



# Computer Programming I: การเขียนโปรแกรมคอมพิวเตอร์ I

## โครงสร้างภาษาซี ตัวแปร และการแสดงผลอย่างง่าย



อ.ดร.ปัญญานต์ อ้นพงษ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

aonpong\_p@su.ac.th

# ทบทวนครั้งที่แล้ว



- จากการเรียนชั่วโมงที่แล้ว เราได้ทำโจทย์เพื่อเพิ่มทักษะการเขียนโฟลวชาร์ต และซูโดโค้ดในลักษณะต่างๆมาแล้ว สิ่งที่น่าจะทำได้แก่
  - โจทย์ที่จะได้เจอจะมีความหลากหลาย แต่ถ้าฝึกทำโจทย์ใหม่ ๆ ไปเรื่อย ๆ จะพบว่า จำนวนของรูปแบบของโจทย์นั้นมีจำกัด เมื่อนักศึกษาได้ทำถึงระดับหนึ่งจะรู้สึกว่าย่อยว่าโจทย์เริ่มซ้ำไปมา
  - ได้ยกตัวอย่างโจทย์และเทคนิคที่อาจได้ใช้งานในอนาคต

# Outline



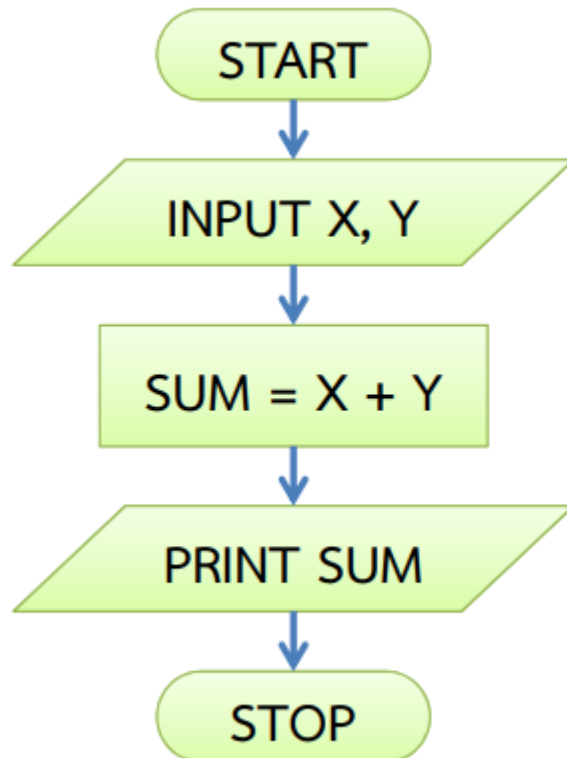
- ความสัมพันธ์ของฟลอชาร์ต ซูโดโค้ด และภาษาคอมพิวเตอร์
- ภาษาซีกับการเขียนโปรแกรม
- การเริ่มต้นและจบโปรแกรม
- ตัวแปร
- ค่าคงที่ในภาษาซี
- การแสดงผลอย่างง่าย

# ความสัมพันธ์ของโฟลวชาร์ต ซูโดโค้ด และภาษาคอมพิวเตอร์

- เราได้เรียนรู้วิธีการเขียนโฟลวชาร์ตและซูโดโค้ดไปแล้ว สิ่งเหล่านี้สามารถทำได้โดยไม่ต้องรู้ภาษาคอมพิวเตอร์
- สิ่งเหล่านี้เป็นส่วนสำคัญของการเริ่มต้นเขียนโปรแกรมคอมพิวเตอร์
- ถ้ากำหนดลำดับขั้นตอนวิธีการออกมาได้ไม่ชัดเจนพอ เราจะเขียนโปรแกรมคอมพิวเตอร์ไม่ได้
- โฟลวชาร์ตจะเข้าใจได้ง่ายที่สุด แต่แปลงเป็นภาษาคอมพิวเตอร์ยากกว่าซูโดโค้ด และยังต้องใช้พื้นที่ในการเขียนมาก
- ซูโดโค้ดแปลงเป็นภาษาคอมพิวเตอร์ได้ง่ายกว่า เขียนได้เร็วกว่า

# ความสัมพันธ์ของโฟลวชาร์ต ชูโตโค้ด และภาษาคอมพิวเตอร์

- เริ่มจากโฟลวชาร์ตและชูโตโค้ดง่าย ๆ ก่อน โจทย์ข้อนี้เราเคยเห็นมาก่อนบ้างแล้ว



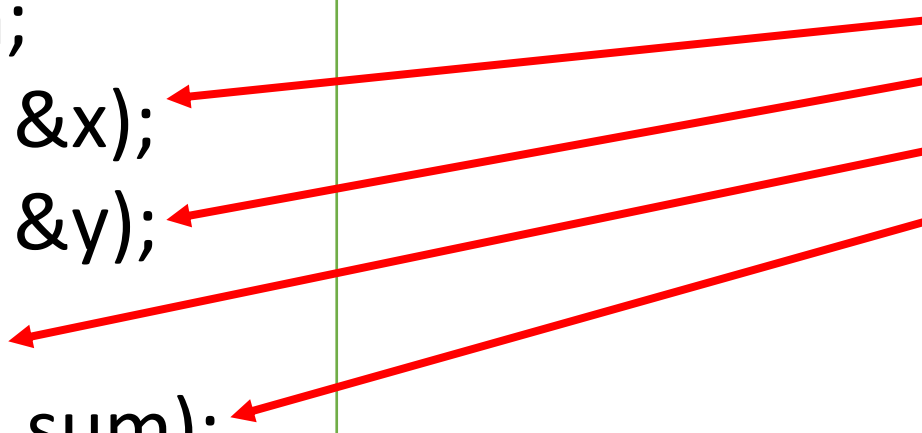
```
START
READ X
READ Y
COMPUTE SUM = X + Y
PRINT SUM
STOP
```

# ความสัมพันธ์ของโฟลวชาร์ต ซูโดโค้ด และภาษาคอมพิวเตอร์

- ถ้าเปลี่ยนเป็นโค้ดในภาษาซี

```
#include <stdio.h>
void main(){
    int x, y, sum;
    scanf("%d", &x);
    scanf("%d", &y);
    sum = x+y;
    printf("%d", sum);
}
```

```
START
READ X
READ Y
COMPUTE SUM = X + Y
PRINT SUM
STOP
```



# ความสัมพันธ์ของโฟลวชาร์ต ซูโดโค้ด และภาษาคอมพิวเตอร์

- ถ้าเปลี่ยนเป็นโค้ดในภาษาซี

```
#include <stdio.h>
void main(){
    int x, y, sum;
    scanf("%d", &x);
    scanf("%d", &y);
    sum = x+y;
    printf("%d", sum);
}
```

