การเขียนโปรแกรมคอมพิวเตอร์ 1

Computer Programming I

คำสั่งควบคุม: คำสั่งทำซ้ำด้วย For Loop และ Do .. While 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

สัปดาห์ที่ 7

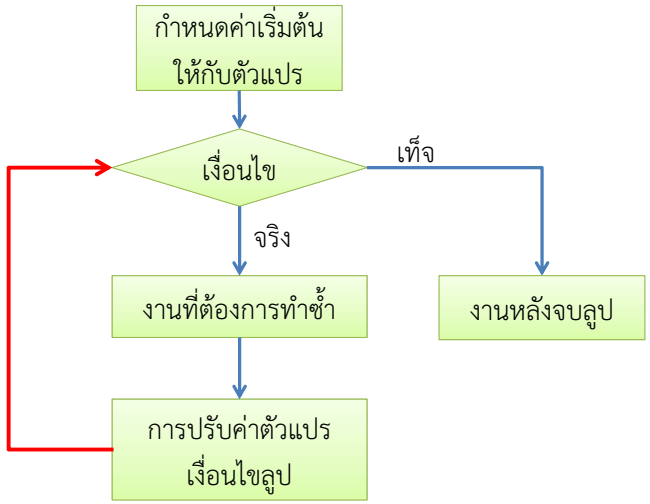
การวนลูปในภาษาซี

- มีอยู่สามแบบ
  - while ( ) { ... }
  - for ( ) { ... }
  - do { ... } while ( );
- แบบ while ( ) { ... } กับ for ( ) { ... } ใช้ทดแทนกันได้เสมอ เพราะหลักการคิดและลำดับการทำงานเหมือนกันทุกอย่าง
- ส่วนแบบ do { ... } while ( ); จะมีเอกลักษณ์เป็นของตัวเอง เพราะลำดับการคิดแตกต่างจากคนอื่น

การทำซ้ำด้วย For Loop

- มีลักษณะเทียบเท่ากับ While Loop ทุกประการ
- แต่งานที่มันถนัดก็คือการวนทำซ้ำจากค่า i = 0 ถึง n
  → แบบนี้ for loop จะได้โค้ดที่กะทัดรัดดูดีกว่า
- ถ้าเป็นงานแบบอื่นคิดด้วย for loop แล้วอาจจะชวนงงกว่า while loop เช่น ถ้าเงื่อนไขการจบลูปคือ x < 0 เป็นต้น การใช้ for loop อาจจะไม่ช่วยอะไรให้ดีขึ้น แถมชวนงงน่าสงสัยด้วย
  → ของทั้งสองทดแทนกันได้เสมอ แต่การเลือกใช้ให้ถูกสุขลักษณะจะนำไปสู่โค้ดที่ดูเป็นธรรมชาติและเข้าใจง่ายกว่า
- for loop มีลูกเล่นชวนงงมากกว่า แต่มักให้โค้ดที่สั้นกว่า (โค้ดที่สั้นกว่า ไม่ได้หมายความว่าดีกว่า)

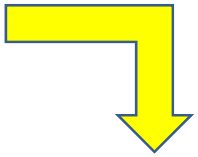
แนวคิดลูปแบบ while ( ) { ... } และ for ( ) { ... }



องค์ประกอบของ for loop ในภาษาซี

มีอยู่สี่ส่วนเหมือน while loop เพียงแต่มีการจัดเรียงตำแหน่งที่ต่างกัน

```
กำหนดค่าเริ่มต้นตัวแปรก่อนเข้าลูป
while ( เงื่อนไข ) {
    งานที่ต้องการทำซ้ำ
    ปรับค่าตัวแปรเงื่อนไขลูป
}
... งานหลังจบลูป ...
```



```
for (กำหนดค่าเริ่มต้นตัวแปรก่อนเข้าลูป; เงื่อนไข; ปรับค่าตัวแปรเงื่อนไขลูป) {
    งานที่ต้องการทำซ้ำ
}
... งานหลังจบลูป ...
```


ตัวอย่างการทำงานของลูป

โจทย์ จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด 1 ถึง 5 (ช่วงปิดจะรวมเลข 1 และ 5 ด้วย) จากนั้นพิมพ์ผลลัพธ์ออกมาทางจอภาพ ให้เขียนด้วยการใช้ while loop และ for loop

