Computer Programming I:
การเขียนโปรแกรมคอมพิวเตอร์ I

คำสั่งควบคุม WHILE LOOP

อ.ดร.ปัญญนัท อ้นพงษ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

aonpong_p@su.ac.th

Outline

- **พื้นฐานของ Loop**
- รูปแบบของ Loop และแนวคิด
- คำสั่งควบคุม While
- คำสั่ง break;
- คำสั่ง continue;
- Loop อนันต์

คำสั่งควบคุม

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

คำสั่งควบคุม

- ในวันนี้เราจะได้เรียนรู้คำสั่งที่ทำให้สามารถเขียนโปรแกรมที่มีการวนซ้ำได้ นั่นทำให้เราสามารถผสมเทคนิคต่าง ๆ เข้าด้วยกันได้อย่างหลากหลาย
  - แบบตรง ๆ บนลงล่างเส้นเดียว
  - แบบมีทางเลือก ซ้ายขวา
  - **แบบมีการวนซ้ำ**
  - **แบบมีทั้งสองอย่าง**

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

พื้นฐานของ Loop

- Loop แปลเป็นภาษาไทยได้ว่า ห่วง, บ่วง, วังวน
- ในภาษาคอมพิวเตอร์ หมายถึงการวนทำงานซ้ำ ๆ
- ถ้าเราต้องการพิมพ์คำว่า “Hello world” เราสามารถใช้ความรู้ของเราในตอนนี้พิมพ์ออกมาได้อย่างง่าย ๆ

```c
#include <stdio.h>

void main()
{
    printf("Hello World");
}
```

พื้นฐานของ Loop

- ทีนี้ถ้าเราอยากพิมพ์ “Hello world” ออกมา จำนวน 5 ครั้ง จะทำอย่างไร
- ถ้าเป็นตอนนี้ก็ไม่ยาก เราสามารถทำแบบนี้ได้

```c
#include <stdio.h>

void main()
{
    printf("Hello World");
    printf("Hello World");
    printf("Hello World");
    printf("Hello World");
    printf("Hello World");
}
```

พื้นฐานของ Loop

- ทีนี้ถ้าเราอยากพิมพ์ “Hello world” ออกมา จำนวน 1,000 ครั้ง จะทำอย่างไร
- อันนี้เริ่มยากแล้ว เพราะ
  - โอกาสทำไม่ครบก็มี
  - โอกาสวางเกิน 1000 ครั้งก็มี
  - ถ้าลูกค้าที่สั่งให้เขียนโปรแกรมนี้บอกว่า..

    “ขอโทษนะครับ ช่วยเปลี่ยนจาก Hello world เป็น Hi world ที”

```c
#include <stdio.h>

void main()
{
    printf("Hello World");
    printf("Hello World");
    printf("Hello World");
    //...
    //ก๊อปปี้วางไปเรื่อยๆ 1000 ครั้ง
    printf("Hello World");
}
```

พื้นฐานของ Loop

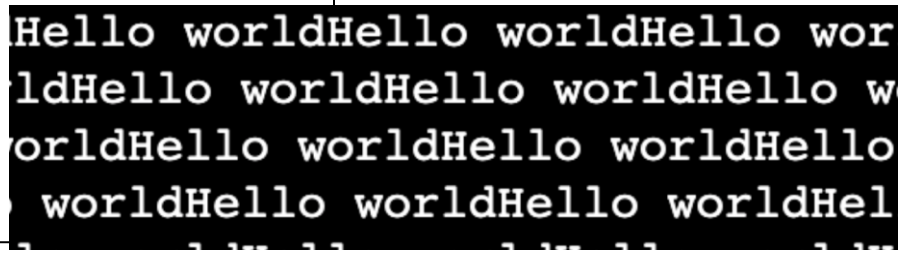
- ถ้าเรารู้จัก Loop เราจะไม่ต้องกังวลว่าจะต้องแก้ไขอะไรมากมายอีกต่อไป

```c
#include <stdio.h>

void main()
{
    int i=0;
    while (i<1000){
        printf("Hello world");
        i++;
    }
}
```

ถ้าจะแก้จำนวนครั้งก็แก้ตรงนี้

ถ้าจะแก้คำก็แก้ตรงนี้



พื้นฐานของ Loop

- ในการทำงานแต่ละรอบของ Loop ไม่จำเป็นต้องทำงานเหมือนกันเป๊ะทุกครั้งก็ได้ เราอาจตั้งเงื่อนไขที่แตกต่างกันในการทำงานแต่ละรอบ
- นอกจากนี้ยังมีเทคนิคอีกหลายอย่างที่จะได้เรียนต่อไป

```
Hello worldHi worldHello worldHi world
Hello worldHi worldHello worldHi world
Hello worldHi worldHello worldHi world
Hello worldHi worldHello worldHi world
Hello worldHi worldHello worldHi world
```

```c
#include <stdio.h>

void main()
{
    int i=0;
    while (i<1000){
        if(i%2==0)
            printf("Hello world");
        else
            printf("Hi world");
        i++;
    }
}
```

Outline

- พื้นฐานของ Loop
- **รูปแบบของ Loop และแนวคิด**
- คำสั่งควบคุม While
- คำสั่ง break;
- คำสั่ง continue;
- Loop อนันต์

รูปแบบของ Loop และแนวคิด

แนวคิดของ Loop ที่ทุกรูปแบบมีร่วมกันคือ

- การที่โปรแกรมของเราต้องทำอะไรซ้ำ ๆ หลาย ๆ ครั้ง หรือไม่สามารถกำหนดจำนวนครั้งได้อย่างแน่นอน เช่น
  - ให้รับค่าจำนวนเต็ม N และพิมพ์คำว่า Hello world ออกมาจำนวน N ครั้ง
  - ให้รับค่าจำนวนเต็มมาเรื่อย ๆ จนกว่าค่าที่รับจะเป็นเลข -1
- การที่โปรแกรมต้องการทำงานเกี่ยวกับอนุกรมเลขคณิตต่าง ๆ เช่น
  - การหาเลขในอนุกรม
  - การหาผลรวมของตัวเลขในขอบเขต
- โจทย์อื่น ๆ สุดแล้วแต่จะคิดได้ (ที่มีการวน ๆ กลับไปทำสิ่งเดิมในโฟลวชาร์ต)

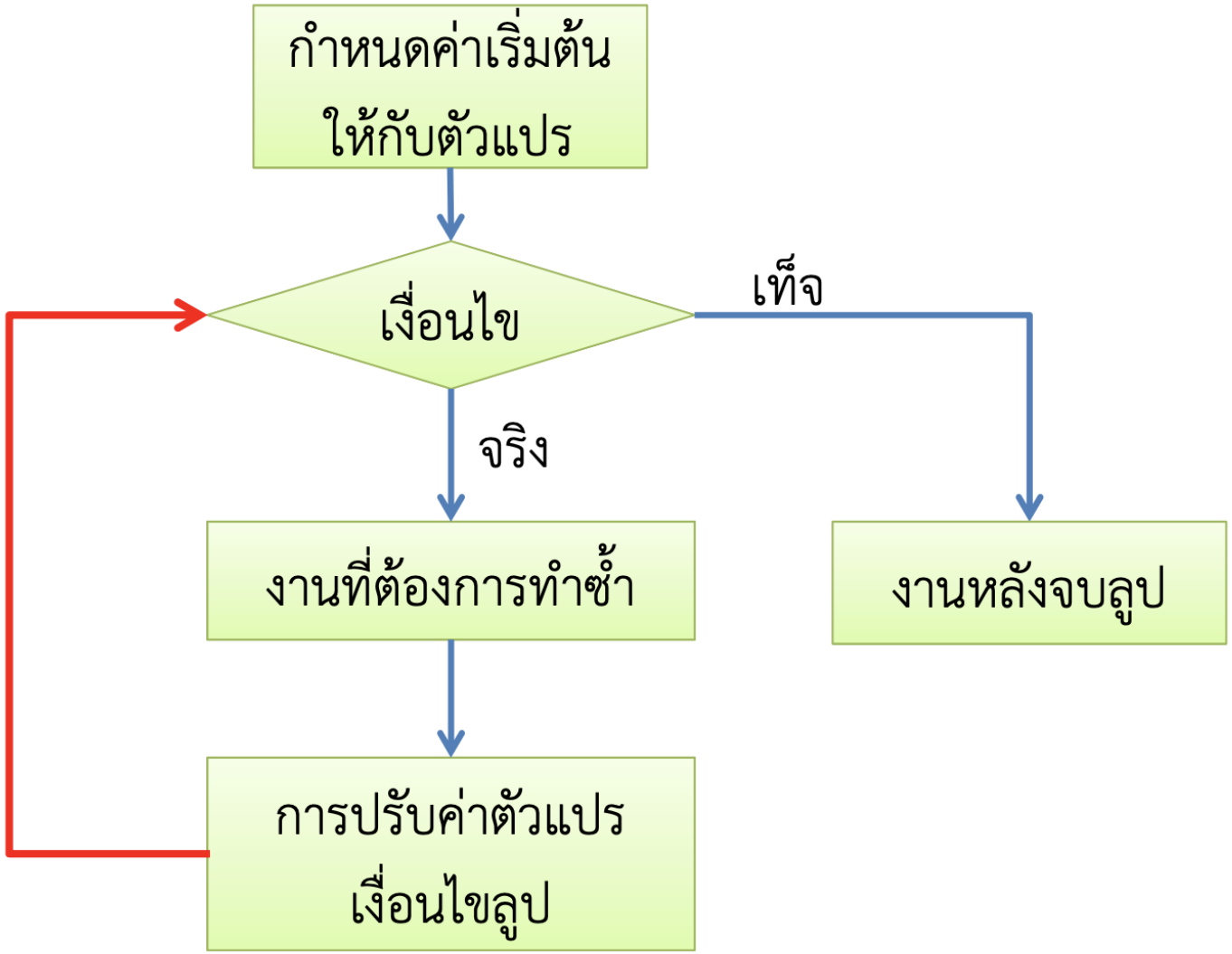
รูปแบบของ Loop และแนวคิด

รูปแบบของ Loop ในภาษาซี มี 3 รูปแบบ

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

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

รูปแบบของ Loop และแนวคิด: while, for



ให้ดูภาพนี้สร้างความคุ้นเคยก่อน แล้วลองคิดทบทวนว่าตอนเราหัดเขียน Flowchart เราเคยเขียนออกมาในลักษณะคล้ายๆแบบนี้บ้างหรือไม่

