การเขียนโปรแกรมคอมพิวเตอร์ 1

Computer Programming I

บทนำเกี่ยวกับคอมพิวเตอร์และการโปรแกรม

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

สัปดาห์แรก

แนะนำวิชา

รหัสและชื่อวิชา: 517 111 การโปรแกรมคอมพิวเตอร์ 1

(Computer Programming I)

จำนวนหน่วยกิต: 3 (2 – 2 - 5)

เป็นวิชาบังคับวิทยาการคอมพิวเตอร์ และเทคโนโลยีสารสนเทศปี 1

ตัวเลข 2 – 2 – 5 ที่จำนวนหน่วยกิตมีความหมายว่า

- เรียนทฤษฎีสัปดาห์ละ 2 ชั่วโมง
- เรียนปฏิบัติสัปดาห์ละ 2 ชั่วโมง
- ศึกษาด้วยตัวเองสัปดาห์ละ 5 ชั่วโมง

สำหรับหลักสูตรเข้มข้น

เนื่องจากนักศึกษาภาควิชาคอมพิวเตอร์ต้องใช้ทักษะจากวิชานี้อย่างจริงจัง

- การเรียนภาคบรรยายจะเพิ่มขึ้น 1 คาบต่อสัปดาห์
- การเรียนภาคปฏิบัติจะเพิ่มขึ้น 2 คาบต่อสัปดาห์

ดังนั้นที่จริงแล้วในหลักสูตรเข้มข้นเรา

- เรียนทฤษฎีสัปดาห์ละ 3 ชั่วโมง
- เรียนปฏิบัติสัปดาห์ละ 4 ชั่วโมง
- ศึกษาด้วยตัวเองสัปดาห์ละ 5 ชั่วโมง
  (หรือมากกว่าขึ้นอยู่กับพื้นฐานแต่ละคน)

แนะนำผู้สอน

อาจารย์ ดร. ภิญโญ แท้ประสาทสิทธิ์

ห้องทำงาน ห้อง 1642/3 ชั้น 6 อาคารวิทยาศาสตร์ 1
(ห้องพักอาจารย์โซน 3 เป็นโซนที่ดูลึกลับมาก
จุดสังเกตนอกจากป้ายก็คือว่ามีการเจิมยันต์ไว้ตรงทางเข้าด้วย)

ติดต่อ pinyotae+111@gmail.com,
pinyo@su.ac.th

เฟซบุ๊ค Pinyo Taeprasartsit

การประเมินผลการศึกษา

- ตัด F ที่ 40 คะแนน เหมือนปีที่ผ่านมา
- สำหรับคะแนนถูกแบ่งไว้ดังนี้
  - สอบทฤษฎีกลางภาค + ปลายภาค 15 + 15 = 30 คะแนน
  - สอบปฏิบัติกลางภาค + ปลายภาค 25 + 25 = 50 คะแนน
  - สอบปฏิบัติการย่อย 25 คะแนน

  คะแนนรวมแบบปรกติคือ 105 คะแนน
- นอกจากนี้มีตัวช่วยในการเก็บคะแนนดังนี้
  - การเข้าเรียนและส่งการบ้าน 4 คะแนน
  - ทดสอบความเร็วในการพิมพ์ดีดภาษาอังกฤษ 2 คะแนน
  - คะแนนแถมในการสอบปฏิบัติอื่น ๆ → ประมาณ 20 คะแนน

รูปแบบการเรียนการสอนและการวัดผล

- ชั่วโมงปฏิบัติจะมีการสอนเทคนิคบางอย่าง จะได้เรียนแล้วลองทำตามทันที **แต่ควรอ่านโจทย์และลองทำมาก่อน ไม่เช่นนั้นจะตามไม่ทัน**
- มีการสอบปฏิบัติเพิ่มเติมในชั่วโมงเรียนปฏิบัติการ**ตอนเย็นวันอังคาร**
- ข้อสอบย่อยจะง่ายกว่าข้อสอบกลางภาคและปลายภาคทั้งในเรื่องของความลึกของโจทย์และเวลาที่ให้ **ดังนั้นควรเก็บคะแนนจากการสอบย่อยให้มาก**
- ในชั่วโมงปฏิบัติที่มีการสอนจะใช้เอกสาร ถามเพื่อน ถามพี่ ลอกกันได้ เพราะใครพยายามด้วยตัวเองก็เก่งขึ้นเอง และ**ตอนสอบก็ตัวใครตัวมัน**
- **การสอบทุกครั้งไม่มีการนำเอกสารเข้าสอบและต้องทำด้วยตัวเองเท่านั้น**

หนังสือประกอบการเรียนการสอน

- ผมได้ค้นหาหนังสือภาษาซีมาศึกษาดูหลายเล่มและพบว่าเล่มที่น่าสนใจที่สุดสำหรับพวกเราคือ “คู่มือเรียนภาษาซี ฉบับปรับปรุงใหม่” โดย อรพิน ประวัติบริสุทธิ์ สำนักพิมพ์โปรวิชัน ราคา 199 บาท อ่านง่าย มี แบบฝึกหัดท้ายบทพร้อมเฉลย
- ในเอกสารของรายวิชาที่ให้บนเว็บไปมีบอกด้วยว่าเนื้อหาในชั้นเรียนตรงกับส่วนไหนของหนังสือ
- แต่จริง ๆ แล้วขยันทำจากแบบฝึกหัดและข้อสอบเก่าที่ให้ไปจะได้ผลมากที่สุด ได้ทักษะที่ตรงกับความต้องการของตลาดแรงงาน และปูพื้นสำหรับวิชาอื่น ๆ ที่ต้องใช้การเขียนโปรแกรม

เนื้อหาและโจทย์สำหรับการเรียนภาคปฏิบัติ

- ส่วนใหญ่โจทย์มีเตรียมไว้ให้ล่วงหน้าให้ดาวน์โหลดไปซ้อมก่อนได้
- ต้นฉบับมีให้ที่ร้านถ่ายเอกสารบางร้าน
- **ให้ลองทำก่อนเข้าเรียน** เนื่องจากเวลาในแล็บมีน้อยมากเมื่อเทียบกับความเร็วในการคิดของนักศึกษามือใหม่
- ถ้าลองทำมาก่อน → เวลาที่ทำไม่ได้จะได้มีเวลาถามอาจารย์หรือผู้ช่วยสอนในตอนทำแล็บทันที
- บางครั้งเนื้อหามีเฉลยให้ด้วย แต่นักศึกษาไม่ควรจะเปิดอ่านเฉลยทันที
  - **ความเข้าใจไม่ได้เกิดจากการจำ แต่เกิดจากคิด วิเคราะห์ และนำไปใช้**
  - ถ้าไม่ได้คิดด้วยตัวเองก่อนจะไม่เข้าใจและโอกาสผ่านจะเกือบเป็นศูนย์

แบบฝึกหัดสำหรับซ้อมทำด้วยตัวเอง

- จะมีโจทย์ข้อสอบเก่าเป็นแบบฝึกหัดให้ทำแล้วมากกว่า 250 หน้า
- การทำแบบฝึกหัดจะส่งผลให้พัฒนาตัวเองและเข้าใจเนื้อหาวิชาได้อย่างที่ควรจะเป็น ถ้าทำด้วยความเข้าใจจบปีหนึ่งแล้วจะออกไปทำงานเลยก็ยังได้
- อย่าลืมซ้อมทำโจทย์พวกนี้ด้วย เพราะมันก็เหมือนกับโจทย์เลขในวิชาแคลคูลัส **ถ้าไม่ซ้อมทำก็จะทำข้อสอบไม่ได้ เพราะการเขียนโปรแกรมได้เราต้องมีพร้อมทั้งความจำและความชำนาญ**
- เทคนิคต่าง ๆ ในแบบฝึกหัดที่จริงเป็นเทคนิคเดียวกับโจทย์ในข้อสอบ
- **อย่าลืมว่าวิชานี้มีผลกับการเรียนตลอดทั้งสี่ปีในภาควิชาคอมและไอที**

เนื้อหาที่จะเรียนในวันนี้

- **รู้จักกับคอมพิวเตอร์และการเขียนโปรแกรม**
- คอมพิวเตอร์กับโปรแกรม
  - การวางแผนการเขียนโปรแกรม
  - ปัญหาที่แก้ด้วยคอมพิวเตอร์ได้
  - การแก้ปัญหาการคำนวณด้วยมนุษย์
  - การแก้ปัญหาการคำนวณด้วยคอมพิวเตอร์
- ขั้นตอนและตัวอย่างการวางแผนการเขียนโปรแกรม
  - การวิเคราะห์ปัญหา
  - การอธิบายลำดับการคำนวณด้วยซูโดโค้ดและโฟลวชาร์ต

รู้จักกับคอมพิวเตอร์

- คอมพิวเตอร์เป็นเครื่องคำนวณแบบหนึ่ง
- ในปัจจุบันคอมพิวเตอร์มีสมรรถนะการคำนวณที่สูงมาก
  - เรามักจะรันโปรแกรมสำเร็จได้ในเวลาไม่กี่วินาที
  - สมรรถนะของเครื่องคอมพิวเตอร์พกพา (แล็ปท็อป/โน้ตบุ๊ค) สามารถคำนวณตัวเลขได้หลายสิบล้านคำสั่งในหนึ่งวินาที เช่น เครื่องแล็ปท็อปปัจจุบันสามารถบวกลบคูณหารเลขทศนิยมได้มากกว่า 50 ล้านคู่ในหนึ่งวินาที
  - สำหรับซุปเปอร์คอมพิวเตอร์ที่เร็วที่สุดในปัจจุบันสามารถทำได้มากกว่า 17.5×10^{15} คำสั่งในหนึ่งวินาที (10,000 ล้านล้าน, 10 Peta) เครื่องที่เร็วที่สุดในปัจจุบันคือ (Titan Supercomputer ของ Cray)
- ด้วยสมรรถนะที่สูงเช่นนี้ เราจึงนำมันมาประยุกต์ใช้ช่วยงานมนุษย์หลายอย่าง

องค์ประกอบของคอมพิวเตอร์

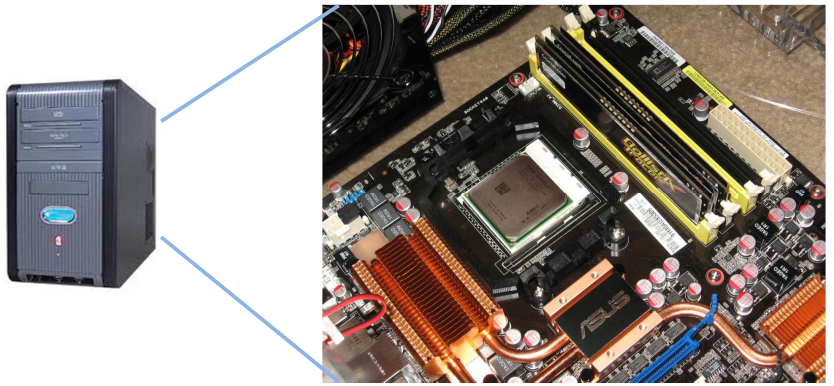
- ถ้าเราสังเกต เราจะพบว่าเวลาเราทำงานเราใช้ร่างกายหลายส่วนร่วมกัน
  - เราใช้ตาเพื่ออ่านหรือคำสั่ง เช่น เนื้อหาในหนังสือ หรือ โจทย์
  - หรือบางทีเราก็ใช้หูฟังคำถามหรือคำอธิบาย
  - เราต้องมีการใช้สมองคิด
  - เราใช้มือเขียนบันทึกหรือคำตอบ
  - ร่างกายเรานั่งอยู่บนเก้าอี้
- คอมพิวเตอร์เองก็เหมือนกันจะทำงานได้ก็ต้องมีอุปกรณ์หลายส่วนเช่นกัน
  - เมาส์กับคีย์บอร์ดเป็นเหมือนตากับหูที่ใช้รับคำสั่งจากเราว่าเราจะให้ทำอะไร
  - ซีพียู (พร้อมทั้งหน่วยความจำ) เป็นเหมือนสมอง
  - จอภาพเป็นเหมือนมือใช้เขียนคำตอบออกมาให้คนอื่นได้เห็น

เปรียบเทียบการคำนวณในมนุษย์และคอมพิวเตอร์

- คนเราเรียนรู้และทำงานโดยอาศัยความรู้ความจำในอดีตเป็นตัวช่วย เช่น เราหาความยาวด้านสามเหลี่ยมจากมุมได้เพราะเรารู้เรื่องตรีโกณมิติ
- คอมพิวเตอร์เองก็เช่นกัน จะทำงานได้ก็ต้องมีการเก็บวิธีการคำนวณบางอย่างไว้ เช่น วิธีการคำนวณค่า sin, cos, และ tan
- เวลาคนเราคำนวณตัวเลข เราก็ต้องจำตัวเลขที่เกี่ยวข้องไว้ในหัวได้ เช่น “จงหาค่าของ 5 + 3” เราคำนวณได้ว่ามันมีค่าเท่ากับ 8
  - ➢ถ้าเราลองทบทวนดูเราจะพบว่า ถ้าเราไม่สามารถจำเลข 5 และ 3 ไว้ในหัว เราได้เลยล่ะก็ เราจะหาผลลัพธ์ออกมาไม่ได้เลย
- **คอมพิวเตอร์ก็ต้องเก็บข้อมูลที่เกี่ยวข้องกับการคำนวณไว้ด้วย** เช่นจากตัวอย่างเดิม เครื่องก็ต้องจำเลข 5 กับ 3 ไว้เพื่อใช้ในการหาผลบวก

