Computer Programming I:
การเขียนโปรแกรมคอมพิวเตอร์ I

คำสั่งควบคุม FOR LOOP และ DO ... WHILE LOOP

อ.ดร.ปัญญนัท อ้นพงษ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

aonpong_p@su.ac.th

Outline

- **รูปแบบของ Loop และแนวคิด**
- คำสั่งควบคุม For
- คำสั่งควบคุม Do... while

คำสั่งควบคุม

- ที่ผ่านมาเราได้เขียนโฟลวชาร์ตในหลากหลายรูปแบบ
  - แบบตรง ๆ บนลงล่างเส้นเดียว
  - แบบมีทางเลือก ซ้ายขวา
  - แบบมีการวนซ้ำ
  - แบบมีทั้งสองอย่าง
- ตอนนี้ เราได้เรียนจนสามารถเขียนโปรแกรมเพื่อรองรับโฟลวชาร์ตได้ทุกรูปแบบแล้ว คำสั่งที่เราเขียนได้มีดังนี้
  - ส่วนที่ต้องเขียนอยู่แล้ว #include<stdio.h>, void main(){...ชุดคำสั่ง...}
  - ส่วนการรับข้อมูลเข้า-ส่งผลลัพธ์ออก printf, scanf
  - คำสั่งควบคุมแบบเงื่อนไข if, else, else if, if ใน if
  - *คำสั่งควบคุมการวนซ้ำ* while

คำสั่งควบคุม

- จริง ๆ แล้วคำสั่ง while สามารถทำงานเกี่ยวกับการวนซ้ำได้ทุกรูปแบบ แต่จะมีบางรูปแบบที่เมื่อใช้แล้วอาจดูซับซ้อนกว่าที่ควรจะเป็น (ถ้าถนัดและมั่นใจก็ใช้ได้)
- ในบทเรียนนี้จะนำเสนอเกี่ยวกับคำสั่งทางเลือกที่สามารถใช้แทนและอาจเหมาะสมกว่า while ในบางกรณี
- บางคนเรียนแล้วอาจรู้สึกว่ายากกว่า บางคนเรียนแล้วอาจรู้สึกว่าน่าใช้กว่า
- แต่ยังไงก็ควรรู้เป็นพื้นฐานเอาไว้ทุกวิธี และการเลือกใช้ให้เหมาะสมจะทำให้เขียนโปรแกรมได้ง่ายที่สุด

คำสั่งควบคุม

- คำสั่งควบคุม คือคำสั่งที่ทำให้โปรแกรมมีทิศทางไปในทิศทางที่กำหนด
- ด้วยคำสั่งควบคุม ทำให้เราสามารถเขียนโปรแกรมตามโฟลวชาร์ตในลักษณะต่าง ๆ ได้
- คำสั่งควบคุม มี 2 ประเภท
  - คำสั่งเงื่อนไข (Condition Statement)
    - if-else
    - switch-case

    ใช้แทนกันได้ (แต่ต้องมีการปรับแต่งโค้ด) ถ้าถนัดอันไหนอาจเลือกใช้อันนั้นตลอดก็ได้
  - คำสั่งทำซ้ำ (Iteration Statement)
    - while
    - **do-while**
    - **for**

    ใช้แทนกันได้ (แต่ต้องมีการปรับแต่งโค้ด) ถ้าถนัดอันไหนอาจเลือกใช้อันนั้นตลอดก็ได้

รูปแบบของ Loop และแนวคิด

รูปแบบของ Loop ในภาษาซี มี 3 รูปแบบ

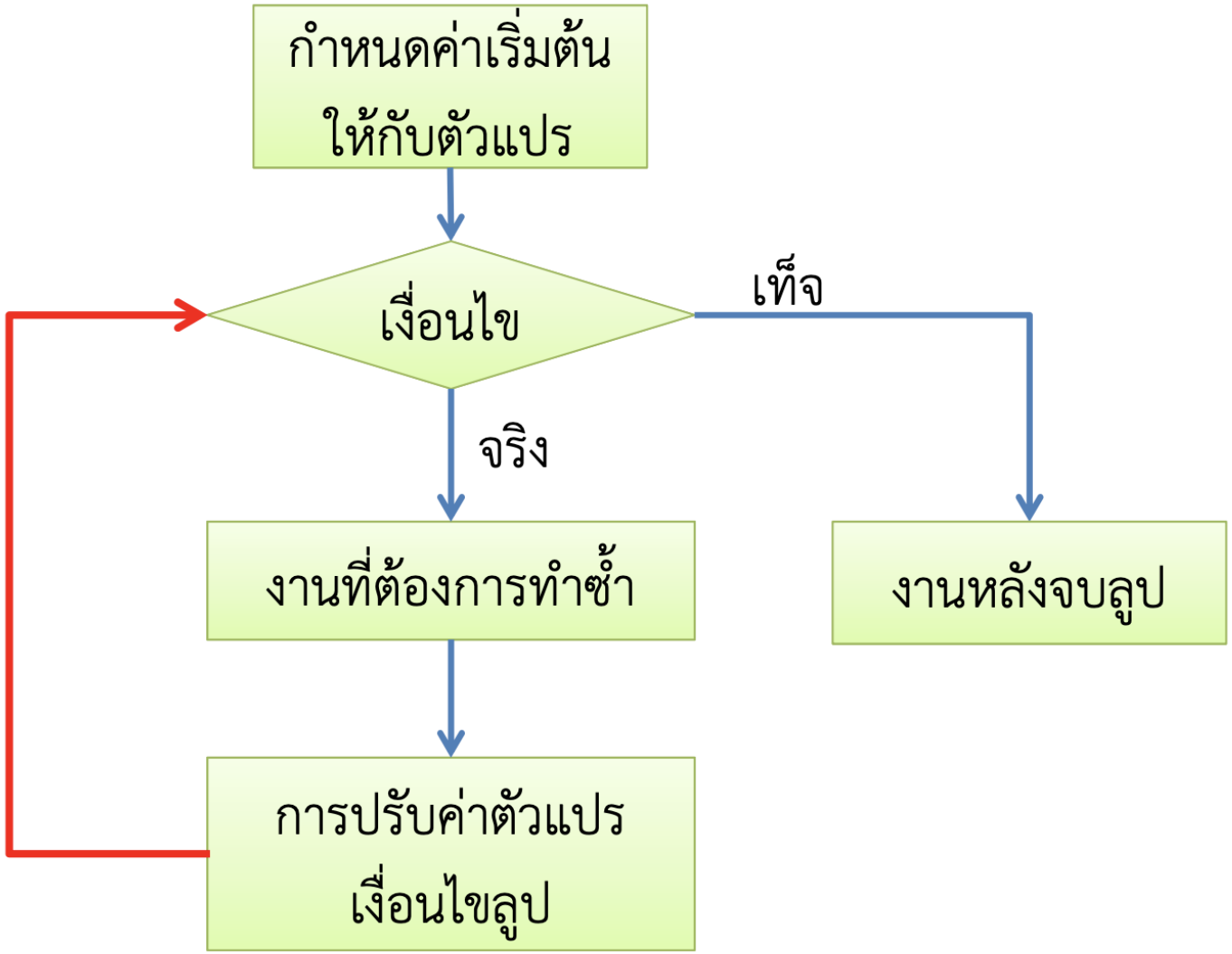
- while
- **for**
- **do ... while**

Note: แต่ละรูปแบบก็มีแนวคิดและวิถีทางเป็นของตนเอง อย่างไรก็ตาม Loop ทุกๆ รูปแบบสามารถทำงานแทนกันได้ แม้อาจต้องมีการปรับแก้บางจุด ดังนั้นในทางปฏิบัติ เราอาจเลือกใช้เฉพาะคำสั่งที่เราถนัดก็ได้ (พอรู้พื้นฐานทุกๆ รูปแบบไว้สอบ Lecture พอ)

