



การเขียนโปรแกรมคอมพิวเตอร์ 1

Computer Programming I

การซ้อนลูป (Nested Loop)

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



การซ้อนลูปคืออะไร

การซ้อนลูปคือการกำหนดขั้นตอนการทำงานที่มีการวนซ้ำมากกว่าหนึ่งระดับ

- ถ้ามีสองระดับเรามักเรียกว่าลูปสองชั้น

```
for (int j = 0; j < M; ++j) { → ลูปชั้นนอก
    for (int i = 0; i < N; ++i) { → ลูปชั้นใน
        printf("%d %d", j, i);
    }
}
```

- ถ้ามีสามระดับเรามักเรียกว่าลูปสามชั้น
- โดยปรกติจะไม่ค่อยเจอลูปที่มากกว่าสามชั้นโดยตรง เพราะผู้เขียนโปรแกรมจะเลี่ยงไปใช้ฟังก์ชันประกอบเพื่อให้โค้ดในโปรแกรมเข้าใจง่ายขึ้น
 - แต่ในบริบทของการทำงานก็อาจจะเป็นลูป 4 ชั้น (หรือมากกว่า) เช่นเดิม



เราเอาลูปสองชั้นไปทำอะไร

ก่อนที่จะเรียนการเขียนลูปสองชั้น เรามาดูกันก่อนดีกว่าว่าเราเอาไปทำอะไรได้บ้าง

- ใช้ในการจัดการตาราง (คือมีข้อมูลหลายแถวและหลายคอลัมน์)
- ใช้ในการจัดการรูปภาพ (ที่จริงภาพดิจิทัลก็คือตารางที่มีหลายแถวและหลายคอลัมน์)
- ใช้ในการคำนวณที่ข้อมูลชุดเดิมถูกอ่านหรือเขียนซ้ำหลาย ๆ รอบ
 - ประกติเท่าที่ผ่านมามีข้อมูลหนึ่งชุดใหญ่ ๆ ที่เราอ่านเขียนครั้งเดียว
→ เราใช้ลูปหนึ่งชั้นสำหรับอ่านเขียนข้อมูลชุดนั้นในแต่ละรอบ
 - ถ้าต้องอ่านเขียนข้อมูลชุดเดิมหลายรอบก็มักจะต้องการลูป 2 ชั้น
 - เช่น จากแถวข้อมูลข้างล่างซึ่งมีช่องข้อมูลต่อกันไปรวม 15 ช่อง จงหาช่องตัวเลขที่ติดกัน 5 ช่องที่มีผลรวมตัวเลขข้างในมากที่สุด

7	2	3	1	0	4	4	6	5	3	5	3	6	5	2
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

* ในวิชานี้เราจะเรียนเฉพาะลูป 2 ชั้น เพราะต้องการเน้นที่พื้นฐาน แต่ถ้าใครเข้าใจหลักการทำงานก็มักจะทำลูป 3 ชั้นหรือ 4 ชั้นได้เองอย่างเป็นธรรมชาติ



ลองใช้รูปสองชั้นกับข้อมูลในรูปแบบตาราง

สมมติว่าตารางของเรามีจำนวน 5 แถว และ 6 คอลัมน์

เราต้องการพิมพ์หมายเลขแถวและคอลัมน์ของตารางลงไปในรูปแบบข้างล่าง

(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)
(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)

นั่นคือเราต้องการแสดงตำแหน่งออกมาเป็นคู่ลำดับ โดยแสดงตำแหน่งแถวออกมาก่อนตำแหน่งคอลัมน์ เมื่อจบแต่ละแถวเราก็สั่งขึ้นบรรทัดใหม่ แล้วพิมพ์คู่ลำดับออกมาในลักษณะเดิม

โค้ดแสดงลำดับของหมายเลขแถวและคอลัมน์



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

```
void main() {
```

```
    for(int row = 1; row <= 5; ++row) {
```

ลูปด้านในพิมพ์ข้อความออกมา ขอให้เข้าใจด้วยว่าลูปด้านในจะต้องวนจนครบ 6 รอบก่อน มันถึงจะหลุดออกมา และใน 6 รอบนี้ค่า row จะเหมือนเดิมเพราะลูปด้านนอกไม่ถูกแตะต้อง ค่า row จึงไม่เปลี่ยน

```
        for(int col = 1; col <= 6; ++col) {
```

```
            printf(" (%d, %d) ", row, col);
```

```
        }
```