วิเคราะห์

1. ไม่มีการรับข้อมูลเข้าจากผู้ใช้ แต่จะต้องสร้างตัวเลขขึ้นมาเอง
2. งานที่ต้อง**ทำซ้ำ**แน่ ๆ คือการบวกเลข
3. ต้องมีการนับเลขที่จะบวกเพิ่มขึ้นเรื่อย ๆ เพื่อให้เปลี่ยนตัวบวกจาก 1 ไปเป็น 2, 3, 4 และ 5 ได้
4. เงื่อนไขที่ควรใช้ในการทำงานคือ ‘ตัวบวกต้องอยู่ในช่วง 1 ถึง 5’

โค้ดที่ใช้ While Loop

```
void main() {
    int sum = 0;
    int adder = 1;

    while (adder <= 5) {
        sum = sum + adder;
        adder = adder + 1;
    }

    printf("%d", sum);
}
```

การกำหนดค่าตัวแปรเริ่มต้นก่อนเข้าลูป

เงื่อนไขที่จะให้ทำลูป

งานที่ต้องการทำซ้ำ

การปรับค่าตัวแปรเงื่อนไขลูป

งานหลังจบลูป

โค้ดที่ใช้ For Loop แบบที่ 1

```
void main() {
    int sum = 0;
    int adder;
    for(adder = 1; adder <= 5; ++adder) {
        sum = sum + adder;
    }

    printf("%d", sum);
}
```

การกำหนดค่าตัวแปรเริ่มต้นก่อนเข้าลูป

เงื่อนไขที่จะให้ทำลูป

การปรับค่าตัวแปรเงื่อนไขลูป

งานที่ต้องการทำซ้ำ

งานหลังจบลูป

การใช้ for loop จะทำให้โค้ดดูสั้นลง เพราะงานสามอย่างจะกระจุกอยู่ที่ตอนต้นของลูป คนที่เริ่มเรียนอาจจะงงได้ว่าคำสั่งแต่ละอันถูกทำตอนไหน

ลำดับการทำงานของคำสั่งใน for loop (1)

- คนจำนวนมากมีความเข้าใจที่ผิดพลาดหรือตื้นเขินเกี่ยวกับ for loop
  - ไม่รู้ว่าคำสั่งที่อยู่ในวงเล็บกลมทุกอันจะมีการถูกเรียกใช้ตามลำดับใด
  - มักจะคิดอยู่แค่ว่ามันวนไปกี่รอบแค่นั้น ไม่ได้เข้าใจว่ามันทำงานอย่างไร
  - ทำให้งงเมื่อต้องใช้ลูปกับงานที่มีการวนรอบแบบไม่จำกัดจำนวน

- ที่จริงแล้วลำดับการทำงานของคำสั่งต่าง ๆ ในลูปมีกฎเกณฑ์ที่แน่นอน
  - มีลำดับเช่นเดียวกันกับ while loop แต่ของ for loop จะเข้าใจยากกว่า (แม้ว่าโค้ดจะดูสั้นกว่าก็ตาม)
  - ต้องวกกลับไปที่พื้นฐานว่าลูปมีสี่องค์ประกอบ คือ 1. เตรียมค่าตัวแปรลูป 2. เงื่อนไขลูป 3. งานที่ให้ทำซ้ำ และ 4. การปรับตัวแปรลูป

ทบทวนลำดับการทำงานในลูป while

ลูปมีสี่องค์ประกอบ คือ 1. เตรียมค่าตัวแปรลูป 2. เงื่อนไขลูป 3. งานที่ให้ทำซ้ำ และ 4. การปรับตัวแปรลูป

```
int sum = 0;
int adder = 1;

while (adder <= 5) {
    sum = sum + adder;
    adder = adder + 1;
}
...
```

(1) กำหนดค่าตัวแปรเริ่มต้นก่อนเข้าลูป (ทำครั้งเดียว)

(2) เงื่อนไขที่จะให้ทำลูป

(3) งานที่ต้องการทำซ้ำ

(4) ปรับค่าตัวแปรเงื่อนไขลูป

ลำดับการทำงานคือ (1) → (2) → (3) → (4) → (2) → (3) → (4) → ..... (2) → (3) → (4) → (2) → จบลูป

