การเขียนโปรแกรมคอมพิวเตอร์ 1

Computer Programming I

คำสั่งควบคุม: คำสั่งทำซ้ำด้วย 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 6

การทำซ้ำ

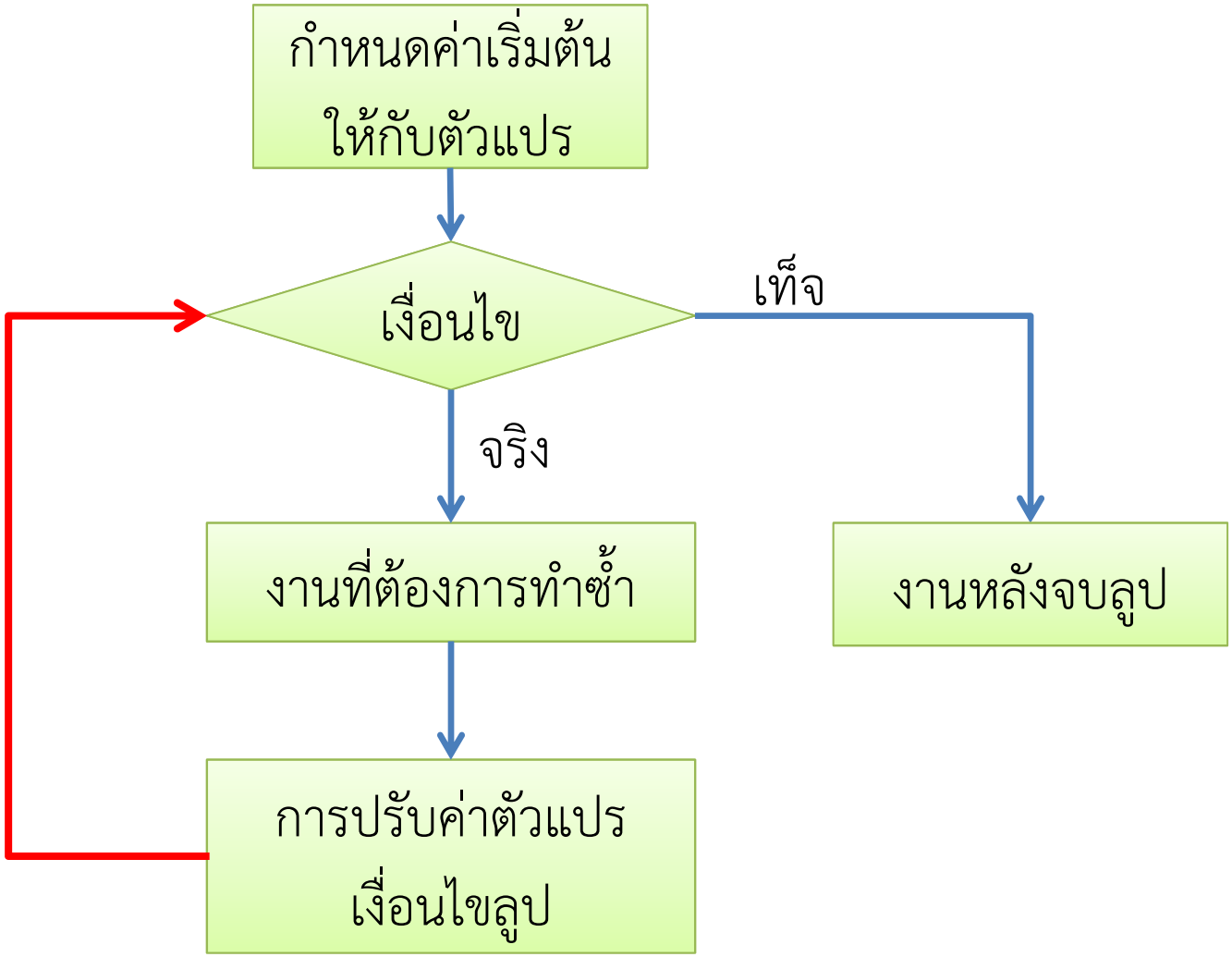
การทำซ้ำหรือการวนลูป แท้จริงแล้วไม่ใช่เรื่องยากเกินไปนัก หากเราเข้าใจองค์ประกอบพื้นฐานของการวนลูป ซึ่งมีของอยู่สี่อย่าง

1. การกำหนดค่าเริ่มต้นของตัวแปรต่าง ๆ ก่อนเข้าลูป
2. เงื่อนไขที่จะให้ทำลูป (จะอยู่ก่อนหรืออยู่ด้านหลังคำสั่งที่อยากทำก็ได้)
3. งานที่ต้องการทำซ้ำ
4. การปรับค่าตัวแปรเงื่อนไขลูป (คือการเปลี่ยนค่าตัวแปรที่เกี่ยวพันกับเงื่อนไขที่จะให้ทำหรือจบลูป)

การวนลูปในภาษาซี

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

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



รูปแบบทั่วไปของลูป while ( ) { ... }

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

- หลักการทำงานก็คือว่า ถ้าเงื่อนไขของลูปเป็นจริง โปรแกรมจะทำสิ่งที่อยู่ภายในลูป ซึ่งก็คือ ‘งานที่จะให้ทำ’ และ ‘การแก้ไขตัวแปรเงื่อนไขลูป’
- เป็นไปได้ที่เงื่อนไขของลูปจะไม่เป็นจริงตั้งแต่แรก ทำให้ไม่มีการทำงานใด ๆ ภายในลูปเลยแม้แต่ครั้งเดียว

ข้อควรจำเกี่ยวกับลูป while ( ) { ... }

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

- **<u>สิ่งใหนที่จะให้ทำซ้ำบ่อย ๆ ต้องอยู่ในลูป สิ่งไหนไม่ต้องทำซ้ำอยู่ข้างนอก</u>**
- สิ่งที่คนจำนวนมากลืมคิดก็คือเรื่องการกำหนดค่าตัวแปรก่อนเข้าลูป
- ถ้าลูปไม่มีการแก้ไขตัวแปรที่เกี่ยวกับเงื่อนไขลูปเลย เป็นไปได้มากว่าลูปจะวนไม่รู้จบ เพราะเงื่อนไขที่เป็นจริงในตอนแรกจะเป็นจริงต่อไปหากไม่มีการแก้ไข (นักศึกษาเข้าใจประเด็นนี้หรือไม่ ?)