Outline

- รูปแบบของ Loop และแนวคิด
- **คำสั่งควบคุม For**
- คำสั่งควบคุม Do... while

คำสั่งควบคุม for



ภาพนี้เคยให้ดูไปแล้วในบทเรียนที่ 6 คำสั่งควบคุม while

ในบทนี้ คำสั่งควบคุม for ก็จะทำงานในลักษณะเดียวกัน

คำสั่งควบคุม for

- มีลักษณะเทียบเท่ากับ While Loop ทุกประการ
- งานที่เหมาะสมกับ for คือการวนซ้ำจากค่า i = 0 ถึง n แบบนี้ for loop จะได้โค้ดที่กะทัดรัดดูดีกว่า
- ถ้าเป็นงานที่ไม่ได้กำหนดจำนวนครั้งที่แน่นอนมา ถ้าคิดด้วย for loop แล้วจะวุ่นวายกว่าการใช้ while loop เช่น ถ้าเงื่อนไขการจบ loop คือการกรอกค่าเป็นเลข 0 เป็นต้น
- for loop มีลูกเล่นมากกว่า มักให้โค้ดที่สั้นกว่า (โค้ดที่สั้นกว่า ไม่ได้หมายความว่าดีกว่า)

- *ลูกเล่นของ for loop ไม่ต้องพยายามเล่นให้ครบก็ได้ เล่นแค่ที่เป็นประโยชน์ก็พอ*

คำสั่งควบคุม for


```
for (การกำหนดค่าตัวแปรเริ่มต้น ; เงื่อนไข ; การอัพเดทตัวแปรเงื่อนไข) {
        // คำสั่งที่ต้องการให้ทำซ้ำ
}
```

- การทำซ้ำจะทำเฉพาะคำสั่งที่อยู่ใน loop
- จากที่สังเกต ยังคงมีนักศึกษาไม่เข้าใจว่าตรงไหนคือใน loop ตรงไหนคือนอก loop
  - ใน loop คือ ในเครื่องหมายปีกกา ส่วนนอก loop คือนอกเครื่องหมายปีกกา

คำสั่งควบคุม for (เปรียบเทียบกับ while)

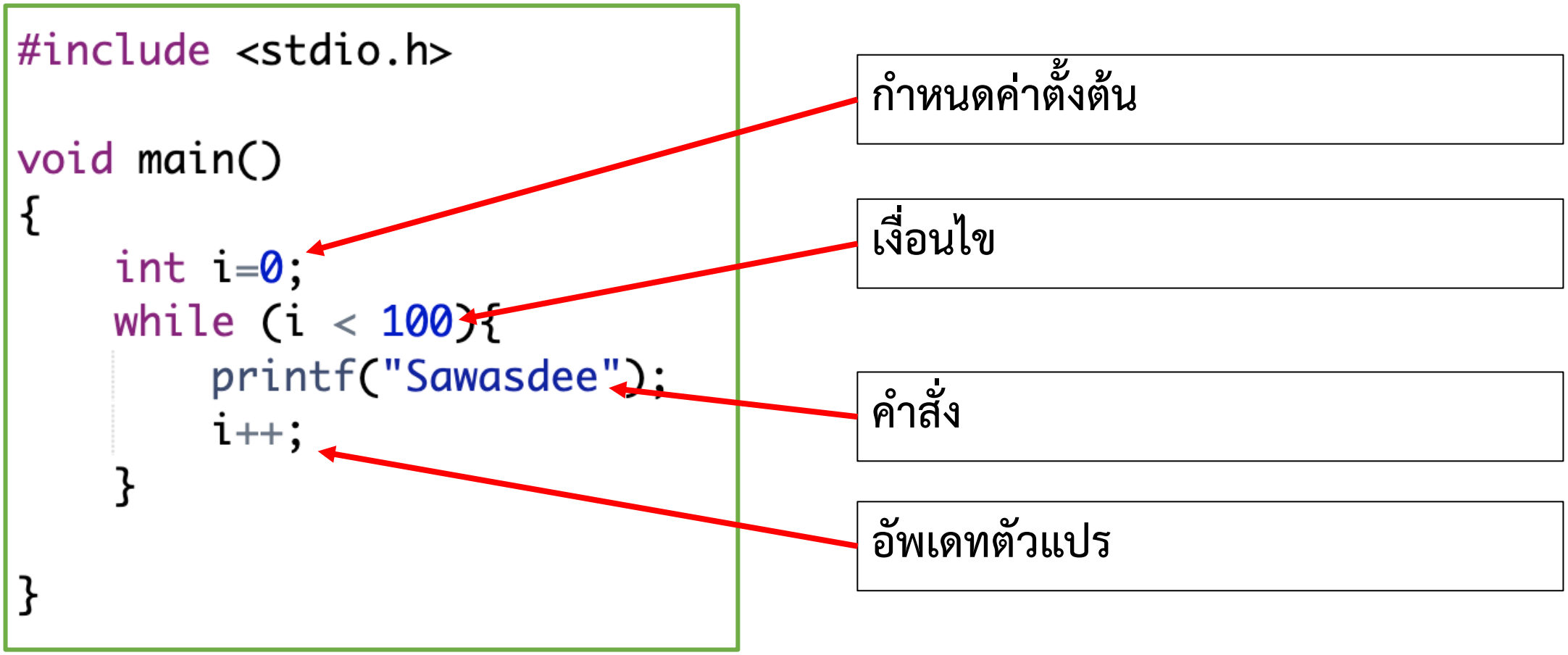

```
for (การกำหนดค่าตัวแปรเริ่มต้น ; เงื่อนไข ; การอัพเดทตัวแปรเงื่อนไข) {
        // คำสั่งที่ต้องการให้ทำซ้ำ
}
```

```
การกำหนดค่าตัวแปรเริ่มต้น
while (เงื่อนไข){
        // คำสั่งที่ต้องการให้ทำซ้ำ
        การอัพเดทตัวแปรเงื่อนไข
}
```

คำสั่งควบคุม for

จากบทที่แล้วผู้เขียนโปรแกรมต้องการพิมพ์คำว่า “Sawasdee” 100 ครั้ง (ใช้ while เหมือนเดิม)

