการทางคณิตศาสตร์ทางตรรกศาสตร์ หรือ อื่น ๆ
 - ที่เราเคยทำความรู้จักไปแล้ว เช่น $+$, $-$, $*$, $/$, $\%$ (mod), $>$, $<$ เป็นต้น
 - จริง ๆ แล้วยังมีสัญลักษณ์อื่น ๆ อีก จะแนะนำให้รู้จักในวันนี้
- โอเปอเรนด์ หรือ Operand คือตัวถูกดำเนินการ ซึ่งก็คือข้อมูลที่ใช้กับตัวดำเนินการต่าง ๆ อาจเป็นค่าคงที่ นิพจน์ ตัวแปร หรือฟังก์ชันก็ได้
- ตัวอย่างเช่น $y+1$
 - เครื่องหมาย $+$ เป็น operator ส่วน y และ 1 เป็น operand

นิพจน์



- คือการนำเอา operator และ operand หลาย ๆ ตัวมารวมกัน เพื่อพิจารณาเป็นประโยคเดียวหรือค่าข้อมูลตัวเดียว
- ตัวอย่าง

$5 + 7$ เป็นนิพจน์

$y + 3$ เป็นนิพจน์

$x + y$ เป็นนิพจน์

$x + 2 + z - y$ เป็นนิพจน์

- นิพจน์จะมี Data type กำกับอยู่เสมอ (ทำหน้าที่เหมือนค่าคงที่หรือตัวแปร)
 - ผลของนิพจน์ $5 + 7$ คือ 12 (int)
 - ผลของนิพจน์ $y + 3$ ขึ้นอยู่กับ y
 - *ถ้า y เป็น int ผลของมันจะเป็น int ด้วย
 - **ถ้า y เป็น float/double ผลของมันจะเป็น float/double ด้วย
 - ผลของนิพจน์ $x + 2 + z - y$ ไม่ว่าจะทำการคำนวณสักกี่ครั้ง ผลของแต่ละคู่ก็จะมี Data type กำกับ นั่นทำให้ไม่ว่านิพจน์จะยาวแค่ไหน สุดท้ายก็จะมี Data type กำกับด้วยเสมอ


นิพจน์ในนิพจน์ และนิพจน์ที่ถูกต้อง

- พิจารณานิพจน์นี้

$$5 * (a * b - (3 + c))$$

เราสามารถมองทุกอย่างรวมเป็นนิพจน์เดี่ยวได้ และเราก็สามารถมอง คู่ หรือกลุ่มของการคำนวณที่สมบูรณ์ว่าเป็นนิพจน์ได้ นั่นคือ ในนิพจน์ก็มีนิพจน์ย่อยรวมอยู่ได้ เช่น

$a * b$ เป็นนิพจน์, $(3 + c)$ เป็นนิพจน์ และมองสองนิพจน์นี้รวมกันเป็น


$$a * b - (3 + c) \quad \leftarrow \text{ก็เป็นนิพจน์}$$

นิพจน์ในนิพจน์ และนิพจน์ที่ถูกต้อง

- พิจารณานิพจน์นี้

$$5 * (a * b - (3 + c))$$

แบบไหนถึงจะเรียกว่า **ไม่ใช่** นิพจน์

$$5 * (a \quad \leftarrow \text{ไม่ใช่นิพจน์}$$

$$a * \quad \leftarrow \text{ไม่ใช่นิพจน์}$$

$b - (3 + c) \leftarrow$ เป็นนิพจน์แล้ว ถึงแม้ว่าถ้าพิจารณากับนิพจน์ด้านบนแล้วลำดับการคำนวณจะไม่ถูกต้อง