?

```
START
READ X
READ Y
COMPUTE SUM = X + Y
PRINT SUM
STOP
```

# ความสัมพันธ์ของฟลอชาร์ต ซูโดโค้ด และภาษาคอมพิวเตอร์

- ย้อนกลับไปเมื่อสัปดาห์แรก เราได้พูดถึงการคำนวณว่าจำเป็นต้องใช้หน่วยความจำในการคิด
  - ส่วนแรก ต้องจำให้ได้ว่าการ  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $%$  และอื่น ๆ มีขั้นตอนวิธีการอย่างไร
  - ส่วนที่สอง สมมติ เวลาเรานำเลขสองตัวมาบวกกัน เช่น  $5+2$  เราต้องนำ 5 ไว้ในใจแล้วนับนิ้วต่อ การนำ 5 ไว้ในใจและการนับนิ้วนี้แหละคือการใช้ความจำ
  - ถ้าจำไม่ได้ว่าจะเอาเลขอะไรมาบวกกัน เราก็จะบวกมันไม่ได้
- ทั้งหมดที่กล่าวมา คอมพิวเตอร์ก็เช่นกัน
  - คอมพิวเตอร์มี CPU สำหรับการคำนวณ และจำเป็นต้องใช้ RAM เพื่อเป็นหน่วยความจำ



# การจดจำของคอมพิวเตอร์

- แบ่งเป็น 2 ส่วน

1. ส่วนจดจำวิธีการทำงาน

ส่วนนี้เป็นส่วนที่เราต้องเรียกใช้ด้วยทุกครั้ง นั่นคือ `#include <stdio.h>` ส่วนนี้จะทำหน้าที่บอกวิธีการนำข้อมูลเข้าและแสดงผลอย่างง่ายที่เราต้องใช้ในการเรียนตลอดเทอมนี้ คำสั่งนี้เปรียบได้กับการอ่านหนังสือและเขียนหนังสือของเราที่เราจะต้องจำตัวอักษรและวิธีการอ่านและเขียนให้ได้

2. จดจำข้อมูลในการคำนวณ

อย่างที่คุยกันไปก่อนหน้านี้ว่าเราต้องจำตัวเลขให้ได้ก่อนจึงจะสามารถคำนวณได้ คอมพิวเตอร์ก็เช่นกัน แต่ไม่ใช่ว่าอยู่ๆ คอมพิวเตอร์จะสามารถจดจำข้อมูลทุกอย่างได้เลย ก่อนจะให้คอมพิวเตอร์ทำอะไร เราต้องทำการจองพื้นที่หน่วยความจำก่อน ซึ่งในการจองพื้นที่ก็จะต้องนิยามค่าที่มันจะต้องจำให้มันด้วย

`int x, y, sum;` บรรทัดนี้จึงหมายถึงตัวแปร 3 ตัวที่คอมพิวเตอร์จะต้องจำ ซึ่งมีประเภทเป็น `int` (เดี๋ยวจะคุยกันในภายหลัง)

# ความสัมพันธ์ของโฟลวชาร์ต ชูโดโค้ด และภาษาคอมพิวเตอร์

- ถ้าเปลี่ยนเป็นโค้ดในภาษาซี

```
#include <stdio.h>
void main(){
    int x, y, sum;
    scanf("%d", &x);
    scanf("%d", &y);
    sum = x+y;
    printf("%d", sum);
}
```

จะอ่านจะเขียน ต้องรู้ภาษา

จองพื้นที่ในหน่วยความจำ (ไม่มีในชูโดโค้ด)

```
START
READ X
READ Y
COMPUTE SUM = X + Y
PRINT SUM
STOP
```

# ความแตกต่างของโฟลวชาร์ต ซูโดโค้ด และภาษาคอมไพเตอร์

- โฟลวชาร์ตและซูโดโค้ดเป็นตัวบอกลำดับวิธีการทำงาน (อัลกอริทึม) ไม่จำเป็นต้องเกี่ยวกับคอมไพเตอร์ก็ได้
  - ไม่ต้องการอธิบายการจำและทักษะ (int x,y,sum; #include<stdio.h>) อยู่ในนั้นก็ได้
- ภาษาคอมไพเตอร์ (ในที่นี้ ภาษาซี) เป็นตัวที่ใช้บอกคอมไพเตอร์ว่าควรทำงานอย่างไร จึงต้องการความชัดเจน
  - เพราะเหตุนี้จึงต้องการอธิบายการจำและทักษะให้กับคอมไพเตอร์ด้วย เพื่อให้เครื่องคอมไพเตอร์ของเราเข้าใจมันได้

# การจดจำค่าในภาษาซี (Memory)

- ส่วนจดจำเรื่องทักษะมักจะไม่ค่อยมีความเปลี่ยนแปลงในระดับนี้ คำสั่งที่ใช้บ่อย (ทุกโปรแกรมที่เขียนในรายวิชานี้) ได้แก่

```
#include <stdio.h>
```
- คำสั่งอื่นก็พอมีบ้างเช่นกัน เช่น `math.h` เป็นต้น แต่เราจะไม่พูดถึงในตอนนี้นี้
- สิ่งที่มีปัญหาเพราะมีรายละเอียดจุกจิก คือส่วนจดจำเกี่ยวกับตัวแปร
  - นักศึกษาจะต้องระบุนิคมของตัวแปรให้เหมาะสมกับการใช้งาน
  - จุดนี้ต้องอาศัยทั้งความเข้าใจและความชำนาญ

# ตัวแปร (Variable)

- ตัวแปร คือ สิ่งที่ใช้เก็บข้อมูลในโปรแกรมคอมพิวเตอร์ (ได้ทั้งตัวเลข ตัวอักษร และข้อมูลชนิดอื่นๆบางชนิด)
- ถ้าเปรียบเทียบกับสิ่งที่เราเคยได้ทดลองในคาบแล็บ ตัวแปรจะเป็นตัวอักษรภาษาอังกฤษ
- ตัวอย่างการใช้ตัวแปรในการคำนวณ

$$x = 15$$

$$y = 15$$

$$f(x, y) = x+y \quad \leftarrow \text{เมื่อมาอยู่ในระบบคอมพิวเตอร์ จะถูกทำให้เรียบง่ายกว่านี้}$$

# ตัวแปร (Variable)

- เมื่อนำตัวแปรมาใช้ในการเขียนโปรแกรม เราจะไม่สามารถเรียกใช้ได้แบบทันทีทันใด จะต้องมีการเกริ่นให้คอมพิวเตอร์รู้ก่อนว่าจะมีตัวแปรอะไร เก็บค่าอะไร

เช่น เราอยากเขียนโปรแกรมรับค่าจำนวนเต็ม  $x$ ,  $y$  และหาผลรวม ( $sum$ )



# ตัวแปร (Variable)

- เราจะทราบได้ว่าที่แน่ๆ จะต้องใช้ตัวแปร x, y และ sum ดังนั้นเราจึงต้องประกาศให้โปรแกรมรู้ตั้งแต่ก่อนจะใช้งาน
  - ต้องบอกด้วยว่าประเภทข้อมูลที่จะใช้เป็นข้อมูลประเภทใด (จำนวนเต็ม, ทศนิยม, ตัวอักษร, ฯลฯ)
- เนื่องจากโจทย์ต้องการการคำนวณเป็นจำนวนเต็ม จึงต้องประกาศเป็น **ชนิดข้อมูล** แบบ int (จะอธิบายภายหลัง)

```
int x;           // จะใช้ตัวแปร x ในรูปแบบจำนวนเต็ม
int y;           // จะใช้ตัวแปร y ในรูปแบบจำนวนเต็ม
int sum;         // จะใช้ตัวแปร sum ในรูปแบบจำนวนเต็ม
```