ตัวอย่างการทำงานของลูป

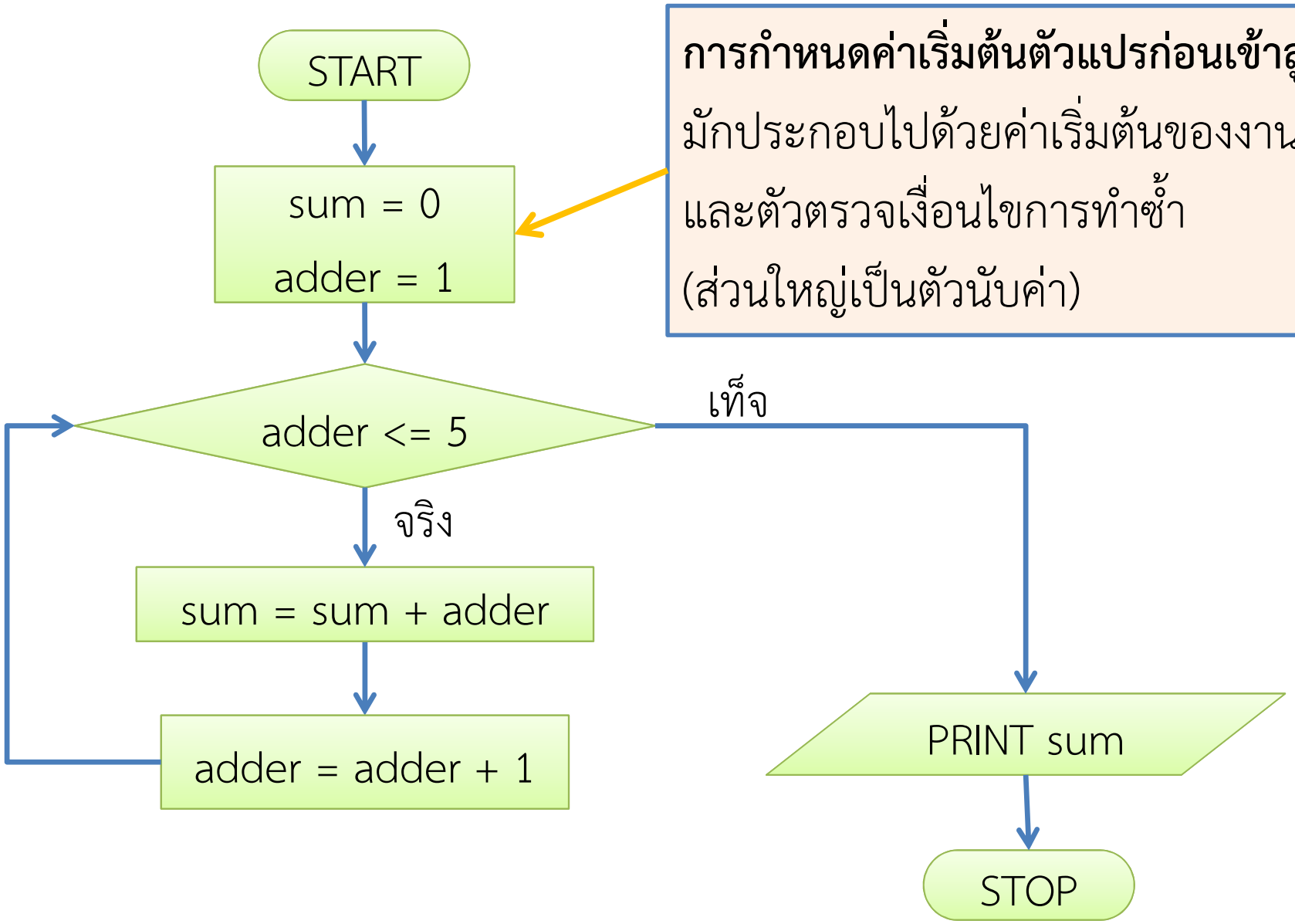
(ขอเริ่มจากตัวอย่างที่เข้าใจง่าย ความซับซ้อนจะค่อย ๆ เพิ่มขึ้นในแต่ละตัวอย่าง)

โจทย์ จงเขียนโฟลวชาร์ตและโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด 1 ถึง 5 (ช่วงปิดจะรวมเลข 1 และ 5 ด้วย) จากนั้นพิมพ์ผลลัพธ์ออกมาทางจอภาพ (บังคับให้ใช้ลูปค่อย ๆ บวกเลขทีละค่า)

วิเคราะห์

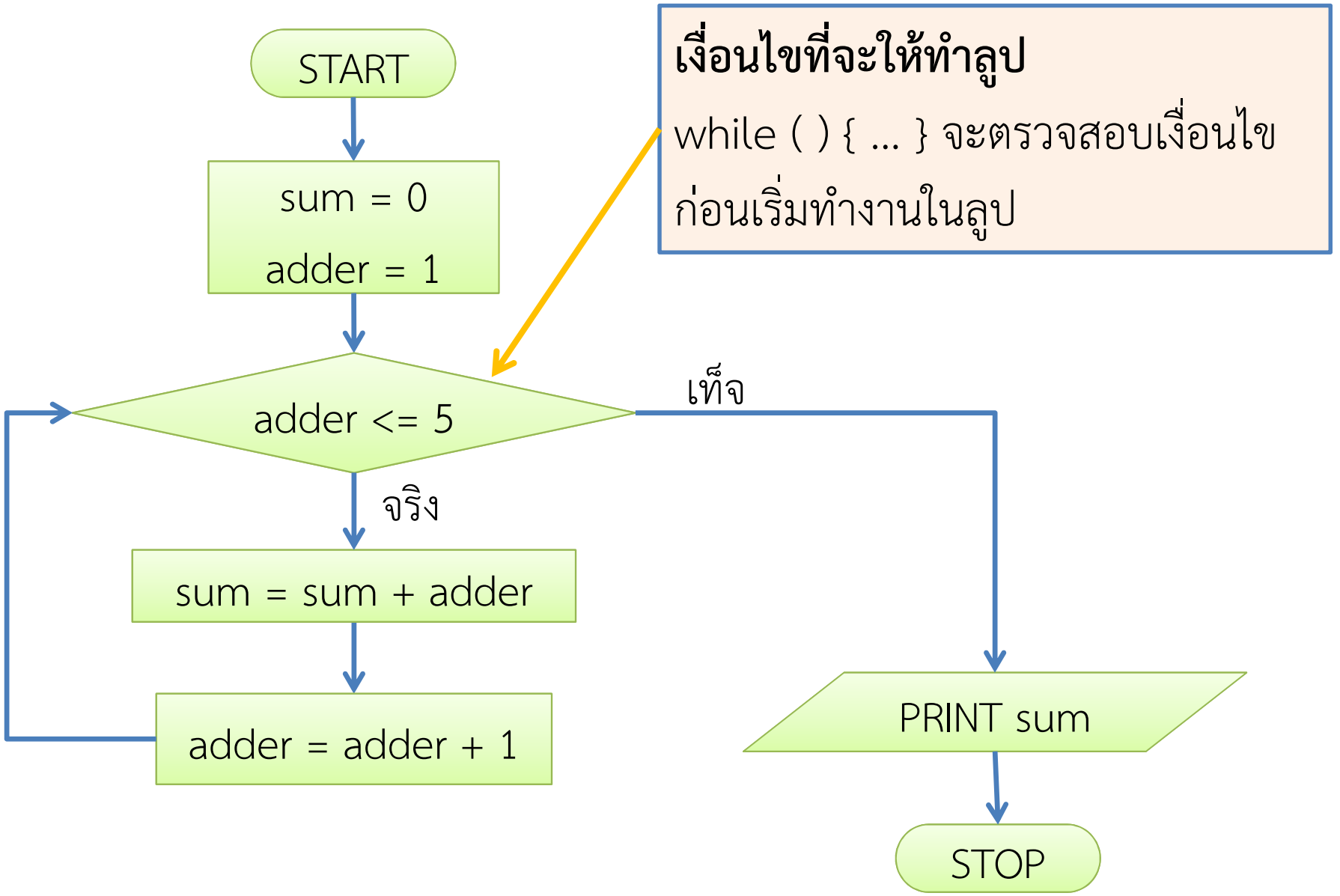
1. ไม่มีการรับข้อมูลเข้าจากผู้ใช้ แต่จะต้องสร้างตัวเลขขึ้นมาเอง
2. งานที่ต้อง**ทำซ้ำ**แน่ ๆ คือการบวกเลข
3. ต้องมีการนับเลขที่จะบวกเพิ่มขึ้นเรื่อย ๆ เพื่อให้เปลี่ยนตัวบวกจาก 1 ไปเป็น 2, 3, 4 และ 5 ได้
4. เงื่อนไขที่ควรใช้ในการทำงานคือ ‘ตัวบวกต้องอยู่ในช่วง 1 ถึง 5’