ตัวดำเนินการทางคณิตศาสตร์

- ตัวดำเนินการทางคณิตศาสตร์เป็นพื้นฐานของโปรแกรมจำนวนมาก
 - เป็นการนำโอเปอเรนด์มาหาผลลัพธ์
 - มี 7 ตัวที่ใช้บ่อย
 - กลุ่มที่ใช้กับตัวแปรหรือตัวเลขหรือนิพจน์ก็ได้ เช่น $+$, $-$, $*$, $/$, $\% \text{ (mod)}$
 - กลุ่มที่ใช้กับตัวแปรเท่านั้น คือ $++$, $--$
 - $++$ คือการเพิ่มค่าตัวแปรนั้นขึ้น 1 เช่น $x++$; มีค่าเท่ากับ $x = x + 1$;
 - $--$ คือการลดค่าตัวแปรนั้นลง 1 เช่น $x--$; มีค่าเท่ากับ $x = x - 1$;
- * ไม่สามารถใช้ $++$ และ $-$ กับตัวเลขหรือนิพจน์ได้ เช่น $8++$; จะไม่สามารถทำได้

การบวกลบคูณหาร

- ทำตัวเหมือนวิชาคณิตศาสตร์ทั่วไป
- **การหารจะมีความพิเศษนิดหน่อย เพราะผลลัพธ์จะขึ้นอยู่กับชนิดของข้อมูล**
 - ถ้าทั้งตัวตั้งและตัวหารเป็นจำนวนเต็ม ผลลัพธ์ก็จะเป็นจำนวนเต็มด้วย แม้จะหารไม่ลงตัว เศษก็จะถูกปัดทิ้ง เช่น $8/3$ จะให้คำตอบเป็น 2
 - แต่ถ้าเป็นเลขทศนิยม ผลลัพธ์ก็จะให้ค่าตามปกติ อย่างที่คุ้นเคย (ความเที่ยงตรงขึ้นอยู่กับชนิดของข้อมูล; Float/Double)

เครื่องหมายลบ



- ใช้ได้ 2 ลักษณะ นักศึกษาน่าจะคุ้นเคยอยู่แล้ว
 - ใช้ลบตัวเลขสองตัว เช่น $9 - 2$, $x - y$ เป็นต้น
 - ใช้สลับเลขจากบวกเป็นลบ เช่น -3 , $-x$ เป็นต้น
- อาจทำให้สับสนได้บางกรณี เช่น $8*-3$
- แนะนำว่าถ้าจะใช้กรณีสลับเลขบวกเป็นลบให้ใส่วงเล็บเข้าไปด้วย จะทำให้ดูง่ายขึ้น เช่น $8*(-3)$

เครื่องหมายหาเศษ (mod)



- ใช้หาเศษ สามารถเรียกใช้โดยการใส่เครื่องหมาย %
- ประยุกต์ใช้ได้หลายสถานการณ์
- ตัวอย่าง
 - $9 \% 4$ ได้ผลลัพธ์เท่ากับ 1
 - $8 \% 3$ ได้ผลลัพธ์เท่ากับ 2
 - $9 \% 3$ ได้ผลลัพธ์เท่ากับ 0
- ปกติการหารจะปัดเศษทิ้ง เราใช้วิธี % เพื่อช่วยปัดเศษขึ้นได้หรือไม่

เครื่องหมาย ++ และ --



- เครื่องหมาย ++ และ -- มีการใช้งานที่คล้ายกัน (เพิ่ม 1-ลด 1) ในที่นี้จะอธิบายเฉพาะเครื่องหมาย ++ เท่านั้น
- ใช้ได้เฉพาะกับตัวแปรเท่านั้น

```
int x = 5;
```

```
x++; //แบบนี้ใช้ได้ ทำเสร็จแล้ว x จะเพิ่มขึ้นหนึ่งเป็น 6
```

```
++x; //แบบนี้ก็ใช้ได้ ทำเสร็จแล้ว x จะเพิ่มขึ้นอีกหนึ่ง (จะเอาเครื่องหมายไว้ข้างหน้าหรือข้างหลังก็เพิ่มขึ้นหนึ่ง)
```

ไม่ว่าจะใส่ด้านหน้าหรือด้านหลังตัวแปร ค่าของตัวแปรก็จะเพิ่มขึ้น 1 เท่ากัน
แต่...!!