- แล้วจึงจะนำไปใช้ในการคำนวณ

```
sum = x + y;     // นำค่าที่คำนวณได้จากฝั่งขวามาเก็บในฝั่งซ้าย
```

# ชนิดข้อมูล (Data type)

- มีหน้าที่บอกคอมพิวเตอร์ว่าตัวแปรตัวนั้นจะเก็บข้อมูลชนิดใด (จำนวนเต็ม, ทศนิยม, ตัวอักษร, ฯลฯ)
- **ทำไมจึงต้องบอก?** เพราะว่าวิธีการจัดจำข้อมูลแต่ละชนิดในหน่วยความจำของคอมพิวเตอร์ไม่เหมือนกัน
- การระบุชนิดข้อมูลให้ตรงกับความต้องการใช้งานจึงมีความสำคัญ หลายคนจะทำผิดจุดนี้เพราะกระบวนการคิดในหัวเราไม่ต้องคำนึงถึงจุดนี้มากนัก แต่คอมพิวเตอร์กลับมีความเปราะบางมาก

ชนิดข้อมูลตัวแปรแบ่งออกเป็นสองกลุ่ม

1. กลุ่มพื้นฐาน (Basic Data Type) มีสี่ประเภท (1) จำนวนเต็ม (2) ทศนิยม (3) ตัวอักษร และ (4) Void
2. กลุ่มขั้นสูง (Advanced Data Type) จะมีความซับซ้อนมากขึ้น เช่น อาร์เรย์, ตัวชี้, สตริง เป็นต้น เราจะยังไม่พูดถึงในครั้งแรกนี้



# ข้อมูลชนิด จำนวนเต็ม



ชนิดข้อมูล (เก็บจำนวนเต็ม)	ขนาด (Bytes)	Min	Max
int	4	-2,147,483,648	2,147,483,647
short int	2	-32,768	32,767
unsigned int	4	0	4,294,967,295
long long int (มาตรฐานใหม่)	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

- ตัวแปรประเภท int ถูกเลือกใช้บ่อยที่สุด
- ตัวแปรประเภท int กับ signed int คือสิ่งเดียวกัน (แต่ไม่มีใครเขียนว่า signed int กันแม้ว่าจะไม่ทำให้เกิด error)
- ตัวแปร short int มักถูกใช้เมื่อต้องการประหยัด memory แต่ด้วยความที่มันเก็บตัวเลขได้น้อยมาก (ยังไม่ถึงแสน) จึงต้องระมัดระวังในการใช้งาน (ปกติก็ไม่ค่อยมีใครใช้เท่าไรหรอก)
- เราสามารถประกาศตัวแปรประเภท short int ว่า short สั้นๆ เฉยๆ ก็ได้

# ข้อมูลชนิด ทศนิยม



- เลขทศนิยม คือเลขที่มีจุด ไม่ว่าจะเป็น 3.5, 2.44, 12.75333, 55.5656 หรือแม้กระทั่ง 20.00
- คอมพิวเตอร์จะมีวิธีการจำเลขทศนิยม 2 แบบ คือ
  - Single Precision (ความเที่ยงเดียว)
  - Double Precision (ความเที่ยงทวิคูณ)
- ทั้งสองแบบมีแนวคิดเหมือนกัน แต่แบบ Double Precision จะมีความเที่ยงตรงมากกว่า แต่ก็ต้องแลกมาด้วยความต้องการหน่วยความจำที่มากกว่า
- **เราจะเลือกแบบไหน?** ถ้าการใช้งานทั่วไปก็แล้วแต่ความต้องการของเราว่าจะนำผลไปใช้ทำอะไร แต่ถ้าเป็นการสอบ ก็ให้เลือกแบบที่โจทย์ระบุมา (โจทย์จะบอกมาให้เพราะบางครั้งผลลัพธ์ของทั้งสองรูปแบบจะไม่เท่ากัน)

# ข้อมูลชนิด ทศนิยม



ชนิดข้อมูล (เก็บทศนิยม)	ขนาด (Bytes)	Min (-)	Max (-)	Min (+)	Max (+)
float (single precision)	4	$-3.4 \times 10^{-38}$	$-3.4 \times 10^{38}$	$3.4 \times 10^{-38}$	$3.4 \times 10^{38}$
		* ความเที่ยงของเลขฐานสิบ ประมาณ 7 หลัก			
double (double precision)	8	$-1.7 \times 10^{-308}$	$-1.7 \times 10^{308}$	$1.7 \times 10^{-308}$	$1.7 \times 10^{308}$
		* ความเที่ยงของเลขฐานสิบ 15 หลัก			

- ความแตกต่างของจำนวนเต็มและทศนิยมยังเกิดขึ้นเมื่อมีการดำเนินการทางคณิตศาสตร์ด้วย เราจะเรียนเรื่องนี้โดยละเอียดในภายหลัง

# ข้อมูลชนิด ตัวอักษร

- เกี่ยวข้องกับการแสดงผล มีขนาด 1 byte
- ตัวแปรชนิดตัวอักษร (char) 1 ตัวแปร สามารถเก็บตัวอักขระได้ 1 ตัวเท่านั้น ไม่ใช่ทั้งประโยค
- ต้องใช้เครื่องหมายอัฒประกาศเดี่ยวครอบตัวอักขระที่ต้องการเก็บใน char
- นักศึกษาต้องแยกประโยคทั้งสองแบบนี้ออก

`sum = x;`

`sum = 'x';`

// ผีงซ้ายคือการเก็บค่า x ไว้ในตัวแปร sum เช่น ถ้า x เป็น int เก็บค่า 15 ตัวแปร sum ก็จะมีค่า 15

// ผีงขวาคือการเก็บตัวอักขระ x เข้าไปใน sum

//ประเภทตัวแปร sum ทางผีงซ้ายและผีงขวาควรเป็นประเภทเดียวกันหรือไม่ และควรเป็นประเภทอะไรบ้าง?