โฟลวชาร์ต การวนซ้ำบวกเลข

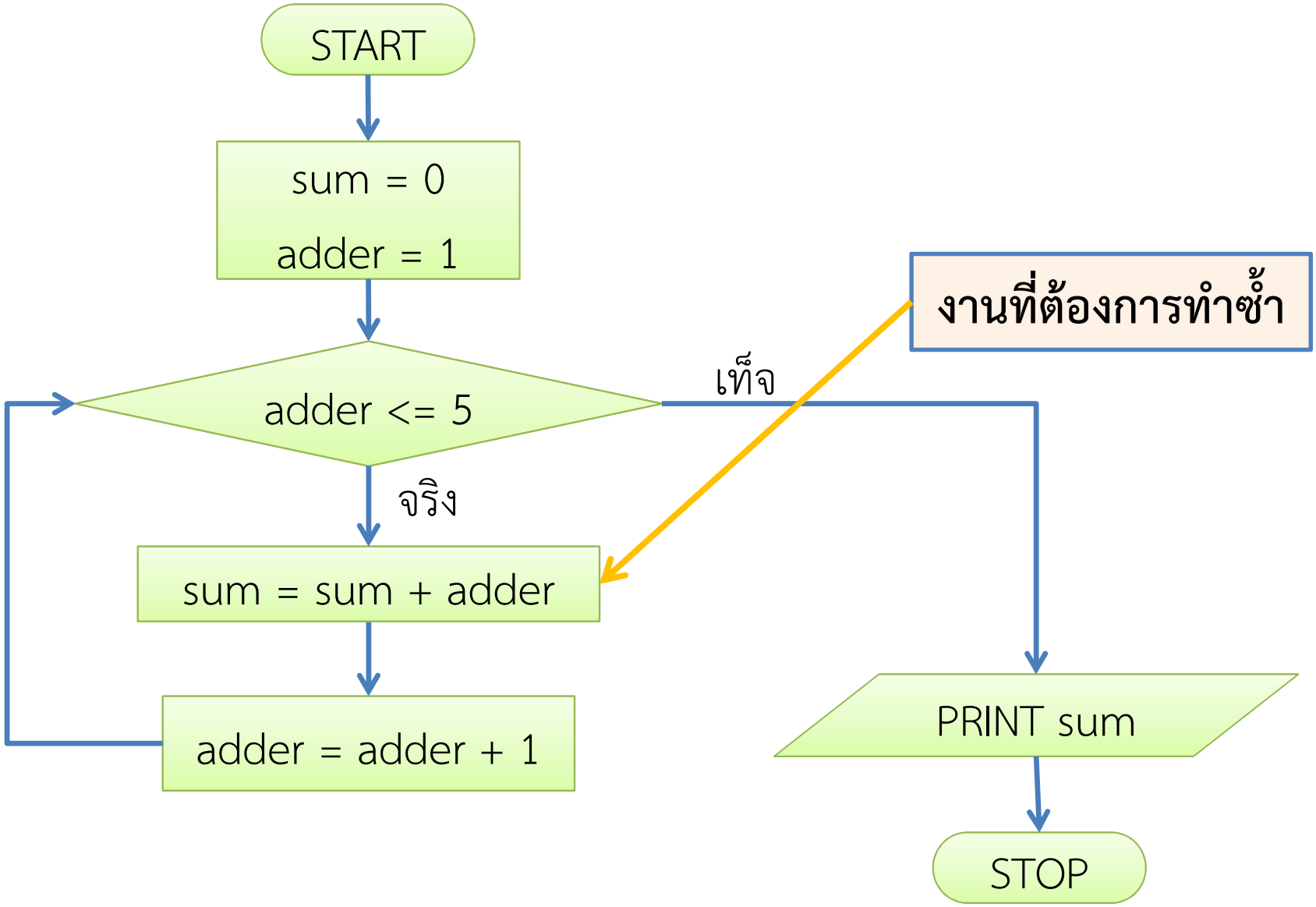


การกำหนดค่าเริ่มต้นตัวแปรก่อนเข้าลูป
มักประกอบไปด้วยค่าเริ่มต้นของงาน
และตัวตรวจเงื่อนไขการทำซ้ำ
(ส่วนใหญ่เป็นตัวนับค่า)

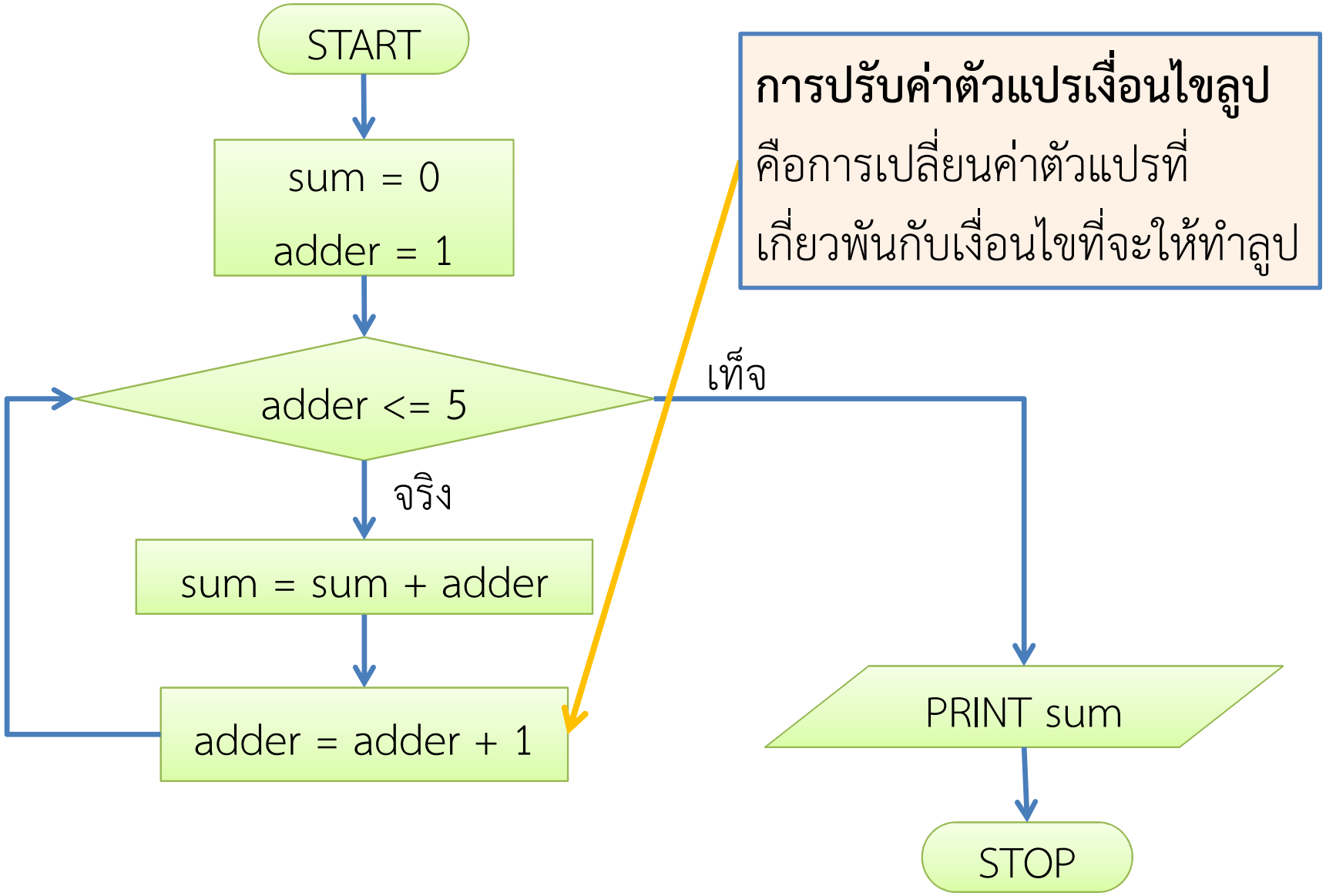
โฟลวชาร์ต การวนซ้ำบวกเลข (2)