เครื่องหมาย ++ และ --



- ถ้าเป็นกรณีแบบนี้ (สมมติปัจจุบัน $x = 5$)

$y = x++;$ // y มีค่า 5 จากนั้นจึงเพิ่มค่า x ขึ้น 1 เป็น 6

และ

$y = ++x;$ // x จะเพิ่มขึ้น 1 เป็น 6 ก่อนเก็บใน y (มีค่า 6)

- แบบนี้จะให้ผลลัพธ์ของ y ไม่เหมือนกัน
- ถึงจะชวนสับสน แต่ก็ยังมีวิธีหลีกเลี่ยงความสับสนนี้

เครื่องหมาย ++ และ --



ถ้าอยากให้ y เพิ่มขึ้นตาม x แยกเป็นสองประโยคแบบนี้ ยังไงก็ไม่พลาด

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

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

ถ้าไม่อยากให้ y เพิ่มขึ้นตาม x แยกเป็นสองประโยคแบบนี้ ยังไงก็ไม่พลาด

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

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

ตัวอย่างโค้ด (credit ผศ.ดร.ภิญโญ แท้ประสาทสิทธิ์)

```
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

ตัวอย่างโค้ด (credit ผศ.ดร.ภิญโญ แท้ประสาทสิทธิ์)



```
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

ตัวอย่างโค้ด (credit ผศ.ดร.ภิญโญ แท้ประสาทสิทธิ์)



```
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

ลำดับการทำงานของ Operator

1. เครื่องหมาย ++, -- (เสมือนใส่วงเล็บให้โดยอัตโนมัติ)
2. วงเล็บ
3. เครื่องหมาย - ที่ทำหน้าที่กลับเครื่องหมาย
4. *, /, % โดยถ้านิพจน์มีหลายเครื่องหมายพร้อมกันจะคิดตามลำดับปรากฏ (ตรงนี้ผิดกันเยอะเพราะไปทำ / ก่อนตามความเคยชิน)
 - 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 $((3 * 2) / 5) \% 7$
5. เครื่องหมาย +, - ถ้ามีหลายเครื่องหมายพร้อมกันจะคิดตามลำดับปรากฏ

ลำดับการทำงานของ Operator

1. $5 + 3 * 7 - 4 / 2$

2. $5 + 3 * 7 / 4 - 2$

เมื่อ $x = 9$;

3. $x++ * 9$

4. $++x * 9$

5. $10 --x++$

เวลาเขียนโปรแกรมจริงจึงควรใส่วงเล็บให้ชัดเจน

การแปลงชนิดข้อมูล (casting)

คอมพิวเตอร์มีการจัดเก็บข้อมูลแต่ละชนิดไม่เหมือนกัน เมื่อคำนวณค่าบางอย่างออกมาได้ จึงต้องมีการกำหนดประเภทข้อมูลให้เหมาะสม

1. การแปลงข้อมูลโดยปริยาย (Implicit type conversion)
 - กำหนดโดยตัวโปรแกรมโดยอัตโนมัติ ตามกฎการเปลี่ยนชนิดข้อมูล
2. การแปลงข้อมูลโดยชัดแจ้ง (Explicit type conversion)
 - กำหนดโดยผู้เขียนโปรแกรมโดยตรง ผู้เขียนจะระบุชนิดข้อมูลลงไปโดยชัดแจ้ง

การแปลงข้อมูลโดยปริยาย (Implicit type conversion)

- คอมไพเลอร์เป็นคนจัดการ
- เกิดขึ้นก่อนการดำเนินการกับ Operator
- การแปลงจะแปลงไปยังข้อมูลที่มีนัยสำคัญมากกว่า
- เช่น ถ้า `int + float` ตัวที่เป็น `int` จะถูกเปลี่ยนเป็น `float` ก่อนทำการบวก

High

ลำดับนัยสำคัญ

double

float

unsigned int

int

short

char

การแปลงข้อมูลโดยปริยาย (Implicit type conversion)



| นิพจน์ | การแปลงชนิดข้อมูล |
|------------------------|-------------------|
| char + int | |
| float + double | |
| int / double | |
| (short + int)/float | |
| double / short * float | |
| double * short / float | |

การแปลงข้อมูลโดยปริยาย (Implicit type conversion)