เดี๋ยวรูปนี้จะกลับมาให้ดูอีกครั้งในส่วนถัดไป

Outline

- พื้นฐานของ Loop
- รูปแบบของ Loop และแนวคิด
- **คำสั่งควบคุม While**
- คำสั่ง break;
- คำสั่ง continue;
- Loop อนันต์

คำสั่งควบคุม while

while เป็นคำสั่งที่ใช้งานคล้ายกับ if

เอาเฉพาะพื้นฐานจริงๆ ก่อน ให้พิจารณาโค้ดด้านขวา

- ถ้าเงื่อนไขใน ( ) เป็นจริง คำสั่งถัดมาจะถูกทำ
- ถ้าต้องการให้ทำหลายคำสั่ง เราก็ใช้ { } ครอบทุกคำสั่งเอาไว้เป็นขอบเขต **แม้มีคำสั่งเดียวก็ควรใส่เอาไว้*
- ทุกอย่างดูเหมือนจะเหมือน if หมดเลย เพียงแต่ ในคำสั่ง while **เมื่อทุกอย่างทำงานจนครบหมดแล้ว จะทำการกลับไปตรวจเงื่อนไขใหม่ แล้วทำซ้ำไปเรื่อย ๆ**

```
while ( เงื่อนไข ) {
        //      คำสั่ง
}
```

คำสั่งนี้จะถูกทำซ้ำจนกว่าเงื่อนไขจะเป็นเท็จ

คำสั่งควบคุม while

ก่อนจะคุยเรื่อง while กันต่อ พิจารณาโค้ดนี้ก่อน

โค้ดนี้ทำการประกาศค่า i เป็นจำนวนเต็มแล้ว print ออกมา

- กรณีนี้ โค้ดจะไม่ error แต่ค่าของ i จะออกมาแบบสุ่ม (มักไม่ใช่ค่าที่พึงประสงค์)

```c
#include <stdio.h>

void main()
{
    int i;
    printf("%d", i);

}
```

คำสั่งควบคุม while

ก่อนจะคุยเรื่อง while กันต่อ พิจารณาโค้ดนี้ก่อน

โค้ดนี้ทำการประกาศค่า i เป็นจำนวนเต็มแล้ว print ออกมา

- ถ้ามีการใช้ scanf มาให้ input ค่า อันนี้ไม่มีปัญหา เพราะแม้ตอนประกาศ ค่า i จะเป็นค่าสุ่ม แต่พอเก็บค่าจากคีย์บอร์ด ค่าที่เราป้อนจะเข้ามาแทนที่ทันที

```c
#include <stdio.h>

void main()
{
    int i;
    scanf("%d", &i);
    printf("%d", i);

}
```

บรรทัดนี้ค่าเป็น 88 (สุ่มมา แต่ละครั้งไม่เท่ากัน)

รับค่าเข้า ถ้าผู้ใช้กรอก 0 ก็จะไปแทน 88 ทันที

แทนที่เมื่อ print จะออกค่าสุ่ม (88) ก็จะออกเป็นค่ารับเข้า (0)

คำสั่งควบคุม while

พิจารณาโค้ดนี้ ผู้เขียนต้องการพิมพ์คำว่า “Sawasdee” 100 ครั้ง **ลองหาข้อผิดพลาด**

```
#include <stdio.h>

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

}
```

คำสั่งควบคุม while

พิจารณาโค้ดนี้ ผู้เขียนต้องการพิมพ์คำว่า “Sawasdee” 100 ครั้ง **ลองหาข้อผิดพลาด**

```c
#include <stdio.h>

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

}
```

บรรทัดนี้ค่าเป็นค่าสุ่มมา

- ถ้าสุ่มเป็นเลข >= 100 ก็จะไม่ได้ทำอะไรใน while ต่อ
- ถ้าสุ่มเป็นเลข < 100 ก็จะเข้า while ทันที แล้วไม่ได้ออกมาอีก (เพราะต้องวนซ้ำรัวๆ)

คำสั่งควบคุม while

พิจารณาโค้ดนี้ ผู้เขียนต้องการพิมพ์คำว่า “Sawasdee” 100 ครั้ง **ลองหาข้อผิดพลาด**

```
#include <stdio.h>

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

}
```

ตอนนี้บรรทัดนี้ไม่ใช่ค่าสุ่มแล้ว แต่ถูกกำหนดเป็น 0 ตั้งแต่ต้น

- ลองไล่โค้ดดูแล้วยังพบปัญหาว่า เมื่อเข้าลูปไปแล้วไม่มีทางกลับออกมา เพราะ i จะเป็น 0 เสมอ การจะทำให้ i มากกว่าหรือเท่ากับ 100 ได้ ต้องมีการอัพเดทค่า i

คำสั่งควบคุม while

พิจารณาโค้ดนี้ ผู้เขียนต้องการพิมพ์คำว่า “Sawasdee” 100 ครั้ง โค้ดนี้ใช้งานได้

```c
#include <stdio.h>

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

}
```

กำหนดค่าตั้งต้น

เงื่อนไข

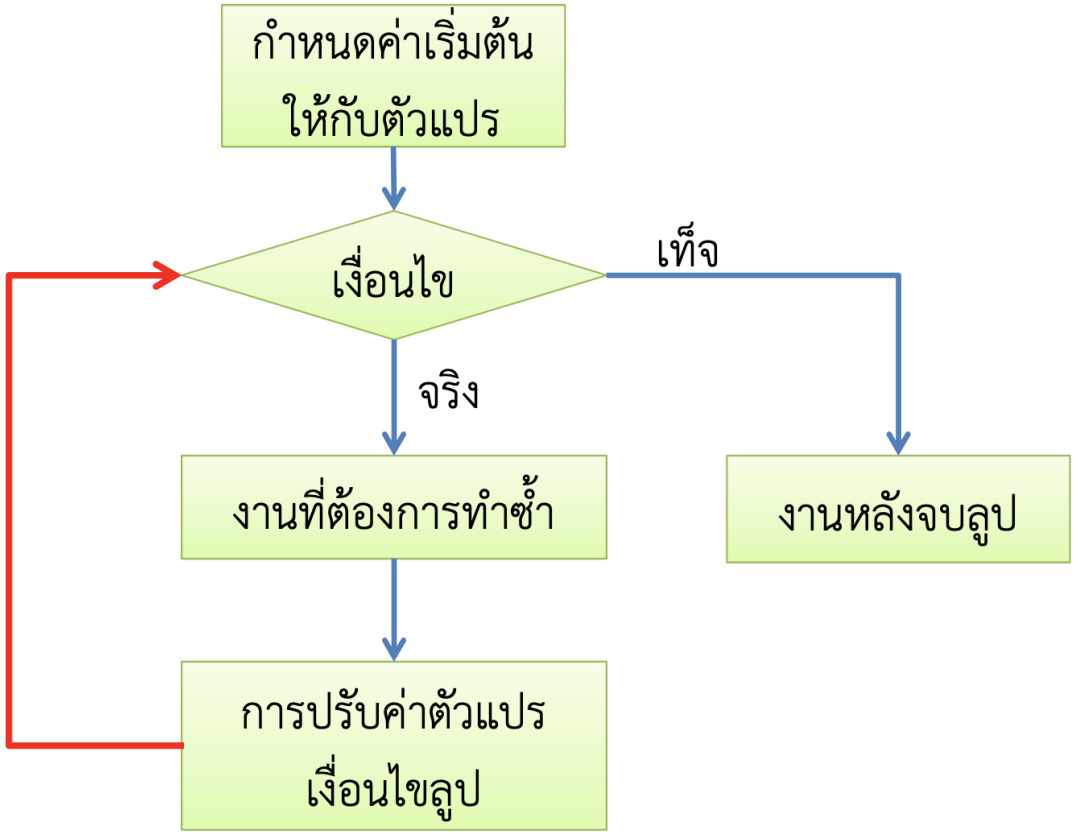
คำสั่ง

อัพเดทตัวแปร

คำสั่งควบคุม while

โค้ดที่ใช้คำสั่ง while **มัก**พบในลักษณะนี้ (ไม่ใช่ทุกกรณี)

```
การกำหนดค่าเริ่มต้นตัวแปรก่อนเข้าลูป
while ( เงื่อนไข ) {
    งานที่ต้องการทำซ้ำ
    การปรับค่าตัวแปรเงื่อนไขลูป
}
... งานหลังจบลูป ...
```