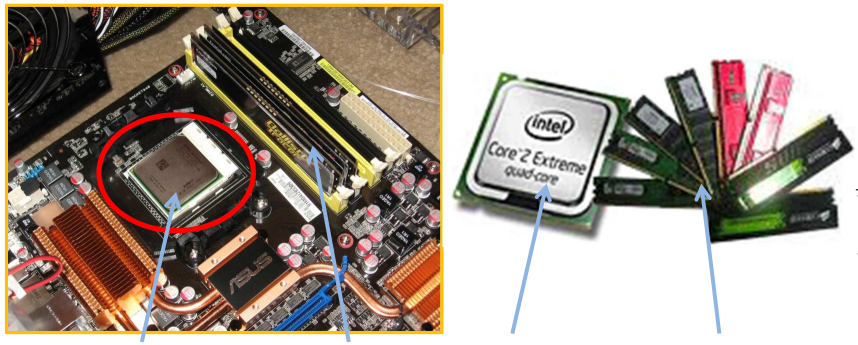
หน่วยความจำกับการคำนวณในคอมพิวเตอร์

- ในขณะทำการคำนวณ ซีพียู (CPU, หน่วยประมวลผลกลาง) จะมีการติดต่อกับหน่วยความจำ (Memory, RAM) บ่อย ๆ
- ซีพียูกับหน่วยความจำเปรียบเหมือนสมองคนละส่วน : ส่วนความคิดและจำ



ภาพ overclockzone.com และ jedihawk.com

ซีพียูและหน่วยความจำ



ซีพียู หน่วยความจำ ซีพียู หน่วยความจำ

ภาพ jedihawk.com และ comcenter.co.th

เรื่องสำคัญ : การคำนวณอยู่คู่กับการเก็บข้อมูลในหน่วยความจำ ในการเขียนโปรแกรมก็เช่นกัน เราต้องระบุว่ามีข้อมูลอะไรและจะเอาข้อมูลไปทำอะไรบ้าง

คีย์บอร์ดและเมาส์

- คีย์บอร์ด (แป้นพิมพ์) และ เมาส์ เป็นอุปกรณ์รับคำสั่งจากเรา
- ทำหน้าที่รับฟังว่าเราต้องการอะไร
- เนื่องจากคอมพิวเตอร์ทั่วไปไม่ได้สังการด้วยภาพและเสียง คีย์บอร์ดและเมาส์จึงเปรียบเหมือนตาและหูของคอมพิวเตอร์
- มีไว้เพื่อสื่อสารรับคำสั่งจากเรา (เราใช้ตาอ่านคำสั่งและใช้หูฟังคำสั่ง)

ภาพ thaidelclub.com

เนื้อหาที่จะเรียนในวันนี้

- รู้จักกับคอมพิวเตอร์และการเขียนโปรแกรม
- **คอมพิวเตอร์กับโปรแกรม**
  - **การวางแผนการเขียนโปรแกรม**
  - **ปัญหาที่แก้ด้วยคอมพิวเตอร์ได้**
  - **การแก้ปัญหาการคำนวณด้วยมนุษย์**
  - **การแก้ปัญหาการคำนวณด้วยคอมพิวเตอร์**
- ขั้นตอนและตัวอย่างการวางแผนการเขียนโปรแกรม
  - การวิเคราะห์ปัญหา
  - การอธิบายลำดับการคำนวณด้วยซูโดโค้ดและโฟลวชาร์ต

คอมพิวเตอร์กับโปรแกรม

- คอมพิวเตอร์ทำการคำนวณต่าง ๆ ได้ เพราะได้รับคำสั่งเกี่ยวกับการคำนวณ
- คำสั่งในการคำนวณมักจะมาเป็นชุด ๆ ไม่ได้เป็นแค่การบวกลบคูณหารครั้งเดียวจบ
  - เหมือนกับการแก้สมการสองตัวแปร เช่น
    $x + y = 5$
    $x - y = 3$
  - เราสามารถคำนวณได้ $x = 4$ และ $y = 1$ แต่ไม่ใช่ว่าเราบวกลบตัวเลขมั่ว ๆ แล้วจะได้คำตอบ
  - เรามีลำดับการคิดที่เป็นระบบ ประกอบด้วยหลายขั้นตอน
  - ขั้นตอนในการคำนวณเหล่านี้ สามารถแปลงไปเป็นการคำสั่งด้านการคำนวณด้วยคอมพิวเตอร์ได้

ว่าแต่โปรแกรมคืออะไรกันแน่

- โปรแกรมเป็นชุดคำสั่งด้านการคำนวณ ซึ่งอาจจะรวมไปถึงการอ่านข้อมูลเข้า (input) และการแสดงผลลัพธ์ (output)
- โปรแกรมมีอยู่ในสองรูปแบบใหญ่ ๆ คือ
  - แบบที่เป็นภาษาที่เราอ่านออก (มนุษย์เข้าใจ แต่เครื่องไม่เข้าใจ) อันนี้เป็นโปรแกรมที่เราเขียนขึ้นมานั่นเอง และเป็นสิ่งที่เราต้องทำในวิชานี้
  - แบบที่เป็นภาษาเครื่องเลขฐานสอง (มนุษย์ไม่เข้าใจ แต่เครื่องเข้าใจ) นี่เป็นโปรแกรมประยุกต์ต่าง ๆ ที่เราใช้ เช่น โปรแกรม Microsoft Office และ Firefox เป็นต้น
- โดยทั่วไป เราจะเขียนโปรแกรมเป็นภาษาที่เราอ่านออก เช่น ภาษาซี แล้วใช้ตัวแปลโปรแกรม (compiler) แปลภาษาซีให้เป็นภาษาเครื่องเพื่อให้นำไปคำนวณด้วยเครื่องได้

การวางแผนการเขียนโปรแกรม

- ณ ตอนนี้เรารู้แล้วว่าโปรแกรมคือชุดคำสั่งที่ใช้ระบุขั้นตอนการคำนวณ และเรารู้ด้วยว่าเราระบุขั้นตอนเหล่านี้ได้ผ่านภาษาโปรแกรม เช่น ภาษาซี
- แต่เราจะวางแผนการคำนวณอย่างไรดี ถึงจะเป็นระบบและแก้ปัญหาได้ ?
  - อันดับแรก เราต้องแน่ใจก่อนว่าปัญหาที่เราจะแก้เป็นสิ่งที่เครื่องคำนวณได้
  - โดยมากแล้วปัญหาที่มนุษย์คำนวณได้ก็จะเป็นปัญหาที่คอมพิวเตอร์คำนวณได้เช่นกัน
  - ถ้าปัญหาใดที่มนุษย์ไม่รู้แม้กระทั่งขั้นตอนในการคิด โอกาสที่มันจะเป็นงานที่คอมพิวเตอร์คำนวณได้แทบจะเป็นศูนย์

เรื่องสำคัญ : โปรแกรมทำหน้าที่ระบุขั้นตอนการทำงาน ถ้าเรายังคิดขั้นตอนแก้ปัญหาในกระดาษไม่ออก แสดงว่าเราไม่รู้ขั้นตอนการทำงานที่ถูกต้อง เราย่อมไม่สามารถสั่งคอมพิวเตอร์ให้แก้ปัญหาได้

ปัญหาที่แก้ด้วยคอมพิวเตอร์ได้

- คือปัญหาที่สามารถคำนวณได้ (computable) หรือสามารถระบุออกมาเป็นขั้นตอนที่แน่ชัดได้
- เราจะเขียนโปรแกรมได้ เราก็ต้องรู้ขั้นตอนที่แน่ชัดก่อน
  - เพราะคอมพิวเตอร์ทำตามที่เราสั่ง ไม่ได้สร้างวิธีคิดแทนเรา
  - ถ้าปัญหาไม่มีวิธีคิดที่แน่นอน เราจะเขียนโปรแกรมไม่ได้ ถ้าเราเขียนโปรแกรมอย่างนั้นออกมาแล้ว ผลลัพธ์มักจะผิด
- ปัญหาต้องไม่ใช้ทรัพยากรเกินกว่าที่เครื่องคอมพิวเตอร์มี เช่น ต้องไม่ใช้หน่วยความจำเกินกว่าที่เครื่องจะจัดหาให้ได้ เป็นต้น

การแก้ปัญหาการคำนวณด้วยมนุษย์

- ปัญหาด้านการคำนวณได้รับการแก้ไขด้วยมนุษย์มาช้านานแล้ว
  - ➢เช่น การแก้สมการสองตัวแปร สามตัวแปร ... N ตัวแปร
    การทดสอบว่าเลขเป็นจำนวนเฉพาะหรือไม่
    การหาตัวหารร่วมมาก ตัวคูณร่วมน้อย เป็นต้น
- สังเกตด้วยว่าปัญหาพวกนี้มีวิธีแก้ที่ชัดเจน
  - ➢ในระยะหลัง เราเพียงใช้คอมพิวเตอร์มาช่วยคิดเพื่อให้ได้คำตอบที่เร็วขึ้นและมีความผิดพลาดน้อยลง
- อุปสรรคสำหรับโปรแกรมเมอร์จำนวนมากก็คือ การที่ไม่ทราบว่าวิธีแก้ปัญหาคืออะไร หรือรู้แต่ก็ไม่สามารถอธิบายมาเป็นขั้นตอนได้
  - หลายคนแก้ปัญหาพวกนี้ด้วยการลองผิดลองถูกไปเรื่อย
  - คนที่ฝึกคิดมาอย่างดี จะรู้ขั้นตอนพวกนี้ชัดเจน อธิบายวิธีที่ถูกต้องได้ไม่ติดขัด

เรารู้วิธีแก้ปัญหาอย่างถูกต้องแน่นอนหรือไม่

- ดูตัวอย่างต่อไปนี้แล้วถามตัวเราเองว่าเรารู้วิธีหาคำตอบที่แน่นอนหรือไม่
  - การแก้สมการหนึ่งตัวแปร $3x + 2 = 8$
    เราต้องการหาค่า x ซึ่งเป็นผลลัพธ์ จากข้อมูลเข้าคือตัวเลขค่าคงที่ต่าง ๆ
  - การแก้สมการสองตัวแปร
    $2x + 3y = 25$
    $3x + 2y = 20$
    เราต้องการหาค่า x และ y ซึ่งเป็นผลลัพธ์ จากข้อมูลเข้าคือตัวเลขค่าคงที่
- คำถามที่เราต้องถามตัวเราเองก่อนเขียนโปรแกรม
  1. เราสามารถหาคำตอบเหล่านี้ได้ โดยไม่ต้องลองถูกลองผิดหรือไม่ ?
  2. เราสามารถอธิบายวิธีแก้ปัญหาที่ถูกต้องให้คนอื่นเข้าใจได้หรือไม่ ?
- ถ้าหากเรายังอธิบายวิธีแก้ปัญหาไม่ได้ อย่าเพิ่งลงมือเขียนโปรแกรม

การแก้ปัญหาการคำนวณด้วยคอมพิวเตอร์

- มนุษย์เป็นผู้กำหนดวิธีการคำนวณลงไปในโปรแกรม เพื่อให้คอมพิวเตอร์ทำตามขั้นตอนที่ระบุไว้ในโปรแกรม
- การกำหนดวิธีการคำนวณต้องมีสื่อกลาง ซึ่งก็คือภาษาคอมพิวเตอร์นั่นเอง
- เราจึงต้องมีการเรียนภาษาคอมพิวเตอร์เพื่อทำให้เราสอนคอมพิวเตอร์ให้ทำตามที่เราต้องการได้
- คอมพิวเตอร์จะทำการคำนวณตามกระบวนการที่เราระบุไว้ในภาษา

ภาษาคอมพิวเตอร์

- เป็นเครื่องมือในการสื่อสารกับคอมพิวเตอร์
- มีหลายภาษา แต่ละภาษามีจุดอ่อนจุดแข็งแตกต่างกันไป
- ภาษาคอมพิวเตอร์จำนวนมาก มีลักษณะดังต่อไปนี้
  - มีกฎเกณฑ์ที่แน่นอน มีความเข้มงวดทางภาษามาก
  - เต็มไปด้วยการนิยามและกำหนดค่าต่าง ๆ
  - ต้องการให้เราระบุวิธีคิดลงไปอย่างชัดเจน
  - ไม่ยอมรับความกำกวม ถ้าเกิดขึ้นจะไม่ยอมทำงานให้เรา หรือจะมีกฎตายตัวว่าจะตีความเป็นอย่างใดอย่างหนึ่ง
- **คนที่จะเขียนภาษาคอมพิวเตอร์ได้ถูกต้อง จะต้องเข้าใจกฎเกณฑ์ทางภาษาอย่างชัดเจน**
  - ถ้าเราไม่เข้าใจกฎอย่างชัดเจน เราจะงงอยู่ตลอดเวลา (สาเหตุการสอบตก)