```
int i = 10;
```

```
float f = 3.2;
```

ถ้าเราจะหาค่าของ i / f เราควรจะนำตัวแปรชนิด int หรือ float มาเก็บผลลัพธ์นี้ ?

การแปลงข้อมูลโดยปริยาย (Implicit type conversion)

```
int i = 10;
```

```
float f = 3.2;
```

```
float result_float = i / f;
```

```
int result_int = i / f;
```

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

3.125000

```
printf("%d\n", result_int);
```

3

การแปลงข้อมูลโดยปริยาย (Implicit type conversion)

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 ดังนั้นความรู้เกี่ยวกับการแปลงชนิดข้อมูลจึงเป็นสิ่งจำเป็น แม้แต่กับเรื่องแสดงผลลัพธ์ให้ถูกต้อง

การแปลงข้อมูลโดยชัดแจ้ง (Explicit type conversion)

- เรียกสั้น ๆ ได้ว่า cast หรือ casting
- วิธีการ cast

ตัวอย่าง

ต้องการ cast ข้อมูลทศนิยม 2.5 ให้กลายเป็นจำนวนเต็ม

ทำได้แบบนี้ (int) 2.5

นิพจน์ที่ได้ก็จะตัดทศนิยมทิ้งไป

การแปลงข้อมูลโดยชัดแจ้ง (Explicit type conversion)

```
float x = 2.5;  
printf("%d", (int) x);
```

Output

2

การแปลงข้อมูลโดยชัดแจ้ง (Explicit type conversion)

- อย่างไรก็ตาม การ cast เป็นการเปลี่ยนค่านิพจน์นั้น แต่ไม่ได้เปลี่ยนค่าในตัวแปร
- ถึงแม้จะทำการ cast ไปแล้ว แต่ค่าในตัวแปร x ก็ยังคงมีค่าเท่าเดิม เช่น

```
float x = 2.5;
```

```
(int) x;           // เปลี่ยนเป็น int แค่ในบรรทัดนี้
```

ค่าในตัวแปร x ยังคงเป็น float และมีค่าเท่าเดิม (ชนิดข้อมูล ถ้าประกาศไปแล้ว ยังไงก็เปลี่ยนแปลงไม่ได้ในภาษาซี)

การแปลงข้อมูลโดยชัดแจ้ง (Explicit type conversion)

- เรามัก cast จำนวนเต็มให้กลายเป็นเลขทศนิยมเพื่อหาผลหารที่เที่ยงตรง

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

ทั้ง x และ y เป็น int ทั้งคู่ การแปลงชนิด
ข้อมูลจึงไม่เกิดขึ้น แบบนี้ต้องบังคับการ cast

```
float f2 = x / (float) y;
```

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

2.000000

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

2.500000

การแปลงข้อมูลโดยชัดแจ้ง (Explicit type conversion)

- หนึ่งในการใช้งานที่นิยมที่สุดในการ cast ก็คือการปิดเศษทศนิยม

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