สังเกตให้ดีว่าจุดสุดท้ายที่ได้ทำในลูปคือการตรวจเงื่อนไขลูป

ลำดับการทำงานของคำสั่งใน for loop (2)

```
int sum = 0;
int adder;
for(adder = 1; adder <= 5; ++adder) {
    sum = sum + adder;
}
...
```

(1) กำหนดค่าตัวแปรเริ่มต้นก่อนเข้าลูป

(2) เงื่อนไขที่จะให้ทำลูป

(4) การปรับค่าตัวแปรเงื่อนไขลูป

(3) งานที่ต้องการทำซ้ำ

ลำดับการเขียนส่วนต่าง ๆ จะเปลี่ยนไป แต่ลำดับการทำงานจะคงเดิม นั่นคือ

(1) → (2) → (3) → (4) → (2) → (3) → (4) → ..... (2) → (3) → (4) → (2) → จบลูป

สังเกตให้ดีว่าส่วนที่ (1) จะถูกทำแค่ครั้งเดียว และเป็นส่วนที่ (2) ที่เป็นส่วนสุดท้าย แต่ส่วนที่ (4) ถูกยกไปอยู่ตรงต้นขั้วลูป

โค้ดที่ใช้ For Loop แบบที่ 2

- หลายคนอาจเห็นว่า ในตัวอย่างที่ยกมา มีตัวแปรที่เกี่ยวกับลูปมีมากกว่าหนึ่งตัว เราจะทำการกำหนดค่าตัวแปรเริ่มต้นหลาย ๆ ตัวใน for loop ได้อย่างไร
- เรื่องนี้สบายมาก ให้เราคั่นคำสั่งกำหนดค่าเริ่มต้นในกลุ่ม (1) ด้วยคอมมา แทนที่จะเป็นเซมิโคลอนก็เป็นอันเสร็จพิธี

```
void main() {
    int sum, adder;
    for(sum = 0, adder = 1; adder <= 5; ++adder) {
        sum = sum + adder;
    }
    printf("%d", sum);
}
```

(1) กำหนดค่าตัวแปรเริ่มต้นก่อนเข้าลูป

(2) เงื่อนไขที่จะให้ทำลูป

(4) การปรับค่าตัวแปรเงื่อนไขลูป

(3) งานที่ต้องการทำซ้ำ

ลึก ๆ กับความเป็นไปใน for loop

- จากสไลด์ที่ผ่านมา เห็นได้ว่าเราสามารถเขียนคำสั่งกำหนดค่าเริ่มต้นให้มีมากกว่าหนึ่งได้ด้วยการใช้เครื่องหมายคอมมาคั่นคำสั่ง
- ส่วนเครื่องหมายเซมิโคลอนยังใช้คั่นกลุ่ม (1) กับ (2) และ (2) กับ (4) เหมือนเดิม (กฎเหล็ก ห้ามเปลี่ยนแปลงเป็นอันขาด)
- แล้วถ้าเราต้องการจะให้มีเงื่อนไขมากกว่าหนึ่งอย่าง หรือว่าให้มีการปรับตัวแปรลูปมากกว่าหนึ่งตัวแปรล่ะ จะทำอย่างไร ?
- คำตอบก็คือ เราควรใช้ตัวดำเนินการทางตรรกะเข้ามาช่วย ไม่ควรใช้เครื่องหมาคอมมา เพราะผลที่ได้มักจะผิดไปจากที่เราคิด
  - เช่น ถ้าเงื่อนไขคือ (a < 7) || (b < 5), c < 10 แบบนี้โปรแกรมจะตรวจเพียงว่า c < 10 เป็นจริงหรือไม่ ถ้าเราอยากให้ตรวจค่าความจริงทั้งสองชุดว่าเป็นจริงทั้งคู่หรือไม่ ให้เราใช้ ((a < 7) || (b < 5)) && (c < 10)

ตัวอย่างเรื่องลึก ๆ ใน for loop (ปัญหาอนุกรมสองส่วน)