เนื้อหาที่จะเรียนในวันนี้

- รู้จักกับคอมพิวเตอร์และการเขียนโปรแกรม
- คอมพิวเตอร์กับโปรแกรม
  - การวางแผนการเขียนโปรแกรม
  - ปัญหาที่แก้ด้วยคอมพิวเตอร์ได้
  - การแก้ปัญหาการคำนวณด้วยมนุษย์
  - การแก้ปัญหาการคำนวณด้วยคอมพิวเตอร์
- **ขั้นตอนและตัวอย่างการวางแผนการเขียนโปรแกรม**
  - **การวิเคราะห์ปัญหา**
  - **การอธิบายลำดับการคำนวณด้วยซูโดโค้ดและโฟลวชาร์ต**

ขั้นตอนการพัฒนาโปรแกรม

- ประกอบด้วย 5 ขั้นตอนหลัก (อ้างอิงตามบทที่ 4 ในหนังสือ)
  1. **วิเคราะห์ปัญหา (Analysis)**
  2. **วางแผนและออกแบบ (Planning and Design)**
  3. **การเขียนโปรแกรม (Coding)**
  4. การทดสอบโปรแกรม (Testing)
  5. จัดทำเอกสารและคู่มือการพัฒนาหรือใช้งาน (Documentation)
- ในวิชานี้จะเน้นสามขั้นตอนแรกคือ วิเคราะห์ปัญหา, วางแผนและออกแบบ, และ การเขียนโปรแกรม
  - การทดสอบโปรแกรมเป็นส่วนที่ต้องเรียนรู้จากการปฏิบัติเป็นหลัก
  - ส่วนการจัดทำเอกสารและคู่มือจะไม่กล่าวถึงในวิชานี้

การวิเคราะห์ปัญหา (Analysis)

เนื่องจากขั้นตอนการเขียนโปรแกรมเกี่ยวข้องกับการแก้ปัญหา จึงมีความจำเป็นที่จะต้องอธิบายการวิเคราะห์ไปพร้อมกับตัวอย่างปัญหา

โจทย์: จงเขียนโปรแกรมรับค่าจำนวนเต็ม 2 จำนวน และหาผลบวกของเลขทั้งสองจำนวนนั้น

1. **วิเคราะห์ปัญหา**
   - การวิเคราะห์ปัญหาเป็นขั้นตอนที่สำคัญที่สุด เพราะถ้าเราคิดไม่ออก เราก็บอกคอมพิวเตอร์ให้ทำตามเราไม่ได้
   - ควรเริ่มจากการการคัดแยกว่าอะไรคือข้อมูลเข้า และแยกให้ได้ว่าผลลัพธ์ที่ต้องการคืออะไร
   - จากนั้นวิเคราะห์ว่าข้อมูลเข้าและผลลัพธ์มีความสัมพันธ์กันอย่างไร

โจทย์ดัดแปลงมาจากตัวอย่าง 4.1 หน้า 41 – 43 ในหนังสือเรียน

วิเคราะห์ข้อมูลเข้าและผลลัพธ์

- **ข้อมูลเข้า** : จำนวนเต็มสองตัว
  - กำหนดให้จำนวนเต็มตัวแรกชื่อ x
  - กำหนดให้จำนวนเต็มตัวที่สองชื่อ y
  - **หมายเหตุ** การเขียนโปรแกรมมักเกี่ยวข้องกับการนิยามชื่อและกำหนดค่า
- **ผลลัพธ์** : ผลบวกของจำนวนเต็มทั้งสองตัว

  ➢กำหนดให้ผลลัพธ์ชื่อ sum
- วิเคราะห์ความสัมพันธ์ระหว่างข้อมูลเข้าและผลลัพธ์
  - เราต้องนำข้อมูลเข้ามาบวกกัน คือ หาค่า x + y
  - ผลบวกคือผลลัพธ์ นั่นคือ sum = x + y
  - **<u>การกำหนดชื่อจะทำให้หาความสัมพันธ์กันได้โดยสะดวก</u>**

    → นอกจากนี้ยังทำให้เราสามารถเขียนอธิบายเป็นขั้นตอนได้ง่ายด้วย

วางแผนและออกแบบ (Planning and Design)

- คือการนำปัญหาที่วิเคราะห์ได้จากขั้นตอนที่หนึ่ง มาวางแผนอย่างเป็นขั้นตอน ขั้นตอนดังกล่าวต้องอธิบายลำดับการทำงานโปรแกรมโดยชัดเจน
  - แผนการแก้ปัญหาที่เป็นขั้นตอนนี้เรียกว่า ***อัลกอริทึม*** (Algorithm)
  - วิธีที่นิยมใช้อธิบายอัลกอริทึมมีอยู่สองแบบคือ ***ซูโดโค้ด*** (Pseudocode) และ ***โฟลวชาร์ต*** (Flowchart)
- **ซูโดโค้ด** คือ การอธิบายขั้นตอนออกมาเป็นภาษาที่สื่อความหมายง่าย ๆ
  - จะบอกเป็นขั้นตอนสั้น ๆ หลายขั้นตอนต่อกันไป
  - ไม่บอกเป็นข้อความยาว ๆ เป็นย่อหน้า
- **โฟลวชาร์ต** คือ การอธิบายขั้นตอนโดยใช้สัญลักษณ์รูปภาพเป็นตัวสื่อความหมาย

ซูโดโค้ด (Pseudocode)

- การอธิบายขั้นตอนด้วยซูโดโค้ดควรระบุถึงรายละเอียดต่อไปนี้
  - จุดเริ่มต้น (Start)
  - ข้อมูลเข้า (มักมาจากการอ่านจากผู้ใช้)
  - วิธีการคำนวณ
  - การแสดงผลลัพธ์
  - จุดสิ้นสุด (Stop, End)
- **หมายเหตุ** Pseudo- เป็นคำนำหน้า (prefix) ที่มีรากมาจากภาษากรีก แปลว่า ‘ปลอม’; Pseudocode จึงมีความหมายว่า ‘โค้ดปลอม’ นั่นเอง คือหน้าตามันคล้ายโค้ด แต่ก็ไม่ใช่โค้ดจริงที่เราเขียนลงในเครื่อง

ตัวอย่างซูโดโค้ด

- ย้ำ: สิ่งที่เราควรจะอธิบายในซูโดโค้ด คือ จุดเริ่มต้น, ข้อมูลเข้า, การคำนวณ, การแสดงผลลัพธ์, จุดสิ้นสุด
- จากตัวอย่างโจทย์ เราเขียนเป็นซูโดโค้ดได้ดังนี้

เขียนแบบเต็ม
(ไม่นิยม โดยเฉพาะการใส่คำว่า COMPUTE)

```
START
  READ  X
  READ  Y
  COMPUTE  SUM = X + Y
  PRINT  SUM
STOP END
```

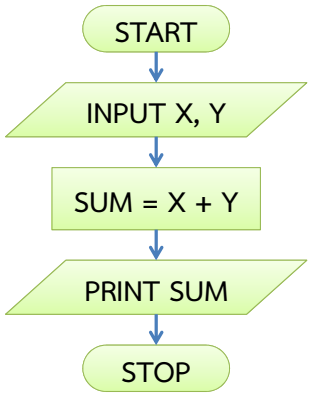
เขียนแบบทั่วไป (นิยมกว่ามาก)

```
START
  READ  X, Y
  SUM = X + Y
  PRINT  SUM
STOP END
```

โฟลวชาร์ต (Flowchart)

- เช่นเดียวกับซูโดโค้ด เรามักจะระบุของห้าอย่างลงไปด้วยคือ
  จุดเริ่มต้น, ข้อมูลเข้า, การคำนวณ, การแสดงผลลัพธ์, และ จุดสิ้นสุด
- ข้อแตกต่างคือจะมีรูปมากำกับ ทำให้มองออกได้ง่ายขึ้นว่าขั้นตอนทำอะไร

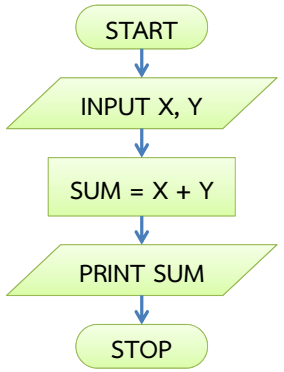
ตัวอย่าง



องค์ประกอบพื้นฐานในโฟลวชาร์ต

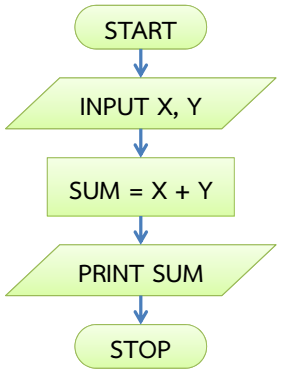
-  คือ ตัวบอกจุดเริ่มต้นหรือสิ้นสุด (Terminator)

-  ใช้ระบุลำดับการทำงาน



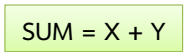
องค์ประกอบพื้นฐานในโฟลวชาร์ต

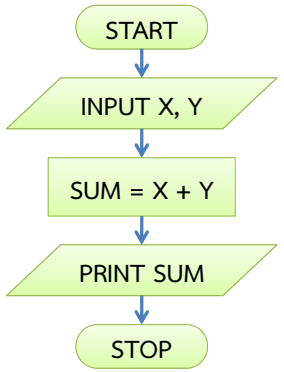
-  ระบุการรับข้อมูลเข้าหรือแสดงผลลัพธ์



องค์ประกอบพื้นฐานในโฟลวชาร์ต

-  แทนขั้นตอนการประมวลผลและการคำนวณต่าง ๆ
  เป็นส่วนที่สำคัญที่สุดในกระบวนการอธิบายลำดับการทำงาน





ข้อสังเกตเกี่ยวกับซูโดโค้ดและโฟลวชาร์ต

- ในซูโดโค้ดเราระบุ READ X และ READ Y แยกกันต่างหาก แต่เราใช้ INPUT X, Y ในโฟลวชาร์ตรวมกันในช่องเดียว
  - ถูกทั้งคู่ แต่สำหรับโฟลวชาร์ต ส่วนที่เราเข้าใจได้ง่าย เช่นการรับข้อมูลเข้า มักจะถูกยุบรวมกันเพื่อประหยัดพื้นที่ แต่จะเขียนแยกกันก็ได้
  - ในซูโดโค้ดถ้าเราจะเขียนรวมกันเป็น READ X, Y ก็ได้เช่นกัน
- คำว่า READ กับคำว่า INPUT หมายถึง การอ่านข้อมูลเข้า เราจะใช้คำไหนก็ได้ ถือว่าถูกต้องทั้งคู่
  - วิชานี้ไม่ได้เน้นภาษาอังกฤษ นักศึกษาจะใช้คำไทยว่า 'อ่านค่า' ก็ได้
  - แต่ชื่อของค่าต่าง ๆ ควรเป็นภาษาอังกฤษ จะได้นำไปใช้ต่อในโปรแกรมได้ง่าย
  - ควรเลี่ยงการตั้งชื่อว่า ก ข อะไรทำนองนี้ ไม่งั้นจะงงตอนเขียนโปรแกรม

ขั้นตอนการพัฒนาโปรแกรม

- ประกอบด้วย 5 ขั้นตอนหลัก (อ้างอิงตามบทที่ 4 ในหนังสือ)
  1. **วิเคราะห์ปัญหา (Analysis)**
  2. **วางแผนและออกแบบ (Planning and Design)**
  3. **การเขียนโปรแกรม (Coding)**
  4. การทดสอบโปรแกรม (Testing)
  5. จัดทำเอกสารและคู่มือการพัฒนาหรือใช้งาน (Documentation)

การเขียนโปรแกรม (Coding)

- เป็นการนำอัลกอริทึมจากขั้นตอนที่สองมาเขียนให้ถูกต้องตามหลักไวยากรณ์ของภาษาคอมพิวเตอร์
- เรื่องน่ารู้เกี่ยวกับไวยากรณ์ของภาษาคอมพิวเตอร์
  - ภาษาไทย ภาษาอังกฤษมีหลักไวยากรณ์; ภาษาคอมพิวเตอร์ก็มีหลักไวยากรณ์เหมือนกัน
  - แต่หลักไวยากรณ์ของภาษาคอมพิวเตอร์จะเคร่งครัดมาก ผิดแล้วผิดเลย ดังนั้นคนเขียนต้องแม่นเรื่องนี้
  - เราใช้คำว่า Grammar แทนไวยากรณ์ของภาษามนุษย์ แต่เราใช้คำว่า Syntax แทนไวยากรณ์ของภาษาคอมพิวเตอร์
  - ราชบัณฑิตแปลคำว่า Syntax ว่า 'วากยสัมพันธ์' แปลคำว่า Syntax error สองแบบคือ 'ผิดวากยสัมพันธ์' และ 'ผิดไวยากรณ์'

ตัวอย่างโปรแกรมบวกตัวเลข

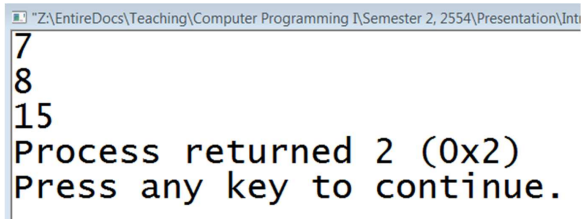
```
#include <stdio.h>   // เตรียมการทำงาน (ส่วนพิเศษสำหรับภาษาซี)
void main() {        // เริ่มการทำงาน (START)
   int x, y, sum;    // นิยามชื่อต่าง ๆ (ไม่ปรากฏในซูโดโค้ดหรือโฟลวชาร์ต)
   scanf("%d", &x);  // อ่านค่า X  (INPUT X)
   scanf("%d", &y);  // อ่านค่า Y  (INPUT Y)
   sum = x + y;      // หาค่าผลบวก แล้วเก็บไว้ที่ sum (SUM = X + Y)
   printf("%d", sum);// แสดงผลบวกทางจอภาพ  (PRINT SUM)
}                    // จบการทำงาน (STOP)
```