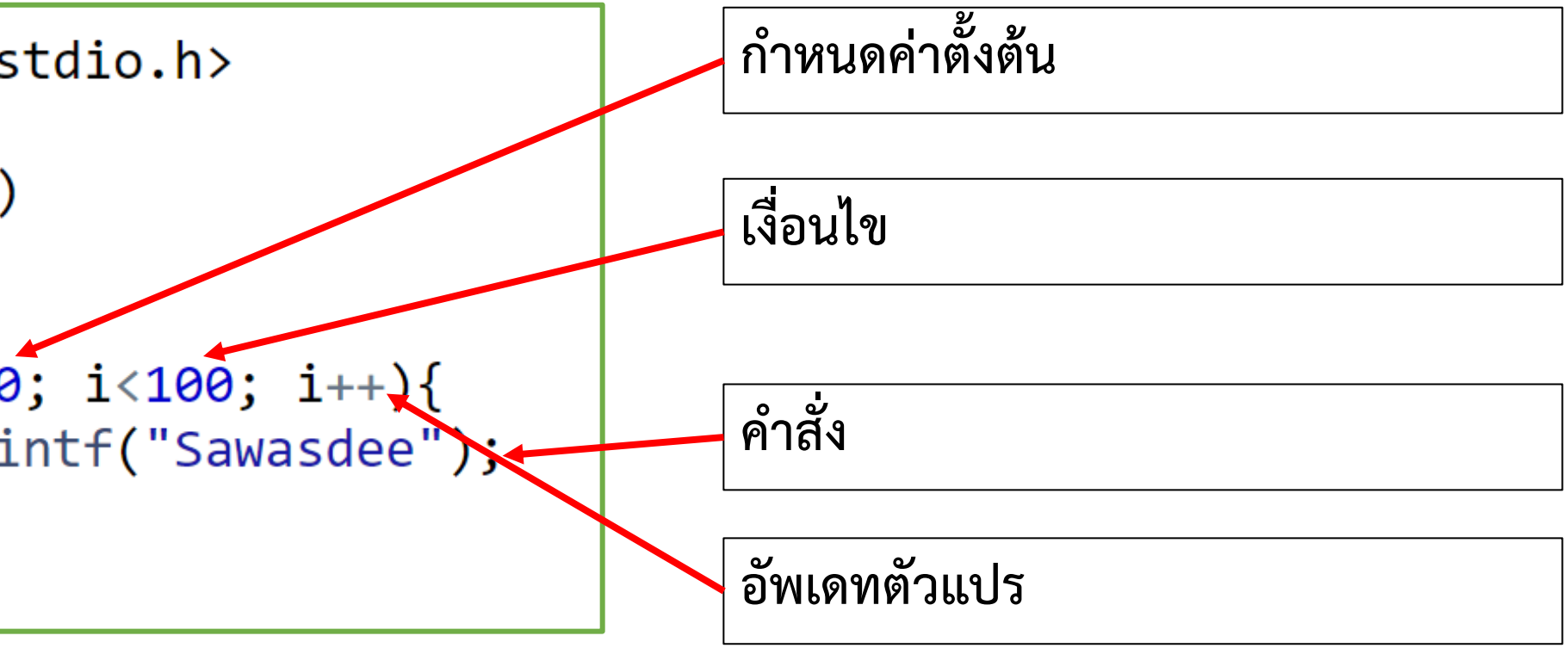
```c
#include <stdio.h>

void main()
{
    int i=0;
    while (i < 100){
        printf("Sawasdee");
        i++;
    }

}
```



คำสั่งควบคุม for

จากบทที่แล้วผู้เขียนโปรแกรมต้องการพิมพ์คำว่า “Sawasdee” 100 ครั้ง (ใช้ for)

```
#include <stdio.h>

void main()
{
    int i;
    for(i=0; i<100; i++){
        printf("Sawasdee");
    }
}
```



คำสั่งควบคุม for

จะเห็นว่า ในกรณีโจทย์นี้ ถ้าเราใช้ for โค้ดจะกระชับขึ้นมาก เพราะทุกอย่างที่สำคัญถูกรวมไว้ที่เดียว

```c
#include <stdio.h>

void main()
{
    int i=0;
    while (i < 100){
        printf("Sawasdee");
        i++;
    }

}
```

```c
#include <stdio.h>

void main()
{
    int i;
    for(i=0; i<100; i++){
        printf("Sawasdee");
    }
}
```

คำสั่งควบคุม for

ถ้าเราต้องการให้มีการวนเป็นจำนวนครั้งที่แน่นอน (รู้จำนวนรอบตั้งแต่ต้น) ใช้ for จะเหมาะสมกว่า เพราะท่องแพทเทิร์นได้เลย

```c
#include <stdio.h>

void main()
{
    int i;
    for(i=0; i<100; i++){
        printf("Sawasdee");
    }
}
```

อยากให้วนซ้ำกี่รอบก็เปลี่ยนเลขนี้

คำสั่งควบคุม for: ตัวอย่างโจทย์

ตัวอย่างโจทย์ 1 จงเขียนโฟลวชาร์ตและเขียนโปรแกรมภาษาซีสำหรับการพิมพ์คำว่า “I love you” จำนวน 3000 ครั้ง (ขึ้นบรรทัดใหม่ทุกครั้ง) ก่อนจบโปรแกรมให้พิมพ์ว่า “Good night” อีก 1 ครั้ง

คำสั่งควบคุม for: ตัวอย่างโจทย์

ตัวอย่างโจทย์ 2 จงเขียนโฟลวชาร์ตและเขียนโปรแกรมภาษาซีสำหรับการพิมพ์คำว่า “I love you ” จำนวน 3000 ครั้ง โดยใส่เลขครั้งที่พิมพ์ไว้ข้างหลังข้อความด้วย เช่น

I love you 1

I love you 2

I love you 3

...

I love you 3000

Good night

(ขึ้นบรรทัดใหม่ทุกครั้ง) ก่อนจบโปรแกรมให้พิมพ์ว่า “Good night” อีก 1 ครั้ง

คำสั่งควบคุม for: ตัวอย่างโจทย์

ตัวอย่างโจทย์ 3 จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด 1 ถึง 5 (ช่วงปิดจะรวมเลข 1 และ 5 ด้วย) จากนั้นพิมพ์ผลลัพธ์ออกมาทางจอภาพ ให้เขียนด้วยการใช้ while loop และ for loop