โฟลวชาร์ต การวนซ้ำบวกเลข (3)



โฟลวชาร์ต การวนซ้ำบวกเลข (4)



โค้ดภาษาซี

```
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

แยกให้ออกด้วยว่า งานไหนที่ต้องทำซ้ำ งานไหนที่ไม่ต้องทำซ้ำ

ธรรมชาติของปัญหาเกี่ยวกับลูป

- ความยากของเรื่องลูปก็คือว่า ‘มันมีงานแฝงที่ต้องทำเพื่อให้ลูปมันทำตามวัตถุประสงค์ได้’ และงานแฝงนี่แหละที่เป็นสิ่งที่คนจำนวนมากคิดไม่ออก การจะคิดของพวกนี้ได้จะต้องผ่านการฝึกคิดด้วยตัวเองสักระยะหนึ่ง
- เราจะต้องเข้าใจความสัมพันธ์ระหว่างงานหลักที่จะให้ทำเป็นอันดับแรก
- จากนั้นงานหลักจะบอกเราได้ว่า มีอะไรที่โปรแกรมยังขาดไป และเราต้องหาทางเติมเต็มส่วนที่ขาดไปนั้น
- ในตอนแรกที่เรายังไม่มีประสบการณ์เราต้องใช้การสังเกตจากตัวอย่างและยืม ‘กระบวนท่า’ ต่าง ๆ มาประยุกต์ใช้กับปัญหาอื่น
  → จำเป็นต้องใช้ความคิดสร้างสรรค์ในการแก้ปัญหา เพราะปัญหามันไม่เหมือนกันแบบเป๊ะ ๆ เราต้องต้องรู้จักสังเกตและประยุกต์ใช้วิธีการ

สิ่งที่ได้เรียนรู้จากตัวอย่างบวกเลข

- งานนี้เราต้องสังเคราะห์ข้อมูลขึ้นมาสำหรับทำการบวก ตรงนี้เป็น*งานแฝง* ไม่ปรากฏในตัวปัญหาโดยตรง
- เงื่อนไขในการทำลูปที่บอกว่า adder <= 5 เป็นสิ่งที่ไม่มีการระบุไว้ในตัวปัญหาเลย แต่มันเป็นเงื่อนไขที่เกิดขึ้นมาเพื่อให้ทำวัตถุประสงค์หลักได้ → เงื่อนไขใช้ทำงานแฝง ซึ่งก็คือการสังเคราะห์ข้อมูลเข้าขึ้นมา
- เรามีการปรับค่าตัวแปรลูปคือ adder = adder + 1; เพื่อทำงานแฝง และยังให้ผลสอดคล้องกับการสังเคราะห์ค่าขึ้นมา
  แต่อย่าไปคิดว่าปัญหาทุกอย่างจะเป็นแบบนี้กันหมด ยังมีเทคนิคที่จำเป็นอย่างอื่นอีกมาก และปัญหาจำนวนมากไม่ได้สำเร็จรูปขนาดนี้
- การรู้ว่าอะไรคืองานแฝงและการสร้างเงื่อนไขลูปที่สอดคล้องกัน ต้องอาศัยทั้งความรู้ การวิเคราะห์ และทักษะประสบการณ์ → ไม่ง่าย

ตัวอย่าง : บวกเลขในช่วง x ถึง y (โดยที่ y > x)

โจทย์ จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด x ถึง y (ช่วงปิดจะรวมเลข x และ y ด้วย) โดยที่ค่า x และ y มาจากผู้ใช้ และ y > x สมมติว่าผู้ใช้ใส่ค่าที่สอดคล้องเงื่อนไขนี้ไม่มีพลาด เมื่อทำการบวกจนเสร็จแล้ว ให้พิมพ์ผลลัพธ์ออกมาทางจอภาพ

วิเคราะห์

หมายเหตุ เพื่อเรียนรู้พื้นฐานของลูป เราจะไม่ใช้สูตรบวกเลขอนุกรม

1. การบวกควรเริ่มจาก x, x + 1, ... ไปสิ้นสุดที่ y
2. ค่า x และ y นี้เป็นตัวกำหนดขอบเขตของตัวบวกและสามารถใช้เป็นเงื่อนไขของลูปในปัญหานี้ได้
3. ในเมื่อโจทย์กำหนดไว้แล้วว่าผู้ใช้ใส่ค่า y ที่มากกว่า x ตลอดเวลาเราไม่ต้องกังวลคอยตรวจค่าว่าจะผิด

โค้ดภาษาซี (1)

```
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);
}
```

การกำหนดค่าตัวแปรเริ่มต้นก่อนเข้าลูป

เงื่อนไขที่จะให้ทำลูป

งานที่ต้องการทำซ้ำ

การปรับค่าตัวแปรเงื่อนไขลูป

งานหลังจบลูป

โค้ดภาษาซี (2)

```
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);
}
```

การกำหนดค่าตัวแปรเริ่มต้นก่อนเข้าลูป

เงื่อนไขที่จะให้ทำลูป

งานที่ต้องการทำซ้ำ

การปรับค่าตัวแปรเงื่อนไขลูป

งานหลังจบลูป

ตัวอย่าง : บวกเลขในช่วง x ถึง y (แบบค่าอิสระ)

โจทย์ 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. แล้วเงื่อนไขลูปล่ะ? → ระหว่าง x >= adder >= y กับ y >= adder >= x จะใช้อันไหนดีเพราะไม่รู้ว่าตัวไหนมีค่ามากกว่า

จะสร้างเงื่อนไขลูปอย่างไรดี

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)

โค้ดภาษาซีสำหรับการบวกจาก x ถึง y (ค่าอิสระ)