การทดสอบโปรแกรม

- เป็นการทดสอบว่าผลลัพธ์จากโปรแกรมที่ได้ถูกต้องหรือไม่
- โดยมากทำด้วยการทดลองใส่ข้อมูลเข้าให้กับโปรแกรมแล้วดูผลลัพธ์ที่ได้
- จากตัวอย่างเดิมสิ่งที่เราทำก็คือการใส่ค่า x และ y เข้าไป จากนั้นก็ตรวจดูว่าโปรแกรมของเราพิมพ์ค่า sum ออกมาถูกต้องหรือไม่
- เราจะรู้ได้ว่าโปรแกรมถูกต้องหรือไม่ เราก็ต้องทราบก่อนจากการคำนวณด้วยตัวเราเองว่าผลลัพธ์ที่ได้เป็นอย่างไร
  - ถ้าเราไม่สามารถคำนวณด้วยตัวเราเองได้ เราย่อมไม่รู้ว่าผลลัพธ์ที่ถูกต้องเป็นอย่างไร และตัดสินไม่ได้ว่าโปรแกรมถูกต้องหรือไม่
  - โดยปรกติแล้ว เรารู้วิธีคำนวณตั้งแต่ขั้นตอนที่หนึ่งและสอง
- การทดสอบที่ดีจะต้องทดสอบกับข้อมูลเข้าหลาย ๆ แบบ

ตัวอย่างการทดสอบโปรแกรม

- ทดสอบรอบแรก : กำหนดให้ค่า x และ y คือ 7 และ 8 ผลลัพธ์ที่ควรได้จากโปรแกรมคือ 15 เพราะ 7 + 8 = 15



- เนื่องจากการทดสอบควรทำกับข้อมูลเข้าหลาย ๆ ชุด หลาย ๆ แบบ ดังนั้นจึงควรทำการทดสอบเพิ่มเติม เช่น
- ทดสอบรอบที่สอง : กำหนดให้ค่า x และ y คือ 1 และ -8 ผลลัพธ์ที่ควรได้จากโปรแกรมคือ -7

ระเบียบวิธีในการกำหนดขั้นตอนการทำงานของโปรแกรม

- ถึงจุดนี้เรารู้แล้วว่าปัญหาแบบไหนที่น่าจะคำนวณด้วยคอมพิวเตอร์ได้
- และเราก็รู้ด้วยว่าเราต้องระบุขั้นตอนการทำงานให้คอมพิวเตอร์ทราบ
  - เราต้องทราบขั้นตอนวิธีแก้ปัญหาอย่างชัดเจนเป็นขั้นเป็นตอน
  - เราต้องอธิบายขั้นตอนดังกล่าวในรูปภาษาคอมพิวเตอร์ได้
- เนื่องจากว่าคนจำนวนมากไม่ทราบว่าจะระบุขั้นตอนอย่างไร
  - จึงมีระเบียบวิธีในการอธิบายขั้นตอนการทำงานขึ้นมา
  - ระเบียบวิธีนี้จะให้ผลที่นำประยุกต์ใช้กับการเขียนโปรแกรมได้ง่าย

ขั้นตอนการพัฒนาโปรแกรม

- ประกอบด้วย 5 ขั้นตอนหลัก (อ้างอิงตามบทที่ 4 ในหนังสือ)
  1. **วิเคราะห์ปัญหา (Analysis)**
  2. **วางแผนและออกแบบ (Planning and Design)**
  3. **การเขียนโปรแกรม (Coding)**
  4. การทดสอบโปรแกรม (Testing)
  5. จัดทำเอกสารและคู่มือการพัฒนาหรือใช้งาน (Documentation)

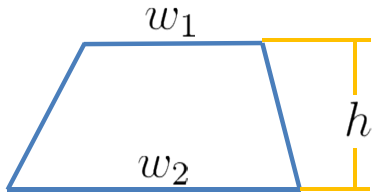
ตัวอย่างปัญหา : คำนวณพื้นที่สี่เหลี่ยมคางหมู

ปัญหา จงวิเคราะห์ปัญหาและเขียนโฟลวชาร์ตสำหรับการหาพื้นที่สี่เหลี่ยมคางหมู เมื่อทราบความยาวด้านคู่ขนานทั้งสองและความสูงของรูปสี่เหลี่ยม

การแก้ปัญหา

1. **วิเคราะห์ปัญหา** : ข้อมูลเข้ามีสามค่า คือ ค่าความยาวด้านคู่ขนานด้านที่หนึ่ง (w1), ค่าความยาวด้านคู่ขนานด้านที่สอง (w2), และค่าความสูง (h) ผลลัพธ์มีหนึ่งอย่าง คือ พื้นที่สี่เหลี่ยม (A)

   ข้อมูลเหล่านี้มีความสัมพันธ์กันตามสมการ $A = \frac{1}{2}(w_1 + w_2)\,h$

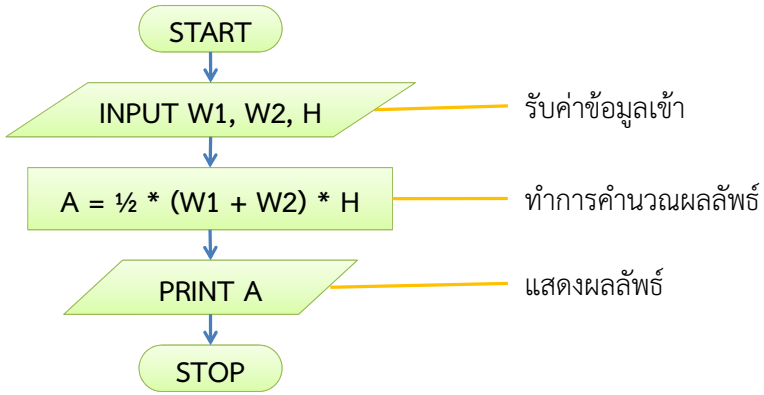


โจทย์ดัดแปลงมาจากตัวอย่าง 4.2 หน้า 45 – 46 ในหนังสือเรียน

ตัวอย่างปัญหา : หาพื้นที่สี่เหลี่ยมคางหมู (2)

2. วางแผนและออกแบบ

จากความสัมพันธ์ระหว่างข้อมูลเข้าและผลลัพธ์ เราสามารถสรุปออกมาเป็นขั้นตอนการทำงานในรูปของโฟลวชาร์ตได้ดังนี้



ตัวอย่างที่ซับซ้อนขึ้น : การตัดเกรด

โจทย์ การตัดเกรดในบางมหาวิทยาลัยจะแบ่งออกเป็นสามระดับคือ ตก, ผ่าน, และ ยอดเยี่ยม โดยมีเกณฑ์การตัดเกรดดังนี้ น้อยกว่า 40 คะแนนคือตก (F) ได้ถึง 40 คะแนนแต่น้อยกว่า 80 คะแนนคือผ่าน (P) และได้ 80 คะแนนขึ้นไปคือยอดเยี่ยม (A) จงวิเคราะห์ปัญหาและเขียนโฟลวชาร์ตสำหรับการตัดเกรด

การแก้ปัญหา

1. **วิเคราะห์ปัญหา** : ข้อมูลเข้ามีอยู่ค่าเดียวคือคะแนนของนักศึกษา (SCORE) ส่วนผลลัพธ์มีอยู่ค่าเดียวเช่นกันคือเกรด (GRADE) ซึ่งมีค่าที่เป็นไปได้สามค่าเท่านั้น คือ ‘F’, ‘P’, และ ‘A’
   ความสัมพันธ์ระหว่างข้อมูลเข้ากับผลลัพธ์ก็คือเราใช้คะแนนเป็นตัวระบุเกรด โดยที่มีเกณฑ์การระบุเกรดที่แน่นอนตายตัวกำหนดมาให้แล้ว ปัญหานี้ไม่ใช่การเอาค่ามาคำนวณโดยตรงแต่เป็นการแยกประเภทค่า

โจทย์ดัดแปลงมาจากตัวอย่าง 4.3 หน้า 46 – 48 ในหนังสือเรียน

ตัวอย่างที่ซับซ้อนขึ้น: การตัดเกรด (2)

2. วางแผนและออกแบบ

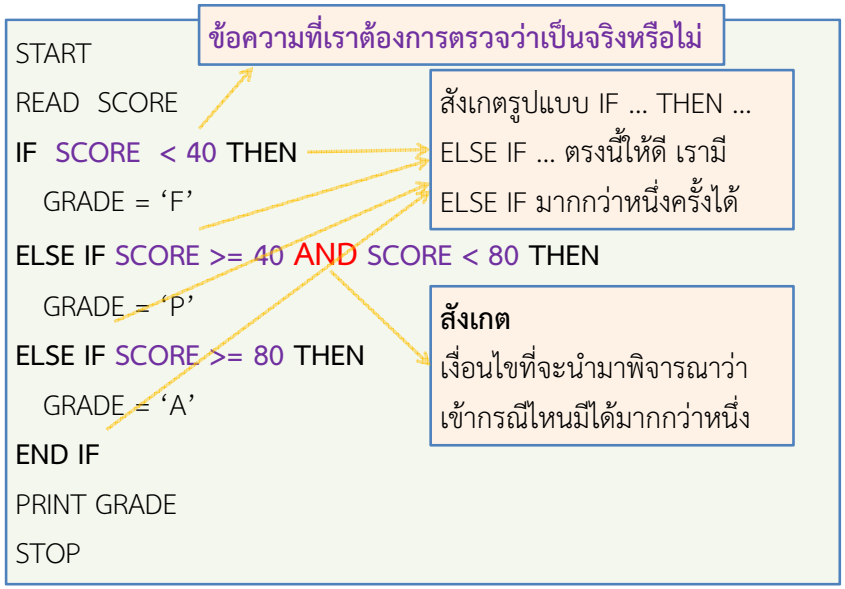
การพิจารณาแยกประเภทค่าไม่ใช่ปัญหาบวกลบคูณหารตัวเลข เป็นปัญหาที่เกี่ยวกับการตัดสินค่า มีการแยกกรณี

รูปแบบของการแยกกรณีทำให้ซูโดโค้ดมีการใช้คำสั่ง

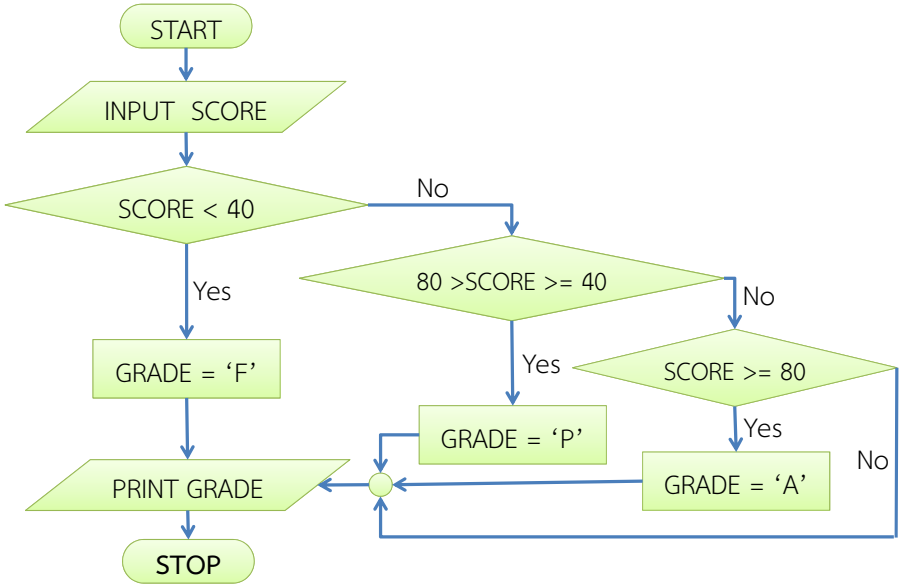
```
IF … THEN
   …
ELSE
   …
END IF
```

และทำให้โฟลวชาร์ตแยกออกเป็นสองทางด้วยการใช้สัญลักษณ์

ซูโดโค้ดของการตัดเกรด

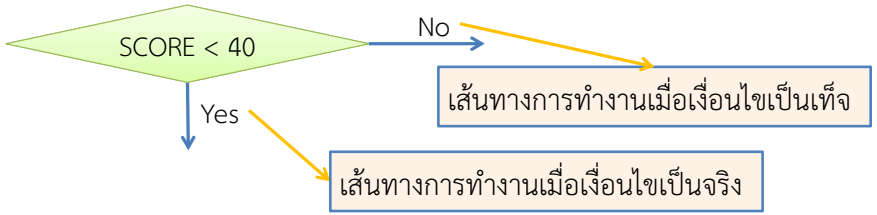


โฟลวชาร์ตของการตัดเกรด



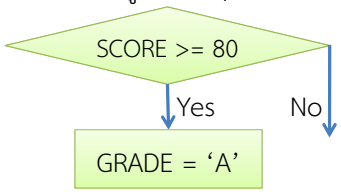
ข้อสังเกตจากโฟลวชาร์ตที่ผ่านมา

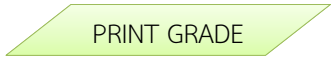
- เวลาแบ่งกรณีเราจะใส่เงื่อนไขไว้ใน 
- ลูกศรจะถูกกำกับด้วยคำว่า Yes กับ No ซึ่งแทนเส้นทางการทำงานเมื่อเงื่อนไขเป็น 'จริง' และเป็น 'เท็จ'



เส้นทางการทำงานในโฟลวชาร์ต

- **คำถาม** จำเป็นหรือไม่ที่จะต้องมีทั้งเส้นทาง Yes และเส้นทาง No พร้อมกัน?
- **คำตอบ** โดยทั่วไปเป็นสิ่งที่จำเป็น เพราะเส้นทางการทำงานจะต้องมีโอกาสไปถึงจุดสิ้นสุดเสมอ
- **ข้อสังเกต** เส้นทางบางอันดูแปลก ๆ เช่น



เส้นทาง 'No' จะไม่มีการกำหนดค่าเกรด แต่สุดท้ายจะพยายามพิมพ์ค่าเกรดออกมาด้วย 

แล้วโปรแกรมข้างบนผิดหรือเปล่า ?