โจทย์ จงเขียนโปรแกรมที่หาผลรวมเลขอนุกรม 1*1 + 2*2 + 3*4 + 4*8 + 5*16 + ... + 10*512 ด้วยการใช้ลูป for และพิมพ์ผลรวมนั้นออกมาทางจอภาพ (โจทย์สั้น ๆ แต่อาจจะน่ากลัวสำหรับมือใหม่ อย่าคิดว่าโจทย์ยาวแล้วจะยาก โจทย์สั้นแล้วจะง่าย)

วิเคราะห์

1. ไม่มีอินพุตจากผู้ใช้ เพราะว่าตัวเลขกำหนดไว้ตายตัวในโจทย์
2. ต้องพิมพ์ผลลัพธ์จากการคำนวณออกมาทางจอภาพ
3. การเปลี่ยนแปลงของตัวเลขมีสองแบบ แบบแรกจะขึ้นทีละหนึ่ง คือ 1, 2, 3, 4, ..., 10 และแบบที่สองขึ้นทีละเท่าตัวคือ 1, 2, 4, 8, ..., 512 ดังนั้นเราต้องแยกการปรับค่าออกเป็นสองชุด (เงื่อนไขมีชุดเดียวก็ได้)

โค้ดสำหรับปัญหา อนุกรมสองส่วน

```
void main() {
    int sum, add, multiply;
    for(sum = 0, add = 1, multiply = 1;
        add <= 10;
        add++, multiply *= 2)
    {
        sum += add * multiply;
    }
    printf("%d", sum);
}
```

ส่วนประกอบลูปที่มีหลายคำสั่งถูกคั่นด้วยคอมมาในลักษณะเดียวกัน แต่การคั่นส่วนประกอบด้วยเซมิโคลอนยังเป็นเช่นเดิม

นี่เป็นโจทย์ลักษณะเดียวกับมิดเทอมครั้งที่แล้ว ถ้าใครไม่คุ้นว่าตัวเองทำอะไรคล้าย ๆ แบบนี้ได้ ก็ไม่ต้องสงสัยเลยว่าทำไมสอบไม่ผ่าน

ปัญหาอนุกรมสามส่วน

โจทย์ จงเขียนโปรแกรมที่หาผลรวมเลขอนุกรม

2*3*5 + 4*9*13 + 8*27*35 + 16*81*97 + 32*234*250 + ... + 2048*177147*179195 ด้วยการใช้ลูป for และพิมพ์ผลรวมนั้นออกมาทางจอภาพ (เนื่องจากตัวเลขผลรวมมีค่ามากกว่าที่ตัวแปรแบบ int จะเก็บค่าได้ แนะนำว่าให้ใช้ตัวแปรแบบ double ในการเก็บค่าตัวเลขทุกตัวแทน)

วิเคราะห์

1. ตัวเลขตัวที่สามจาก 2*3*5 มาจากผลบวกของเลขสองตัวหน้า
2. บางทีการจะรู้ว่าต้องวนกี่รอบกันแน่ก็ไม่ใช่เรื่องที่ตรงไปตรงมาเสียทีเดียว แบบนี้เราต้องเลี่ยงด้วยการใช้การเปรียบเทียบความมากน้อยแทน
3. อย่าคิดแต่จะกำหนดให้ sum เป็น double ไม่อย่างนั้นการเปลี่ยนชนิดข้อมูลจะเกิดขึ้นหลังคูณเสร็จ ซึ่งได้ค่าที่ผิดไปเรียบร้อยแล้ว

โค้ดสำหรับปัญหาอนุกรมสามส่วน

```
void main() {
    double sum, term1, term2, term3;
    for(sum = 0, term1 = 2, term2 = 3, term3 = 5;
        term1 <= 2048;
        term1 *= 2, term2 *= 3, term3 = term1 + term2)
    {
        sum += term1 * term2 * term3;
    }
    printf("%lf", sum);
}
```

ใช้การเปรียบเทียบความมากน้อย แบบนี้จะง่ายกว่านับจำนวนรอบ

ว่าแต่เราแยกการทำงานยาก ๆ อย่าง term3 = term1 + term2 ไปไว้ในลูปได้หรือเปล่า ?