# ข้อมูลชนิด ตัวอักษร



- ตัวแปรชนิดตัวอักษร จริงๆแล้วเก็บอักขระเป็นรหัสตัวเลข ตามมาตรฐาน ASCII (แอสกี) ซึ่งย่อมาจาก American Standard Code for Information Interchange
- ทำไมเป็นแบบนี้? เพราะคอมพิวเตอร์จำได้แค่ตัวเลข จึงมีการสร้างรหัสแทนตัวอักษรขึ้นมา เพื่อให้คอมพิวเตอร์จำได้ เช่น 'a' แทนด้วยตัวเลข 97 และ 'b' แทนด้วยเลข 98 เป็นต้น
- เพราะแบบนี้ทำให้เราบวกลบตัวแปรชนิดตัวอักษรได้

# ข้อมูลชนิด ตัวอักษร



```
#include <stdio.h>

void main(){

    char A = 'a';

    printf("%c", A);

    A = A + 1;

    printf("%c", A);

}
```

// ผลลัพธ์เป็น 'a', ตอนนี้คอมพิวเตอร์เข้าใจว่า 97  
// เอา 97 ไปบวก 1 ทำให้ตอนนี้ A เก็บค่า 98 แทน  
// ผลลัพธ์เป็น b, เพราะ 98 เป็นรหัสของ 'b'

//ถ้าการบวก ไม่ได้บวกเพิ่มแค่ 1 นักศึกษาคิดว่าจะเกิดอะไรขึ้น?

# ข้อมูลชนิด ตัวอักษร: อักขระพิเศษ

- นอกจากอักขระ a-z และ A-Z แล้ว char ยังเก็บสัญลักษณ์และเครื่องหมายวรรคตอนด้วย
- **ตัวเลขและสัญลักษณ์ทางคณิตศาสตร์ก็เก็บได้** ตรงนี้ต้องระมัดระวังเพราะถ้าเผลอนำเลขจากอักขระไปคำนวณจะให้ผลลัพธ์ผิดพลาด (เว้นแต่ว่าจงใจประยุกต์ใช้สมบัตินี้)
- ตัวอย่างอักขระพิเศษ ? ! : ; , @ \$ % ^ & \* (พวกนี้ยังไม่ค่อยประหลาด)
- จะมีอักขระพิเศษบางตัวที่ไม่ใช่ตัวอักษรเสียทีเดียว แต่เหมือนเป็น mark จัดหน้าจอบอกมากกว่า

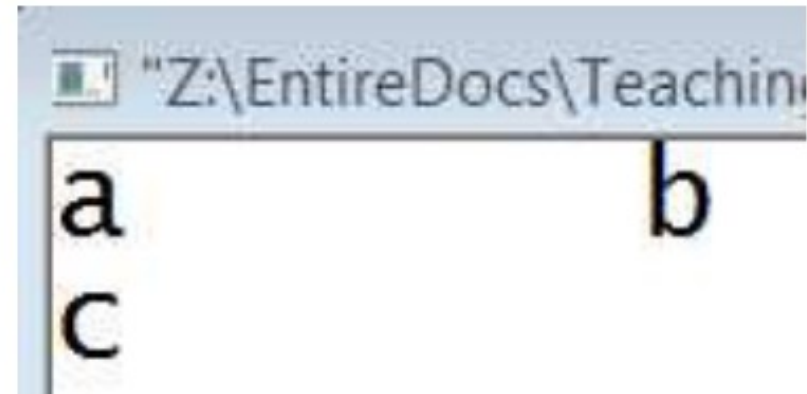
อักขระพิเศษ	ความหมาย
\n	ขึ้นบรรทัดใหม่
\t	แท็บ (Tab)
\0	สิ้นสุดข้อความ (NULL)
\a	เสียงเตือน (ไม่ได้ใช้ในห้องเรียนนี้)

# ข้อมูลชนิด ตัวอักษร: อักขระพิเศษ

เช่น `printf("a\tb\nc\a");` จะพิมพ์ข้อความ

a            b

c



บนจอแล้วตามด้วยเสียงเตือนหนึ่งครั้ง

ทั้งนี้เป็นผลมาจาก a ตามด้วย Tab (\t) จากนั้นตามด้วย b และการขึ้นบรรทัดใหม่ (\n) จากนั้นตามด้วย c และปิดท้ายด้วยเสียงเตือน (\a)



# ข้อมูลชนิด ตัวอักษร: อักขระหลีก (Escaped Character)

- ถ้าเราใช้ `\n` ในการขึ้นบรรทัดใหม่ไปแล้ว ถ้าเราอยากพิมพ์ทั้ง `\` และ `n` ติดกันออกทางจอภาพจริงๆ จะทำได้หรือไม่? **ทำได้**
- เราต้องใช้ **อักขระหลีก** สำหรับตัวอักษรที่มีหลายหน้าที่ในภาษา C

อักขระหลีก	ความหมาย
<code>\\</code>	เครื่องหมาย <code>\</code>
<code>\'</code>	เครื่องหมาย <code>'</code>
<code>\"</code>	เครื่องหมาย <code>"</code>

```
printf("\\n");
```

// ผลลัพธ์ที่ได้ คือ `\n` ตามที่ตั้งใจ

# ข้อมูลชนิด ตัวอักษร: สายอักขระ

- ตัวอักษรตัวเดียว ยังไม่เพียงพอที่จะสื่อสารอะไรได้
- แต่ถ้าเราประกอบตัวอักษรหลาย ๆ ตัวเข้าด้วยกันเป็นคำ หรือประโยคก็จะสื่อสารได้
- ข้อมูลที่เก็บอักขระเป็นสาย (หลายตัว) เรียกว่า string (เป็นตัวแปรข้อมูลชั้นสูง)
- ข้อมูล string จะใช้สัญลักษณ์ “ ” ครอบสายอักขระทั้งหมดเอาไว้ (ต่างจาก char)
- คำสั่งแสดงข้อความออกทางจอภาพ รองรับแค่ข้อมูล string เช่น

```
printf("Welcome to Silpakorn");    //สังเกตว่าใช้ “ ” (double quote)
```

- แม้ว่าจะพิมพ์ตัวอักษรโดดเพียงตัวเดียวด้วยคำสั่ง printf แต่ก็ต้องใช้ “ ” เพราะ printf รองรับแค่ข้อมูล string (อักขระตัวเดียวก็สามารถมองให้เป็น string ได้)

```
printf("a");
```

# ข้อมูลชนิด void (ว่างเปล่า; ไม่ระบุชนิดข้อมูล)

- มักใช้ในจุดเริ่มต้นของโปรแกรม (main)

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

```
void main(){
```

```
// ...
```

```
}
```

# การประกาศตัวแปร: พื้นฐาน

- เรา รู้จักชนิดตัวแปรไปหลายชนิดแล้ว ทั้งจำนวนเต็ม ทศนิยม และตัวอักษร
- ในการประกาศตัวแปร เราจะระบุชนิดตัวแปรก่อน เว้นวรรคแล้วตามด้วยชื่อตัวแปร

**ชนิดตัวแปร ชื่อตัวแปร;**                      เช่น

- ต้องการประกาศตัวแปร ประเภทจำนวนเต็ม ชื่อ x

```
int x;                      //อย่าลืม ;
```

- ลองประกาศตัวแปร
  - ประเภททศนิยม Double Precision ชื่อ silpa
  - ประเภทจำนวนเต็ม เก็บค่า ในช่วง -1 ถึง 3000 ล้าน ชื่อ very\_big
  - ประเภทจำนวนเต็ม เก็บค่า ในช่วง 100 ถึง 3000 ล้าน ชื่อ still\_big