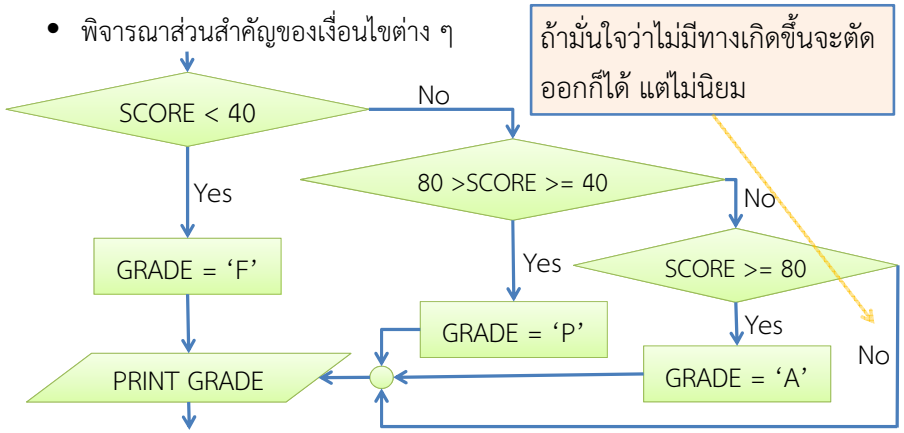
การที่เราคิดจะพิมพ์ค่าเกรดออกมาโดยไม่มีการกำหนดค่าก่อนเป็นสิ่งที่ผิด

- **คำถาม** แต่ถ้าเหตุการณ์ที่เราไม่ได้กำหนดค่าก่อนพิมพ์เป็นสิ่งที่ไม่มีทางเกิดขึ้นล่ะ ?
- **คำตอบ** แสดงว่าโปรแกรมเราไม่ผิด และอันที่จริงโฟลวชาร์ตที่ให้ไปก็ไม่มีที่ผิดด้วย

- **คำถาม** เรื่องนี้มันเป็นอย่างไรกันแน่ ? งง ช่วยอธิบายหน่อย
- **คำตอบ** ดูคำอธิบายหน้าถัดไปได้เลย

โฟลวชาร์ตกับเหตุการณ์ที่ไม่มีทางเกิดขึ้น

- พิจารณาส่วนสำคัญของเงื่อนไขต่าง ๆ



- เราจะพบว่าถ้า SCORE >= 80 เป็นเท็จ แสดงว่า SCORE < 80 ซึ่งการแยกประเภทในสองรอบแรกจัดการไปให้เรียบร้อยแล้ว ดังนั้น No ตรงเงื่อนไข SCORE >= 80 จะไม่มีทางเกิดขึ้นแน่นอน

เปรียบเทียบซูโดโค้ดกับโฟลวชาร์ต

```
START
READ  SCORE
IF  SCORE  < 40 THEN
   GRADE = 'F'
ELSE IF SCORE >= 40 AND SCORE < 80 THEN
   GRADE = 'P'
ELSE IF SCORE >= 80 THEN
   GRADE = 'A'
END IF
PRINT GRADE
STOP
```

ELSE เทียบได้กับเส้นทาง 'No' ในโฟลวชาร์ต

IF อันสุดท้ายนี้ไม่มี ELSE เพราะเรามั่นใจว่าไม่มีทางที่จะมาถึงจุดนี้ได้โดยที่เงื่อนไข SCORE >= 80ไม่เป็นจริง

ซูโดโค้ดที่เทียบเท่ากัน (ให้ผลลัพธ์เหมือนกัน)

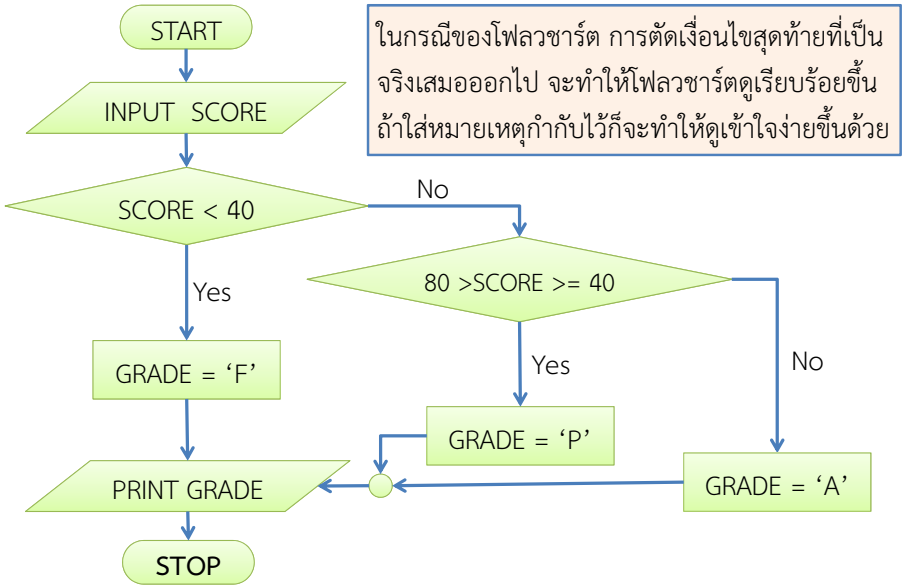
```
START
READ  SCORE
IF  SCORE  < 40 THEN
   GRADE = 'F'
ELSE IF SCORE >= 40 AND SCORE < 80 THEN
   GRADE = 'P'
ELSE
   GRADE = 'A'
END IF
PRINT GRADE
STOP
```

ในกรณีที่เรามั่นใจว่าเงื่อนไขสุดท้ายจะเป็นจริงโดยอัตโนมัติ เราไม่จำเป็นต้องใส่เงื่อนไขอะไรเข้าไปก็ได้ (ลองเอาไปคิดทบทวนดู ถ้าเข้าใจตรงนี้แล้วจะเก่งขึ้น)

แต่คนจำนวนมากก็เลือกที่จะใส่เงื่อนไขสุดท้ายไว้ ทั้งนี้ก็เพื่อความชัดเจนของซูโดโค้ด

ในกรณีที่เงื่อนไขออก ควรจะใส่หมายเหตุกำกับไว้ว่าตรงนี้มันเกิดขึ้นได้เพราะ SCORE >= 80 ทำแบบนี้ก็จะชัดเจนขึ้นด้วย

โฟลวชาร์ตที่เทียบเท่ากัน (ให้ผลลัพธ์เหมือนกัน)



ในกรณีของโฟลวชาร์ต การตัดเงื่อนไขสุดท้ายที่เป็นจริงเสมอออกไป จะทำให้โฟลวชาร์ตดูเรียบร้อยขึ้น ถ้าใส่หมายเหตุกำกับไว้ก็จะทำให้ดูเข้าใจง่ายขึ้นด้วย

มี IF แล้วต้องมี ELSE เสมอหรือไม่ ?

- **ไม่จำเป็น ขึ้นอยู่กับปัญหาที่เราต้องการแก้**
- ถ้าไม่มี ELSE โครงสร้างของซูโดโค้ดก็จะลดรูปมาเป็น

```
IF … THEN
  …
END IF
```

- ตัวอย่าง : จงพิมพ์คำว่า “Hello” ถ้าเลขที่อ่านเข้ามามีค่าเท่ากับ 1

```
START
READ X
IF  X = 1 THEN
   PRINT “Hello”
END IF
END
```


วนทำงานที่คล้ายกันซ้ำหลายรอบ

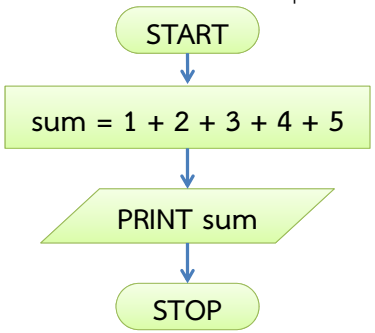
- ในชีวิตจริงของเรา เราพบว่าเราต้องทำงานที่คล้ายกันซ้ำกันหลายรอบ
  - เช่นการทานข้าว เราต้องตักข้าวเข้าปากแล้วลุยเคี้ยว 30 รอบ
  - จากนั้นเราก็ต้องตักข้าวอีก เพราะช้อนเดียวคงไม่พออิ่ม
  - การเคี้ยว 30 รอบนั้นเป็นการทำอะไรซ้ำ ๆ
  - การตักข้าวช้อนที่ 2, 3, 4, … ก็คือการทำซ้ำแบบหนึ่ง
- สังเกตให้ดีว่าแม้งานเราจะดูเหมือนซ้ำ แต่มันใช่ว่าจะเหมือนเดิมโดยสมบูรณ์
  - เช่นการเคี้ยวข้าวครั้งแรก ข้าวยังไม่ค่อยละเอียด แต่ครั้งต่อ ๆ มาข้าวละเอียดขึ้นแล้ว การเคี้ยวจะดูเหมือนง่ายขึ้น
  - การตักข้าวช้อนแรกกับช้อนที่สองใช่ว่าจะได้กับข้าวแบบเดียวกันมา ถึงจะดูเหมือนกันในแง่เป็นการกินข้าว มันก็ไม่ได้เหมือนกันโดยสมบูรณ์

แล้วงานการคำนวณที่วนซ้ำล่ะ

- มีธรรมชาติเป็นเช่นนั้นเหมือนกัน คือเป็นงานที่คล้ายกัน แต่มีอะไรเปลี่ยนแปลงไปบ้าง ใช่ว่าจะต้องเหมือนกันหมด
- เช่น ถ้าเราต้องคิดหาผลบวกของ 1 + 2 + 3 + 4 + 5
  - เราจะพบว่าถ้าเราไม่ได้ใช้สูตรลัดคำนวณเลขอนุกรม เราก็ต้องทำการบวกเลขหลายรอบ
  - ตัวของการบวกเลขแต่ละครั้งก็เหมือนการตักข้าวเข้าปากแต่ละช้อน เลขที่ได้มาแต่ละครั้งใช่ว่าจะเหมือนเดิม และเลขก็บวกสะสมขึ้นเรื่อย ๆ ไม่ต่างอะไรกับการกินข้าวที่กินสะสมเพิ่มขึ้นเรื่อย ๆ
- เนื่องจากงานคำนวณที่ต้องวนซ้ำมีอยู่มากมาย ไม่ใช่แค่การบวกเลข
  → เรามีวิธีอธิบายกระบวนการคำนวณพวกนี้ไว้อยู่แล้ว สำคัญมากด้วย

ตัวอย่าง จงหาผลบวกของ 1 + 2 + 3 + 4 + 5

- แน่นอนว่าด้วยเลขไม่กี่ตัว เราจะบวกเลขตรง ๆ เลยก็ได้ เช่น



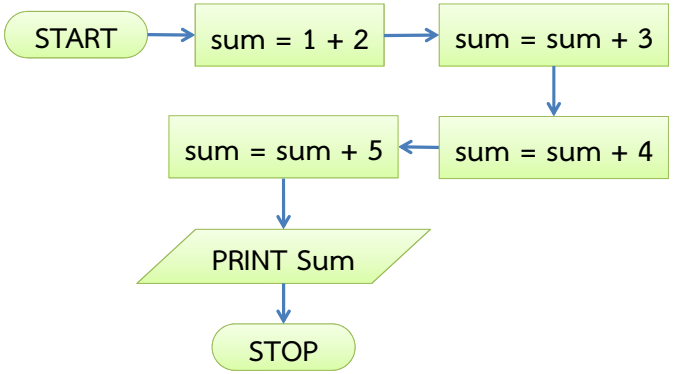
- แต่อย่าลืมว่าถ้าเราต้องบวกเลขกว่าร้อยตัว และอาจจะไม่ใช่อนุกรมที่ตายตัว เรื่องคงไม่ง่าย ทว่าจะให้เริ่มอธิบายเรื่องการวนซ้ำด้วยตัวอย่างที่ท้าทายทันทีก็ใช่ที่ ดังนั้นเราจะทำการรู้จักกับการวนซ้ำจากตัวอย่างนี้แหละ