```
...
int adder = x;
while((adder <= y && adder >= x) ||
      (adder <= x && adder >= y))
{
    sum = sum + adder;
    if(x <= y) {
        adder = adder + 1;
    } else {
        adder = adder - 1;
    }
}
```

การกำหนดค่าตัวแปรเริ่มต้นก่อนเข้าลูป

เงื่อนไขที่จะให้ทำลูป

งานที่ต้องการทำซ้ำ

การปรับค่าตัวแปรเงื่อนไขลูป

บทเรียนจากตัวอย่าง

- ไม่น่าเชื่อว่าโจทย์ที่ดูเหมือนไม่มีอะไรจะทำให้เราสะดุดได้ บางทีเงื่อนไขลูปมันจะซับซ้อนเป็นพิเศษ ถ้าเราเลือกทางไม่ค่อยเหมาะสม
- สังเกตในหน้าที่แล้ว งานที่ต้องการทำซ้ำมีนิดเดียว แต่ทั้งเงื่อนไขลูปและการปรับตัวแปรลูปมันหนักหน่วงมาก
  → นี่เป็นเหตุผลที่ทำให้มือใหม่รู้สึกเหมือนถูกหมัดขวาตรงเข้าที่หน้า
  → รู้สึกมึนงงจนจะยืนสองขาไม่ไหว อยากร้องไห้ ทั้งที่โจทย์ดูพื้น ๆ
- ตรรกะในการคิดเป็นสิ่งที่สำคัญมากในการแก้ปัญหา จำเป็นต้องฝึกจนชำนาญ
- อย่างไรก็ตามวิธีคิดที่ดีกว่า จะนำเราไปสู่เป้าหมายได้เหมือนกับว่าเราใช้ทางลัดมุ่งหน้าสู่เส้นชัย → ดูวิธีคิดอีกวิธีแล้วจะรู้ซึ้ง

ตัวอย่าง : บวกเลขในช่วง x ถึง y (แบบค่าอิสระ)

โจทย์ จงเขียนโค้ดภาษาซีสำหรับการหาผลบวกของเลขจำนวนเต็มที่มีค่าอยู่ในช่วงปิด x ถึง y โดยที่ค่า x และ y มาจากผู้ใช้ เมื่อทำการบวกจนเสร็จแล้วให้พิมพ์ผลลัพธ์ออกมาทางจอภาพ

วิเคราะห์

1. ปัญหานี้ไม่ได้ระบุว่า y > x หรือ x > y และที่จริง x จะเท่ากับ y ก็ได้
2. เห็นได้ว่าตอนที่เรากำหนดให้ค่า y > x ตลอดทุกอย่างมันดูง่าย ดังนั้นเราจะใช้วิธีคิดแบบเดิมเข้าช่วย คือถ้าเราพบว่า x > y เราจะทำการสลับค่า x และ y เช่นถ้า เราพบว่า x = 5 และ y = 1 เราจะสลับค่าให้ x = 1 และ y = 5 ส่งผลให้ y > x อย่างที่เป็นกับปัญหาดั้งเดิม
3. เหลือแค่ว่าจะสลับค่า x กับ y ได้อย่างไร

เทคนิคสลับค่าตัวแปร

- เป็นเทคนิคพื้นฐานที่โปรแกรมเมอร์ทุกคนใช้หากินได้ตลอด ต้องเรียนรู้ไว้
- อุปสรรคของการสลับค่าก็คือ การเขียนว่า

  x = y;

  y = x;

  ดูเหมือนเป็นการสลับค่า แต่ที่จริงแล้วไม่ได้ทำให้เกิดการสลับค่า

  (ดูออกหรือไม่ว่าทำไม? ลองแทนค่าเริ่มต้นให้ x = 1 และ y = 2 แล้วจะรู้)
- หัวใจของเทคนิคคือ **การสร้างตัวแปรชั่วคราว** ซึ่งนำไปสู่โค้ดยอดนิยมที่ว่า

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

ประยุกต์ใช้เทคนิคสลับค่าในการแก้ปัญหา

```
scanf("%d %d", &x, &y);
if(x > y) {
    int temp = x;
    x = y;
    y = temp;
}
```

ตรวจค่า x และ y จากนั้นทำการสลับค่าถ้าจำเป็น

```
int adder = x;
while(adder <= y) {
    sum = sum + adder;
    adder = adder + 1;
}
```

ขอแค่เข้าใจการสลับค่าตัวแปร เราจะสามารถใช้วิธีเดิม ๆ มาแก้ปัญหาได้

บทเรียนจากวิธีคิดนี้

- การเลือกวิธีที่เหมาะสมสามารถลดความซับซ้อนของโค้ดเราได้
- ความคิดสร้างสรรค์และประสบการณ์จึงเป็นของที่มีค่ามากในการแก้ปัญหา
- การจะทำเรื่องแบบนี้ได้ ผู้เขียนโปรแกรมจะต้องพร้อมด้วยความรู้ ความสามารถในการวิเคราะห์ปัญหา ทักษะ และ ประสบการณ์
- สังเกตด้วยว่าโจทย์ตัวอย่างนี้ไม่ได้ดูยากเย็นอะไร แต่มันก็จำเป็นต้องใช้เทคนิคที่ดี หรือไม่ก็ต้องใช้วิธียาก ๆ และตัววิธียาก ๆ เองจะคิดออกได้ก็ต้องมีความเข้าใจในตรรกศาสตร์ที่ดีมาก

ปัญหาในโปรแกรมทั่วไปรวมทั้งโปรเจ็คสำหรับจบการศึกษายากกว่านี้มาก นักศึกษาจำเป็นต้องเตรียมความพร้อมไว้ให้ดี

คำสั่งหยุดลูป (break)

บางครั้งเราต้องการออกจากลูปจากช่วงตรงกลาง แทนที่จะการออกจากลูปจากการตรวจเงื่อนไขตอนต้นลูป เช่น หากเราต้องการให้ผู้ใช้ใส่เลขจำนวนเต็มบวกมา 10 ค่าเพื่อให้โปรแกรมหาผลบวกของค่าทั้ง 10 แต่ถ้าผู้ใช้เผลอใส่เลขศูนย์หรือติดลบมา ถือว่าผิดพลาด และโปรแกรมจะหยุดรับค่าทันที

- จากตัวอย่างข้างต้นแสดงว่าเงื่อนไขที่จะหยุดลูปมีสองอย่างคือ (1) ผู้ใช้ใส่ค่าครบ 10 จำนวน และ (2) ผู้ใช้ใส่เลขศูนย์หรือค่าติดลบมา
- เงื่อนไขทั้งสองเป็นอิสระจากกัน ไม่ควรเอามาคิดรวมกันตรงต้นลูปพร้อมกัน
- เงื่อนไขที่เป็นอิสระแบบนี้ถ้าจะนำมารวมกันมันจะซับซ้อนมาก
  → การใช้คำสั่ง break; เพื่อหยุดลูปจะทำให้ตรรกะในการคำนวณง่ายขึ้น
- คำสั่ง break; จะทำให้ลูปที่มันอยู่ข้างในจบการทำงานทันที

การใช้คำสั่ง break;

- คำสั่ง break; โดยตัวของมันเองเป็นการออกจากลูปอย่างไม่มีเงื่อนไข
- ถ้าเราใช้มันตรง ๆ มันก็จะหยุดลูปทุกครั้งไป ลูปจะไม่มีการวนซ้ำเพราะโดนคำสั่ง break; สั่งหยุดทำงานทุกครั้ง เช่น