# การประกาศตัวแปร: ยุบรวม

- ถ้าเป็นตัวแปรประเภทเดียวกันหลายตัว เช่น

```
int x;
```

```
int y;
```

```
int sum;
```

- เราสามารถยุบรวมให้สั้นลงด้วยเครื่องหมาย , ได้ ดังนี้

```
int x, y, sum;
```

ลองประกาศตัวแปรเก็บเลขทศนิยมแบบ single precision ชื่ออะไรก็ได้ จำนวน 5 ตัว

# การประกาศตัวแปร: กำหนดค่าเริ่มต้น

- สามารถกำหนดค่าให้ตัวแปรเลยก็ได้

```
int x = 5;
```

```
int y = 10;
```

```
float m = 25.5;
```

```
char A = 'a';
```

```
int sum = 2, div = 5;
```

- ให้คิดเสมอว่า เครื่องหมายเท่ากับ จะนำค่าทางขวาไปแทนในตัวแปรทางซ้าย ดังนั้นจะทำแบบนี้ไม่ได้

```
int 5 = x; //แบบนี้เป็นการเก็บค่า x ลงในเลข 5 ซึ่งทำไม่ได้
```

# การประกาศตัวแปร: กำหนดค่าเริ่มต้น

- ลองคิดเล่นๆ

```
int x = 10, y = 5;
```

```
x=y;
```

- ณ จุดนี้ค่าของ x เป็นเท่าไร และ y เป็นเท่าไร?

# การประกาศตัวแปร: การตั้งชื่อ

- ชื่อตัวแปรมีของปนกันได้อยู่สามอย่างคือ (1) ตัวอักษรภาษาอังกฤษ, (2) ตัวเลข และ (3) เครื่องหมาย underscore (เครื่องหมาย `_`) เช่น
  - `int s_service`; เป็นการประกาศตัวแปรประเภทจำนวนเต็ม ชื่อ `s_service`
  - `float p_value`; เป็นการประกาศตัวแปรประเภททศนิยม ชื่อ `p_value`
- ตัวแปรขึ้นต้นด้วยตัวเลขไม่ได้
  - `int 1s_int`; ไม่สามารถตั้งชื่อนี้ได้
- ตัวแปรขึ้นต้นด้วยเครื่องหมาย underscore ก็ได้
  - `double _coef`; เป็นการประกาศตัวแปรประเภททศนิยม ชื่อ `_coef`
- ห้ามตั้งชื่อตัวแปรซ้ำกันในเขตพื้นที่เดียวกัน แม้ว่าจะเป็นข้อมูลคนละชนิดก็ไม่สามารถทำได้



# การประกาศตัวแปร: การตั้งชื่อ

- ตัวพิมพ์เล็กใหญ่มีความสำคัญ เพราะภาษา C จะมองตัวอักษรเป็นคนละตัว
- แม้ว่าจะเป็นอย่างนั้นก็อย่าพยายามตั้งชื่อตัวแปรเดียวกันแต่ต่างแค่พิมพ์เล็กใหญ่ เพราะเป็นการเพิ่มความสับสนให้ตัวเอง

int, INT, iNT, InT ถือว่าเป็นคนละความหมายกันทั้งหมด

int x; int X; สามารถทำได้ คอมไพล์ผ่านได้

- ผู้เริ่มเรียนมักสับสนกับเรื่องตัวเล็กตัวใหญ่ของชื่อตัวแปร ทำให้โปรแกรม คอมไพล์ไม่ผ่าน เช่น ตอนประกาศ เขียนว่า int Number; แต่พอเอาไปใช้ ไปเขียนว่า number = 5; แบบนี้จะใช้ไม่ได้ คอมไพล์ไม่ผ่าน

# การประกาศตัวแปร: จำเป็นจะต้องประกาศตัวแปรทุกครั้ง?

- **ไม่จำเป็น** ในกรณีที่เรารู้ตัวเลขที่แน่นอนตั้งแต่ตอนเขียนโปรแกรม เราสามารถระบุตัวเลขลงไปใน  
การคำนวณได้เลย
- เช่น ในตัวอย่างเราเขียนว่า `int sum = x + y`; ถ้าเรารู้ค่า `x` และ `y` ตั้งแต่ตอนเริ่มเขียนโปรแกรม  
เราสามารถเขียนว่า `int sum = 5 + 7`; ไปเลยก็ได้
- ถึงไม่จำเป็นแต่บางทีก็ควรใช้ เพราะค่าคงที่บางอย่างใช้บ่อยและเขียนยาก เช่น ค่า `pi = 3.14159265` ถ้าหากเราจะเขียนเลขนี้ทุกครั้ง โค้ดเราจะดูยุ่งเหยิงและเสี่ยงที่จะพิมพ์ผิดไปเป็นค่า  
อื่นได้ (ถ้าจะแก้ค่าก็แก้ทีเดียว)
- การประกาศตัวแปรจะ**จำเป็น** ในกรณีที่มีการรับข้อมูลจากผู้ใช้ เราต้องส่งตัวแปรไปรับค่าจากผู้  
ใช้ มา ความจำเป็นนี้เกิดขึ้นเพราะเราไม่ทราบค่าของตัวเลขจนกว่าผู้ใช้งานจะระบุมาในภายหลัง

# สรุปพื้นฐานตัวแปร



- เราอ้างถึงตัวแปรที่ประกาศไว้ก่อนหน้าได้ แต่จะอ้างถึงตัวแปรก่อนการประกาศไม่ได้ ไม่ว่าจะเป็นการอ่านค่าหรือบันทึกค่าตัวแปร เช่น

```
void main() {  
    int x;  
    x = 7;  
}
```