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

```
    }
```

```
}
```

เนื่องจากผลลัพธ์ถูกจัดการตามแถวก่อน เราจึงเอาแถวออกมาเป็นลูปด้านนอก ส่วนคอลัมน์เป็นลูปด้านใน (ทราบหรือไม่ว่าทำไมจึงเป็นเช่นนี้)

พิมพ์ขึ้นบรรทัดใหม่หลังจากพิมพ์คอลัมน์จนครบ (จัดเป็นส่วนของลูปด้านนอก)



ลำดับการทำงานของลูป 2 ชั้น

- อันที่จริง ลำดับการทำงานของลูป 2 ชั้นนั้นก็จะเป็นไปตามแนวคิดของลูปชั้นเดียวทุกประการ เพียงแต่ผู้เรียนบางท่านยังไม่คล่องเรื่องลูปชั้นเดียว
→ ทำให้งงหนักยิ่งกว่าเดิม และเราควรกลับมาดูที่พื้นฐานตรงนี้ก่อน
- ขอใช้ลูปด้านในจากตัวอย่างที่แล้วเป็นกรณีศึกษา โดยกำหนดให้ row = 1 เป็นค่าคงที่ตายตัวไว้ก่อน

```
int row = 1;
for(int col = 1; col <= 6; ++col) {
    printf(" (%d, %d) ", row, col);
}
printf("\n");
```

- ตอนนี้เราคงเห็นได้ชัดเจนขึ้นว่าทำไมตอนที่วนลูปด้านในแล้วค่าตัวเลขคอลัมน์จึงเปลี่ยนไปเรื่อย ๆ แต่ตัวเลขแสดงแถวเป็นค่าคงที่ตลอด



ถ้าทำแบบเดิมซ้ำ ๆ ต่อกันไปละ

ถ้าเราเขียนลูปแบบเดิมซ้ำ แต่เปลี่ยนค่า row ไปด้วย เราก็จะได้ผลลัพธ์ ออกมาสองแถว โดยมีเลขแถวเปลี่ยนไป ส่วนเลขคอลัมน์จะมีการวน เปลี่ยนแปลงในลักษณะเดิม

```
int row = 1;
for(int col = 1; col <= 6; ++col) {
    printf("(%d, %d) ", row, col);
}
printf("\n");

row = 2;
for(int col = 1; col <= 6; ++col) {
    printf("(%d, %d) ", row, col);
}
printf("\n");
```

ที่จริงการเอาลูปมาครอบอีกชั้นก็คือการทำแบบเมื่อกี่นี้แหละ



ถ้าเราเอาลูปด้านนอกมาครอบ ผลก็คือเวลาที่มันทำลูปด้านในเสร็จ มันจะออกมาที่ `printf("\n");` เสร็จแล้วก็ตีกลับขึ้นไปเปลี่ยนค่า `row` ใหม่ วนเช่นนี้ไปเรื่อย ๆ จนกว่าเงื่อนไขลูปด้านนอกจะไม่เป็นจริง

```
for(int row = 1; row <= 5; ++row) {  
  
    for(int col = 1; col <= 6; ++col) {  
        printf(" (%d, %d) ", row, col);  
    }  
    printf("\n");  
  
}
```




ลองตอบคำถามนี้ดู

- โปรแกรมนี้จะพิมพ์เลขอะไรออกมาบ้าง

```
#include <stdio.h>

void main() {
    int sum = 0;
    for(int i = 0; i < 7; ++i) {
        for(int j = 0; j < 3; ++j) {
            sum = sum + 1;
        }
        printf("%d ", sum);
    }
}
```

- คำตอบ

3 6 9 12 15 18 21



แล้วคำถามนี้ล่ะ

- โปรแกรมนี้จะพิมพ์เลขอะไรออกมาบ้าง

```
#include <stdio.h>

void main() {
    for(int i = 0; i < 7; ++i) {
        for(int j = 0; j < 3; ++j) {
            printf("%d ", i + j);
        }
    }
}
```