```
while (i < 10)  {
    scanf("%d", &x);
    break;
    sum = sum + x;
    i++;
}
```

ถ้าทำแบบนี้พอเจอคำสั่ง break ลูปจะหยุดทันที คำสั่งหลัง break จะไม่มีโอกาสได้ทำงานเลย

จะเห็นได้ว่าคำสั่ง break; อยู่ในลูปโดยไม่อยู่ใต้เงื่อนไขของ if ดังนั้นโปรแกรมนี้หลังจากรับค่าจากผู้ใช้มาเก็บไว้ ก็จะออกจากลูปทันที ไม่มีการวนทำซ้ำ

การทำให้คำสั่ง break; มีประโยชน์

ดังนั้นเราจึงใช้มันคู่กับเงื่อนไข if เช่นจากตัวอย่างที่ป้องกันไม่ให้ผู้ใช้ใส่เลขศูนย์และค่าติดลบ

- ถ้าข้อมูลเข้าจากผู้ใช้คือ x เราก็จะได้ตรรกะของ if ที่ควรจะเป็นคือ ‘ถ้า x <= 0 ให้โปรแกรมออกจากลูป’ และได้ลูปเป็น

แบบที่ถูกต้องตามข้อกำหนด

```
while (i < 10) {
  scanf(“%d”, &x);
  if (x <= 0){
    break;
  }
  sum = sum + x;
  i++;
}
```

แบบเดิมที่ใช้ไม่ได้

```
while (i < 10) {
  scanf(“%d”, &x);
  if (x <= 0){
    break;
  }
  sum = sum + x;
  i++;
}
```

ตัวอย่างการใช้ break;

โจทย์ จงเขียนโปรแกรมที่รับจำนวนเต็มบวกจากผู้ใช้ได้มากถึง 10 จำนวน และหาผลบวกของเลข 10 จำนวนดังกล่าว แต่หากผู้ใช้ใส่เลขศูนย์หรือติดลบเข้ามาโปรแกรมจะไม่นำค่าดังกล่าวไปบวกกับตัวเลขอื่น ๆ ก่อนหน้า นอกจากนี้โปรแกรมจะหยุดรับค่าจากผู้ใช้ และก่อนจบโปรแกรมจะพิมพ์ข้อความว่า Error แต่หากผู้ใช้ใส่จำนวนเต็มบวกมาทั้ง 10 จำนวน โปรแกรมจะพิมพ์ผลบวกของเลขทั้ง 10 ออกมา

(เช่นเดิม โจทย์ข้อนี้ดูเหมือนไม่มีอะไร แต่มือใหม่ต้องใช้เวลาคิดนานพอสมควร)

วิเคราะห์ปัญหาการหยุดลูป

- จุดยากของปัญหาอยู่ที่ว่าทำอย่างไรตอนที่โปรแกรมออกจากลูปแล้วจะแยกได้ว่าจะพิมพ์ผลบวกหรือคำว่า Error ดี
- วิธีที่ได้ผลดีในข้อนี้คือให้ตรวจว่าค่าตัวเลขที่ได้จากผู้ใช้อันล่าสุดคือค่าบวกหรือว่าเป็นอย่างอื่น
- วิธีอีกอันหนึ่งที่ได้ผลดีก็คือการตรวจว่าลูปวนไปจนครบสมบูรณ์กี่รอบ ถ้าครบสมบูรณ์ดีทั้ง 10 รอบก็แสดงว่าเราควรแสดงผลบวกออกมา

วิธีตรวจความเป็นค่าบวกของเลขสุดท้าย

```
int x;
int sum = 0;
int 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);
}
```

ถ้าถูก break ตรงนี้แสดงว่าลูปจะจบลง โดยที่ค่า x ไม่เป็นบวก

ดังนั้นถ้าเราตรวจตรงนี้ได้ว่า x เป็นศูนย์ หรือติดลบก็สรุปได้เลยว่าผู้ใช้ใส่ค่าผิด

วิธีตรวจจำนวนรอบที่สมบูรณ์

```
int x;
int sum = 0;
int 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);
}
```

ถ้าถูก break ตรงนี้แสดงว่าลูปจะจบลงก่อนได้ทำ i++ ทางด้านใต้ ส่งผลให้ i < 10 ในขณะที่การจบลูปแบบปรกติจะได้ค่า i == 10

ดังนั้นถ้าเราตรวจตรงนี้ได้ว่า x น้อยกว่าสิบ ก็สรุปได้เลยว่าผู้ใช้ใส่ค่าผิด

แล้วถ้าไม่อยากใช้คำสั่ง break เลยล่ะ