→ แบบนี้ใช้ได้ เพราะเราอ้างถึง x หลังประกาศตัวแปร

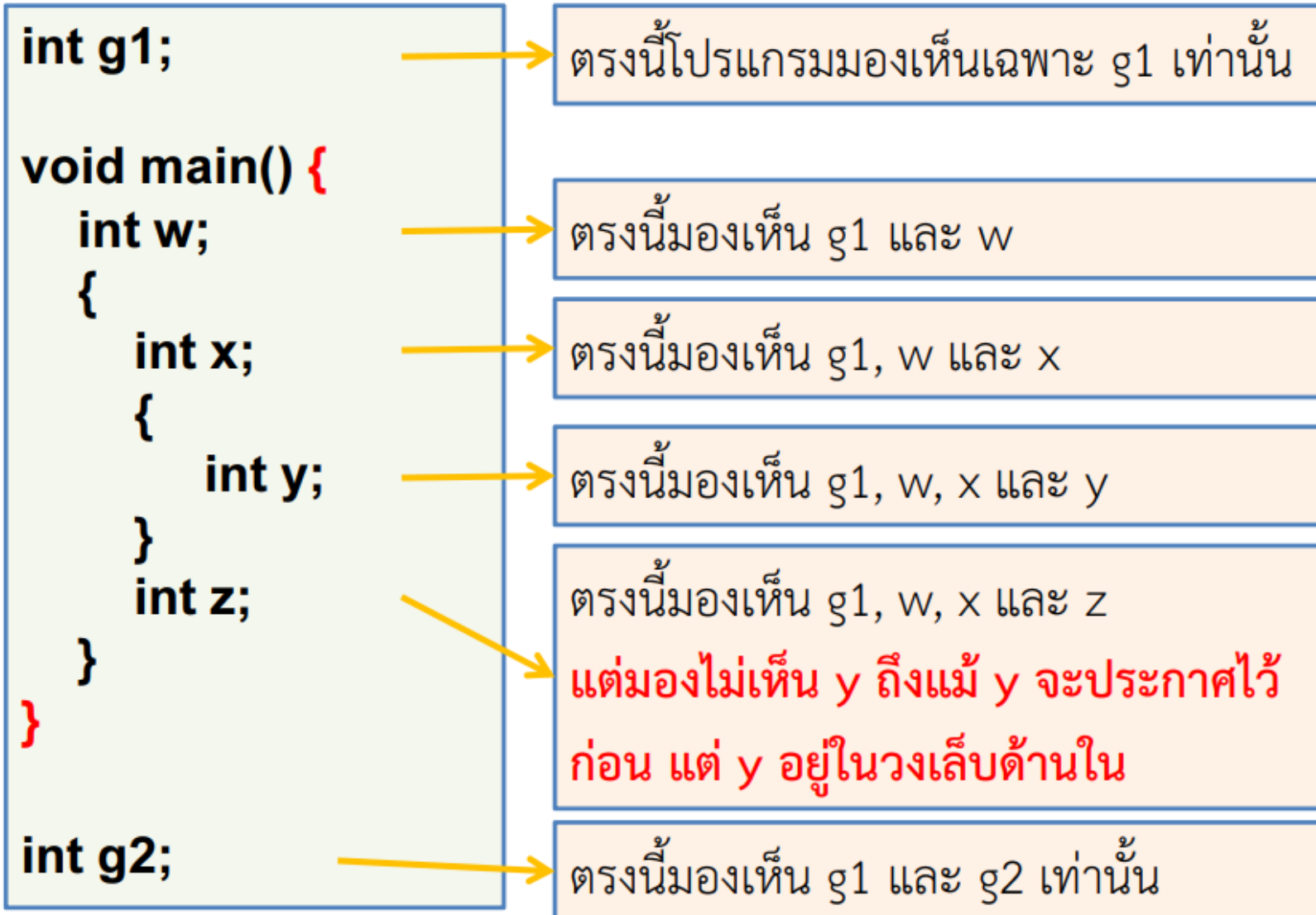
```
void main() {  
    x = 7;  
    int x;  
}
```

→ แบบนี้ใช้ไม่ได้ เพราะเราอ้างถึง x ก่อนการประกาศตัวแปร

# ขอบเขตการมองเห็นตัวแปร (Variable scope)

- ตัวแปรที่ประกาศไว้ในวงเล็บปีกกา เรียกว่าตัวแปรเฉพาะที่ (local variable)
  - ตัวแปรที่ประกาศไว้นอกวงเล็บปีกกาเรียกว่าตัวแปร ครอบคลุม (global variable)
1. โปรแกรมจะมองเห็นตัวแปรภายหลังการประกาศแล้วเท่านั้น ไม่ว่าจะเป็นแบบ Local หรือ Global
  2. โปรแกรมจะมองเห็นตัวแปรแบบ Local ภายในบริเวณวงเล็บปีกกาเดียวกัน และ วงเล็บปีกกาที่ซ้อนอยู่ข้างใน (จะซ้อนอยู่กี่ชั้นก็มองเห็นหมด) แต่ก็มองเห็นหลังการประกาศตัวแปรแล้วเท่านั้น (กฎข้อที่ 1 สำคัญที่สุด)
  3. โปรแกรมจะมองเห็นตัวแปรแบบ Global ภายในวงเล็บปีกกาทุกอันที่ตามหลังการประกาศตัวแปร Global นั้น

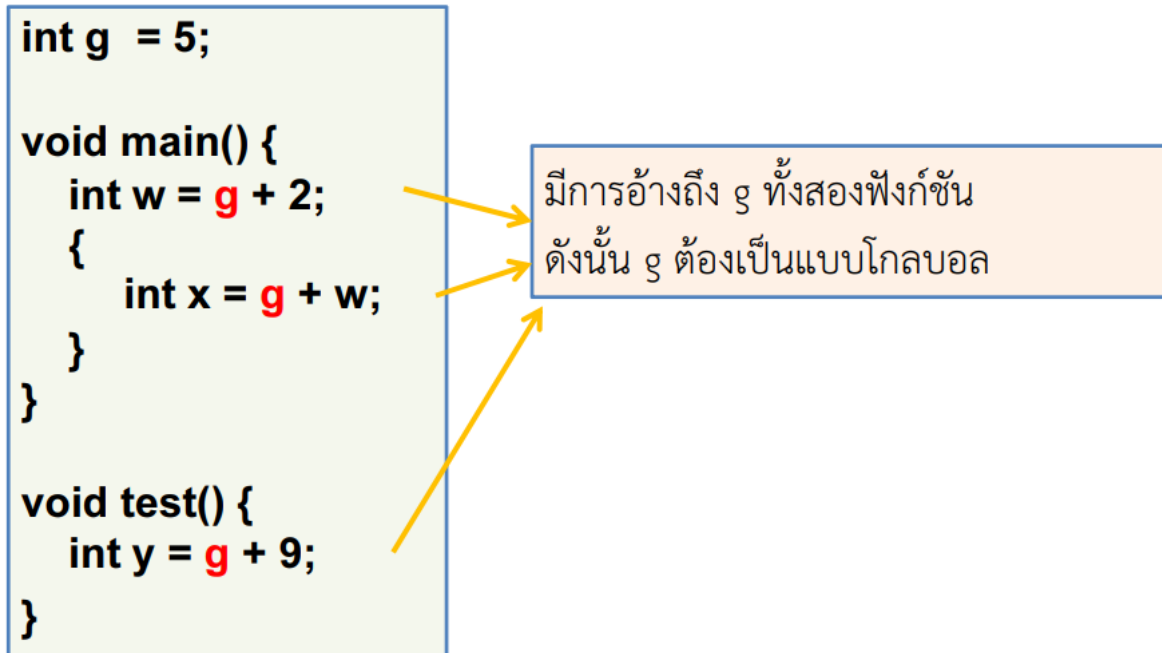
# ขอบเขตการมองเห็นตัวแปร (Variable scope)



# ขอบเขตการมองเห็นตัวแปร (Variable scope)

จะประกาศตัวแปรนอก main ทำไม แตกต่างจากใน main อย่างไร?