```
float rf2 = (int) f2;  เราบังคับการปิดเศษทศนิยมด้วยวิธีนี้ได้
```

```
printf("%f\n", rf2);  2.0000000
```

ตัวดำเนินการทางตรรกะ (Logical Operators)

- ใช้ในการดำเนินการเปรียบเทียบค่าต่าง ๆ
- แบ่งเป็นสองกลุ่ม
 - กลุ่มเปรียบเทียบค่า (<, >, <=, >=, ==, !=)
 - กลุ่มพิจารณาค่าตรรกะ (&&, ||, !)

Logical Operators: กลุ่มเปรียบเทียบค่า

| เครื่องหมาย | ความหมาย |
|-------------|----------------------------------|
| < | น้อยกว่า |
| > | มากกว่า |
| <= | น้อยกว่าหรือเท่ากับ |
| >= | มากกว่าหรือเท่ากับ |
| == | เท่ากับ (เทียบระหว่างซ้ายกับขวา) |
| != | ไม่เท่ากับ |

Logical Operators: กลุ่มเปรียบเทียบค่า

- ผลลัพธ์ที่ได้จากเครื่องหมายเหล่านี้จะเป็น จริง หรือ เท็จ เท่านั้น
 - เช่น ถ้า $1 > 0$ คือ จริง
 - ถ้า $x = 2$ และ $y = 3$
 $x > y$ คือ เท็จ และ $x \leq y$ คือจริง เป็นต้น

Logical Operators: กลุ่มพิจารณาคำตรรกะ

| เครื่องหมาย | ความหมาย |
|-------------|-------------------------------|
| && | และ หรือ and ($p \wedge q$) |
| | หรือ หรือ or ($p \vee q$) |
| ! | นิเสธ หรือ not ($\sim p$) |

Logical Operators: กลุ่มพิจารณาค่าตรรกะ

| เครื่องหมาย | การคืนค่าความจริง |
|-------------|---|
| && | ต้องเป็นจริงทั้งซ้ายและขวา |
| | อย่างน้อยตัวใดตัวหนึ่งทางซ้ายและขวาต้องเป็นจริง |
| ! | ถ้าเป็นจริงจะกลายเป็นเท็จ ถ้าเป็นเท็จจะกลายเป็นจริง |

- ก่อนจะไปดูตัวอย่างโปรแกรม นักศึกษาจะต้องทำความเข้าใจเรื่องของค่าความจริงกันก่อน
- เพราะคอมพิวเตอร์เก็บได้แต่ตัวเลข คอมพิวเตอร์จึงไม่สามารถจำค่าจริงค่าเท็จตรง ๆ ได้ แต่มันสามารถจำได้ในรูปแบบของตัวเลขรหัสบางอย่าง
- รหัสนี้เข้าใจไม่ยาก
 - 0 คือเท็จ
 - ค่าอื่น ๆ คือจริง (ถ้าภาษาซีเป็นผู้กำหนดอัตโนมัติจะเป็น 1)

ตัวอย่าง (credit ผศ.ดร.ภิญโญ แท้ประสาทสิทธิ์)

| นิพจน์ | ค่าตรรกะ (ตัวเลขในวงเล็บคือค่าที่เป็นผลลัพธ์) |
|--------------------------------|--|
| $(10 > 1) \ \&\& \ (1 \geq 1)$ | จริง (1) |
| $(10 > 1) \ \ (1 \geq 1)$ | 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) |

ตัวอย่าง (credit ผศ.ดร.ภิญโญ แท้ประสาทสิทธิ์)