คำสั่งควบคุม while

<u>จุดสังเกต</u>

- **งานที่ต้องการให้ทำซ้ำจะอยู่ใน loop ส่วนงานอื่นที่ทำครั้งเดียวก็ต้องอยู่ข้างนอก** เช่น ถ้าต้องการพิมพ์ผลลัพธ์สุดท้ายออกมาครั้งเดียว แต่นำ printf ไปใส่ใน loop คุณก็จะพิมพ์ผลออกมาทางหน้าจอซ้ำ ๆ ด้วย (คล้ายที่ได้แสดงการ print คำว่า hello ซ้ำ ๆ ก่อนหน้านี้)
- การอัพเดทตัวแปรที่เป็นเงื่อนไขภายใน loop จะต้องอยู่ใน loop มิฉะนั้นจะเกิด loop อนันต์หรือที่เรียกกันติดปากนักเขียนโปรแกรมว่า infinite loop
- ตัวแปรที่เป็นเงื่อนไขอาจถูกอัพเดทด้วยอะไรก็ได้ ไม่ว่าจะเป็น สมการ หรือ input หรือแม้แต่การใช้ ++, -- ธรรมดาก็ได้
- เป็นไปได้ที่ตัวแปรจะไม่เข้าเงื่อนไขแต่แรก กรณีนี้คำสั่งใน loop จะไม่ถูกกระทำเลย

คำสั่งควบคุม while: ตัวอย่างโจทย์

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

คำสั่งควบคุม while: ตัวอย่างโจทย์

<u>ตัวอย่างโจทย์ 2</u> 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

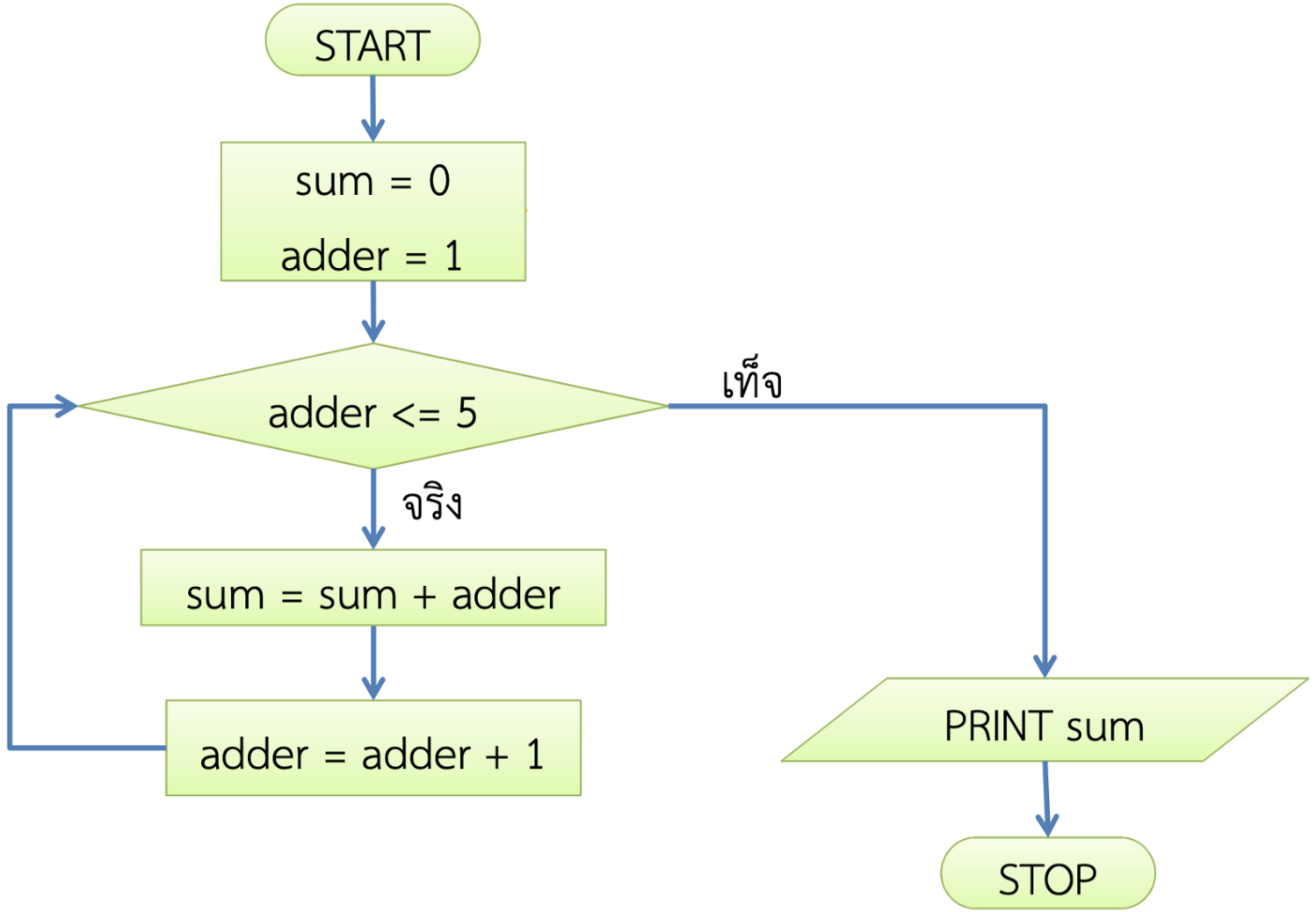
คำสั่งควบคุม while: ตัวอย่างโจทย์

<u>ตัวอย่างโจทย์ 3</u> จงเขียนโฟลวชาร์ตและโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด 1 ถึง 5 (ช่วงปิดจะรวมเลข 1 และ 5 ด้วย) จากนั้นพิมพ์ผลลัพธ์ออกมาทางจอภาพ (บังคับให้ใช้ลูปค่อย ๆ บวกเลขทีละค่า)