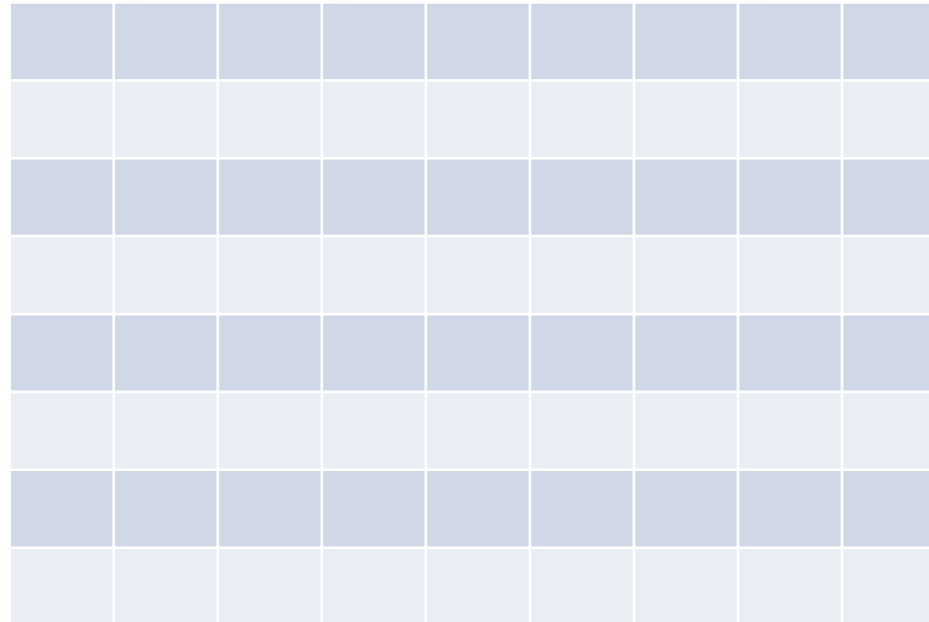
- คำตอบ

0 1 2 1 2 3 2 3 4 3 4 5 4 5 6 5 6 7 6 7 8



อุปสองชั้นในการจัดการรูปภาพ

- ที่จริงภาพดิจิทัลนั้นมีลักษณะเป็นตารางเก็บข้อมูลที่มีช่องตารางอยู่อย่างหนาแน่น

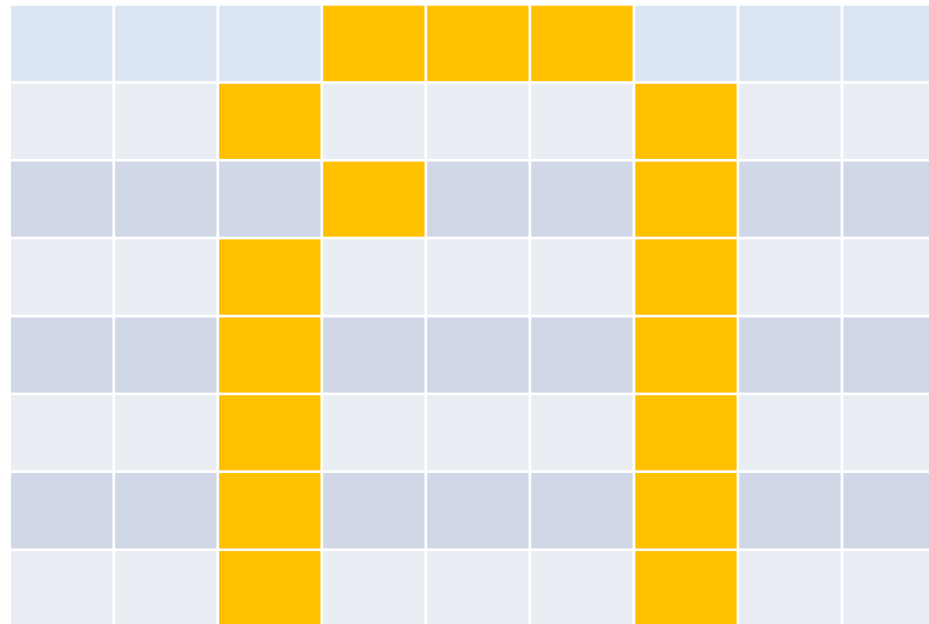


- ช่องแต่ละช่องในตารางแท้จริงแล้วก็คือองค์ประกอบของภาพ



อุปสองชั้นในการจัดการรูปภาพ

- ที่จริงภาพดิจิทัลนั้นมีลักษณะเป็นตารางเก็บข้อมูลที่มีช่องตารางอยู่อย่างหนาแน่น



- ช่องแต่ละช่องในตารางแท้จริงแล้วก็คือองค์ประกอบของภาพ
- ถ้าเราเลือกใส่ค่าสีที่เหมาะสมลงไปในตารางมันก็จะกลายเป็นภาพได้