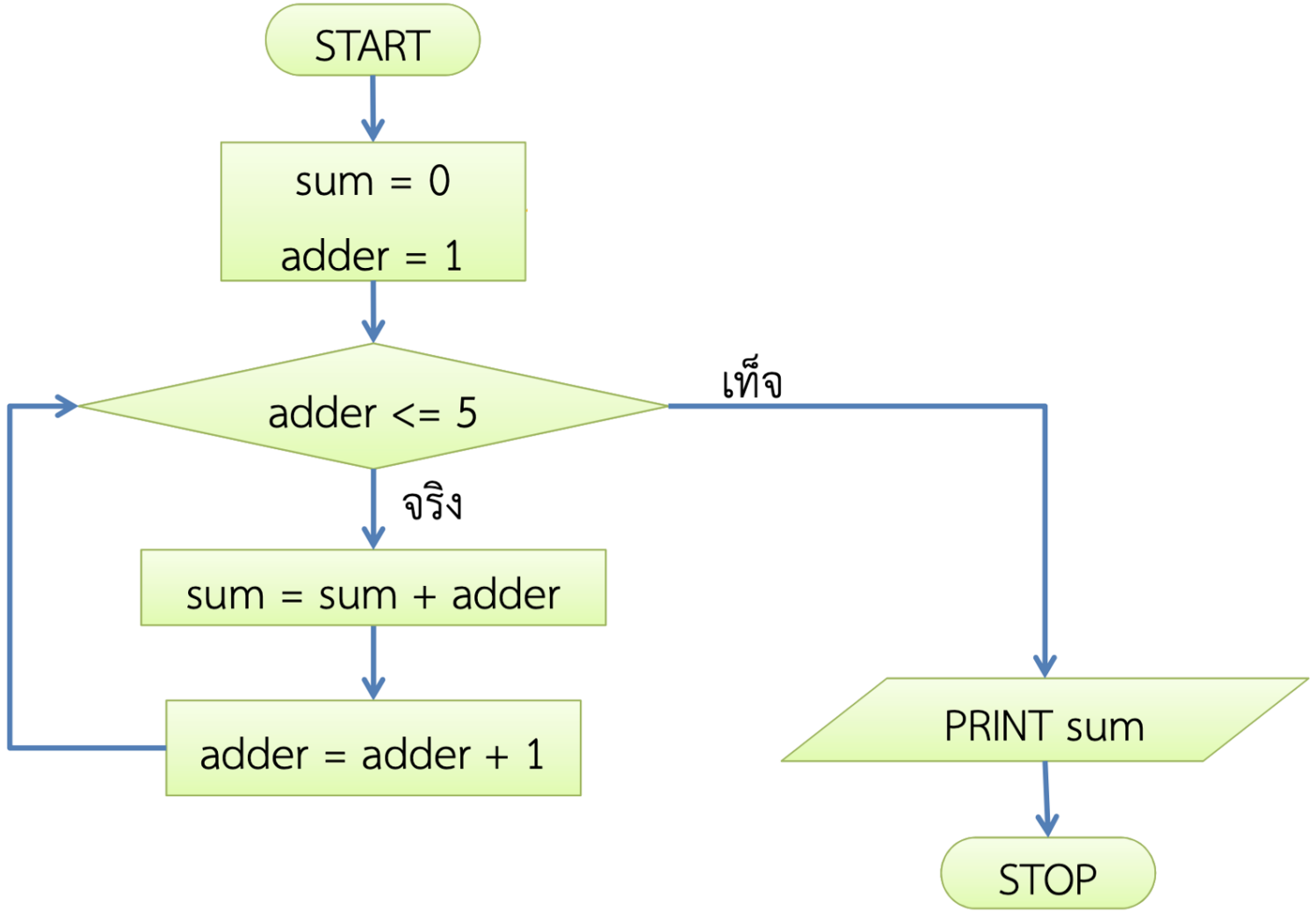
วิเคราะห์

ข้อมูลขาเข้า ไม่มี

ข้อมูลขาออก ผลบวกของเลข 1 ถึง 5 (sum)

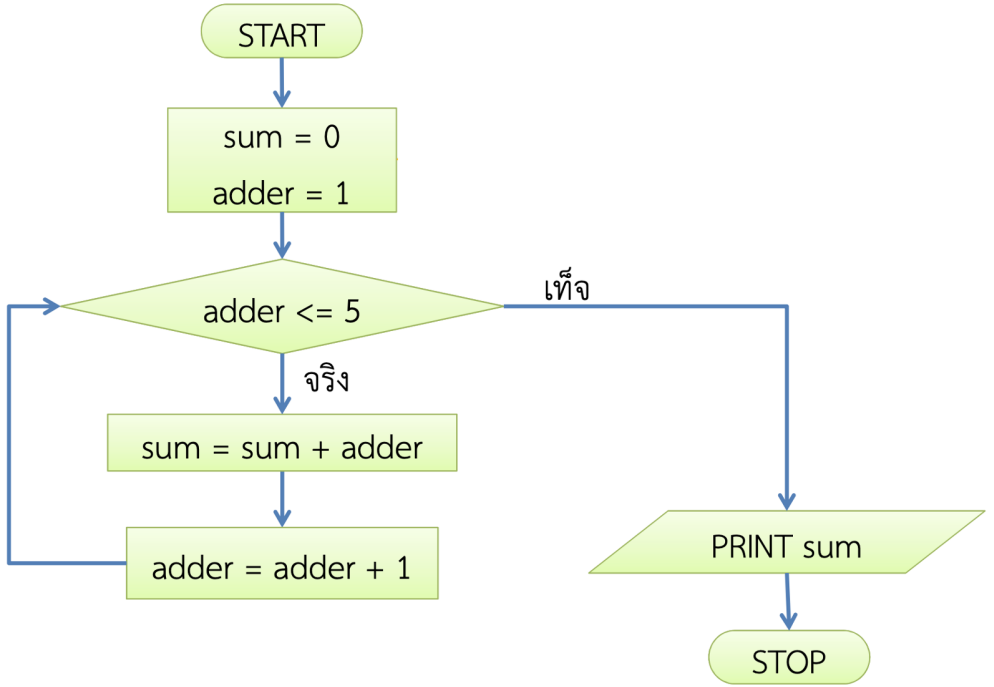
แนวคิด วน loop แล้วเก็บค่าผลรวมจาก 1 ไป 2 ไป 3 ต่อเนื่องไปเรื่อย ๆ

คำสั่งควบคุม for: ตัวอย่างโจทย์



คำสั่งควบคุม for: ตัวอย่างโจทย์

```c
#include <stdio.h>

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



คำสั่งควบคุม for: ตัวอย่างโจทย์

ใช้ for

```c
#include <stdio.h>

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

ใช้ while

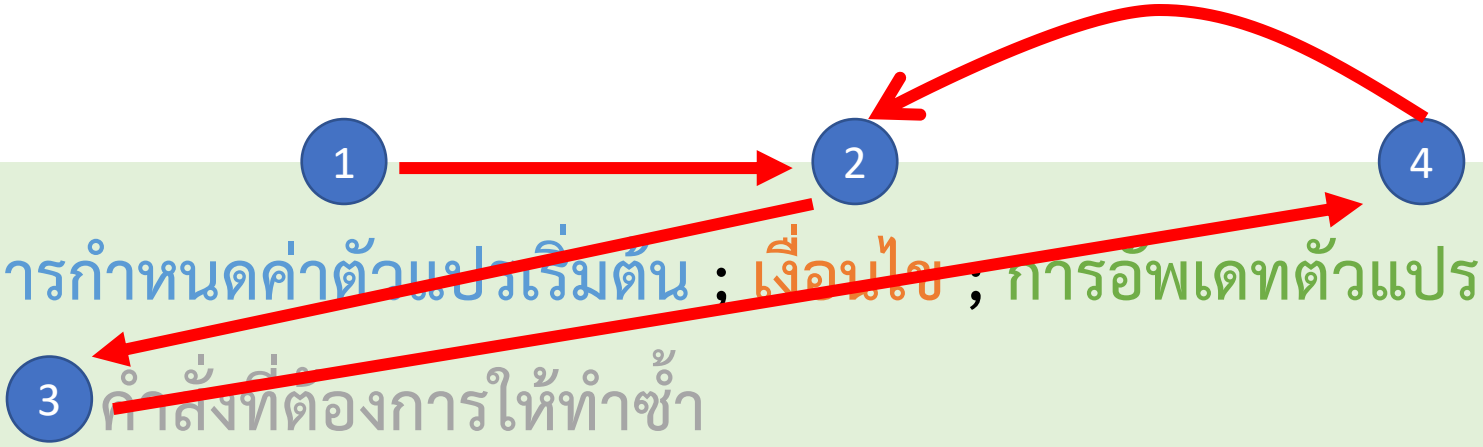
```c
#include <stdio.h>

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

คำสั่งควบคุม for: ลำดับการทำงาน