```
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
```

ตัวอย่าง

- นิเสธ สามารถซ้อนกันได้ เช่น `!!p` ซึ่งก็จะให้นิเสธของผลนิเสธอีกที (เหมือน - สองอันซ้อนกันจะกลายเป็นบวก) แต่ถ้าจะทำแบบนี้ก็ควรคิดเหตุผลก่อนว่าจะทำไปเพื่ออะไร
- สามารถใช้นิเสธกับนิพจน์ยาว ๆ ก็ได้
เช่น `!((x < y) && (x < z))` เป็นต้น
- จะใช้กับตัวดำเนินการเลขคณิตก็ได้
เช่น `x > s + 1` หรือ `p + 10 > q * 2` เป็นต้น (ใช้บ่อยมาก)

สรุปเรื่องโอเปอเรเตอร์



- เครื่องหมาย $+$, $-$, $*$, $/$, $\%$ ใช้กับ ตัวแปร นิพจน์ และตัวเลขได้
- เครื่องหมาย $++$, $--$ ใช้กับตัวแปรเท่านั้น
- เครื่องหมาย $-$ ใช้ได้สองความหมาย คือ สลับเครื่องหมาย และลบเลขตามปกติ
- ลำดับการคำนวณมีการกำหนดตายตัว แต่่วงเล็บสำคัญสุดเสมอ
- เครื่องหมาย $++$, $--$ สามารถใส่ก่อนหรือหลังตัวแปรก็ได้ แต่ให้ผลไม่เหมือนกัน (ใส่ก่อนคือบวก1ก่อนค่อยนำไปใช้ ใส่หลังคือนำไปใช้ก่อนค่อยบวกหนึ่ง)

สรุปเรื่องการแปลงข้อมูล



- เหตุเกิดจากคอมพิวเตอร์ไม่สามารถคำนวณข้อมูลต่างชนิดกันได้ จึงต้องทำการแปลงข้อมูลให้เหมือนกันก่อนคำนวณ ถ้าเราไม่เปลี่ยนให้ คอมไพเลอร์จะเปลี่ยนให้
- การแปลงอัตโนมัติจะแปลงข้อมูลหนึ่งไปเป็นข้อมูลที่มีนัยสำคัญสูงกว่า
- บางครั้งเราก็ต้องทำการแปลงข้อมูลเองเพื่อประโยชน์บางอย่าง
 - บังคับให้หารแล้วมีทศนิยม
 - บังคับปิดเศษ
- ทศนิยมมีนัยสำคัญสูงกว่าจำนวนเต็ม

สรุปเรื่องตัวดำเนินการทางตรรกะ



- มีสองกลุ่ม คือ กลุ่มเปรียบเทียบค่า กับ กลุ่มพิจารณาค่าความจริง
- ทั้งสองกลุ่มให้ผลลัพธ์เป็น จริง (1) และ เท็จ (0) เท่านั้น
- ถ้าเรากำหนดเอง 0 หมายถึงเท็จ ส่วนเลขอื่นๆ จะหมายถึงจริง
- ผลการทำงานจะมีชนิดข้อมูลเป็นจำนวนเต็ม
- ถ้าเป็นการเปรียบเทียบตัวแปร ต้องระวัง เพราะต้องไล่โปรแกรมตั้งแต่ต้นจนถึงบรรทัดนั้น
- ระวังเรื่อง $==$ และ $=$

การรับและแสดงผลข้อมูล

- ตอนนี้เราได้เรียนรู้เรื่องการแสดงผลออกทางจอภาพด้วยคำสั่ง printf ไปแล้ว
 - แสดงค่าตายตัว printf(“Hello”);
 - แสดงค่าตัวแปร printf(“%d”, out_var);
 - แสดงค่าแบบผสมทั้งข้อความและตัวแปร printf(“The result is %d”, x);
 - แสดงค่าตัวแปรหลายตัว printf(“The 1st result is %d and the 2nd is %f”, x, y)
- ต่อจากนี้จะเป็นรายละเอียดเพิ่มเติมของการใช้ printf

`printf(“string_format”, data_list);`

การแสดงผลข้อมูล



- ตัวอย่างโปรแกรม

```
#include<stdio.h>
```

```
void main() {
```

```
    float f = 0.5;
```

```
    int i = 2;
```

```
    printf("%f %d", f, i);
```

```
}
```

โดยการปรับ %f ตรงนี้

เลขทศนิยมหลายตำแหน่งกว่าที่
เราต้องการ เราสามารถกำหนด
จำนวนที่ต้องการได้

//output : 0.500000 2

การแสดงผลข้อมูล

- เราสามารถใส่จำนวนทศนิยมที่ต้องการได้โดยการพิมพ์ **%.<จำนวนทศนิยม>f**
- ถ้าต้องการทศนิยม 2 ตำแหน่ง ก็พิมพ์ **%.2f**
- ถ้าต้องการทศนิยม 4 ตำแหน่ง ก็พิมพ์ **%.4f**
- ถ้าไม่ต้องการทศนิยมเลย ก็พิมพ์ **%.0f**
- ถ้าไม่พิมพ์จะตั้งเป็น 6 โดยมาตรฐาน
- **ไม่มีผลกับค่าของตัวแปร** มีผลกับสิ่งที่จะแสดงผลทางจอภาพเท่านั้น

การแสดงผลข้อมูล



```
#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

การรับข้อมูล



- จริงๆ แล้วในภาษาซีมีหลายคำสั่งที่สามารถรับค่าจากคีย์บอร์ดได้
- คำสั่ง scanf น่าจะเหมาะกับการเริ่มต้นที่สุด
- ใช้หลักการคล้าย printf โดยมีรูปแบบการใช้งานคือ