แต่เรายังทำขนาดนั้นไม่ได้ในตอนนี

- เนื่องจากการเก็บข้อมูลในภาพนั้นจำเป็นต้องมีตารางหรือแถวข้อมูลที่ทำให้เราอ่านเขียนค่าได้หลายครั้ง ทำให้เราจำเป็นที่จะต้องใช้โครงสร้างข้อมูลที่เหมาะสมกับงานแบบนี้
- โครงสร้างที่เหมาะสมคืออาเรย์ ซึ่งเราจะเรียนในสัปดาห์ถัดไป
- ในตอนนี้เราจะมาเรียนรู้วิธีใช้ลูปสองชั้นโดยไม่ต้องใช้อาเรย์ไปก่อน
 - ดังนั้นรูปแบบการใช้งานจะค่อนข้างจำกัด
 - แต่เป็นขั้นตอนการเรียนรู้ที่สำคัญมาก เพราะลูปสองชั้นถ้าเอาไปใช้กับอาเรย์ได้อย่างถูกต้องและเหมาะสมมันจะมีประโยชน์มากอย่างไม่น่าเชื่อ



พิมพ์ตัวเลขชั้นบันได

- เพื่อที่จะเรียนรู้แนวคิดอุปสองชั้น เรามาดูตัวอย่างเพิ่มเติม
- สมมติว่าผู้ใช้ใส่เลข 5 เข้ามาแล้วเราต้องการพิมพ์ว่า

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

- ถ้าผู้ใช้ใส่เลข 10 เข้ามา โปรแกรมก็ต้องไล่ไปจนถึง 10 ถ้าใส่จำนวนเต็มบวก N เข้ามา ก็ต้องไล่ไปเรื่อย ๆ จนถึง N
- แสดงว่าต้องมี N แถว ส่วนจำนวนคอลัมน์ก็ตรงกับหมายเลขแถวที่โปรแกรมกำลังพิมพ์นั่นเอง



พิจารณาการพิมพ์ในแต่ละแถว

- ลองกำหนดเลขแถวตายตัวไว้ที่ `int row = 4;` ก่อน
→ แสดงว่าเราจะพิมพ์เลข 1 2 3 4
- โค้ดที่ได้ก็จะมีหน้าต่างทำนองนี้

```
int row = 4;  
for(int col = 1; col <= row; ++col) {  
    printf("%d ", col);  
}  
printf("\n");
```

- ที่เหลือก็คือเราต้องเอาลูปอีกชั้นมาครอบเพื่อให้มันเปลี่ยนเลขแถวได้อย่างที่ควรจะเป็น
(ลองคิดดูด้วยตัวเองให้ดีก่อนว่าควรเขียนลูปด้านนอกอย่างไร)



เอาลูปด้านนอกมาแปะเพื่อเปลี่ยนค่า row

- จุดหลักคือเปลี่ยนค่า row ให้ได้อย่างที่ควรเป็น ถ้าทำได้ตัวเลขที่ถูกพิมพ์ออกมาในแต่ละแถวก็จะออกมาอย่างที่เราต้องการ (เพราะเลขแถวมันถูกนั่นเอง)

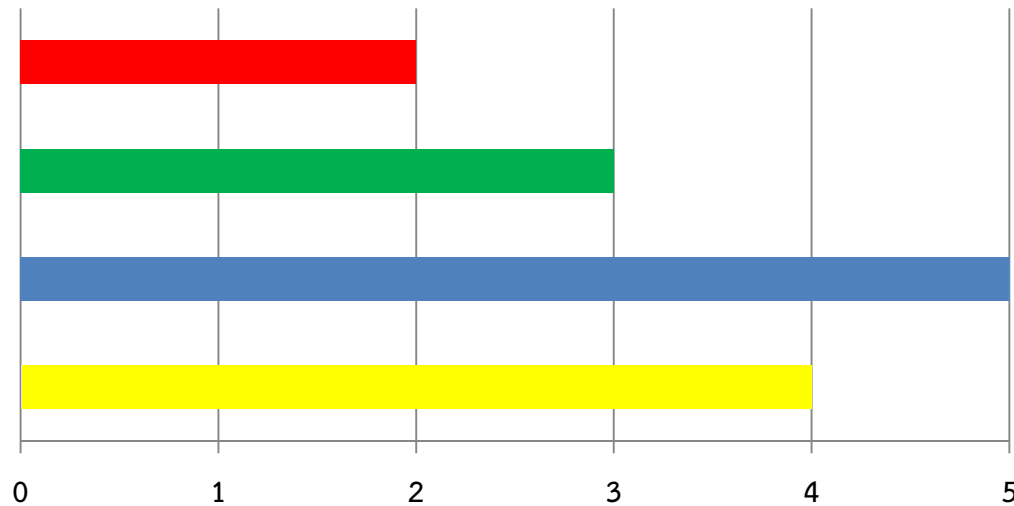
```
int N;  
scanf("%d", &N);  
for(int row = 1; row <= N; ++row) {  
    for(int col = 1; col <= row; ++col) {  
        printf("%d ", col);  
    }  
    printf("\n");  
}
```