```
for (การกำหนดค่าตัวแปรเริ่มต้น ; เงื่อนไข ; การอัพเดทตัวแปรเงื่อนไข) {
        // คำสั่งที่ต้องการให้ทำซ้ำ
}
```

คำสั่ง for เป็นคำสั่งที่ทำให้การวนซ้ำที่กำหนดจำนวนครั้งได้แน่นอนสามารถทำได้สะดวกขึ้น แต่ลึกๆ ของมันก็มีสิ่งที่อาจดูไม่สมเหตุสมผล นั่นคือลำดับการทำงาน

คำสั่งควบคุม for: ลำดับการทำงาน




```
for (การกำหนดค่าตัวแปรเริ่มต้น ; เงื่อนไข ; การอัพเดทตัวแปรเงื่อนไข) {
        คำสั่งที่ต้องการให้ทำซ้ำ

}
```

1 > 2 > 3 > 4 > 2 > 3 > 4 > 2 > 3 > 4 > 2 > จบ loop

สังเกตว่า เริ่มที่ 1 ครั้งเดียวแล้วไม่กลับไปอีก และจบ loop ที่ 2

Outline

- รูปแบบของ Loop และแนวคิด
- **คำสั่งควบคุม For**
  - **ลูกเล่นของ for (ที่ควรรู้ แต่ไม่ควรพยายามใช้)**
- คำสั่งควบคุม Do... while

ลูกเล่นของ For

- จากโครงสร้างของคำสั่ง for จะเห็นว่าในเครื่องหมายวงเล็บ () จะถูกแบ่งคำสั่งออกเป็น 3 ส่วนด้วยเครื่องหมายเซมิโคล่อน ;

for(คำสั่งส่วนที่ 1; คำสั่งส่วนที่ 2; คำสั่งส่วนที่ 3)

- เรารู้หน้าที่ของคำสั่งแต่ละส่วนไปแล้ว
- แต่จริงๆแล้ว คำสั่งแต่ละส่วน ไม่จำเป็นต้องมีคำสั่งเดียวก็ได้

ลูกเล่นของ For

- เราสามารถกำหนดค่าเริ่มต้นของตัวแปรใน loop มากกว่า 1 ตัวก็ได้

```
for ( i = 0, sum = 0 ; i < 100 ; i++)
```

- แม้ว่า sum จะไม่ได้เป็นส่วนหนึ่งของส่วนที่ 2 ที่ทำหน้าที่เป็นเงื่อนไข ก็กำหนดค่าในบริเวณนี้ได้ (ในทางปฏิบัติ ลองหาเหตุผลสักข้อที่จะทำแบบนี้ดู)
- ใช้คอมม่าขั้นระหว่างคำสั่ง (,)
- ใช้เซมิโคล่อนขั้นระหว่างส่วน (ต้องมี 3 ส่วนเสมอ)

ลูกเล่นของ For

- เราสามารถอัพเดทค่าของตัวแปรเงื่อนไขใน loop มากกว่า 1 ตัวก็ได้

```
for ( i = 0 ; i < 100; i++, j=j*2)
```

- **แต่การอัพเดทตัวแปรพวกนี้จะอยู่ลำดับสุดท้าย (ทำคำสั่งที่ต้องการวนซ้ำก่อน แล้วจึงกระโดดกลับขึ้นมา)**

ลูกเล่นของ For

- หรือจะทำทั้งสองอย่างพร้อมกันเลยก็ได้

```
for ( i = 0, sum = 0; i < 100; i++, j=j*2)
```

- **ทำกรอบฟ้าก่อน แล้วมากรอบส้ม จากนั้นไปทำคำสั่งใน loop แล้วค่อยกลับมาทำกรอบเขียว**

ลูกเล่นของ For

- ไม่ใช่แค่นั้น เราใส่คำสั่งลงไปในแต่ละส่วนของ for เลยก็ได้

```
for ( printf("Sup man"), sum = 0; i < 100; printf("%d", i), i++)
```

- **ทำกรอบฟ้าก่อน แล้วมากรอบส้ม จากนั้นไปทำคำสั่งใน loop แล้วค่อยกลับมาทำกรอบเขียว**

ลูกเล่นของ For

- แต่จะใส่คำสั่งก็ระวังตรงนี้นิดนึง **ถ้าใส่คำสั่งในส่วนที่ 2 (กรอบส้ม) ประโยคที่จะใช้พิจารณาเงื่อนไขจะเป็นประโยคสุดท้ายประโยคเดียว**

```
for ( i = 0 ; i < 100, printf("GGez"); i++)
```

- แบบนี้ for ตัวนี้จะไม่ได้พิจารณา i < 100 แต่จะมอง printf("GGez"); เป็นเงื่อนไข ซึ่งไม่เท่ากับ 0 ซึ่งเป็นจริงเสมอ (วนไม่รู้จบ)

```
for ( i = 0 ; printf("GGez"), i < 100 ; i++)
```

- ถ้าแบบนี้จะพิจารณา i < 100 รันวนซ้ำ 100 รอบ

ลูกเล่นของ For

- หรือแต่ละส่วนของ for จะไม่มีอะไรเลยก็ได้

```
for (  ;  ;  )
```

- **แต่เซมิโคล่อนแบ่งส่วนก็ยังต้องครบ ไม่ว่าส่วนไหนส่วนหนึ่งจะว่างไป**
- **ก็จะเป็นการวนไม่รู้จบแบบ while(1)**

ลูกเล่นของ For

- หรือถ้าทำทุกอย่างในวงเล็บเงื่อนไขเสร็จแล้ว คำสั่งที่ต้องการให้วนซ้ำอาจไม่ต้องมีก็ได้ แล้วใส่เป็นปีกกาเปล่า ๆ

```
for ( printf("Sup man"), sum = 0; i < 100; printf("%d", i), i++)
{ }
```

ลูกเล่นของ For

- อย่าพยายามใช้ เข้าใจไว้ทำข้อสอบข้อเขียนก็พอ

ลูกเล่นของ For

ตัวอย่างโจทย์ 4 จงเขียนโปรแกรมที่หาผลรวมเลขอนุกรม

1*1 + 2*2 + 3*4 + 4*8 + 5*16 + ... + 10*512

ด้วยการใช้ลูป for และพิมพ์ผลรวมนั้นออกมาทางจอภาพ

ลูกเล่นของ For