วิเคราะห์

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

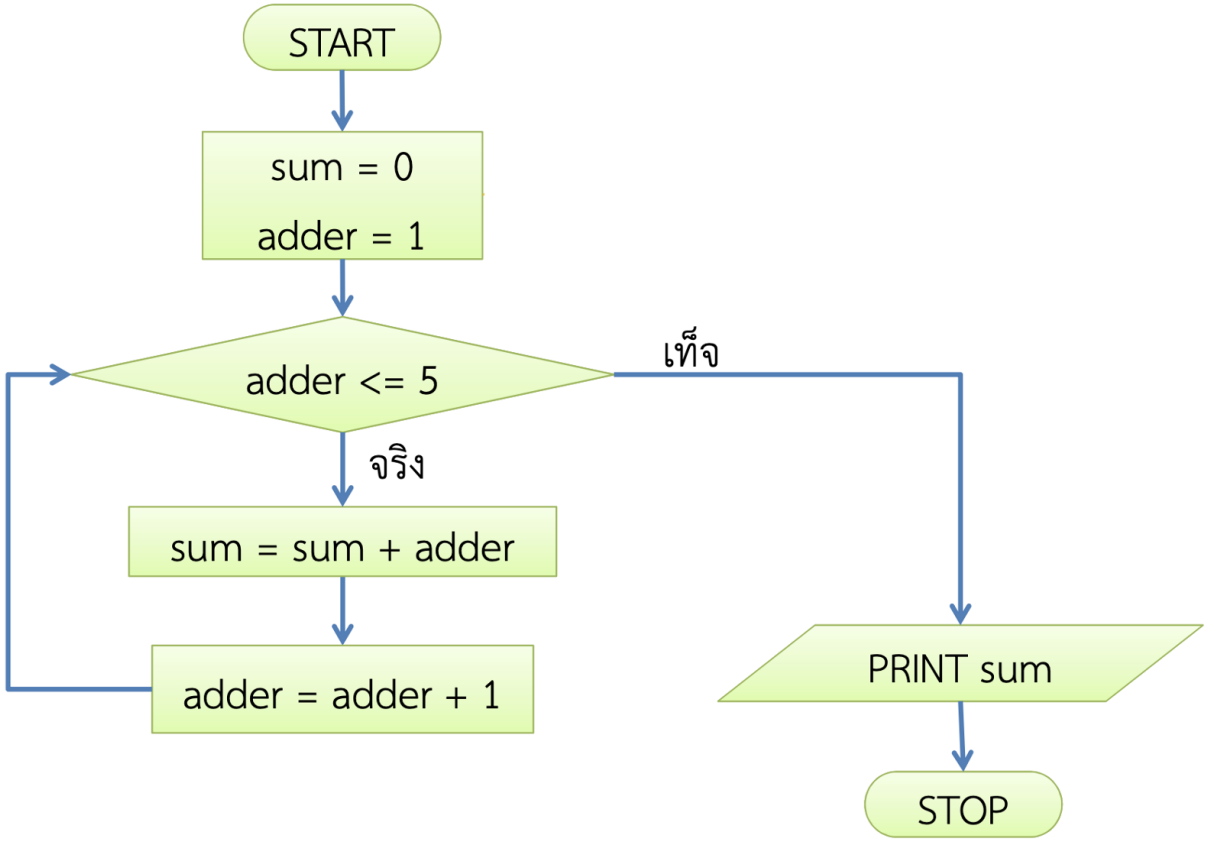
คำสั่งควบคุม while: ตัวอย่างโจทย์



คำสั่งควบคุม while: ตัวอย่างโจทย์

```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 while

<u>ความท้าทายของ loop</u>

จากที่เราได้ลองทำตัวอย่างโจทย์ไปแล้ว เราจะพบกับความท้าทายหลายประการ

- ความยากของเรื่อง loop ก็คือว่า ‘loop มักจะมีงานแฝงที่ต้องทำเพื่อให้มันสามารถทำตามวัตถุประสงค์ได้’ งานตัวนี้มักไม่ปรากฏในโจทย์และเราต้องเข้าใจมันเอง
- จำเป็นต้องผ่านการฝึกคิดและทำโจทย์ด้วยตัวเองระยะหนึ่ง

<u>เทคนิค</u>

- เราจะต้องเข้าใจความสัมพันธ์ระหว่างงานหลักที่จะให้ทำเป็นอันดับแรก
- จากนั้นงานหลักจะบอกเราได้ว่า มีอะไรที่โปรแกรมยังขาดไป และเราต้องหาทางเติมเต็มส่วนที่ขาดไปนั้น
- อย่างที่เคยบอกบ่อยๆ เมื่อทำโจทย์เยอะๆ จะรู้สึกว่าโจทย์มันเริ่มซ้ำๆ (แม้จะไม่เหมือนกันเป๊ะๆ)

คำสั่งควบคุม while

<u>จากตัวอย่างที่ 3</u>

- งานนี้เราต้องสังเคราะห์ข้อมูลขึ้นมาสำหรับทำการบวก ตรงนี้เป็นงานแฝงที่ไม่ปรากฏในตัวปัญหาโดยตรง
- เงื่อนไขในการทำ loop ที่บอกว่า adder <= 5 เป็นสิ่งที่ไม่มีการระบุไว้ในตัวปัญหาเลย แต่เป็นเงื่อนไขที่เกิดขึ้นมาเพื่อให้ทำวัตถุประสงค์หลักได้
- เรามีการปรับค่าตัวแปร loop คือ adder = adder + 1; ตัวแปรนี้นอกจากจะใช้ในเงื่อนไขแล้วยังถูกนำไปใช้ในการคำนวณในฐานะงานแฝงด้วย (สังเคราะห์ข้อมูล)

คำสั่งควบคุม while: ตัวอย่างโจทย์

<u>ตัวอย่างโจทย์ 4</u> จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด x ถึง y (ช่วงปิดจะรวมเลข x และ y ด้วย) โดยที่ค่า x และ y มาจากผู้ใช้ และ y > x สมมติว่าผู้ใช้ใส่ค่าที่สอดคล้องเงื่อนไขนี้ไม่มีพลาด เมื่อทำการบวกจนเสร็จแล้ว ให้พิมพ์ผลลัพธ์ออกมาทางจอภาพ

วิเคราะห์

1. การบวกควรเริ่มจาก x, x + 1, ... ไปสิ้นสุดที่ y

2. ค่า x และ y นี้เป็นตัวกำหนดขอบเขตของตัวบวกและสามารถใช้เป็นเงื่อนไขของลูปในปัญหานี้ได้

3. ในเมื่อโจทย์กำหนดไว้แล้วว่าผู้ใช้ใส่ค่า y ที่มากกว่า x ตลอดเวลาเราไม่ต้องกังวลคอยตรวจค่าว่าจะผิด

คำสั่งควบคุม while: ตัวอย่างโจทย์

โค้ดสำหรับตัวอย่างโจทย์ 4 แบบที่ 1

```c
#include <stdio.h>

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

คำสั่งควบคุม while: ตัวอย่างโจทย์

โค้ดสำหรับตัวอย่างโจทย์ 4 แบบที่ 2

```c
#include <stdio.h>

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

คำสั่งควบคุม while: ตัวอย่างโจทย์

<u>ตัวอย่างโจทย์ 5</u> จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด x ถึง y โดยที่ค่า x และ y มาจากผู้ใช้ เมื่อทำการบวกจนเสร็จแล้วให้พิมพ์ผลลัพธ์ออกมาทางจอภาพ

วิเคราะห์

1. ปัญหานี้ไม่ได้ระบุว่า y > x หรือ x > y และที่จริง x จะเท่ากับ y ก็ได้
2. วิธีแก้ปัญหาก็คือเรากำหนดให้ adder เปลี่ยนค่าจาก x ไป y โดยดูว่าจะต้องเพิ่มหรือลดค่า adder ถ้าหาก y > x เหมือนข้อที่แล้วก็ให้เพิ่มค่า adder แต่ถ้า y < x ก็ให้ลดค่า adder
3. เงื่อนไข loop จะตั้งว่าอย่างไรดี ลองคิดเองก่อนเปิดหน้าถัดไป

คำสั่งควบคุม while: ตัวอย่างโจทย์

เรามีทางเลือก 2 ทางเลือกที่น่าจะเป็นไปได้

1. x >= adder >= y เงื่อนไขนี้เป็นจริงได้เมื่อ x >= y
2. y >= adder >= x เงื่อนไขนี้เป็นจริงได้เมื่อ x <= y

คำสั่งควบคุม while: ตัวอย่างโจทย์

เนื่องจาก x >= adder >= y และ y >= adder >= x จะมีอันใดอันหนึ่งที่เป็นจริงได้เท่านั้น หาก adder ยังมีค่าอยู่ระหว่าง x และ y จริง

เช่น

- ถ้า x = 1, y = 5 และ adder = 3 จะได้ว่า x >= adder >= y เป็นเท็จ แต่ y >= adder >= x เป็นจริง
- ถ้าเราสลับค่าให้ x = 5 และ y = 1 เราก็จะได้ผลว่า x >= adder >= y เป็นจริง แต่ y >= adder >= x จะกลับกลายเป็นเท็จ
- หากค่า adder ออกไปนอกช่วง เช่น adder = 7 เราก็จะพบว่า 1 >= 7 >= 5 และ 5 >= 7 >= 1 เป็นเท็จทั้งคู่
- ดังนั้นเราสามารถทำการใช้ OR ของเงื่อนไขทั้งสองกลุ่มนี้ได้ เพราะขอแค่มีตัวเดียวเป็นจริงก็เพียงพอแล้ว
- สรุปเงื่อนไขที่ควรใช้คือ (x >= adder >= y) || (y >= adder >= x)

คำสั่งควบคุม while: ตัวอย่างโจทย์

โค้ดสำหรับตัวอย่างโจทย์ 5 แบบที่ 1

```c
#include <stdio.h>

void main()
{
    int x, y;
    scanf("%d%d", &x, &y);
    int adder = x, sum = 0;
    while((adder <= y && adder >= x) || (adder <= x && adder >= y)){
        sum = sum + adder;
        if(x <= y) {
            adder = adder + 1;
        } else {
            adder = adder - 1;

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

คำสั่งควบคุม while: ตัวอย่างโจทย์

<u>จากตัวอย่างโจทย์ 5</u>

- จะเห็นว่าโจทย์ดูไม่ซับซ้อนและไม่มีอะไร แต่จริงๆแล้วแฝงความยากไว้ให้เราสะดุดได้ ถ้าเราเลือกทางที่ไม่เหมาะสม เงื่อนไขของ loop อาจมีความซับซ้อนมากกว่าที่ควรจะเป็น แม้ว่าบางครั้งจะใช้ได้เหมือนกัน (และถ้าใช้ได้เหมือนกันจริงๆก็จะยังคงได้คะแนนเต็มจำนวน)
- จากข้อ 5 การทำงานใน loop มีเพียงนิดเดียว แต่เงื่อนไขของ loop กลับมีความซับซ้อนมาก
  - แค่ให้ตัวแปร adder วิ่งจาก x ไปยัง y แต่เงื่อนไขกลับยาวเป็นบรรทัดเพื่อให้รองรับทุกกรณี
  - ความชำนาญในการแก้ปัญหาแบบนี้ หาไม่ได้จากการอ่าน ต้องฝึกทำเท่านั้น
- อย่างไรก็ตาม ยังมีวิธีแก้ปัญหานี้ที่ดีอีกวิธีหนึ่ง และเทคนิคจากวิธีนี้ก็นำไปใช้กับโจทย์ในลักษณะคล้ายกันได้อีกหลายรูปแบบ นั่นคือ**เทคนิคการสลับค่า**

คำสั่งควบคุม while: ตัวอย่างโจทย์

<u>ตัวอย่างโจทย์ 5 (โจทย์เดิม)</u> จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด x ถึง y โดยที่ค่า x และ y มาจากผู้ใช้ เมื่อทำการบวกจนเสร็จแล้วให้พิมพ์ผลลัพธ์ออกมาทางจอภาพ

วิเคราะห์

1. ปัญหานี้ไม่ได้ระบุว่า y > x หรือ x > y และที่จริง x จะเท่ากับ y ก็ได้
2. ถ้าเรากำหนดให้ x < y ทุกครั้งได้ โจทย์นี้จะง่ายขึ้น ดังนั้น ถ้า x > y เราจะลองสลับค่าระหว่างค่าของ x และค่าของ y ทีนี้ เมื่อนำไปเขียนโค้ด x จะมีค่าน้อยกว่าหรือเท่ากับ y เสมอ
3. เรารู้แล้วว่าจะสลับค่า แต่จะสลับยังไงดี

คำสั่งควบคุม while: เทคนิคสลับค่าตัวแปร

*เทคนิคนี้ไม่ใช่สูตรสำเร็จของการทำโจทย์ข้อนี้ แต่เป็นเหมือนตัวต่อจิ๊กซอว์ที่นำไปใช้ในโจทย์ที่หลากหลาย และบังเอิญโจทย์ข้อนี้ก็สามารถใช้เทคนิคนี้ได้พอดี

พิจารณา

ถ้าอยากสลับค่า x และ y ใช้โค้ดแบบนี้ได้หรือไม่

ตอบ

```c
#include <stdio.h>

void main()
{
    int x, y;
    scanf("%d %d", &x, &y);
    x = y;
    y = x;
    printf("%d %d", x, y);
}
```