```
int x;
int sum = 0;
int 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);
}
```

ทำได้เหมือนกัน มีมากกว่าหนึ่งวิธี แต่ว่าแต่ละวิธีก็จะดูประหลาดในความรู้สึกของหลาย ๆ คน

ถ้าค่า x ผิดเราจะใช้การเพิ่มค่า i เพื่อให้หลุดออกจากลูปทางด้านบน

ด้วยวิธีข้างบนเราต้องตรวจ x <= 0 เราใช้การตรวจค่า i แบบตัวอย่างที่แล้วไม่ได้

แบบไม่ใช้ break อันที่สอง

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

การใช้ scanf สองที่และเงื่อนไข i < 10 อันที่สองทำให้วิธีการดูขัดตาพอสมควร

ถ้าใช้วิธียุ่ง ๆ นี้ มันจะทำให้เราตรวจเงื่อนไขด้านใต้นี้ด้วยวิธีไหนก็ได้ จะใช้ x <= 0 หรือ i < 10 ก็ได้

ใช้ลูป do while แทน (เรียนสัปดาห์หน้า)

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

วิธีนี้ดูเป็นธรรมชาติที่สุด แต่หลายคนอาจจะไม่ชอบ

ใช้ลูป do while ทำให้คิดวิธีนี้ไม่ออก

คำสั่งวกกลับไปต้นลูป (continue)

- บางครั้งจุดที่เราอยากให้โปรแกรมวกกลับไปต้นลูปอาจจะไม่ใช่แค่ตรงด้านท้ายของลูปเท่านั้น เราอาจจะอยากให้มีการวกกลับที่จุดอื่น ๆ ด้วย
- แม้จะเป็นไปได้ที่เราจะแก้ปัญหานี้ผ่านการใช้ if-else ที่ซับซ้อนขึ้น แต่การใช้คำสั่ง continue; เพื่อสั่งให้โปรแกรมวกกลับไปด้านบนของลูปจะทำให้ตรรกะในการคิดดูง่ายขึ้น
- คำสั่ง continue; ไม่ใช่สิ่งที่จำเป็นอย่างยิ่งยวด แต่มันทำให้เรามีอิสระในการวางแผนการคิดในการเขียนโปรแกรมมากขึ้น จึงควรเรียนรู้ไว้
- เช่นเดียวกับ break; การใช้ continue; ที่มีประโยชน์ ต้องใช้คู่กับเงื่อนไขของ if ไม่เช่นนั้นลูปจะวนกลับไปด้านบนทุกครั้ง ไม่มีทางไปถึงคำสั่งที่อยู่หลังจากมัน

ตัวอย่างการใช้ continue;

โจทย์ จงเขียนโปรแกรมที่รับค่าจำนวนเต็มจากผู้ใช้มา 10 จำนวน หากจำนวนเต็มนั้นหารด้วย 5 ลงตัว โปรแกรมจะไม่พิมพ์ข้อความใด ๆ ออกมา และวนกลับไปเตรียมรับตัวเลขตัวต่อไปจากผู้ใช้ แต่หากไม่เป็นเช่นนั้น โปรแกรมจะพิมพ์คำว่า Accept และนับจำนวนตัวเลขแบบนี้ว่ามีกี่ตัว สุดท้ายเมื่อผู้ใช้ใส่เลขครบสิบตัว โปรแกรมจะพิมพ์จำนวนครั้งที่โปรแกรมแสดงคำว่า Accept ออกมาและจบการทำงาน

วิเคราะห์ โปรแกรมมีการวนกลับกลางทาง เราสามารถใช้คำสั่ง continue; เพื่อให้โปรแกรมวนกลับไปตรวจเงื่อนไขของลูปด้วย

*** สิ่งที่คนจำนวนมากทำพลาดในการใช้ continue; กับ while loop ก็คือว่าลืมปรับค่าตัวแปรเงื่อนไขก่อนสั่งให้วนกลับไปที่เงื่อนไขลูป

โปรแกรมนับตัวเลขที่หาร 5 ไม่ลงตัว

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

เมื่อโปรแกรมทำงานมาถึงจุดนี้โปรแกรมจะตีกลับไปที่ด้านบนของลูป และตรวจเงื่อนไขของลูปอีกครั้ง

สังเกตดูให้ดีว่า เป็นไปได้ว่าโปรแกรมจะมีการปรับค่าตัวแปรเงื่อนไขสองที่ในลูปก็ได้

การลดความซ้ำซ้อนของลูป

โปรแกรมที่โค้ดมีความซ้ำซ้อนไม่ผิด แต่มันก็มักจะยาวขึ้นโดยไม่จำเป็น

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

แท้จริงแล้วการปรับตัวแปรเงื่อนไขลูปจะทำแต่เนิ่น ๆ ก็ได้ ตราบใดที่ผลลัพธ์ตรงตามวัตถุประสงค์ โปรแกรมก็ไม่ผิด

การวนลูปแบบไม่จำกัดจำนวนครั้ง

- โดยปรกติลูปจะวนทำงานไปเรื่อย ๆ จนกว่าเงื่อนไขลูปจะเป็นเท็จ
- ส่วนมากเงื่อนไขที่จะให้หยุดลูปเป็นสิ่งที่ระบุไว้ด้านบนของลูป
- ปัญหาบางอย่างไม่เหมาะที่จะคิดแบบนั้น เพราะตรรกะกระบวนการคิดจะซับซ้อนขึ้น เช่น เราต้องการให้โปรแกรมวนรับค่าไปเรื่อย ๆ ไม่จำกัด จนกว่าจะพบว่าผู้ใช้ใส่เลขศูนย์เข้ามา
  - เราอาจจะบอกว่าให้ใช้เงื่อนไขของลูปคือค่าที่ใส่เข้ามา (ซึ่งก็ไม่ผิด)
  - แต่ลูปที่ชัดเจนในตัวเองมักจะใช้คำสั่ง break; เข้าช่วยตรงกลาง และกำหนดเงื่อนไขลูปที่เป็นจริงเสมอ เช่น กำหนดเงื่อนไขว่า 0 < 1 หรือไม่ก็ใส่เลข 1 เข้าไปเลย (เลข 1 มีค่าความจริงเป็นจริงในภาษาซี) ทั้งนี้เพื่อบ่งชี้ว่าโดยธรรมดาแล้วลูปจะวนไปเรื่อย ๆ ไม่จบ เว้นแต่จะพบเงื่อนไขบางอย่างที่กลางลูป

ตัวอย่างการวนรับค่าไม่จำกัด 1

โจทย์ จงเขียนโปรแกรมที่รับค่าตัวเลขจำนวนเต็มจากผู้ใช้เข้ามาเรื่อย ๆ โปรแกรมจะทำการนับและบวกเลขที่เป็นบวก แต่หากผู้ใช้ใส่เลขที่เป็นลบหรือศูนย์เข้ามา โปรแกรมจะหยุดรับค่าจากผู้ใช้ แล้วพิมพ์จำนวนตัวเลขค่าบวกที่รับมาทั้งหมด รวมทั้งผลรวมของเลขบวกเหล่านี้

วิเคราะห์ ที่ผ่านมาเรามักจะหยุดลูปเมื่อผู้ใช้ใส่ตัวเลขเข้ามาถึงจำนวนหนึ่ง แต่ในปัญหานี้ ผู้ใช้สามารถใส่ตัวเลขเข้ามาได้ไม่จำกัด ดังนั้นการตั้งเงื่อนไขลูปโดยการจำกัดจำนวนครั้งไว้จึงเป็นเรื่องที่ผิด เพราะแท้จริงผู้ใช้จะใส่เลขเข้ามากี่ตัวก็ได้

1. เงื่อนไขที่จะใช้หยุดลูปจึงควรผูกอยู่กับค่าที่ผู้ใช้ใส่เข้ามา
2. ต้องป้องกันการนับและบวกค่าที่ไม่เป็นบวก แต่ให้โปรแกรมหยุดลูปแทน

โปรแกรมหาผลรวมตัวเลขบวก แบบไม่ใช้ break;

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

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

แบบนี้ไม่ใช้คำสั่ง break; แต่ก็คงพอจะเห็นได้ว่าเราต้องมีการทำอะไรแปลก ๆ เช่นเริ่มมาก็ให้ x = 1 ไปก่อนเลย ทั้งนี้ก็เพื่อรับประกันว่าเงื่อนไขที่กำหนดไว้จะเป็นจริงในรอบแรกของการทำงานแน่ ๆ

ถ้าผู้ใช้ใส่เลขศูนย์หรือค่าติดลบมา การนับและบวกค่าจะไม่เกิดขึ้น จากนั้นโปรแกรมจะวนกลับขึ้นไปทางด้านบน และลูปจะจบการทำงานเพราะเงื่อนไขลูปไม่เป็นจริง

โปรแกรมหาผลรวมตัวเลขบวก แบบใช้ break;

```
void main() {
  int count = 0;
  int sum = 0;
  int x;

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

แบบนี้ใช้คำสั่ง break; เงื่อนไขลูปคือ 0 < 1 ซึ่งแปลว่า ‘จริง’ ดังนั้นลูปจะวนไปเรื่อย ๆ จนกว่าจะโดนคำสั่ง break; ที่อยู่ด้านใน รูปแบบเงื่อนไขที่นิยมกว่าสำหรับวิธีนี้คือ while (1) { ... }

ถ้าผู้ใช้ใส่เลขศูนย์หรือค่าติดลบมา จะเข้าเงื่อนไขนี้ และจะเกิดการหยุดลูปขึ้น คำสั่งนับและบวกค่าด้านใต้ก็จะถูกข้ามไปด้วย