โค้ดสำหรับอนุกรมสามส่วนแบบกระจายงาน (ที่ผิด)

เราสามารถปรับกระจายงานไปไว้ภายในลูปได้ตามใจชอบ แต่ต้องกระทำด้วยความระมัดระวัง (ตัวอย่างข้างล่างเป็นแบบที่ผิด)

```
double sum, term1, term2, term3;
for(sum = 0, term1 = 2, term2 = 3, term3 = 5;
        term1 <= 2048;
        term1 *= 2, term2 *= 3)
    {
        sum += term1 * term2 * term3;
        term3 = term1 + term2;
    }
```

เรายังงานมาไว้ด้านใต้แบบนี้ได้ ไม่ผิดไวยากรณ์ภาษา แต่ผลลัพธ์ตรงนี้ผิด !!! ทราบหรือไม่ว่าเพราะอะไร

วิเคราะห์ปัญหาเรื่องลำดับการทำงาน

```
for(sum = 0, term1 = 2, term2 = 3, term3 = 5;
        term1 <= 2048;
        term1 *= 2, term2 *= 3, term3 = term1 + term2) ...
```

แบบดั้งเดิม

ลำดับการคิดของแบบดั้งเดิมนั้นจะเปลี่ยน term1 และ term2 ก่อน term3

```
for(sum = 0, term1 = 2, term2 = 3, term3 = 5;
        term1 <= 2048;
        term1 *= 2, term2 *= 3)
    {
        sum += term1 * term2 * term3;
        term3 = term1 + term2;
    }
```

แบบใหม่

แต่แบบนี้ term3 จะเปลี่ยนก่อน term1 และ term2 ทำให้ผลลัพธ์ผิด

โค้ดสำหรับอนุกรมสามส่วน แบบกระจายงาน (ที่ถูกต้อง)

ในกรณีที่เราต้องย้ายสิ่งต่าง ๆ ลงมาจนหมด เราสามารถปล่อยให้เป็นที่ว่างตรงบริเวณต้นขั้วลูปได้เลย

```
for(sum = 0, term1 = 2, term2 = 3, term3 = 5;
        term1 <= 2048; )
{
    sum += term1 * term2 * term3;
    term1 *= 2;
    term2 *= 3;
    term3 = term1 + term2;
}
```

ย้ายงานมาไว้ด้านใต้หมด เพื่อให้ลำดับการทำงานถูกต้อง สังเกตว่าข้างบนตรงส่วนปรับค่าตัวแปรกลายเป็นที่ว่างไปแล้ว

ที่จริงส่วนอื่น ๆ ของ for loop จะว่างด้วยก็ได้ (1)

ย้ายส่วนกำหนดค่าตัวแปรเริ่มต้นออกไปข้างนอกเหมือน while loop ได้

```
sum = 0; term1 = 2; term2 = 3; term3 = 5;
for(; term1 <= 2048; )
{
    sum += term1 * term2 * term3;
    term1 *= 2;
    term2 *= 3;
    term3 = term1 + term2;
}
```

เพราะส่วนแรกทำครั้งเดียวก่อนเริ่มตรวจเงื่อนไข การย้ายไปด้านบนจึงเป็นทางเลือกที่สมเหตุผลที่สุด

ที่จริงส่วนอื่น ๆ ของ for loop จะว่างด้วยก็ได้ (2)

ถ้าย้ายเงื่อนไขออกไปก็จะทำให้ลูปวนไม่รู้จบ นอกจากจะเจอคำสั่ง break

```
sum = 0; term1 = 2; term2 = 3; term3 = 5;
for(; ;)
{
    sum += term1 * term2 * term3;
    term1 *= 2;
    term2 *= 3;
    term3 = term1 + term2;
    if (term1 > 2048)
        break;
}
```

ไม่เหลืออะไรนอกจากเซมิโคลอน แบบนี้ ลูปจะทำงานอย่างไม่มีเงื่อนไข

อย่าลืมว่าเงื่อนไขสำหรับ break ถ้าเป็นจริงจะทำให้ลูปหยุดทำงาน แต่เงื่อนไขของลูปด้านบน ถ้าเป็นจริงลูปจะทำงาน