คำสั่งควบคุม while: เทคนิคสลับค่าตัวแปร

*เทคนิคนี้ไม่ใช่สูตรสำเร็จของการทำโจทย์ข้อนี้ แต่เป็นเหมือนตัวต่อจิ๊กซอว์ที่นำไปใช้ในโจทย์ที่หลากหลาย และบังเอิญโจทย์ข้อนี้ก็สามารถใช้เทคนิคนี้ได้พอดี

พิจารณา

ถ้าอยากสลับค่า x และ y ใช้โค้ดแบบนี้ได้หรือไม่

ตอบ ไม่ได้ เพราะเมื่อ x เก็บค่า y ไว้แล้ว ค่าเดิมของ x จะหายไป

เมื่อสั่งให้ y = x; สิ่งที่ y เก็บไปจึงเป็นค่าของตัวมันเองที่เพิ่งส่งให้ x ไป

```c
#include <stdio.h>

void main()
{
    int x, y;
    scanf("%d %d", &x, &y);
    x = y;
    y = x;
    printf("%d %d", x, y);
}
```

5 3

ตอนนี้ x กลายเป็น 3 และ y ยังคงเป็น 3

ตอนนี้ y เก็บค่าของ x (ซึ่งตอนนี้เป็น 3)

Input 5 3

Output 3 3

คำสั่งควบคุม while: เทคนิคสลับค่าตัวแปร

ทางแก้คือ ใช้ตัวแปรอีกตัวหนึ่งมาเก็บค่า x เอาไว้ก่อน แล้วค่อยไปโอนให้ y

```
#include <stdio.h>

void main()
{
    int x, y, tmp;
    scanf("%d %d", &x, &y);
    tmp = x;
    x = y;
    y = tmp;
    printf("%d %d", x, y);
}
```

5 3

ตอนนี้ tmp กลายเป็น 5 ตาม x

ตอนนี้ x กลายเป็น 3 ส่วน y ยังเป็น 3 เหมือนเดิม

ตอนนี้ y กลายเป็น 5 ตาม tmp

Input `5 3`

Output `3 5`

เก็บฟอร์มนี้ไว้เป็นตัวอย่างเลย

```
int temp = x;
x = y;
y = temp;
```

คำสั่งควบคุม while: เทคนิคสลับค่าตัวแปร

<u>โค้ดสำหรับตัวอย่างที่ 5</u> แบบที่ 2 (ใช้การประยุกต์เทคนิคสลับตัวแปร)

```c
#include <stdio.h>

void main()
{
    int x, y, tmp;
    scanf("%d %d", &x, &y);
    if(x > y) {
        tmp = x;
        x = y;
        y = tmp;
    }
    int adder = x, sum = 0;
    while(adder <= y) {
        sum = sum + adder;
        adder = adder + 1;
    }
    printf("%d", sum);
}
```

Outline

- พื้นฐานของ Loop
- รูปแบบของ Loop และแนวคิด
- คำสั่งควบคุม While
- **คำสั่ง break;**
- คำสั่ง continue;
- Loop อนันต์

คำสั่ง break;

- คำสั่ง break; เป็นคำสั่งที่ใช้หยุด loop (หยุดคือหยุดทันที ไม่สนใจเงื่อนไขหรือการกระทำใดๆต่อจากนั้นใน loop และจะเด้งออกมาบรรทัดต่อไปที่อยู่นอก loop เลย)
- ทำให้เราสามารถหยุด loop ในช่วงกลางของตัว loop ได้
- การใช้งาน ก็เพียงแค่พิมพ์ break; ไว้ในจุดที่ต้องการจะหยุด loop เท่านั้น

คำสั่ง break;

- ตัวอย่างแสดงการทำงานของ break; จากโจทย์ต้นชั่วโมง

```c
#include <stdio.h>

void main()
{
    int cnt = 1;
    while(cnt <= 3000){
        printf("I love you %d\n", cnt);
        cnt++;
    }
}
```

```
I love you 2995
I love you 2996
I love you 2997
I love you 2998
I love you 2999
I love you 3000
```

```c
#include <stdio.h>

void main()
{
    int cnt = 1;
    while(cnt <= 3000){
        printf("I love you %d\n", cnt);
        cnt++;
        break;
    }
}
```

```
I love you 1


...Program finished with exit code 13
Press ENTER to exit console.
```

คำสั่ง break;

- ถ้าเราวาง break; ไว้ใน loop ตรง ๆ loop นั้นก็จะไม่มีการวนซ้ำเป็นรอบที่สอง ซึ่งเราคงไม่ปรารถนาแบบนั้น เพราะที่ต้องการใช้ loop ก็เพื่อให้มันวนทำงานซ้ำมากกว่าหนึ่งรอบ
- แต่คำสั่ง break; มักถูกประยุกต์ใช้กับ if-else เพื่อสร้างเงื่อนไขบางอย่างให้ทำการออกจาก loop

คำสั่ง break;

ตัวอย่างโจทย์ 6 ให้รับค่าจำนวนเต็มเข้าทางคีย์บอร์ดไปเรื่อย ๆ จนผู้ใช้ป้อนค่า 0 จากนั้นแสดงผลรวมของค่าทั้งหมดที่ผู้ใช้ป้อนเข้ามา

ข้อนี้จะใช้ break หรือไม่ใช้ก็ได้ ให้นักศึกษาลองคิดก่อนว่าจะเขียนโปรแกรมอย่างไร

คำสั่ง break;

ตัวอย่างโจทย์ 6 ให้รับค่าจำนวนเต็มเข้าทางคีย์บอร์ดไปเรื่อย ๆ จนกว่าผู้ใช้จะป้อนค่า 0 จึงหยุดรับค่าต่อ จากนั้นแสดงผลรวมของค่าทั้งหมดที่ผู้ใช้ป้อนเข้ามา

โค้ดแบบใช้ break;

```c
#include <stdio.h>

void main()
{
    int input=1, sum = 0;
    while(1){
        scanf("%d", &input);
        if(input==0) break;
        sum = sum + input;
    }
    printf("%d", sum);
}
```

คำสั่ง break;

<u>ตัวอย่างโจทย์ 6</u> ให้รับค่าจำนวนเต็มเข้าทางคีย์บอร์ดไปเรื่อย ๆ จนกว่าผู้ใช้จะป้อนค่า 0 จึงหยุดรับค่าต่อ จากนั้นแสดงผลรวมของค่าทั้งหมดที่ผู้ใช้ป้อนเข้ามา

โค้ดแบบใช้การกำหนดเงื่อนไข

```c
#include <stdio.h>

void main()
{
    int input=1, sum = 0;
    while(input!=0){
        scanf("%d", &input);
        sum = sum + input;
    }
    printf("%d", sum);
}
```

คำสั่ง break;

<u>ตัวอย่างโจทย์ 7</u> จงเขียนโปรแกรมที่รับจำนวนเต็มบวกจากผู้ใช้ได้มากถึง 10 จำนวนและหาผลบวกของเลข 10 จำนวนดังกล่าว แต่หากผู้ใช้ใส่เลขศูนย์หรือติดลบเข้ามาโปรแกรมจะไม่นำค่าดังกล่าวไปบวกกับตัวเลขอื่น ๆ ก่อนหน้า นอกจากนี้โปรแกรมจะหยุดรับค่าจากผู้ใช้ และก่อนจบโปรแกรมจะพิมพ์ข้อความว่า Error แต่หากผู้ใช้ใส่จำนวนเต็มบวกมาทั้ง 10 จำนวน โปรแกรมจะพิมพ์ผลบวกของเลขทั้ง 10 ออกมา

คำสั่ง break;

โค้ดสำหรับตัวอย่างที่ 7 แบบที่ 1 ตรวจค่าความเป็นบวกของเลขสุดท้าย

ถ้า x <= 0 จะเข้าคำสั่ง break; ทำให้ loop จบลง

ถ้าโดน break; loop จะจบลงในขณะที่ x ยังคง <= 0

ถ้าไม่โดน break; loop จะทำงานจนจบ และ x > 0

```c
#include <stdio.h>

void main()
{
    int x, sum=0, i=0;
    while(i < 10) {
        scanf("%d", &x);
        if(x <= 0)
            break;
        sum = sum + x;
        i++;
    }
    if(x <= 0) {
        printf("Error");
    } else {
        printf("%d", sum);
    }
}
```

คำสั่ง break;

โค้ดสำหรับตัวอย่างที่ 7 แบบที่ 2 นับรอบ

ถ้า x <= 0 จะเข้าคำสั่ง break; ทำให้ loop จบลง

คำสั่ง i++; จะทำงานไม่ครบ 10 ครั้ง ค่า i จึงน้อยกว่า 10

ถ้าโดน break; loop จะจบลงในขณะที่ i < 0

ถ้าไม่โดน break; loop จะทำงานจนจบ และ i == 10

```c
#include <stdio.h>

void main()
{
    int x, sum=0, i=0;
    while(i < 10) {
        scanf("%d", &x);
        if(x <= 0)
            break;
        sum = sum + x;
        i++;
    }
    if(i < 10) {
        printf("Error");
    } else {
        printf("%d", sum);
    }
}
```

คำสั่ง break;

โค้ดสำหรับตัวอย่างที่ 7 แบบที่ 3 ใช้เงื่อนไข

ถ้า x > 0 จะรับค่าต่อไปตามปกติ

แต่ถ้า x ไม่มากกว่า 0 โปรแกรมจะปรับค่า i เป็น 10 ซึ่งทำให้หลุดออกจากเงื่อนไขของ loop ทันที