คำตอบคือในอนาคตเราจะได้เขียนโปรแกรมออกไปจนถึงนอกฟังก์ชัน main เลย แต่ตอนนี้ยังก่อน



```
void main() {  
    int w = 5;  
    {  
        int x = w + 3;  
        printf("%d\n", x);  
    }  
    {  
        int x = w + 5;  
        printf("%d\n", x);  
    }  
}
```

- การใช้ปีกการแยงงานทำให้เราสามารถจัดการโค้ดได้ง่ายขึ้น
- สามารถจัดสรรตัวแปรที่เหมาะสมกับงานแต่ละส่วนได้ เพราะตัวแปรจะไม่ไปปะปนกับตัวแปรที่อยู่ด้านนอก
- สังเกตว่าในตัวอย่างมีการใช้ตัวแปร x ซ้ำกันได้ อย่างไม่มีปัญหา เพราะ x ในแต่ละบล็อกก็ทำหน้าที่ของตัวเอง

# ปีกกาแยงงาน: การซำกัันของชื่อตัวแปรต่างบล็อก

- ถึงแม้เราจะมีกฎการตั้งชื่อตัวแปรว่าห้ามซำกััน แต่แท้จริงแล้วกฎนั้นจะจงกับตัวแปรภายในบล็อกเดียวกันเท่านั้น
- ตัวแปรที่อยู่**คนละบล็อก**กันชื่อซำกัันได้ นี่เป็นเหตุผลว่าการแบ่งบล็อก (หรือฟังก์ชัน) ทำให้เราสามารถแบ่ง โปรแกรมเป็นส่วน ๆ ที่อิสระจากกันได้ ทำให้เราไม่ต้องกังวลว่าการเขียนในที่หนึ่งจะไปกระทบกับอีกที่หนึ่ง
- แม้แต่บล็อกที่ซ้อนเข้าไปก็มีชื่อตัวแปรที่ซำกัันได้ ในกรณีนี้ตัวแปรที่ประกาศไว้ล่าสุดจะมีลำดับความสำคัญสูงกว่า ดังตัวอย่าง



# ปีกกาแยงงาน: การซ้ำกันของชื่อตัวแปรต่างบล็อก

```
void main() {  
    int x = 5;  
    int w = 2;  
  
    {  
        int x = w + 1;  
        printf("%d %d\n", x, w);  
    }  
  
    printf("%d %d\n", x, w);  
}
```

มีชื่อ x ซ้ำได้ แม้ว่าจะเป็นบล็อกที่ซ้อนกันก็ตาม

x ตรงนี้เป็นไปตามที่ประกาศล่าสุดที่โปรแกรมมองเห็น มีค่าเท่ากับ 3

x ตรงนี้ก็จะเป็นไปตามที่ประกาศล่าสุดที่โปรแกรมมองเห็น มีค่าเท่ากับ 5

ผลลัพธ์ที่พิมพ์ออกมาคือ 3 2

5 2

# แบบฝึกหัด (1)



จงหาผลลัพธ์ของโปรแกรม

```
void main() {  
    int x = 5;  
    int w = 2;  
    {  
        w = x + 1;  
        int x = w + 1;  
        printf("%d %d\n", x, w); x = x - 1;  
    }  
    printf("%d %d\n", x, w);  
}
```

## แบบฝึกหัด (2)



ตรงไหนอ้างอิงถึงตัวแปรผิด

```
int g1 = 1;
void main() {
    int x = 5;
    int w = g1 + g2;
    {
        y = x + 1;
        int y;
        int x = w + g1;
    }
}
int g2;
```

# Memory Constance

- Memory Constance คือค่าที่เราสั่งให้เครื่องจำไว้ และไม่ให้เปลี่ยนค่าเป็นอันขาด
- ก่อนหน้านี้เราให้ตัวแปรเปลี่ยนค่าได้ เช่น

```
int x = w + 1;    //ประกาศตัวแปรประเภทจำนวนเต็ม ชื่อ x
x = x - 1;        // เปลี่ยนค่า x โดยให้ลดลงจากเดิม 1
```

- แต่ในบางครั้งเราไม่ต้องการให้มีการเปลี่ยนค่า เพราะตัวแปรบางอย่างควรเป็นค่าคงที่ตลอด เพื่อป้องกันการเปลี่ยนค่าพวกนี้โดยบังเอิญ เราจึงมีการใช้คำพิเศษ **const** เพื่อบังคับให้คอมไพเลอร์ป้องกันการแก้ค่าพวกนี้ เช่น

```
const double PI = 3.1415926535;
```

# Memory Constance



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

```
void main() {
```

```
    const double PI = 3.1415926535;
```

```
    double radius = 1.5;
```

```
    double circle_area = PI * radius * radius;
```

```
    radius = radius + 1;
```

```
    PI = PI + 1;           //จะ Compile error เพราะมีการเปลี่ยนค่า constant
```

```
    circle_area = PI * radius * radius;
```

```
}
```

# การแสดงผลขั้นพื้นฐาน



- เราได้แอบใช้คำสั่งสำหรับแสดงผลออกทางจอภาพหลายครั้งแล้ว แต่ยังไม่ได้พูดถึงรายละเอียด
- ถ้าจำกันได้ คำสั่งนั้นคือ **printf**
- **printf** เป็นคำสั่งสำหรับแสดงผลออกทางจอภาพ (ใช้บ่อยมากในวิชานี้)
- เราสามารถพิมพ์ข้อความ (string) ออกทางจอภาพได้ โดยใส่ไว้ในวงเล็บหลัง **printf**

```
printf("Hello World");
```

```
printf("Silpakorn");
```

```
printf("My name is Jeff");
```

# การแสดงผลขั้นพื้นฐาน: แสดงค่าจากตัวแปร

- บางครั้งข้อความของเราไม่ได้ตายตัว เพราะต้องมีการเปลี่ยนแปลงค่าตามตัวแปร
- ตัวอย่างเช่นการบอกว่า ผลบวกของ  $x$  และ  $y$  คือเท่าไร
- เราจะส่งตัวแปรเข้าไปในข้อความด้วยสัญลักษณ์พิเศษ โดยใส่ตัวแปรลงทางด้านหลังข้อความ และใส่สัญลักษณ์พิเศษเข้าไปในประโยค ณ ตำแหน่งที่อยากใส่ค่านั้นเข้าไป

```
sum = x+y;
```

```
printf(“%d”, sum);
```

- ในที่นี้สัญลักษณ์พิเศษคือ `%d` เพราะต้องการพิมพ์จำนวนเต็ม (แต่ละประเภทจะไม่เหมือนกัน)
- ถ้าผลลัพธ์ของข้อนี้เป็น 15 (หมายถึง `sum = 15`) ผลลัพธ์ก็จะออกมาเป็น 15

# การแสดงผลขั้นพื้นฐาน: แสดงค่าจากตัวแปร

```
sum = x+y;
```

```
printf("The Final value is %d.", sum);
```

- ถ้าผลลัพธ์ของข้อนี้เป็น 15 (หมายถึง  $sum = 15$ ) ผลลัพธ์ก็จะออกมาเป็น

**“The Final value is 15.”**

- จะใส่มากกว่า 1 ตัวก็ได้ ผลก็คือตัวแปรข้างหลังจะแทรกเข้าสู่สัญลักษณ์พิเศษตามลำดับ

```
int x = 5, y = 12;
```

```
sum = x+y;
```

```
printf("The summation of %d and %d is %d.", x, y, sum);
```