- สังเกตหรือไม่ว่าจริง ๆ แล้วการคิดมาจากลูปด้านในก่อนเป็นเรื่องธรรมดา
- สำหรับคนที่ยังไม่คล่องควรคิดตามแนวทางนี้ แบบที่คิดรวดเดียวจบมันทำได้ เฉพาะคนที่เข้าใจและเริ่มชำนาญกับเรื่องลูปสองชั้นแล้ว



พิมพ์กราฟแท่งแนวนอนแบบใช้ข้อความแทน

- คิดว่าเราคงเคยเห็นกราฟแท่งทำนองนี้มาก่อน



- เราจะแทนค่าออกมาด้วยจำนวนเครื่องหมาย * จากข้างบนไปเป็น

* *

* * *

* * * * *

* * * *



โปรแกรมพิมพ์กราฟแนวนอน

- **ข้อมูลเข้า** เป็นเลขจำนวนเต็มบวกหรือ 0 จำนวน N ค่าที่บอกความยาวของกราฟแท่งจำนวน N แท่ง จุดสิ้นสุดของข้อมูลคือเลขจำนวนเต็มลบ
- **ตัวอย่าง** 2 3 5 4 -1
- **ผลลัพธ์**
กราฟแท่งแนวนอน หนึ่งแท่งต่อหนึ่งบรรทัด แต่ละแท่งประกอบด้วยเครื่องหมาย * มีจำนวนตามตัวเลขแต่ละค่าที่ผู้ใช้ใส่เข้ามา



แนวคิด

- เรารู้ว่าจากตัวเลขแต่ละตัว เราจะต้องพิมพ์ดอกจันออกมาตามค่าตัวเลขนั้น
 - นั่นคือเราต้องวนลูปพิมพ์ดอกจันตามจำนวนรอบที่ถูกระบุด้วยตัวเลขดังกล่าว
 - เมื่ออ่านเลขเข้ามา โปรแกรมก็ต้องวนลูปทำงานองนี้ เพื่อพิมพ์ดอกจัน ...

```
int k;  
scanf("%d", &k);  
for(int i = 0; i < k; ++i) {  
    printf("*");  
}  
printf("\n");
```

- โค้ดข้างบนคือใจความ แต่เราต้องนำทุกอย่างมารวมกันเพื่อพิมพ์กราฟให้ครบทุกแท่งและหยุดทำงานเมื่อพบเลขติดลบตามข้อกำหนด
- และเพราะเราต้องวนพิมพ์กราฟทุกแท่ง จึงต้องมีลูปมาครอบอีกชั้น



ลูปสองชั้นที่สมบูรณ์

- ในโค้ดทางด้านใต้ เราเอาลูป while(1) มาครอบเพื่อให้โปรแกรมวนรับค่าความยาวกราฟมาพิมพ์เพิ่มได้เรื่อย ๆ จนกว่าจะเจอเลขติดลบ
- เราต้องมีการ break ที่ลูปด้านนอก เพื่อสั่งหยุดลูปด้านนอกและจบโปรแกรมได้