```c
#include <stdio.h>

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

1 2 4 3

ลูกเล่นของ For

โค้ดที่ควรจะเขียน

```c
#include <stdio.h>

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

ลูกเล่นของ For

ตัวอย่างโจทย์ 5 จงเขียนโปรแกรมที่หาผลรวมเลขอนุกรม

2*3*5 + 4*9*13 + 8*27*35 + 16*81*97 +

32*234*250 + ... +2048*177147*179195

ด้วยการใช้ลูป for และพิมพ์ผลรวมนั้นออกมาทางจอภาพ

ลูกเล่นของ For

แทนที่จะนับจำนวนรอบ กรณีนี้เราอาจเทียบค่าของเทอม 1 ตรงๆ น่าจะสะดวกกว่านั่งนับว่าต้องวนกี่รอบ

```c
#include <stdio.h>

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);
}
```

ลูกเล่นของ For

ย้ายประโยคนี้จากในวงเล็บ for มาไว้ตรงนี้ จะไม่ผิดไวยากรณ์ แต่คำตอบจะผิด เพราะอะไร

```c
#include <stdio.h>

void main() {
    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;
    }
    printf("%lf", sum);
}
```

ลูกเล่นของ For

เพราะลำดับการคิดมันไม่เหมือนเดิมนั่นเอง

- ก่อนหน้านี้ term1 จะถูกนำไป *2 และ term2 จะถูกนำไป *3 ก่อน จึงจะนำทั้งสองตัวมาบวกกันเป็น term 3
- แต่การทำแบบตัวอย่างนี้ term1 จะถูกนำมารวมกับ term2 ก่อนจะนำไป *2 และ *3 ตามลำดับ

```c
#include <stdio.h>

void main() {
    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;
    }
    printf("%lf", sum);
}
```

ลูกเล่นของ For

สามารถเขียนโค้ดปล่อยว่างแบบนี้ได้

```c
#include <stdio.h>

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

ส่วนที่ปล่อยว่างย้ายมาอยู่ในส่วนคำสั่งแทน จะเห็นว่าตัวแปรในส่วนนี้ก็นำไปเป็นส่วนหนึ่งของเงื่อนไขได้ ไม่ต้องอยู่ในส่วนที่กำหนดหลัง for เสมอไป

ลูกเล่นของ For

```c
#include <stdio.h>

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

แทนที่จะประกาศใน for ส่วนที่ 1 (ซึ่งไม่รู้จะทำทำไม ก็ประกาศข้างนอกแบบที่เราคุ้นเคย)

สามารถเขียนโค้ดปล่อยว่างแบบนี้ได้

ลูกเล่นของ For

```c
#include <stdio.h>

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

ถ้าจะให้วนไม่รู้จบ (แล้วใช้ break;) จะปล่อยว่างแบบนี้ก็ได้

ถ้าใครชินคำสั่ง break; จะใช้แบบนี้ก็ได้ (ตย. นี้ไม่ genius เกินไป สามารถใช้ได้อยู่)

ลูกเล่นของ For

- ลูกเล่นสุด Genius ที่ได้เกริ่นไปแล้วอีกตัวคือการใส่คำสั่งลงไปในวงเล็บของ for เลย
- **การอ่านโค้ดจริงๆแล้วไม่ยากเท่าไหร่ แค่ไล่ทำไปตามลำดับ (ลำดับต้องถูก)**
- แต่ถ้าให้พยายามเขียนโค้ดแบบนั้นเพื่อแก้โจทย์ปัญหา อย่าเลย ลองหาวิธีอื่น
- ลองดูโจทย์ที่อาจออกเป็นข้อสอบวัดความเข้าใจกัน

ลูกเล่นของ For: โจทย์ความเข้าใจ 1

```c
#include <stdio.h>

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

ผลลัพธ์ที่ได้ควรเป็นอย่างไร?

ลูกเล่นของ For: โจทย์ความเข้าใจ 1

```c
#include <stdio.h>

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

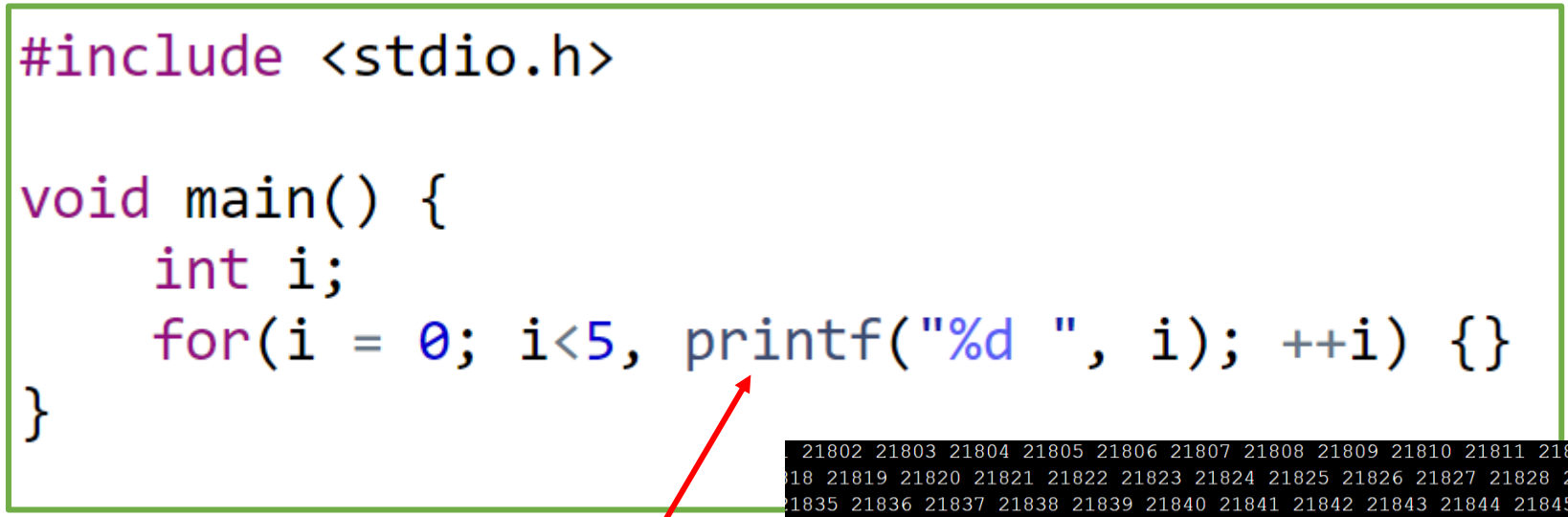
```
0 1 2 3 4 5
```

ลูกเล่นของ For: โจทย์ความเข้าใจ 2

```c
#include <stdio.h>

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

ผลลัพธ์ที่ได้ควรเป็นอย่างไร?

ลูกเล่นของ For: โจทย์ความเข้าใจ 2



```c
#include <stdio.h>

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

```
 21802 21803 21804 21805 21806 21807 21808 21809 21810 21811 21812 21813 21814 21815 21816 21817
18 21819 21820 21821 21822 21823 21824 21825 21826 21827 21828 21829 21830 21831 21832 21833 2183
1835 21836 21837 21838 21839 21840 21841 21842 21843 21844 21845 21846 21847 21848 21849 21850 21
 21852 21853 21854 21855 21856 21857 21858 21859 21860 21861 21862 21863 21864 21865 21866 21867
68 21869 21870 21871 21872 21873 21874 21875 21876 21877 21878 21879 21880 21881 21882 21883 2188
1885 21886 21887 21888
```

วนไม่รู้จบ!