การวนซ้ำบวกเลขจาก 1 ถึง 5

- ลองสังเกตกลไกในการคิดที่เราทำในหัวของเรา
  - เราเริ่มจากเอา 1 + 2 ได้ผลลัพธ์เป็น 3 แล้วเก็บไว้ในหัว
  - จากนั้นเราก็เอาเลข 3 ที่เก็บไว้มาบวกต่อเพื่อหาค่า 1 + 2 + 3 ทำให้ได้ผลเป็น 3 + 3 = 6 เราก็เก็บเลข 6 นี้ไว้ในหัวของเราอีกที
  - แล้วก็ลุยบวกต่อจะได้ผลเป็น 6 + 4 = 10
  - จากนั้นก็ปิดท้ายด้วยการบวก 5 เป็น 10 + 5 = 15
- จุดที่ควรทราบก็คือ “การหาผลบวกแล้วเก็บไว้ก่อน” จากนั้นค่อยนำผลที่เก็บไว้มาบวกต่อ จุดนี้เป็นเทคนิคพื้นฐานที่สำคัญมาก

การบวกเลขด้วยวิธีเก็บค่าไว้บวกต่อ (แบบกระจอก)

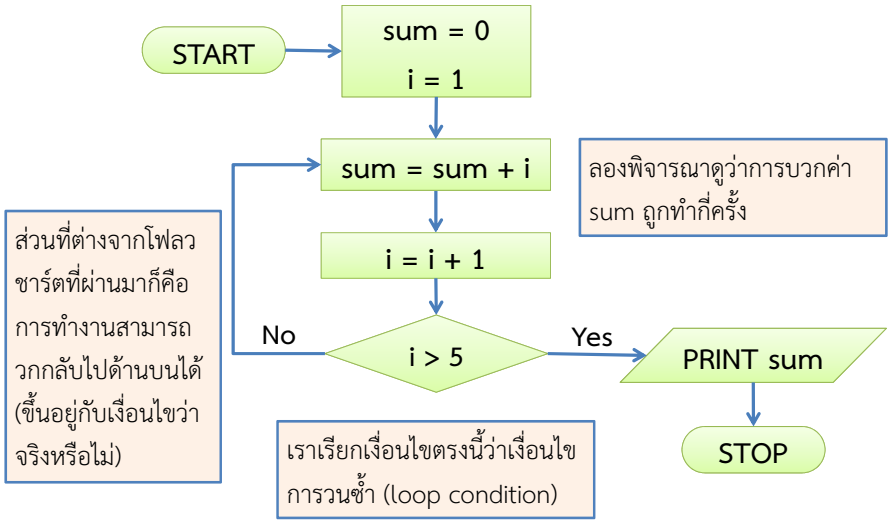
- อย่างที่บอกไว้ก่อนหน้า เราจะไม่ทำอะไรที่ท้าทายทันที แต่จะเริ่มจากเรื่องพื้นฐานก่อน ในที่นี้ก็เหมือนกัน เราจะทำการบวกแบบกระจอกก่อน
- ในตัวอย่างนี้เราจะเก็บไว้ในตัวแปรชื่อ sum และจะบวกใส่มันต่อไปเรื่อย ๆ



การบวกเลขด้วยวิธีเก็บค่าไว้บวกต่อ (แบบเอาจริงแล้ว)

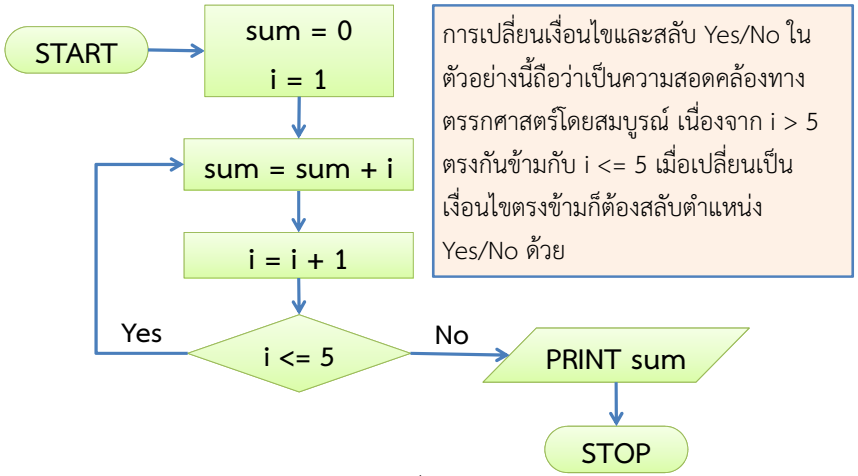
- วิธีที่ไม่ดูกระจอกก็คือเราจะคอยเปลี่ยนตัวบวกไปเรื่อย ๆ แบบอัตโนมัติ
  - คือไม่ต้องคอยใส่เลข 1, 2, 3, 4, และ 5 เอง
  - เราจะใช้ตัวแปรอีกตัวหนึ่งมาคอยนับเลขและเปลี่ยนค่าตัวบวก
  - ใช้ได้ดีกับการบวกที่มีรูปแบบแน่นอนตายตัว
  - ใช้ได้ดีกับงานที่จำเป็นต้องนับรอบการทำงานให้ครบ
  - สามารถประยุกต์ใช้ได้เมื่อจำนวนตัวเลขสูงกว่านี้ทันที
- เรานิยมตั้งค่าตัวแปรนับค่า/นับรอบว่า i แต่จะตั้งเป็นชื่ออื่นที่สื่อความหมายกับงานที่กำลังคำนวณมากกว่านี้ก็นับเป็นความคิดที่ดี
- เรายังนิยมให้ผลบวก sum เริ่มจาก 0 ด้วย ดังนั้นในตัวอย่างนี้ การบวกครั้งแรกจะเริ่มจาก 0 + 1 แทนที่จะเป็น 1 + 2

โฟลวชาร์ตวิธีวนบวกเลข 1 ถึง 5



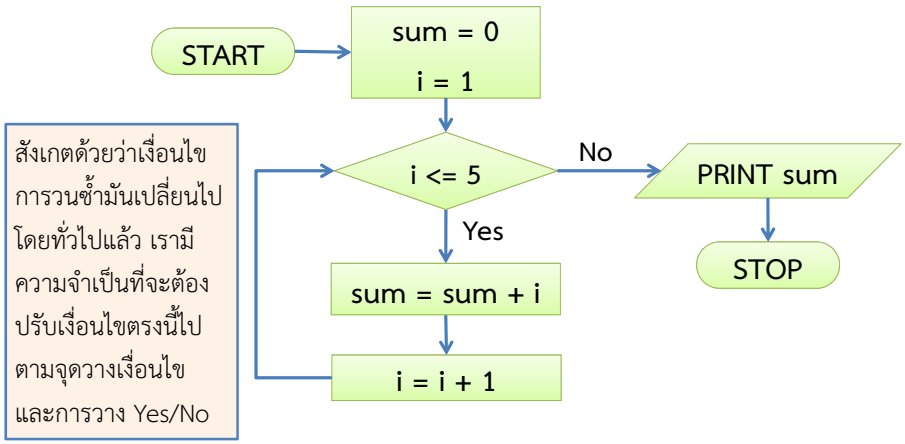
เรื่องน่าคิดเกี่ยวกับเงื่อนไขวนซ้ำ (1)

เงื่อนไขวนซ้ำนั้นเราสามารถตั้งได้หลายแบบและหลายจุด เช่นเราสามารถเปลี่ยนจาก i > 5 ไปเป็น i <= 5 และสลับตำแหน่ง Yes กับ No ได้



เรื่องน่าคิดเกี่ยวกับเงื่อนไขวนซ้ำ (2)

เราอาจจะวางการตรวจเงื่อนไขไว้ทางด้านบนก่อนถึงตัวงานที่ต้องทำซ้ำก็ได้ วิธีนี้จะดูแปลก ๆ สำหรับผู้เริ่มต้น แต่ต่อไปเราจะพบว่ามันเป็นวิธีที่นิยมกว่า



เรื่องน่าสังเกต

- เส้นตีกลับจะครอบตัวเงื่อนไขตรวจสอบและงานที่ต้องทำซ้ำเอาไว้
  - ไม่ว่าจะตั้งตัวตรวจเงื่อนไขวนซ้ำไว้ด้านบนหรือด้านล่างก็ตาม
  - ดังนั้นของที่จะให้ทำซ้ำต้องอยู่ในวงเส้นตีกลับนี้ ถ้าอยู่ข้างนอกแสดงว่าผิด
- ในทางกลับกัน ถ้าของที่จะไม่ให้ทำซ้ำไปอยู่ในวงเส้นตีกลับก็ผิดเหมือนกัน
  - เช่นถ้าเราเอางาน Print sum ไปอยู่ในวง โปรแกรมก็จะพิมพ์ค่า sum ออกมาหลายรอบ
  - ถ้าเราเอา sum = 0 ไปอยู่ในวงค่า sum ก็จะถูกเปลี่ยนให้กลับไปเป็นค่าเดิมอยู่ตลอด
  - ถ้าเราเอา i = 1 ไปไว้ในวง ค่า i ก็จะติดอยู่ที่ 1 กับ 2 อยู่ตลอด วังวนของการทำงานซ้ำจะไม่สามารถหยุดได้ เพราะตรงกับเงื่อนไขทำซ้ำทุกครั้ง

แล้วซูโดโค้ดของการวนซ้ำล่ะ

- ซูโดโค้ดของการวนซ้ำขึ้นอยู่กับตำแหน่งที่เราวางเงื่อนไขการวนซ้ำ
  - ถ้าวางไว้ด้านล่าง เราจะใช้คำสำคัญเริ่มต้นว่า DO จากนั้นตอบท้ายด้วย WHILE ...
  - ถ้าวางไว้ด้านบน (แปลกตอนนี้ ปรกติในวันหน้า) เราจะใช้คู่คำสำคัญว่า WHILE ... DO แล้วตบท้ายด้วย END WHILE เพื่อบอกจุดตีกลับ
  - ตรง ... ที่เขียนไว้ด้านบนคือ เงื่อนไขการวนซ้ำ
- เรื่องซับซ้อนมันมีอยู่ว่า ในซูโดโค้ดทั่วไปจะวนซ้ำเมื่อเงื่อนไขการวนซ้ำเป็นจริง (Yes) และจะเลิกวนเมื่อเงื่อนไขเป็นเท็จ (No)
  - ดังนั้นโฟลวชาร์ตแบบแรกที่ทำให้ดูในหน้า “โฟลวชาร์ตวิธีวนบวกเลข 1 ถึง 5” ทั้งที่ดูเหมือนไม่มีอะไร แต่พอเป็นซูโดโค้ดจะยากขึ้นมาเลย
  - นี่เป็นเหตุผลที่เราเรียนโฟลวชาร์ตก่อน เพราะเราจะมีอิสระในการคิดมากกว่า
  - ยังไงก็อย่าลืมว่าตัวอย่างที่สองแสดงทางแก้ไว้ให้แล้ว คือใช้เงื่อนไขตรงข้ามแทน

ซูโดโค้ดจากตัวอย่างแบบที่สอง

[แบบที่อยู่ในหน้า เรื่องน่าคิดเกี่ยวกับเงื่อนไขวนซ้ำ (1)]

```
START
sum = 0
i = 1
DO
  sum = sum + i
  i = i + 1
WHILE i <= 5
PRINT sum
END
```

บริเวณที่อยู่ตรงช่วง DO รวมจนถึงเงื่อนไขการวนซ้ำของ WHILE ก็คือบริเวณของวงเส้นตีกลับในโฟลวชาร์ตนั่นเอง

ซูโดโค้ดจากตัวอย่างแบบที่สาม

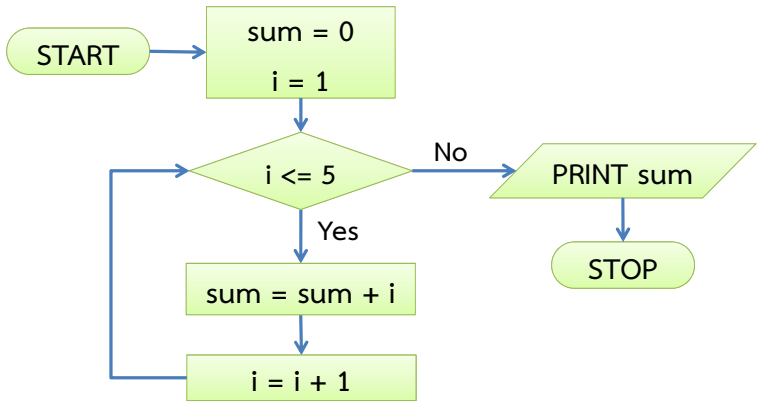
[แบบที่อยู่ในหน้า เรื่องน่าคิดเกี่ยวกับเงื่อนไขวนซ้ำ (2)]

```
START
sum = 0
i = 1
WHILE i <= 5 DO
  sum = sum + i
  i = i + 1
END WHILE
PRINT sum
END
```

บริเวณที่อยู่ตรงช่วง WHILE ไปจนถึง END WHILE ก็คือบริเวณของวงเส้นตีกลับในโฟลวชาร์ตนั่นเอง