จะเอางานอย่างอื่นไปใส่ไว้ตรงต้น for ลูปได้หรือไม่

ภาษาซีแท้จริงไม่ได้มามัวคิดว่าโค้ดแต่ละส่วนทำงานตามวัตถุประสงค์ที่ควรจะเป็นหรือไม่ (อย่าลืมว่าเครื่องมันไม่ได้คิดวิธีแทนเรา มันแค่ทำตามที่เราสั่ง)

- สิ่งที่มันสนใจทำจริง ๆ ก็คือการทำคำสั่งแต่ละกลุ่มตามลำดับ
- ดังนั้นที่จริงเราจะเอาอะไรไปใส่ก็ได้ มันรับหมด เอา printf ไปใส่ก็ยังได้

```
void main() {
    int i;
    for(i = 0; printf("%d ", i), i < 5; ++i) {  }
}
```

ปล่อยให้ด้านในลูปว่าง ๆ ก็ได้ ในกรณีปล่อยว่างจะต้องใส่วงเล็บปีกกาให้

จากตัวอย่างข้างบน โปรแกรมจะพิมพ์ว่า 0 1 2 3 4 5 เพราะค่าความจริงจะตีความตามคำสั่งหลังสุดของเงื่อนไขเท่านั้น

แล้วถ้าเป็นแบบนี้โปรแกรมจะพิมพ์ว่าอะไร

จากโค้ดเดิม เราสลับลำดับเล็กน้อย เอา printf ไปไว้ด้านหลัง i < 5

```
void main() {
    int i;
    for(i = 0; i < 5, printf("%d ", i); ++i) { }
}
```

แบบนี้โปรแกรมจะพิมพ์ค่าไม่หยุด เพราะ printf จะมีค่าเท่ากับจำนวนตัวอักษรที่พิมพ์ออกมาทางหน้าจอ เมื่อจำนวนตัวอักษรที่พิมพ์ออกมาไม่เป็นศูนย์ ภาษาซีก็จะบอกว่าเงื่อนไขเป็นจริง (เพราะตัวเลขที่ไม่เป็นศูนย์แปลว่า ‘จริง’ ในภาษาซี)

ตรงนี้ซับซ้อนมาก ดังนั้นต้องระวังว่าเงื่อนไขตัวจริงคือตัวสุดท้ายเท่านั้น

แล้วแบบนี้ล่ะ

- บอกได้หรือไม่ว่าโปรแกรมนี้จะพิมพ์อะไรออกมา

```
void main() {
    int i;
    for(printf("1 "), i = 0; printf("2 "), i < 3; printf("4 "), ++i) {
        printf("3 ");
    }
}
```

- คำตอบก็คือ 1 2 3 4 2 3 4 2 3 4 2
- หลักการเดิมก็คือ โปรแกรมทำคำสั่งด้านซ้ายของแต่ละส่วนก่อน (ถ้ามีการใช้คอมมาคั่น และเงื่อนไขตัวจริงคือ i < 3 ทำให้เกิดการวนสามครั้ง

บทเรียนจาก for loop

- ถ้าจะใช้มันจัดการกับค่าจาก i = 0 ถึง n-1 การใช้ for loop จะดีมาก
- แต่พอจะเอาไปใช้อย่างอื่น for loop อาจจะทำอะไรแปลก ๆ มาให้เราได้
- ควรใช้อย่างระมัดระวัง เพราะกลไกในการคิดของ for loop ไม่ใช่ง่าย ๆ (เริ่มแรกอาจดูเหมือนง่าย แต่พอจะให้ทำอะไรที่ซับซ้อนจะยากกว่าเดิมมาก)
- ถ้าคุณเข้าใจ while loop เป็นอย่างดี และกำลังงงกับ for loop จะใช้ while loop แทนก็ได้ สิ่งที่สำคัญที่สุดของโปรแกรมคือความถูกต้อง ไม่ใช่ความสั้น
- ให้เลือกใช้วิธีที่รู้สึกมั่นใจ และค่อย ๆ พัฒนาความชำนาญจนทำวิธีที่มันสวยงามที่สุดได้