เมื่อหลุดออกจาก loop แบบไม่สมบูรณ์ ค่า x สุดท้ายที่รับมาจะติดลบหรือเป็น 0

แต่ถ้า loop สมบูรณ์ x จะเป็นค่าบวก ก็แสดงผลรวมตามที่โจทย์ต้องการ

```c
#include <stdio.h>

void main()
{
    int x, sum=0, i=0;
    while(i < 10) {
        scanf("%d", &x);
        if(x > 0) {
            i++;
            sum = sum + x;
        } else {
            i = 10;
        }
    }
    if(x <= 0) {
        printf("Error");
    } else {
        printf("%d", sum);
    }
}
```

คำสั่ง break;

โค้ดสำหรับตัวอย่างที่ 7 แบบที่ 4 ใช้เงื่อนไข

ใช้เงื่อนไขที่ซับซ้อนขึ้น โดยตรวจสอบทั้งค่ารับเข้าและรอบ

ถ้ารับค่า <= 0 เข้ามาจะทำให้ขัดแย้งกับเงื่อนไข loop และออกจาก loop ทันที

เมื่อหลุดออกจาก loop แบบไม่สมบูรณ์ จำนวนรอบสุดท้ายจะไม่ใช่ 10 (รันไม่ครบ)

แต่ถ้า loop สมบูรณ์ I จะเป็น 10 ก็แสดงผลรวมตามที่โจทย์ต้องการ

```c
#include <stdio.h>

void main()
{
    int x, sum=0, i=0;
    while(x > 0 && i < 10) {
        sum = sum + x;
        i++;
        if(i < 10)
            scanf("%d", &x);
    }
    if(i < 10) {
        printf("Error");
    } else {
        printf("%d", sum);
    }
}
```

Outline

- พื้นฐานของ Loop
- รูปแบบของ Loop และแนวคิด
- คำสั่งควบคุม While
- คำสั่ง break;
- **คำสั่ง continue;**
- Loop อนันต์

คำสั่ง Continue

- ปกติแล้ว การทำงานของ loop เมื่อถึงจุดท้ายสุดของชุดคำสั่งแล้วจะมีการวกกลับไปตรวจสอบเงื่อนไขเพื่อทำซ้ำ ต่อเนื่องไป
- แต่บางกรณี เราไม่ต้องการรอให้ถึงจุดท้ายสุดก่อน แต่อยากให้มีการวกกลับไปตรวจสอบเงื่อนไขใหม่และทำซ้ำในรอบใหม่ทันที
- กรณีนี้ เราจะใช้คำสั่ง continue;
- ถ้าเทียบกับ break; แล้ว ก็เรียกได้ว่าเป็นคำสั่งคนละขั้วกัน
  - break; จะออกจาก loop ไปเริ่มคำสั่งถัดไปทันที
  - continue; จะวกกลับไปเริ่มต้น loop เพื่อเช็คเงื่อนไขและทำซ้ำอีกครั้ง

คำสั่ง Continue

- continue; มักจะต้องอยู่ใน if-else ด้วยเหตุผลเดียวกันกับ break;
- ถ้า continue; อยู่นอก if-else จะทำให้คำสั่งอื่น ๆ ใน loop ที่อยู่ถัดจาก continue; ลงไป ไม่มีโอกาสได้ถูกเรียกใช้
- ทั้งคำสั่ง break; และ continue; ไม่ได้จัดเป็นคำสั่งที่จำเป็น กล่าวคืออาจไม่ต้องใช้ก็ได้ แต่การรู้ไว้อาจทำให้เราได้แนวทางการเขียนโปรแกรมที่เปิดกว้างมากยิ่งขึ้น

คำสั่ง Continue

<u>ตัวอย่างโจทย์ 8</u> จงเขียนโปรแกรมที่รับค่าจำนวนเต็มจากผู้ใช้มา 10 จำนวน หากจำนวนเต็มนั้นหารด้วย 5 ลงตัว โปรแกรมจะไม่พิมพ์ข้อความใด ๆ ออกมา และวนกลับไปเตรียมรับตัวเลขตัวต่อไปจากผู้ใช้ แต่หากไม่เป็นเช่นนั้น โปรแกรมจะ พิมพ์คำว่า Accept และนับจำนวนตัวเลขแบบนี้ว่ามีกี่ตัว สุดท้ายเมื่อผู้ใช้ใส่เลข ครบสิบตัวโปรแกรมจะพิมพ์จำนวนครั้งที่โปรแกรมแสดงคำว่า Accept ออกมา และจบการทำงาน

คำสั่ง Continue

โค้ดสำหรับตัวอย่างที่ 8 แบบที่ 1 ใช้ continue;

เมื่อโปรแกรมทำงานมาถึงจุดนี้โปรแกรมจะ ตีกลับไปที่ด้านบนของลูป และตรวจ เงื่อนไขของลูปอีกครั้ง

คำสั่งอัพเดทตัวแปรเงื่อนไข ไม่ได้จำเป็นว่าต้องมีที่เดียวใน loop เท่านั้น และไม่จำเป็นต้องเพิ่มทีละ 1 ก็ได้

```c
#include <stdio.h>

void main()
{
    int count = 0, i=0, x;
    while(i < 10) {
        scanf("%d", &x);
        if(x % 5 == 0) {
            i++;
            continue;
        }
        printf("Accept\n");
        count++;
        i++;
    }
    printf("%d", count);
}
```

คำสั่ง Continue

โค้ดสำหรับตัวอย่างที่ 8 แบบที่ 1+ ใช้ continue; และกระชับโค้ด

```c
#include <stdio.h>

void main()
{
    int count = 0, i=0, x;
    while(i < 10) {
        scanf("%d", &x);
        if(x % 5 == 0) {
            i++;
            continue;
        }
        printf("Accept\n");
        count++;
        i++;
    }
    printf("%d", count);
}
```

รวม i++; ทั้งสองตัวเข้าด้วยกันในตำแหน่งที่เหมาะกว่า

```c
#include <stdio.h>

void main()
{
    int count = 0, i=0, x;
    while(i < 10) {
        scanf("%d", &x);
        i++;
        if(x % 5 == 0) {
            continue;
        }
        printf("Accept\n");
        count++;
    }
    printf("%d", count);
}
```

อัพเดท i ตั้งแต่เนิ่น ๆ ก็ได้ โค้ดจะได้ไม่ยาว (จริงๆ แล้วไม่ต้องซีเรียสมากก็ได้ เอาที่ตัวเองถนัดไว้ก่อน)

คำสั่ง Continue

โค้ดสำหรับตัวอย่างที่ 8 แบบที่ 2 ไม่ใช้ continue; ใช้ if-else

```c
#include <stdio.h>

void main()
{
    int count = 0, i=0, x;
    while(i < 10) {
        scanf("%d", &x);
        i++;
        if(x % 5 != 0) {
            printf("Accept\n");
            count++;
        }
    }
    printf("%d", count);
}
```

Outline

- พื้นฐานของ Loop
- รูปแบบของ Loop และแนวคิด
- คำสั่งควบคุม While
- คำสั่ง break;
- คำสั่ง continue;
- **Loop อนันต์**

Loop อนันต์

- Loop อนันต์คือการวนซ้ำที่ไม่จำกัดจำนวนครั้ง
- โดยปรกติลูปจะวนทำงานไปเรื่อย ๆ จนกว่าเงื่อนไขลูปจะเป็นเท็จ
- ส่วนมากเงื่อนไขที่จะให้หยุดลูปเป็นสิ่งที่ระบุไว้ด้านบนของลูป
- ปัญหาบางอย่างไม่เหมาะที่จะคิดแบบนั้น เพราะตรรกะกระบวนการคิดจะซับซ้อนขึ้น เช่น เราต้องการให้โปรแกรมวนรับค่าไปเรื่อย ๆ ไม่จำกัดจนกว่าจะพบว่าผู้ใช้ใส่เลขศูนย์เข้ามาคำสั่งควบคุม While (ได้เกริ่นนำไปบ้างแล้วในตัวอย่างที่ 6)
- หัวข้อนี้จะมาเจาะลึกโจทย์แนวนี้กัน

Loop อนันต์

<u>ตัวอย่างโจทย์ 9</u> จงเขียนโปรแกรมที่รับค่าตัวเลขจำนวนเต็มจากผู้ใช้เข้ามาเรื่อย ๆ โปรแกรมจะทำการนับและบวกเลขที่เป็นบวก แต่หากผู้ใช้ใส่เลขที่เป็นลบหรือศูนย์เข้ามา โปรแกรมจะหยุดรับค่าจากผู้ใช้ แล้วพิมพ์จำนวนตัวเลขค่าบวกที่รับมาทั้งหมด รวมทั้งผลรวมของเลขบวกเหล่านี้

วิเคราะห์ ที่ผ่านมาเรามักจะหยุด loop เมื่อผู้ใช้ใส่ตัวเลขเข้ามาถึงจำนวนหนึ่ง แต่ในปัญหานี้ ผู้ใช้สามารถใส่ตัวเลขเข้ามาได้ไม่จำกัด ดังนั้นการตั้งเงื่อนไข loop โดยการจำกัดจำนวนครั้งไว้จึงเป็นเรื่องที่ผิด เพราะแท้จริงผู้ใช้จะใส่เลขเข้ามากี่ตัวก็ได้

Loop อนันต์

โค้ดสำหรับตัวอย่างที่ 9 วิธีที่ 1 แบบใช้เงื่อนไข

ถ้า input ไม่ใช่เลขติดลบหรือเลข 0 ก็จะยังคงอยู่ใน loop ต่อไป แต่ถ้าเป็นเลขติดลบหรือเลข 0 ก็จะหลุดออกจาก loop เนื่องจากไม่เข้าเงื่อนไข เป็นโค้ดที่ตรงไปตรงมา

```c
#include <stdio.h>

void main() {
    int count = 0, sum = 0, x = 1;
    while(x > 0) {
        scanf("%d", &x);
        if(x > 0) {
            count++;
            sum += x;
        }
    }
    printf("%d %d", count, sum);
}
```