- ผลจะได้ว่า

**“The summation of 5 and 12 is 17.”**



# การแสดงผลขั้นพื้นฐาน: แสดงค่าจากตัวแปร

## ชนิดของตัวแปรกับสัญลักษณ์พิเศษ

สัญลักษณ์พิเศษที่ใช้แทรกค่าของตัวแปรจะเปลี่ยนไปตามประเภทของข้อมูล

- จำนวนเต็ม (int) ใช้ %d
- ทศนิยม (float & double) ใช้ %f
- ตัวอักษร (char) ใช้ %c
- และยังมีตัวแปรและเทคนิคอื่นๆ อีก จะค่อยๆ เรียนรู้กันไปทีละน้อย

# การแสดงผลขั้นพื้นฐาน: แสดงข้อความผสมอักขระพิเศษ

อักขระพิเศษในที่นี้คืออักขระที่ขึ้นด้วยเครื่องหมาย \ เช่น \n, \t เป็นต้น

- ในการแสดงผล ก็สามารถนำอักขระเหล่านี้ใส่ลงในข้อความตรงๆ ได้เลย เช่น

```
printf("The World.\nis mine");
```

จะได้ผลลัพธ์ดังนี้

The World

is mine

- หมายเหตุตัวใดๆว่า \n ใช้บ่อยมาก

# การแสดงผลขั้นพื้นฐาน: แสดงข้อความผสมอักขระพิเศษ



ทำไม \n จึงถูกใช้บ่อย?

- นักศึกษาคิดว่าผลจากการพิมพ์ข้อความเหล่านี้จะเป็นอย่างไร

```
printf("The World is mine.");
```

```
printf("I am Groot");
```

```
printf("Hello world, again.");
```

```
printf("The result is %f", x_float);
```

```
//สมมติ x_float = 3.01
```

# การแสดงผลขั้นพื้นฐาน: แสดงข้อความผสมอักขระพิเศษ



ทำไม \n จึงถูกใช้บ่อย?

- นักศึกษาคิดว่าผลจากการพิมพ์ข้อความเหล่านี้จะเป็นอย่างไร

```
printf("The World is mine.");
```

```
printf("I am Groot");
```

```
printf("Hello world, again.");
```

```
printf("The result is %f", x_float); //สมมติ x_float = 3.01
```

ผลคือ

The World is mine.I am GrootHello World, again.The result is 3.01

สังเกตว่าสิ่งพิมพ์ไปแค่ไหนก็ได้แค่นั้นจริง ๆ ถ้าไม่สั่งให้ขึ้นบรรทัดใหม่ โปรแกรมก็จะไม่ขึ้นให้

# การแสดงผลขั้นพื้นฐาน: แสดงข้อความผสมอักขระพิเศษ



ทำไม \n จึงถูกใช้บ่อย?

- นักศึกษาคิดว่าผลจากการพิมพ์ข้อความเหล่านี้จะเป็นอย่างไร?

```
printf("The World is mine.\n");
```

```
printf("I am Groot\n");
```

```
printf("Hello world, again.\n");
```

```
printf("The result is %f", x_float);
```

```
//สมมติ x_float = 3.01
```

ผลคือ

# การแสดงผลขั้นพื้นฐาน: แสดงข้อความผสมอักขระพิเศษ



The World is mine.

I am Groot

Hello World, again.

The result is 3.01

# การแสดงผลขั้นพื้นฐาน: แสดงข้อความผสมอักขระพิเศษ

- นักศึกษาคิดว่าผลจากการพิมพ์ข้อความเหล่านี้จะเป็นอย่างไร?

```
printf("The World is mine.\nI am Groot\nHello world, again.\nThe result is %f" , x_float);
```

//สมมติ  $x\_float = 3.01$

ผลคือ?

# การแสดงผลขั้นพื้นฐาน: ตัวอย่างลองทำ

ตัวอย่าง : จงพิมพ์ตัวแปรที่แทนคู่ลำดับสองคู่ หนึ่งคู่ต่อบรรทัด โดยมีตัวแปร  
ชื่อ x1, y1, x2, และ y2 ซึ่งเป็นจำนวนเต็ม ให้มีผลลัพธ์ในรูปแบบ

(x1, y1)

(x2, y2)

โค้ด : แบบม้วนเดียวจบ

```
printf("(%d, %d)\n(%d, %d)", x1, y1, x2, y2);
```

โค้ด : แบบอ่านง่าย

```
printf("(%d, %d)\n", x1, y1);
```

```
printf("(%d, %d)", x2, y2);
```



# สรุป



- ความสัมพันธ์ของฟลอชาร์ต ซูโดโค้ด และภาษาคอมพิวเตอร์
- ภาษาซีกับการเขียนโปรแกรม
- การเริ่มต้นและจบโปรแกรม (stdio.h/void main)
- ตัวแปร (จำนวนเต็ม, ทศนิยม, อักขระ/วิธีการประกาศ/ตั้งชื่อ/Block)
- ค่าคงที่ในภาษาซี (const)
- การแสดงผลอย่างง่าย (printf/ สัญลักษณ์พิเศษ %d, %f, %c/ การขึ้นบรรทัดใหม่ด้วย \n)

# การแสดงผลขั้นพื้นฐาน: ตัวอย่างลองทำ



จงหาคำตอบว่าโปรแกรมจะพิมพ์อะไรออกมาทางจอภาพ

```
int i = 1;
int j = 2;
void main() {
    int x = 3;
    int y = 4;

    printf(" (%d, %d)", y, y);
    printf("%d %d\n", i, j);
    printf("%d%d%d%d", i, j, x, y);
}
```

คำแนะนำ : ระวังการเว้นวรรคและขึ้นบรรทัดใหม่ในคำตอบ

# การแสดงผลขั้นพื้นฐาน: ตัวอย่างลองทำ

จงหาคำตอบว่าโปรแกรมจะพิมพ์อะไรออกมาทางจอภาพ

```
void main() {  
    int w = 5; int x = 3; int y = 2;  
    {  
        int x = w + 3;  
        printf("%d\n", x);  
        {  
            int x = w + y;  
            printf("%d\n", x);  
            w = w - 1; x = x - 1;  
        }  
        printf("(%d, %d)", w, x);  
    }  
    printf("(%d, %d)", w, x);  
}
```

# การแสดงผลขั้นพื้นฐาน: ตัวอย่างลองทำ

ตรงไหนบ้างที่มีการอ้างอิงตัวแปรที่ผิดไป (มี 3 ตำแหน่ง หาให้ครบด้วย)

```
int g1 = 1;
void main() {
    int x = 5;
    int w = g1 + g2;

    {
        y = x + 1;
        int y;
        int x = w + g1;
    }
    printf("%d %d\n", x, y);
}
int g2;
```

# Note