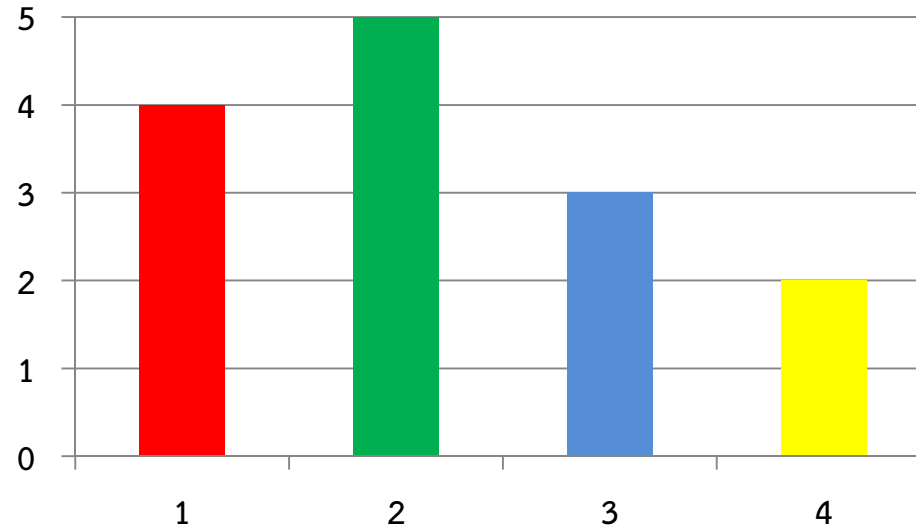
```
while (1) {  
    int k;  
    scanf ("%d" , &k) ;  
    if (k < 0)  
        break ;  
    for (int j = 0; j < k; ++j) {  
        printf ("*") ;  
    }  
    printf ("\n") ;  
}
```

- break อยู่ในลูปชั้นใด โปรแกรมก็จะหยุดลูปในชั้นนั้น



แล้วถ้าเป็นกราฟแท่งแนวตั้งล่ะ

- สมมติว่าเรามีกราฟแท่งแนวตั้งแบบนี้



- แล้วอยากได้แบบนี้

```
      *
*    *
*    *  *
*    *  *  *
*    *  *  *
```



ที่จริงเราทำไมไม่ได้ละ (ด้วยความรู้แค่นี้)

- เมื่อเปลี่ยนมาเป็นกราฟแท่งแนวตั้ง เราอาจจะเข้าใจไปว่าทุกอย่างมันก็น่าจะคล้าย ๆ เดิม และเราใช้วิธีเดิม ๆ ได้
- แต่เอาเข้าจริง ปรากฏว่าเราทำไมไม่ได้ เพราะกว่าจะรู้ว่าควรตั้งฐานกราฟไว้ที่บรรทัดใด เราก็ต้องอ่านข้อมูลให้ครบก่อนเพื่อที่จะหากราฟแท่งที่สูงที่สุด
 - แต่พอรู้ข้อมูลตรงนี้แล้ว ก็จะมีปัญหาว่าค่าตัวเลขที่อ่านมาไม่ได้ถูกเก็บไว้
 - จะสร้างตัวแปรมาเก็บไว้ก็คงต้องสร้างมาเป็นร้อย ถ้าแท่งมันมีเป็นร้อย
 - ปัญหาทำนองนี้เราควรใช้อาเรย์เข้าช่วยเพื่อรวมค่าตัวแปรต่าง ๆ ลงเป็นชุดเดียวกันและใช้โค้ดเดียวกันให้ได้
- ดังนั้นเราต้องเก็บปัญหาแบบที่จำนวนแท่งมีเป็นร้อยเอาไว้ก่อน มาลองทำแบบที่มันมีแค่ 3 แท่งแทนเพื่อเรียนแนวคิดของลูป 2 ชั้นขั้นพื้นฐาน



เขียนกราฟแท่งแนวตั้ง 3 แท่ง

- กำหนดให้กราฟแท่งแนวตั้งมีทั้งหมด 3 แท่ง มีความสูง x , y , และ z ค่าเหล่านี้เป็นจำนวนเต็มบวกหรือศูนย์ (ศูนย์คือเป็นแท่งเปล่า)
- กราฟทั้งสามแท่งต้องมีฐานจากบรรทัดเดียวกัน และบรรทัดแรกจะต้องมีดอกจันของกราฟแท่งที่สูงที่สุดอยู่ด้วย (กล่าวคือห้ามมีบรรทัดเปล่า)