Loop อนันต์

โค้ดสำหรับตัวอย่างที่ 9 วิธีที่ 1 แบบใช้เงื่อนไข

ยังจำได้ไหมว่าเลขอื่น ๆ ที่ไม่ใช่ 0 หมายถึง ‘จริง’ ดังนั้นการใส่เลข 1 ไว้ใน while จึงเป็นการสั่งให้วนซ้ำตลอดไป เพราะยังไงความจริงก็คือความจริง

แม้ว่าจะเป็น loop อนันต์ แต่ก็โดน break; ได้ เพราะคำสั่งนี้จะออกจาก loop โดยไม่สนใจเงื่อนไขนั่นเอง

```c
#include <stdio.h>

void main() {
int count = 0, sum = 0, x;
    while(1) {
        scanf("%d", &x);
        if(x <= 0) {
            break;
        }
        count++;
        sum += x;
    }
    printf("%d %d", count, sum);
}
```

Loop อนันต์

<u>แนวคิดที่ผิดที่พบเห็นได้บ่อย</u>

- ถ้าโจทย์บอกว่าไม่จำกัดครั้ง ก็คือไม่จำกัดครั้ง แต่จะเห็นได้ว่าหลายคนยังคงทำสิ่งที่ตัวเองคุ้นเคย เช่นการจำกัดจำนวนครั้ง (เช่น พยายามตั้งให้ i < 100) แบบนี้เป็นต้น ซึ่งก็ไม่ทราบสาเหตุว่าทำไมถึงต้องทำเช่นนั้น
- วิธีจัดการเงื่อนไขไม่ได้ถูกกำหนดไว้แบบเดียว ให้เลือกแบบที่ตัวเองถนัดและมั่นใจ แค่ทำให้ชำนาญสักทางก็พอ
- ต้องตระหนักถึงความสัมพันธ์ของตัวแปร อย่ายึดติดกับรูปแบบโค้ด วิธีการจัดการให้เก่งคือสร้างความชำนาญด้วยการฝึกฝน

ส่วนตัว ด้วยความเป็นห่วงแหละ <3

- ย้ำเป็นการส่วนตัวว่า วิชานี้เป็นพื้นฐานทั้งการเรียนและการทำงานของภาควิชาของเรา ดังนั้นการฝึกฝนจะไม่ได้เสียเปล่าแน่นอน และความชำนาญที่ได้ถูกสร้างขึ้นจะได้ใช้ไปตลอดจนถึงการทำงานด้านเทคโนโลยี

Loop อนันต์

ตัวอย่างโจทย์ 10 จงเขียนโปรแกรมที่รับค่าตัวเลขจำนวนเต็มจากผู้ใช้เข้ามาเรื่อย ๆ โปรแกรมจะพิมพ์คำว่า even หากตัวเลขเป็นคู่ แต่ถ้าผู้ใช้ใส่เลขคี่เข้ามาโปรแกรมจะพิมพ์เลข -1 ออกมา และถ้าผู้ใช้ใส่เลขคี่เข้ามาติดต่อกันสองตัวโปรแกรมจะจบการทำงานโดยไม่พิมพ์เลข -1 ออกมาสำหรับเลขคี่ตัวที่สอง (ถ้าใส่เลขคู่หลังเลขคี่จะเหมือนกับว่าไม่เคยใส่เลขคี่เข้ามา)

ตัวอย่าง

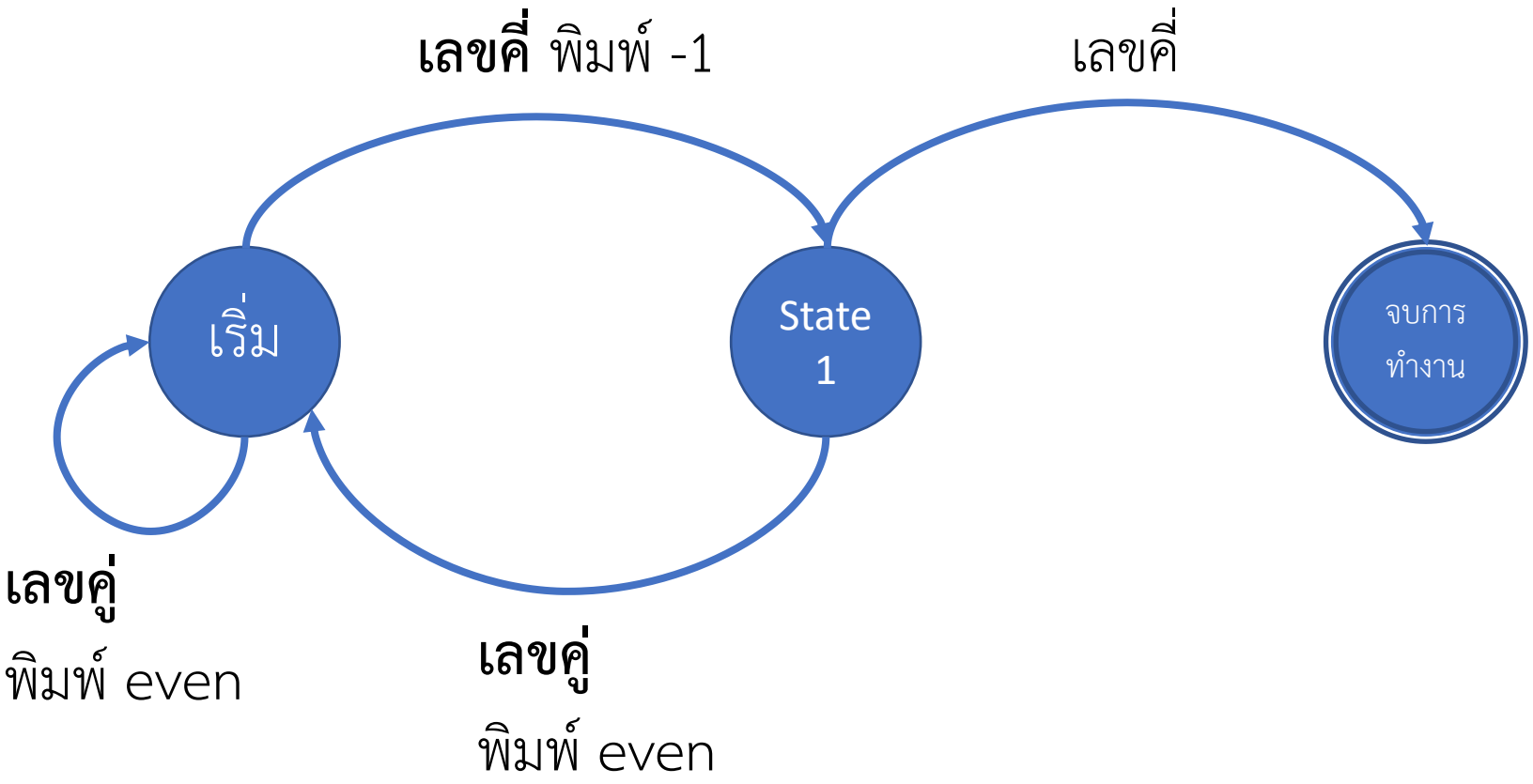
ข้อมูลเข้า	ผลลัพธ์
2 3 0 1 7	even -1 even -1

Loop อนันต์

<u>วิเคราะห์</u> เนื่องจากผลลัพธ์ที่เปลี่ยนไปตามผลที่เกิดขึ้นก่อนหน้า แบบนี้ต้องใช้ ตัวแปรเก็บสถานะเข้าช่วย ตัวแปรนี้ทำหน้าที่ระบุเหตุการณ์ที่เกิดขึ้นก่อนหน้า และมีความสำคัญกับผลลัพธ์ที่จะเกิดขึ้น ในที่นี้ก็คือการระบุว่าตัวเลขตัวที่แล้ว เป็นคี่หรือไม่ สำหรับปัญหานี้

1. ในตอนแรกสถานะคือ ‘เลขก่อนหน้าไม่เป็นคี่’
2. ถ้าผู้ใช้ใส่เลขคี่เข้ามาและ ‘เลขก่อนหน้าไม่เป็นคี่’ ก็ให้เราเปลี่ยนสถานะให้ เป็น ‘เลขก่อนหน้าเป็นคี่’ เราเปลี่ยนแบบนี้เพื่อเตรียมตัวสำหรับเลขถัดไป
3. ถ้าผู้ใช้ใส่เลขคี่เข้ามาและ ‘เลขก่อนหน้าเป็นคี่’ ก็ให้จบการทำงาน
4. ถ้าผู้ใช้ใส่เลขคู่เข้ามา ก็ให้กำหนดสถานะใหม่เป็น ‘เลขก่อนหน้าไม่เป็นคี่’ ย้ำอีกครั้งว่าการเปลี่ยนแปลงเป็นไปเพื่อการตรวจเลขถัดไป ไม่ใช่เลขตัวนี

Loop อนันต์


เลขคี่ พิมพ์ -1
เลขคี่
เริ่ม
State 1
จบการทำงาน
เลขคู่
พิมพ์ even
เลขคู่
พิมพ์ even

Loop อนันต์

โค้ดสำหรับตัวอย่างที่ 10 แบบที่ 1 แบบใช้ break;