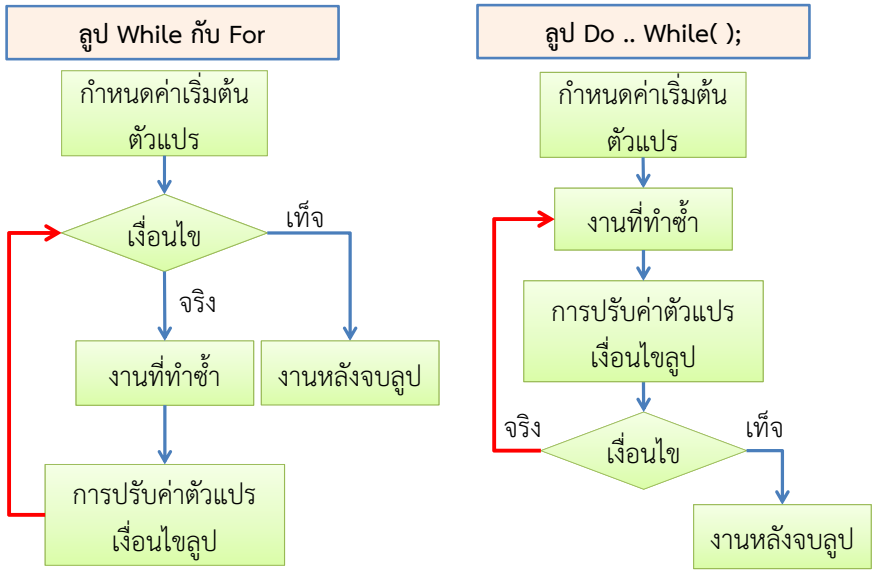
ลูปแบบสุดท้าย do .. while();

เป็นลูปที่มีเอกลักษณ์เฉพาะตัว คือมีลำดับการทำงานที่แตกต่างจากคนอื่น เพราะการตรวจเงื่อนไขถูกกระทำที่ด้านท้ายของลูป แทนที่จะเป็นด้านบน

→ ส่วนตรงกลางของลูปยังไงก็ต้องถูกทำอย่างน้อยหนึ่งครั้ง เพราะไม่มีเงื่อนไขใดจะไปขัดขวางมันได้ (เว้นแต่จะโดนคำสั่ง break;)

(ดูการเปรียบเทียบโครงสร้างลูปในหน้าถัดไป)

เปรียบเทียบโครงสร้างลูป



ตัวอย่างการใช้ลูป do .. while();

โจทย์ จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด 1 ถึง 5 (ช่วงปิดจะรวมเลข 1 และ 5 ด้วย) จากนั้นพิมพ์ผลลัพธ์ออกมาทางจอภาพ ให้เขียนด้วยการใช้ while loop และ for loop

วิเคราะห์

1. งานที่ต้อง**ทำซ้ำ**แน่ ๆ คือการบวกเลข
2. งานทำซ้ำนี้ต้องทำอย่างน้อยหนึ่งครั้ง ดังนั้นเราสามารถใช้ลูป do while ได้

ตัวอย่างโค้ดลูป do .. while();

```
void main() {
    int sum = 0;
    int i = 1;
    do {
        sum = sum + i;
        ++i;
    } while(i <= 5);

    printf("%d", sum);
}
```

ใส่วงเล็บ { } ไว้หลังคำว่า do

ระวังลืมใส่เครื่องหมาย ;

เกร็ดเรื่องลูป do .. while();

- เงื่อนไขลูปอยู่กับคำว่า while เช่นเดิมและอยู่ในวงเล็บด้วย
- ต้องมีเครื่องหมายเซมิโคลอนตามหลังวงเล็บเงื่อนไข (ลักษณะเฉพาะ)
- ความนิยมของลูป do .. while( ); จะมีน้อยกว่าลูป while และ for เพราะบังคับให้ต้องทำงานอย่างน้อยหนึ่งครั้ง ในขณะที่ while กับ for มีอิสระมากกว่า
- ถึงความนิยมจะน้อยกว่า แต่อย่าลืมว่าของพวกนี้มันถูกคิดขึ้นมาเพื่อให้สอดคล้องกับธรรมชาติในการคิดที่หลากหลายของโปรแกรมเมอร์
  → ถ้าหากเรารู้สึกมั่นใจกับการใช้ do .. while( ); และทราบว่าผลลัพธ์จะถูกต้อง ก็อย่าได้ลังเลที่จะใช้มัน