เพราะ for จะตรวจเงื่อนไขประโยคสุดท้ายในส่วนที่ 2

ลูกเล่นของ For: โจทย์ความเข้าใจ 3

```c
#include <stdio.h>

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

ผลลัพธ์ที่ได้ควรเป็นอย่างไร?

ลูกเล่นของ For: โจทย์ความเข้าใจ 3

```c
#include <stdio.h>

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

```
1 2 3 4 2 3 4 2 3 4 2
```

สรุป For loop

- การใช้ for loop เหมาะกับการจัดการ loop ที่รู้จำนวนครั้งแน่นอนมากที่สุด เช่น วนจาก 0-n
- การนำไปใช้อย่างอื่น for loop อาจจะทำให้ยาก แทนที่จะง่าย
- ควรใช้อย่างระมัดระวัง เพราะกลไกในการคิดของ for loop มีความซับซ้อน
- เวลาจะใช้ for loop จงอย่าใช้ให้ถึงจุดที่มันซับซ้อน ให้ใช้แค่ที่มันทำให้ชีวิตง่ายขึ้น
- ถ้าเข้าใจ while loop เป็นอย่างดี และรู้สึกว่า for loop วุ่นวาย จะใช้
- while loop แทนก็ได้ (ใช้แทนกันได้โดยสมบูรณ์)
- ให้เลือกใช้วิธีที่รู้สึกมั่นใจ และค่อย ๆ พัฒนาความชำนาญ เมื่อชำนาญแล้วโค้ดยาวก็ไม่กลัว

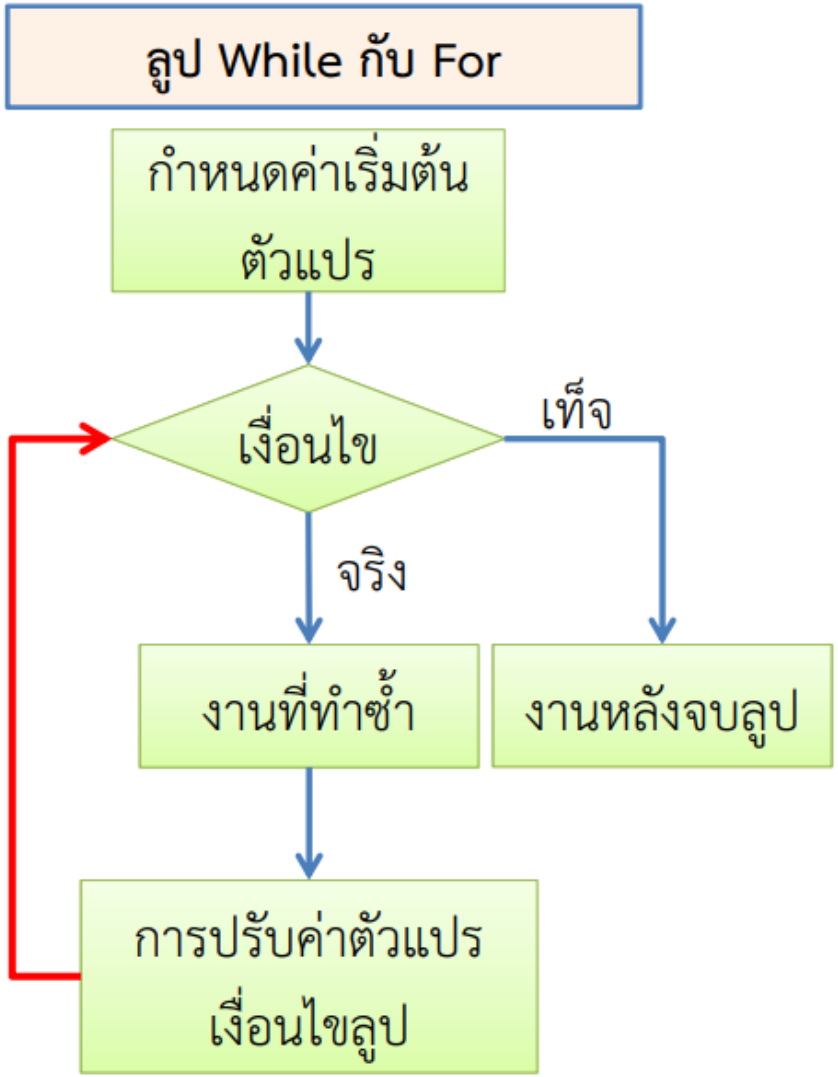
Outline

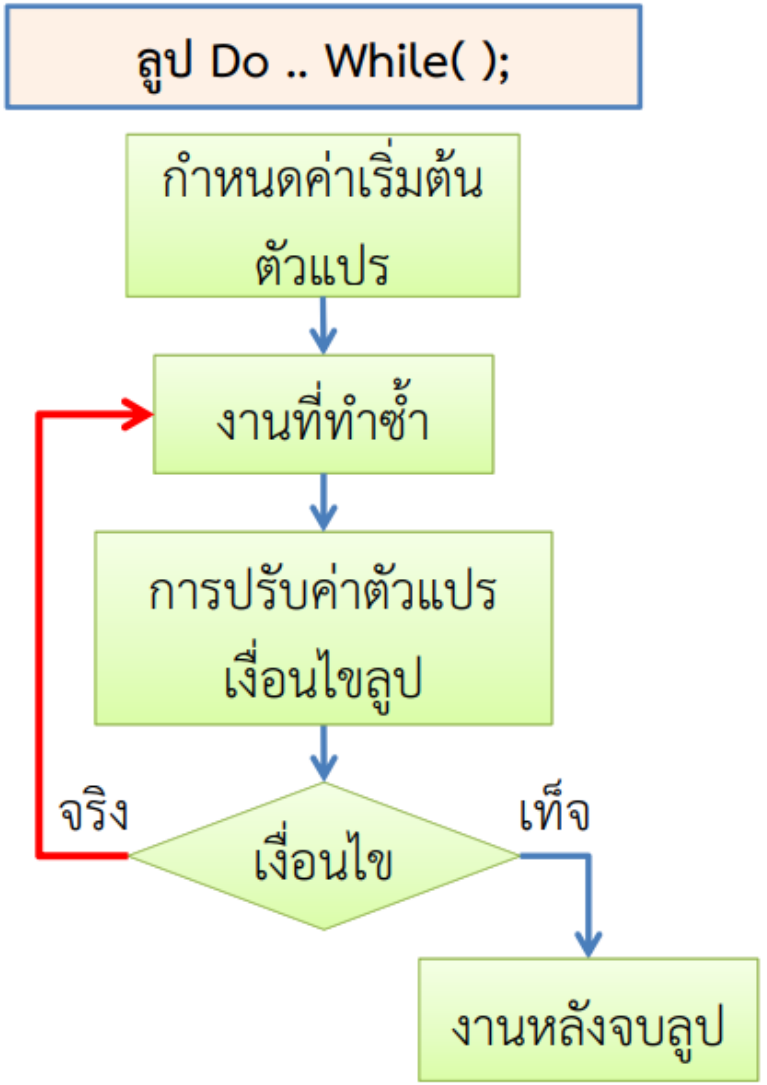
- รูปแบบของ Loop และแนวคิด
- คำสั่งควบคุม For
- **คำสั่งควบคุม do... while**