ตัวแปรเก็บสถานะชื่อ odd
ถ้าเท่ากับ 0 แสดงว่าเลขก่อนหน้าไม่เป็นคี่
ถ้าเท่ากับ 1 แสดงว่าเลขก่อนหน้าเป็นคี่

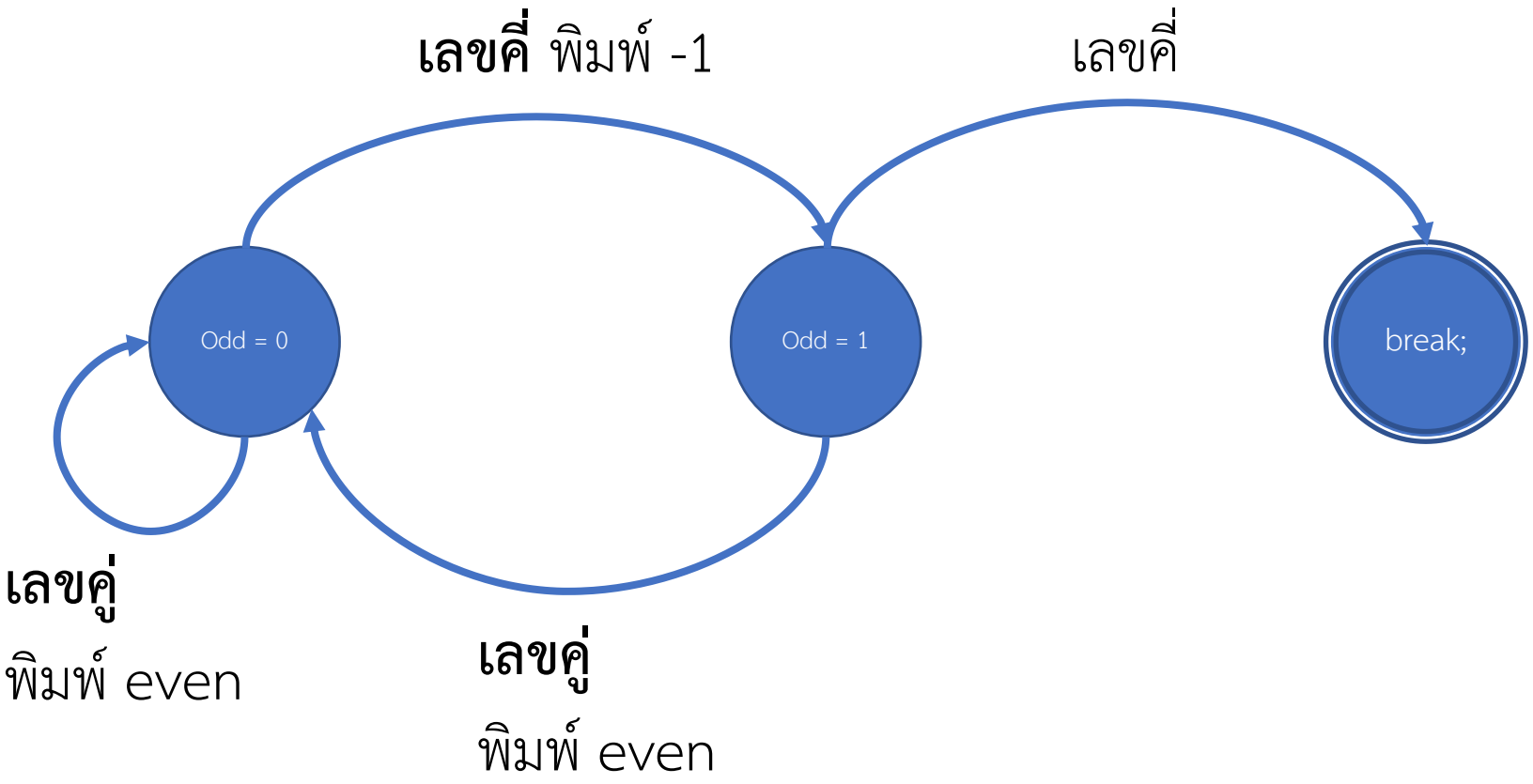
ถ้าเจอเลขคู่ให้ตั้งสถานะสำหรับรอบถัดไปว่าตัวเลขก่อนหน้าเป็นคู่

แต่เมื่อเป็นเลขคี่เราต้องตรวจดูว่าเลขก่อนหน้าเป็นคี่หรือเปล่า ถ้าใช่แสดงว่าเราเจอเลขคี่ติดกันแล้วแน่นอน break ได้เลย

```c
#include <stdio.h>

void main() {
    int x;
    int odd = 0;
    while(1) {
        scanf("%d", &x);
        if(x % 2 == 0) {
            printf("even\n");
            odd = 0;
        } else {
            if(odd == 1) {
                break;
            } else {
                printf("-1\n");
                odd = 1;
            }
        }
    }
}
```

Loop อนันต์



Loop อนันต์

โค้ดสำหรับตัวอย่างที่ 10 แบบที่ 1 แบบใช้ break;

ตัวแปรเก็บสถานะชื่อ odd
เป็นตัวบอกว่ามีเลขคี่ติดกันกี่ตัวแล้ว

เงื่อนไขคือเลขคี่จะติดกันได้ไม่ถึง 2 ตัว

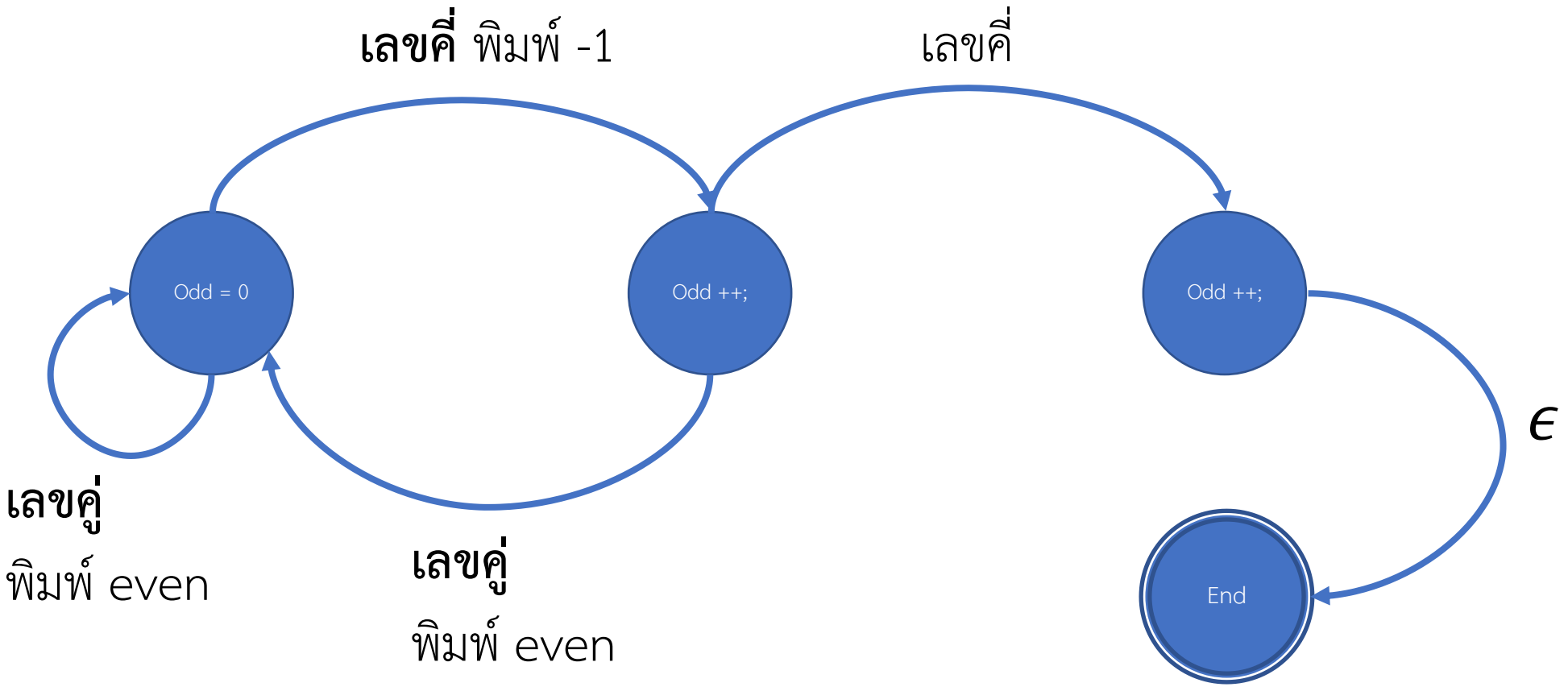
ถ้าเจอเลขคู่ ให้เริ่มนับใหม่

ถ้าเจอเลขคี่ ให้นับจำนวนเลขคี่ที่ติดกันเพิ่ม

```c
#include <stdio.h>

void main() {
    int x;
    int odd = 0;
    while(odd != 2) {
        scanf("%d", &x);
        if(x % 2 == 0) {
            printf("even\n", x);
            odd = 0;
        } else {
            odd++;
            if(odd == 1) {
                printf("-1\n");
            }
        }
    }
}
```

Loop อนันต์


เลขคี่ พิมพ์ -1
เลขคี่
Odd = 0
Odd ++;
Odd ++;
ϵ
เลขคู่
พิมพ์ even
เลขคู่
พิมพ์ even
End

สรุป

- พื้นฐานของ Loop
- รูปแบบของ Loop และแนวคิด
- คำสั่งควบคุม While
- คำสั่ง break;
- คำสั่ง continue;
- Loop อนันต์