แนวคิด

- แบบนี้ก็แสดงว่าเราจะต้องหาให้ได้ก่อนว่าความสูงของกราฟที่สูงที่สุดคือเท่าใด
- และเราก็ต้องคำนวณให้ได้ว่ากราฟแต่ละแท่งจะมีช่องว่างกี่บรรทัดจนกว่าจะมีดอกจันเป็นอันแรก
- จำนวนบรรทัดเปล่าที่ว่าเป็นสิ่งที่สัมพันธ์กับกราฟแท่งที่สูงที่สุด



หาความสูงแห่งกราฟที่มากที่สุด

```
int x, y, z;  
scanf("%d %d %d", &x, &y, &z);  
int max = INT_MIN;  
  
if(x > max)  
    max = x;  
if(y > max)  
    max = y;  
if(z > max)  
    max = z;
```

คำถามชวนคิด เวลาจะวาดกราฟแต่ละแห่ง เราต้องไล่เรียงทีละบรรทัด แล้วเราจะรู้ได้อย่างไรว่า ในแถวที่กำลังดำเนินการอยู่ กราฟแห่งนี้ต้องถูกเขียนด้วยช่องว่างหรือดอกจัน?




ออกแรงคิดสักเล็กน้อย

- ดูกราฟตัวอย่างข้างล่าง ($x = 2, y = 5, z = 3$)

```

*
*
* *
* * *
* * *

```

ที่นี้ดูแถวกับคอลัมน์แต่ละอัน 

สังเกตเห็นหรือไม่ว่า ความแตกต่างระหว่าง
ความสูงของแท่งที่สูงสุดกับแท่งที่เราพิจารณา
ก็คือจำนวนช่องว่างของแท่งกราฟนั้น ๆ
(เช่น แท่งแรกมีช่องว่าง 3 แถวก็เพราะว่า
 $5 - 2 = 3$ นั่นเอง)

	คอลัมน์ 1	คอลัมน์ 2	คอลัมน์ 3
แถว 1		*	
แถว 2		*	
แถว 3		*	*
แถว 4	*	*	*
แถว 5	*	*	*



ส่วนของการพิมพ์แท่งกราฟ

- เราต้องไล่ไปที่ละแถว เพราะมันเป็นข้อจำกัดของการพิมพ์ผลลัพธ์
- จากการวิเคราะห์ในหน้าที่แล้ว ถ้าหากเอาค่าสูงสุดลบด้วยความสูงของแท่งที่สนใจ จะได้จำนวนแถวที่เป็นช่องว่าง → ถ้าหมายเลขแถวเกินจุดนี้ไปก็แสดงว่าพิมพ์ *

```
for(int row = 1; row <= max; ++row) {  
    if(row > max - x) printf("*");  
    else printf(" ");  
  
    if(row > max - y) printf("*");  
    else printf(" ");  
  
    if(row > max - z) printf("*");  
    else printf(" ");  
  
    printf("\n");  
}
```



แล้วตกลงมันเป็นรูปชั้นเดียวใช่หรือเปล่า

- ในกรณีของตัวอย่างที่ถูกแปลงให้ง่ายลงนี้เป็นรูปชั้นเดียว
- เพราะเราเขียนโค้ดซ้ำ ๆ ตรงค่าตัวแปร x , y , และ z
 - ถ้าเรามีร้อยแห่งก็คงจะต้องมี $x_1, x_2, x_3, \dots, x_{100}$
 - แต่การใช้อาเรย์เราจะผนวกตัวแปรเข้าภายใต้ชื่อเดียวกันได้ และทำให้เราสามารถที่จะเปลี่ยนโค้ดที่ดูซ้ำ ๆ ไปเป็นรูปแทน
- เอาไว้มาดูอีกที่ตอนเรียนเรื่องอาเรย์แล้ว จะได้ภาคสมบูรณ์ออกมาที่หลัง



โจทย์พิมพ์กรอบสี่เหลี่ยม

- สมมติว่าผู้ใช้ใส่เลข 7 เข้ามาแล้วเราอยากได้กรอบตามแบบข้างล่างนี้

```
* * * * *
*           *
*           *
*           *
*           *
*           *
*           *
* * * * *
```

- ขอให้สังเกตให้ดีว่ามันมีบางช่วงของงานที่เหมือนกัน และบางช่วงที่ต่างกัน
→ ลูปของเราอาจจะต้องแบ่งเป็นหลาย ๆ ส่วน เราจะใช้ลูปที่เหมือนกันไปเสียทุกรอบก็คงจะไม่ดีเท่าไร (เรื่องทำนองนี้ถือว่าปรกติ)
- **หมายเหตุ** ดอกจันในบรรทัดแรกและสุดท้ายถูกคั่นด้วยช่องว่างหนึ่งช่อง