`scanf("string_format", address_list);`

- แม้จะบอกว่าคล้าย แต่ก็มีข้อแตกต่างพอสมควร

การรับข้อมูล



- คำสั่ง scanf ใช้เครื่องหมาย % เพื่อกำหนดชนิดข้อมูลที่จะรับเข้ามา (เหมือน printf)
- scanf ไม่ได้ใช้เพื่อแสดงผล ดังนั้นจึงไม่สามารถใส่ string อื่นๆลงไปได้ นอกจากสัญลักษณ์ของชนิดข้อมูลและช่องว่างเท่านั้น
 - เช่น scanf(“%d %d”, &x, &y); เป็นต้น (จริงๆแล้วช่องว่างจะใส่หรือไม่ใส่ก็ได้)
 - แบบนี้ไม่ได้ scanf(“Input the number : %d”, &x);
 - แบบนี้ไม่ได้ scanf(“%d\n”, &x);

Address list คืออะไร

- ใน printf ส่วนท้ายของคำสั่งจะเป็น data list จึงสามารถส่งตัวแปรเข้าไปได้เลย
- ใน scanf ส่วนท้ายของคำสั่งจะต่างออกไป คือเป็น address list จึงจำเป็นต้องส่งที่อยู่ของตัวแปรเข้าไป ซึ่งการส่งที่อยู่เข้าไปก็เพียงแค่ใส่เครื่องหมาย & ไว้หน้าชื่อตัวแปรเท่านั้น (แต่ก็พลาดลืมใส่กันเยอะ)
- เช่น ถ้าจะรับค่าให้ตัวแปร x ก็พิมพ์ว่า &x และถ้าจะรับค่าให้ตัวแปร input ก็พิมพ์ว่า &input
- ตอนนี้เราจะยังไม่พูดเรื่องนี้มากนัก ให้เข้าใจว่าตัวแปรข้างหลังของ scanf จะต้องมี & ไปก่อนก็ได้

ตัวอย่างการใช้งาน scanf



```
#include <stdio.h>
```

```
void main() {
```

```
    int x = 3;
```

```
    printf("Value of x is %d\n", x);
```

```
    scanf("%d", &x); //เมื่อโปรแกรมทำงานจนถึงคำสั่งนี้ โปรแกรมจะหยุดรอให้ผู้ใช้ใส่ค่า x เข้าไป เราจะเห็น cursor เครื่องหมายขีดเส้นใต้กระพริบรอผู้ใช้
```

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

```
}
```

รับค่า input หลายตัวพร้อมกัน

- คำสั่ง scanf สามารถรับข้อมูลเข้าหลายตัวพร้อมกันได้ในการคำสั่งเดียว

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)

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

รับค่า input หลายตัวพร้อมกัน

- แม้แต่ผสมกันก็ยังทำได้

```
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
```

ถ้าใส่ข้อมูลเกินกว่า scanf จะรับไหว

- หมายถึงถ้าประกาศให้ scanf รับค่า 2 ตัว แต่ตอนรันกลับเติมค่าไป 3 ค่า จะเกิดอะไรขึ้น
 1. ถ้าหลังจากบรรทัดนั้นในโปรแกรมไม่มีการ scanf อีกเลย ค่านั้นก็จะถูกละทิ้งไป
 2. ถ้าหลังจากบรรทัดนั้นในโปรแกรมยังมีการ scanf อยู่ ค่าที่เกินมาจะถูกผลักไปใส่ใน scanf ครั้งถัดไป

```
int w, x, y, z;
```

```
scanf("%d %d", &w, &x); //บรรทัดนี้รับ 2 ตัวแต่ใส่ไป 5 ตัว
```

```
scanf("%d", &y);
```

```
scanf("%d", &z); //จนถึงบรรทัดนี้ มีการรับค่าแค่ 4 ค่า เลข 5 จึงถูกละทิ้งไป
```

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

```
1 2 3 4 5
Inputs are 1, 2, 3, 4
```

Note