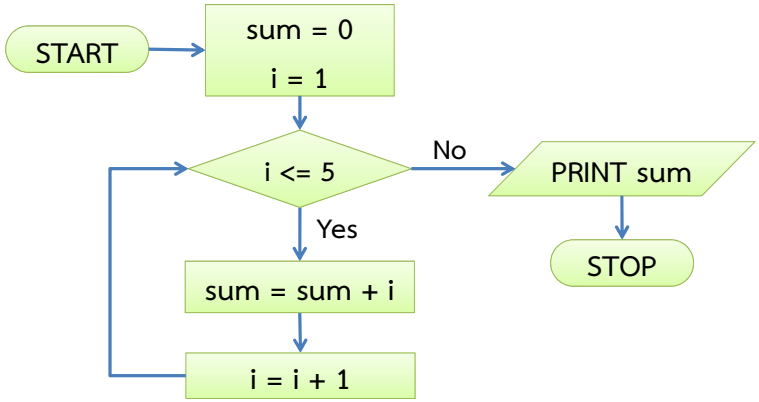
คำถามเพื่อความเข้าใจเกี่ยวกับการวนซ้ำ (1)

ถ้าต้องการบวกเลขจำนวนเต็ม 1 ถึง 100 จะต้องเปลี่ยนโฟลวชาร์ตข้างล่างนี้อย่างไร



คำถามเพื่อความเข้าใจเกี่ยวกับการวนซ้ำ (2)

ถ้าต้องการบวกเลขจำนวนเต็ม 101 ถึง 10,000 จะต้องเปลี่ยนโฟลวชาร์ตข้างล่างนี้อย่างไร



ตัวอย่าง: บวกเลข 10 ค่าที่เราป้อนเข้าไป

โจทย์: จงเขียนโปรแกรมที่รับเลขจำนวน 10 ค่าและพิมพ์รวมของเลขเหล่านั้นออกมา

การแก้ปัญหา

1. วิเคราะห์ปัญหา

เป้าหมาย : เราต้องทำการหาผลบวกของเลขที่ผู้ใช้ป้อนเข้ามาทั้งหมด

ข้อมูลเข้า : อ่านเข้ามาจากผู้ใช้ทีละตัวจำนวน 10 ค่า

ผลลัพธ์ : ผลบวก (sum) ของเลขทั้งหมด

จุดน่าสังเกต : จากตัวอย่างที่ผ่านมาเราค่อย ๆ บวกเลขสะสมค่าเข้าไปเรื่อย ๆ แล้วจึงพิมพ์ผลลัพธ์สุดท้ายออกมาทีเดียว ในงานนี้ก็ทำได้เช่นกัน คือค่อย ๆ รับข้อมูลเข้ามาทีละตัวและก็บวกรอไว้เลย

เริ่มลงมือคิดวิธีการรับค่าและบวกเลข

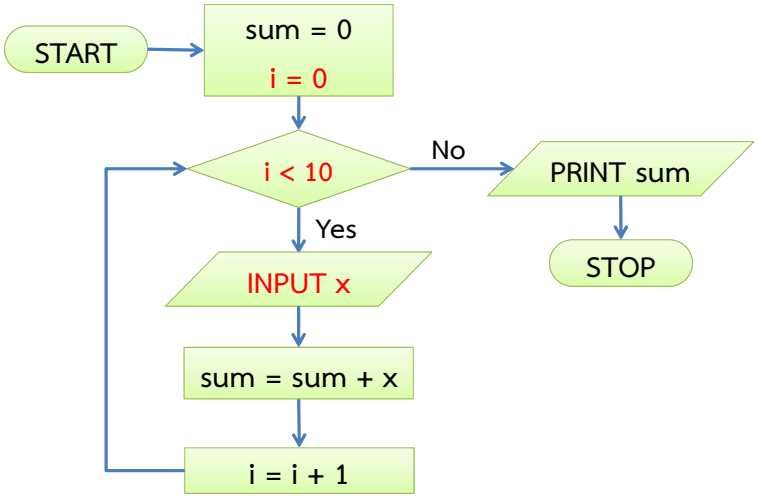
2. วางแผนและออกแบบ

- จากข้อสังเกตเมื่อสักครู่ เดาได้ว่าเราจะใช้วิธีที่คล้าย ๆ เดิมในการจัดการปัญหานี้ นั่นคือแทนที่จะบวกค่า i สะสมเข้าไปใน sum เราจะนำค่าตัวเลขมาจากผู้ป้อนข้อมูลแทน

- แล้วค่า i ล่ะ ยังมีความจำเป็นอะไรอีกหรือไม่?
  - ยังมีความจำเป็น เพียงแต่บทบาทของมันจะถูกนำไปใช้ในการนับว่ารับค่าตัวเลขที่ป้อนเข้ามาไปแล้วกี่ค่า
  - โดยปรกติเราจะนิยมให้ค่า i เริ่มจาก 0 ซึ่งแปลว่า “ยังไม่ได้รับค่าใด ๆ เข้ามา” และเลขของค่า i จะบอกเราว่ารับค่าไปแล้วกี่ค่า ถ้าครบแล้วก็หยุด

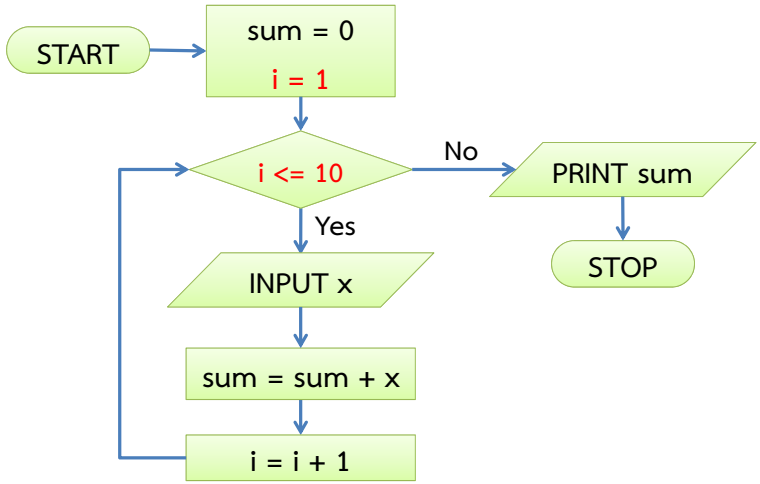
โฟลวชาร์ตสำหรับบวกเลข 10 ค่าที่เราป้อนเข้าไป

โดยรวมแล้วเหมือนเดิมแทบทุกอย่าง



จำเป็นหรือไม่ที่ต้องให้ i เริ่มจากศูนย์

ไม่จำเป็น มันสำคัญเฉพาะเรื่องที่ว่าเราจะตีความค่า i ว่าอย่างไร และหากเราเปลี่ยนจุดเริ่มของค่า i แล้ว เงื่อนไขการวนซ้ำก็จะต้องสอดคล้องกันด้วย



ตัวอย่าง: หาผลบวกของเลขคู่ในช่วงค่า 1 ถึง 100

โจทย์ จงเขียนโฟลวชาร์ตแสดงการหาผลบวกของเลขคู่ที่มีค่าอยู่ในช่วง 1 ถึง 100 (โจทย์จากหนังสือเรียน ตัวอย่างที่ 4.4 หน้า 48 – 50 ในหนังสือเรียน)

การแก้ปัญหา

1. วิเคราะห์ปัญหา

เป้าหมาย : เราต้องทำการหาผลบวกของเลขคู่ในช่วง 1 ถึง 100

ข้อมูลเข้า : ไม่มีการอ่านข้อมูลเข้าจากผู้ใช้ เพราะช่วงค่าถูกกำหนดลงไปในโจทย์ แต่เราต้องจำแนกประเภทของค่าต่าง ๆ ให้ได้ ว่าเป็นเลขคู่หรือเลขคี่

ผลลัพธ์ : ผลบวก (sum) ของเลขคู่ทั้งหมด

จุดน่าสังเกต : มีการแยกประเภทเลขคู่เลขคี่ แสดงว่าน่าจะมีการใช้ IF ... THEN ... ELSE ... หรือไม่ก็การตัดสินเงื่อนไขด้วย 

แต่ข้อนี้ต้องการให้เขียนแค่โฟลวชาร์ตเท่านั้น ดังนั้นไม่ต้องใส่ใจเรื่องซูโดโค้ด

คิดวิธีหาผลบวกเลขคู่ทั้งหมด

การวิเคราะห์ปัญหายังไม่ถึงจุดยุติ ตราบใดที่เรายังไม่ทราบความสัมพันธ์ระหว่างข้อมูลต่าง ๆ กับผลลัพธ์ที่เราต้องการ

- **คำถาม** เลขในช่วง 1 ถึง 100 มีความเกี่ยวข้องอย่างไรกับผลลัพธ์ ?
- **คำตอบ** มีเลขบางตัว คือเลขคู่ที่เราจะต้องนำมาบวกกัน ส่วนเลขคี่นั้น เราไม่ต้องใส่ใจ
- **คำถาม** แล้วเราจะแยกเลขคู่กับเลขคี่ได้อย่างไร
- **คำตอบ** เลขคู่คือเลขที่หารด้วยสองลงตัว (หารด้วยสองแล้วเหลือเศษศูนย์) ส่วนเลขคี่หารด้วยสองไม่ลง (หารด้วยสองแล้วเหลือเศษที่ไม่เป็นศูนย์)
- **คำถาม** ทำอย่างไรจึงจะไล่นับค่าจาก 1 ไปถึง 100 ได้
- **คำตอบ** ต้องมีตัวนับ (count) มาทำการไล่นับค่าขึ้นจาก 1 ไปถึง 100

วางแผนการหาผลบวกของเลขคู่

ณ ตอนนี้เราวิเคราะห์ความสัมพันธ์ของสิ่งต่าง ๆ พร้อมทั้งประเด็นพื้นฐานที่จำเป็นไปหมดแล้ว เราพร้อมที่จะหาผลบวกด้วยการเอาสิ่งต่าง ๆ มาประกอบกันเป็นลำดับที่เหมาะสมแล้ว

2. วางแผนและออกแบบ

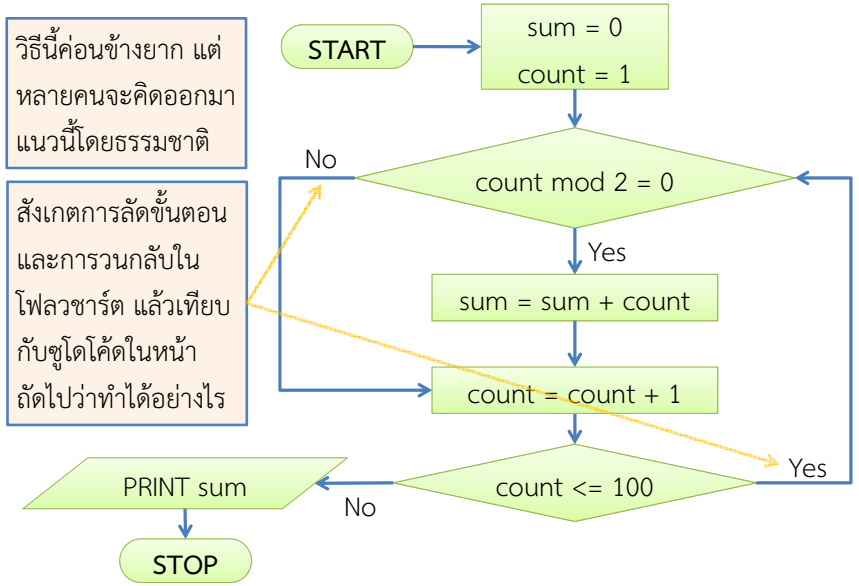
การหาเศษทำได้ด้วยตัวดำเนินการทางคณิตศาสตร์ที่เรียกว่ามอดูโล (Modulo)

เราจะเขียน **x mod y** ว่าเป็น**เศษจากการหาร x ด้วย y**

เช่น 5 mod 2 จะได้ค่าเท่ากับ 1 เพราะเศษจากการหาร 5 ด้วย 2 คือ 1 และ 8 mod 3 จะได้ผลลัพธ์เท่ากับ 2 เป็นต้น

เราจะใช้ตัวดำเนินการนี้ในการวางแผนและออกแบบอัลกอริทึม และอธิบายขั้นตอนออกมาเป็นโฟลวชาร์ตตามที่โจทย์ต้องการ

โฟลวชาร์ตของการบวกเลขคู่จาก 1 ถึง 100



ซูโดโค้ดสำหรับบวกเลขคู่

ถึงแม้โจทย์จะไม่ได้ถามถึงซูโดโค้ด แต่เราจะศึกษาเพื่อเสริมความเข้าใจ

ข้อความที่เราต้องการตรวจว่าเป็นจริงหรือไม่

```
START
sum = 0
count = 1
WHILE  count <= 100  DO
   IF  count  mod  2 = 0  THEN
      sum = sum + count
   END IF
   count = count+ 1
END WHILE
END
```

```
WHILE … DO
   …
END WHILE
```
เป็นโครงสร้างสำหรับการวนซ้ำ เพื่อทำงานที่คล้ายเดิมในซูโดโค้ด

```
IF … THEN
   …
END IF
```
ไม่จำเป็นต้องมี ELSE อยู่ด้วยก็ได้ เพราะเราจะไม่ทำอะไรถ้าเศษจากการหารไม่เป็น 0