แยกงานเป็นส่วน ๆ

- บรรทัดแรกกับบรรทัดสุดท้ายเหมือนกัน
- ส่วนตรงกลางจะเป็นอีกกลุ่มที่ทำแบบเดียวกัน (แต่เป็นคนละพวกกับบรรทัดแรก)
- ดังนั้นงานจะถูกออกแบบมาเป็นสามส่วน
 1. พิมพ์บรรทัดแรก
 2. พิมพ์บรรทัดช่วงกลาง
 3. พิมพ์บรรทัดสุดท้าย (ใช้โค้ดเดียวกับของบรรทัดแรกได้)

รับข้อมูลความกว้างของกรอบ และ พิมพ์ดอกจันบรรทัดแรก



- โค้ดนี้พิมพ์ดอกจันคั่นด้วยช่องว่างและจบท้ายด้วยการขึ้นบรรทัดใหม่
- เราพิมพ์จำนวนดอกจันออกมาเป็นจำนวน N ค่า ในแถวแรก

```
int N;  
scanf ("%d", &N) ;  
  
for(int col = 1; col <= N; ++col) {  
    printf("* ");  
}  
printf ("\n") ;
```



พิมพ์แถวตรงกลาง

- จากทั้งหมด N แถว เราจะพิมพ์แถวที่ 2 ถึง $N - 1$ ในลักษณะที่มีเฉพาะดอกจันตรงขอบซ้ายขวา
- ดังนั้นตำแหน่งคอลัมน์แรกและคอลัมน์สุดท้ายในแถวจึงเป็นจุดพิเศษ
→ ไม่ต้องใช้ลูป
- ส่วนตรงกลางเป็นของที่เหมือนกัน ๆ ใช้การวนซ้ำได้

```
for(int row = 2; row <= N - 1; ++row) {  
    printf("* ");  
    for(int col = 2; col < N; ++col) {  
        printf(" ");  
    }  
    printf("*\n");  
}
```



ปิดท้ายด้วยแถวสุดท้าย

- แถวสุดท้ายทำเหมือนกับแถวแรก แต่เราไม่ต้องสั่งขึ้นบรรทัดใหม่ก็ได้ (เพราะมันเป็นบรรทัดสุดท้าย ขึ้นบรรทัดใหม่ไปก็ไม่ได้อะไรขึ้นมา)

```
for(int col = 1; col <= N; ++col) {  
    printf("* ");  
}
```




รวมร่างเป็นโปรแกรมพิมพ์กรอบ

```
int N;
scanf("%d", &N);
for(int col = 1; col <= N; ++col) {
    printf("* ");
}
printf("\n");

for(int row = 2; row <= N - 1; ++row) {
    printf("* ");
    for(int col = 2; col < N; ++col) {
        printf(" ");
    }
    printf("*\n");
}

for(int col = 1; col <= N; ++col) {
    printf("* ");
}
```



สรุปใจความ

- งานที่ใช้รูปสองชั้นมีลำดับการคิดคล้ายกับรูปชั้นเดียว
 - พอมันวิ่งเข้าไปที่รูปด้านใน มันก็ต้องทำของข้างในให้เรียบร้อยหมดก่อน จึงค่อยหลุดลงมาต่อด้านท้าย และวนขึ้นไปต่อที่รูปด้านนอกได้
 - เวลาที่จินตนาการไม่ออกให้มองว่ารูปด้านในเป็นเหมือนงานบรรทัดหนึ่งที่ทำเสร็จแล้วก็ข้ามไปทำบรรทัดถัดไป
 - ด้วยเหตุนี้เราจึงอาจจะลองเริ่มคิดจากเนื้อหาของรูปด้านในให้เสร็จก่อน แล้วค่อยเอารูปด้านนอกมาครอบอีกชั้น
- ถ้างานมันมีลักษณะแบ่งเป็นหลาย ๆ แบบ รูปของเราจะมีหลาย ๆ ชุดก็สมเหตุผลดี หรือถ้าจะมีชุดเดียวใหญ่ ๆ ก็อาจจะต้องพึ่งพาการใช้ if จำนวนมากเพื่อแยกประเภทงาน (วิธีหลังนี้อาจจะทำให้ยุ่งกว่าเดิม)