บทเรียนจากตัวอย่างโจทย์ ‘รับค่าบวกมาเรื่อย ๆ’

- ถ้าจะให้มันรับค่ามาเรื่อย ๆ ไม่จำกัดจำนวน เงื่อนไขลูปก็ห้ามจำกัดจำนวน (ดูเป็นเรื่องธรรมดา แต่คนจำนวนมากจะหลับหูหลับตาทำแต่สิ่งที่คุ้นเคย และก็จะพยายามบอกว่า $i < 100$ เป็นต้น)
- วิธีจัดการเงื่อนไขมีหลายแบบ ให้เราเลือกวิธีที่เรามั่นใจและตรงกับธรรมชาติในการคิดของเรา ภาษาซีถูกออกแบบมาให้มีทางเลือกหลายทาง มันอยู่แค่ว่าเราจะทำได้สักทางหรือเปล่า และเรามีแนวโน้มจะคิดไปแนวทางใด
- การจัดการลูปให้ได้สำเร็จ ทุกอย่างจะไปด้วยกันเป็นชุดเหมือน ‘คอมโบ’ เพราะจะไปโดด ๆ ไม่ได้ ทุกอย่างต้องลงรอยกันหมด ถ้าเราอ่านโค้ดไม่ออก เราก็จะไม่รู้ว่าสิ่งที่เขียนอยู่นั้นสอดคล้องลงรอยกันดีหรือไม่
- เพราะเราต้องตระหนักถึงความสัมพันธ์ของสิ่งต่าง ๆ เสมอ ไม่ต้องสงสัยเลยว่าทำไมคนทำเรื่องลูปไม่ค่อยได้ (ทั้งที่ตัวโจทย์มันดูง่ายดายเหลือเกิน)

ตัวอย่าง วนรับค่าจนกว่าจะเจอเลขคี่ติดกัน

โจทย์ จงเขียนโปรแกรมที่รับค่าตัวเลขจำนวนเต็มจากผู้ใช้เข้ามาเรื่อย ๆ โปรแกรมจะพิมพ์คำว่า even หากตัวเลขเป็นคู่ แต่ถ้าผู้ใช้ใส่เลขคี่เข้ามา โปรแกรมจะพิมพ์เลข -1 ออกมา และถ้าผู้ใช้ใส่เลขคี่เข้ามาติดต่อกันสองตัว โปรแกรมจะจบการทำงานโดยไม่พิมพ์เลข -1 ออกมาสำหรับเลขคี่ตัวที่สอง (ถ้าใส่เลขคู่หลังเลขคี่จะเหมือนกับว่าไม่เคยใส่เลขคี่เข้ามา)

ตัวอย่าง

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

วิเคราะห์โจทย์ วนรับค่าจนกว่าจะเจอเลขคี่ติดกัน

วิเคราะห์ เนื่องจากผลลัพธ์ที่เปลี่ยนไปตามผลที่เกิดขึ้นก่อนหน้า แบบนี้ต้องใช้ ***ตัวแปรเก็บสถานะ***เข้าช่วย ตัวแปรนี้ทำหน้าที่ระบุเหตุการณ์ที่เกิดขึ้นก่อนหน้า และมีความสำคัญกับผลลัพธ์ที่จะเกิดขึ้น ในที่นี้ก็คือการระบุว่าตัวเลขตัวที่แล้วเป็นคี่หรือไม่ สำหรับปัญหานี้

1. ในตอนแรกสถานะคือ ‘เลขก่อนหน้าไม่เป็นคี่’
2. ถ้าผู้ใช้ใส่เลขคี่เข้ามาและ ‘เลขก่อนหน้าไม่เป็นคี่’ ก็ให้เราเปลี่ยนสถานะให้เป็น ‘เลขก่อนหน้าเป็นคี่’ เราเปลี่ยนแบบนี้เพื่อเตรียมตัวสำหรับเลขถัดไป
3. ถ้าผู้ใช้ใส่เลขคี่เข้ามาและ ‘เลขก่อนหน้าเป็นคี่’ ก็ให้จบการทำงาน
4. ถ้าผู้ใช้ใส่เลขคู่เข้ามา ก็ให้กำหนดสถานะใหม่เป็น ‘เลขก่อนหน้าไม่เป็นคี่’ ย้ำอีกครั้งว่าการเปลี่ยนแปลงเป็นไปเพื่อการตรวจเลขถัดไป ไม่ใช่เลขตัวนี้

โปรแกรมวนรับค่าจนกว่าจะพบเลขคี่ติดกัน แบบใช้ break

```
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;
      }
   }
}
```

ตัวแปรเก็บสถานะชื่อ odd ถ้าเท่ากับ 0 แสดงว่าเลขก่อนหน้าไม่เป็นคี่ ถ้าเท่ากับ 1 แสดงว่าเลขก่อนหน้าเป็นคี่

ถ้าเจอเลขคู่ให้ตั้งสถานะสำหรับรอบถัดไป ว่าตัวเลขก่อนหน้าเป็นคู่

แต่พอเป็นเลขคี่เราต้องตรวจดูว่าเลขก่อนหน้าเป็นคี่หรือเปล่า ถ้าใช่แสดงว่าเราเจอเลขคี่ติดกันแล้วแน่นอน break ได้เลย

โปรแกรมแบบไม่ใช้ break (ใช้ตัวแปรสถานะล้วน ๆ)

```
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");
    }
  }
}
```

ตัวแปรเก็บสถานะชื่อ odd อันนี้นับว่าเป็นเลขคี่ติดต่อกันมากี่ตัวแล้ว ถ้าเป็นศูนย์ก็แสดงว่า ณ จุดนี้มีเลขคี่ติดกันมา 0 ตัว

คราวนี้เงื่อนไขคือตรวจดูว่าเลขคี่ต้องไม่ติดกันถึงสองตัว ถ้าถึงสองตัวก็หยุด

ถ้าเจอเลขคู่ให้ตั้งสถานะว่า ณ จุดนี้มีเลขคี่ติดกันเป็นจำนวน 0 ตัว (เหมือนเริ่มใหม่)

พอเป็นเลขคี่เราก็นับว่ามันติดกันเพิ่มขึ้นอีกหนึ่งครั้ง

แบบฝึกหัดท้ายบท

ทำข้อ 2, 3 และ 4

สังเกตด้วยว่าข้อ 4 โจทย์ยาวมาก มีคนเคยบ่นว่าโจทย์ผมมันยาว

แต่ที่จริงแล้วในการเขียนโปรแกรมโจทย์ยาวมันเป็นเรื่องสามัญ เพราะการอธิบายกระบวนการคิดว่าต้องการให้โปรแกรมทำอะไรบ้าง มันเป็นเรื่องที่ต้องมีการชี้แจงรายละเอียดเพื่อไม่ให้เข้าใจผิดเป็นอย่างอื่น

→ คนที่อ่านจับใจความได้จะไม่มีปัญหาอะไร แต่คนที่อ่านหนังสือไม่คล่อง ชอบอ่านข้าม อ่านแล้วจับสาระใจความไม่ได้ จะมีปัญหากับโจทย์แนวนี้ตลอด