คำสั่งควบคุม do... while

- คำสั่งควบคุมรูปแบบสุดท้าย เป็น loop ที่มีเอกลักษณ์เฉพาะตัว คือลำดับการทำงานต่างจะวิธีการอื่น
- วิธีการอื่นตรวจสอบเงื่อนไขก่อนที่จะเข้า loop แต่ do... while จะทำคำสั่งต่างๆ ใน loop ก่อน 1 รอบ ก่อนที่จะตรวจสอบเงื่อนไข
- ถ้าเงื่อนไขเป็นจริง do...while จะวกกลับไปทำคำสั่งที่กำหนดไว้ใหม่อีกรอบ และตรวจสอบเงื่อนไขใหม่ ทำแบบนี้วนไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นเท็จ
- เกร็ดเล็กน้อย: do...while บางคนเรียกคำสั่งนี้ว่า old school command คือคำสั่งยุคเก่า ไม่ค่อยมีใครใช้ แต่ถ้าเราถนัดแบบนี้ก็ใช้ต่อไปได้

คำสั่งควบคุม do... while



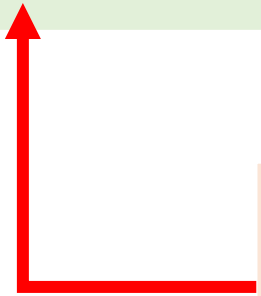


ที่มา ชีต อ.ภิญโญ แท้ประสาทสิทธิ์

คำสั่งควบคุม do...while

การใช้งาน do...while

```
do {
        //คำสั่งที่ต้องการให้ทำซ้ำ (จะทำไปก่อน 1 รอบโดยไม่ตรวจสอบเงื่อนไข)
} while ( เงื่อนไข ) ;
```

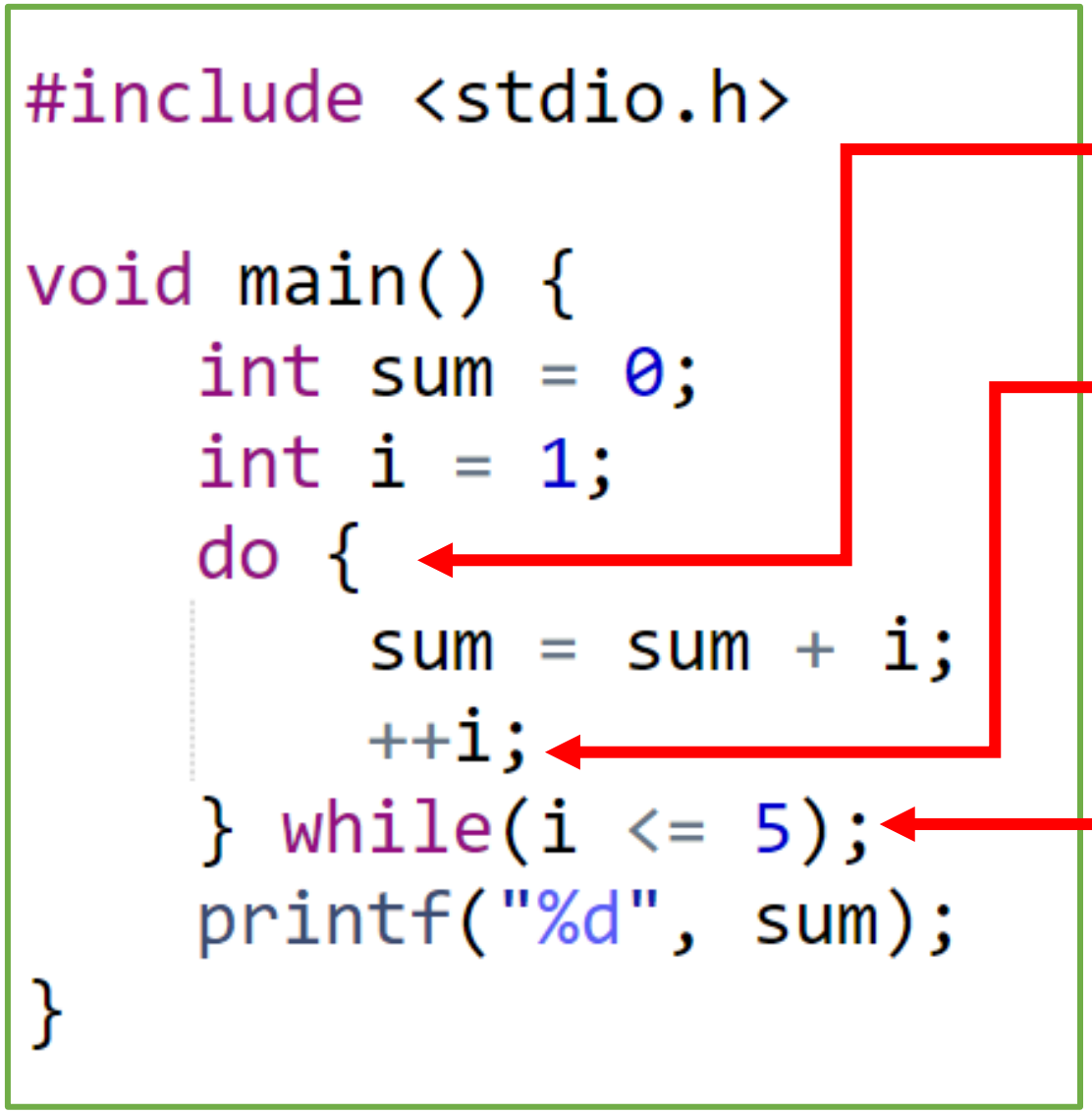


มีเซมิโคล่อนด้วย (;)
เป็นคำสั่งควบคุมคำสั่งเดียวที่มี ;

คำสั่งควบคุม Do... while

<u>ตัวอย่างที่ 6</u> จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด 1 ถึง 5 (ช่วงปิดจะรวมเลข 1 และ 5 ด้วย) จากนั้นพิมพ์ผลลัพธ์ออกมาทางจอภาพ

คำสั่งควบคุม Do... while



คำสั่งที่ต้องการให้ทำซ้ำอยู่ใน {} หลัง do

ยังต้องมีการอัพเดทตัวแปรเงื่อนไขเหมือน while และ for

อย่าลืม ;
(ย้ำว่าเป็นคำสั่งควบคุมตัวเดียวที่มี ;)

คำสั่งควบคุม 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

สรุป

- รูปแบบของ Loop และแนวคิด
- คำสั่งควบคุม for
- คำสั่งควบคุม do... while