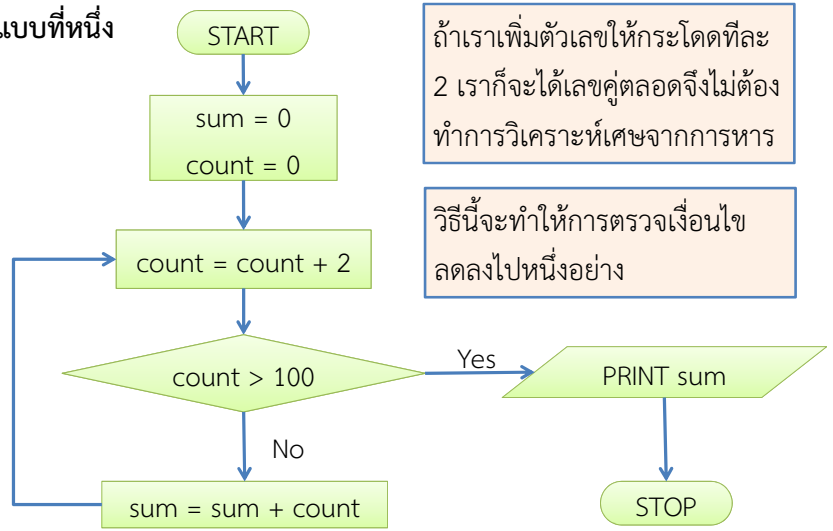
วิธีแก้ปัญหาอีกแบบ

- ปัญหาหนึ่งอาจจะมีวิธีแก้หลายวิธี วิธีถัดไปเป็นวิธีที่หนังสือเรียนใช้ และไม่จำเป็นต้องทำการหาเศษจากการหารด้วยสอง
- วิธีนี้ใช้ประโยชน์จากข้อเท็จจริงที่ว่า ‘เลขคู่สลับกับเลขคี่เสมอ’ ดังนั้นเราไม่ต้องเพิ่ม count ทีละ 1 แต่เพิ่มทีละ 2 เพื่อข้ามไปหาเลขคู่ตัวถัดไปได้ทันที
- กล่าวคือ ถ้า count = 4 เราไม่ต้องเพิ่ม count ให้กลายเป็น 5 แต่เพิ่ม count ให้กลายเป็น 6 เลยด้วยการใช้
  count = count + 2 แทน count = count + 1

สังเกตออกหรือไม่ว่าตอนที่ count = 4 มันเป็นเลขคู่ พอสั่งข้ามไป 6 มันก็ยังเป็นเลขคู่อยู่ดี และพอสั่งข้ามไป 8, 10, 12, …, 100 ก็เป็นเลขคู่ตลอด

โฟลวชาร์ตสำหรับการบวกเลขคู่โดยไม่ต้องหาเศษ

แบบที่หนึ่ง



ถ้าเราเพิ่มตัวเลขให้กระโดดทีละ 2 เราก็จะได้เลขคู่ตลอดจึงไม่ต้องทำการวิเคราะห์เศษจากการหาร

วิธีนี้จะทำให้การตรวจเงื่อนไขลดลงไปหนึ่งอย่าง

ซูโดโค้ดของวิธีที่ไม่มีการหาเศษ (1)

แบบที่หนึ่ง (ซูโดโค้ดนี้มีที่แตกต่างจากโฟลวชาร์ตอยู่เล็กน้อย)

```
START
sum = 0
count = 0
DO
   count = count + 2
   sum = sum + count
WHILE  count < 100
PRINT sum
END
```

DO … WHILE … เป็นโครงสร้างสำหรับการวนซ้ำที่การตรวจเงื่อนไขจะทำหลังจบรอบการทำงาน

สังเกตด้วยว่า WHILE … DO … END WHILE จะตรวจเงื่อนไขก่อนทำงานในแต่ละรอบ

เข้าใจหรือไม่ว่า ทำไมถึงใช้ < แทนที่จะเป็น <=

ซูโดโค้ดแบบที่ตรงกับโฟลวชาร์ตโดยแท้จริง

```
START
sum = 0
count = 0
DO
   count = count + 2
   IF  count > 100  THEN
      BREAK
   END IF
   sum = sum + count
WHILE  0 < 1
PRINT sum
END
```

ลำดับการทำงานจะเหมือนกับในโฟลวชาร์ตทุกอย่าง เงื่อนไขการเปรียบเทียบเพื่อหยุดการวนซ้ำก็เหมือนกัน แต่มีเรื่องที่ดูประหลาดเข้าใจยากปนอยู่ด้วย

เงื่อนไขเหมือนในโฟลวชาร์ตทุกประการ

คำสั่ง BREAK เป็นการหยุดการวนซ้ำทันที

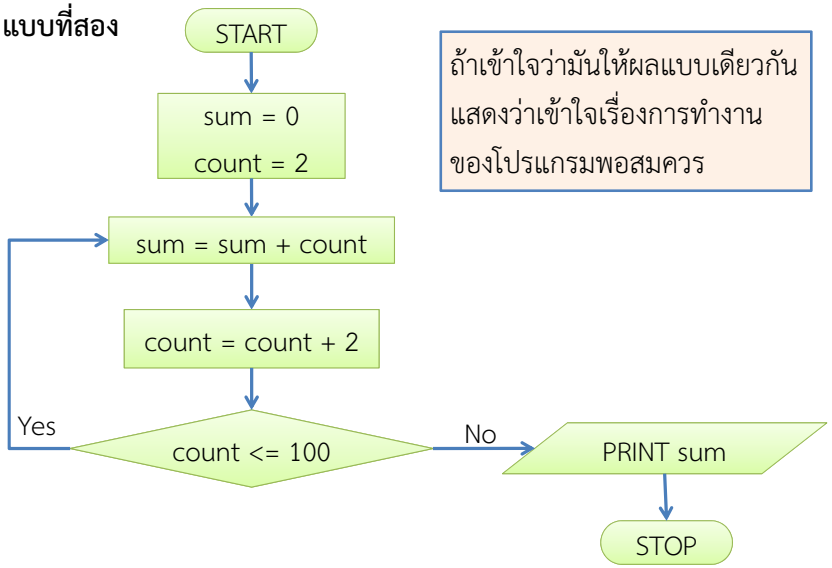
ในโฟลวชาร์ตจะวนกลับเสมอเมื่อทำคำสั่ง sum = sum + count เสร็จ เราจึงนำเงื่อนไขที่เป็นจริงเสมอมาใส่ไว้ตรงนี้

เรื่องนี้สอนให้รู้ว่า ...

- การวนซ้ำที่มีจุดตรวจสอบเงื่อนไขอยู่ตรงกลาง ไม่ได้อยู่ที่ด้านบนสุดหรือล่างสุดจะต้องอาศัยคำสั่ง break เพื่อให้หยุดลูปกลางทางได้
- คำสั่ง break เป็นคำสั่งที่ควรอยู่ภายใต้เงื่อนไขบางอย่าง เพราะถ้ามันอยู่โดด ๆ แสดงว่าการคำนวณต้องมาถึงมันอย่างเลี่ยงไม่ได้ และลูปก็ต้องหยุดทักครั้งไป
- ในโฟลวชาร์ตมันดูง่าย ๆ เหมือนไม่มีอะไร แต่พอเปลี่ยนมาเป็นโค้ดแล้วมันมีเรื่องต้องคิดจุกจิกเพิ่มขึ้นมาทันที
- นี่เป็นอีกเหตุผลที่เราเรียนนเรื่องโฟลวชาร์ตก่อนการเขียนโค้ด เพราะเราจะมีอิสระในแนวทางการคิดสูงกว่า ไม่ถูกบีบด้วยกลไกทางภาษาเขียนโปรแกรมมากนัก

หลักการเดิมแต่ปรับแต่งเล็กน้อยก็ให้ผลแบบเดียวกันได้

แบบที่สอง



ถ้าเข้าใจว่ามันให้ผลแบบเดียวกัน แสดงว่าเข้าใจเรื่องการทำงานของโปรแกรมพอสมควร

ซูโดโค้ดของวิธีที่ไม่มีการหาเศษ (2)

แบบที่สอง (เอาไปคิดทบทวนเป็นการบ้านด้วย สำคัญมาก)

```
START
sum = 0
count = 2
DO
   sum = sum + count
   count = count + 2
WHILE  count <= 100
PRINT sum
END
```

ข้อสังเกต : การจัดโฟลวชาร์ตใหม่บางครั้งจะทำให้เขียนซูโดโค้ดที่เทียบเท่ากันได้ง่ายขึ้น นั่นคือจะเขียนโค้ดภาษาซีง่ายขึ้นด้วย

ถ้าหากตำแหน่งการวนกลับเป็นการตรวจเงื่อนไขจะทำให้ง่าย เพราะจะสอดคล้องกับ DO ... WHILE หรือถ้าการวนซ้ำเริ่มด้วยการตรวจเงื่อนไขก็จะเขียนโค้ดง่ายเช่นกัน เพราะจะตรงกับ WHILE ... DO ... END WHILE

แต่ก็อย่ากังวลกับเรื่องนี้ เพราะไม่ว่าจะเป็นแบบไหนก็มีทางออกในซูโดโค้ดและภาษาซีเสมอ

การวนซ้ำในโฟลวชาร์ตและในซูโดโค้ด

- การเขียนโฟลวชาร์ตเพื่อแสดงลำดับการคิดการทำงานเป็นที่นิยมกว่าในระดับพื้นฐาน
  - สังเกตได้เลยว่า เราอยากจะหยุดวนซ้ำยังไงก็ได้ มันดูง่ายตลอด
  - แต่พอจะมาคิดในรูปแบบซูโดโค้ดปรากฏว่ามันชวนงงเหลือเกิน
- แต่การเขียนโฟลวชาร์ตใช้เนื้อที่หน้ากระดาษเยอะมาก
  - ➢ ในระดับสูงขึ้น เราจะไม่ใช้โฟลวชาร์ต เพราะถือว่าทุกคนเข้าใจหมดแล้ว
- ที่ทุกคนเข้าใจกันหมดก็เพราะว่าแท้จริงแล้วเหตุการณ์วนซ้ำหรือหยุดทำซ้ำมันมีรูปแบบรวมกันแล้วแค่สี่แบบ
  - เราจะมาแจกแจงรูปแบบที่เป็นไปได้ทั้งสี่แบบนี้ในภายหลังก่อนสอบกลางภาค
  - ถ้าแต่ก่อนใครไม่เข้าใจ ก็อาจถึงกลับคิดว่า 'เทอมที่แล้วเราทำอะไรไปเนี่ย'

คำถามส่งท้าย

ที่จริงแล้วซูโดโค้ดสำหรับวิธีที่ต้องหาเศษ ไม่ได้สอดคล้องกับกับโฟลวชาร์ต คือ มีลำดับการเปรียบเทียบที่แตกต่างกันอยู่บ้าง

จงแก้ซูโดโค้ดดังกล่าวให้สอดคล้องกับโฟลวชาร์ตโดยสมบูรณ์

==========

มีแบบฝึกหัดอยู่ท้ายบทที่ 4 ของหนังสือ อย่าลืมเอาไปทำด้วยตนเอง จากนั้นตรวจคำตอบกับเฉลยที่อยู่ท้ายเล่ม (การทำแล้วตรวจคำตอบด้วยตนเองนเป็นวิธีมาตรฐานในการเรียนรู้ที่ดี)

เรื่องควรใส่ใจ

- การวิเคราะห์ปัญหามีพื้นฐานอยู่บนกระบวนการคิดที่คล้ายคณิตศาสตร์
- การฝึกฝนเป็นสิ่งที่จำเป็น เราต้องฝึกทำโจทย์วิชาแคลคูลัสอย่างไร เราก็ต้องฝึกทำโจทย์การเขียนโปรแกรมอย่างนั้น
- พวกเราก็ต้องคิดแก้โจทย์ให้เป็น ถ้าแก้ปัญหาไม่เป็นก็จะไม่ผ่านวิชานี้
- ขอให้นักศึกษาทราบว่าตัวเองคือนักศึกษา กล่าวคือพวกเราต้องเรียนเป็นอาชีพ ออกจากนอกห้องเรียนแล้วก็ต้องเรียนต่อด้วยตนเอง
- โดยปรกติแล้วในวันหนึ่ง ๆ พวกเราควรใช้เวลาไปในเรื่องที่เกี่ยวกับการเรียนประมาณ 8 ชั่วโมง (เรื่องธรรมดาสำหรับการเรียนในสายวิทยาศาสตร์)
- อย่าคิดว่าเกียจคร้านแล้วจะรอด เพราะถ้าเกียจคร้านแล้วรอด อาจารย์และพ่อแม่เราคงไม่บอกให้เราขยัน ดังนั้นเราต้องขยัน เพราะไม่มีทางอื่นแล้วจริง ๆ

สรุปเนื้อหาสาระ

- คอมพิวเตอร์ส่วนบุคคลรับคำสั่งเราผ่านทางคีย์บอร์ดและเมาส์
- เพื่อที่จะสื่อสารกันได้จึงต้องมีโปรแกรมที่อธิบายคำสั่งต่าง ๆ
- โปรแกรมสำเร็จรูปที่เราเห็นอยู่ในรูปแบบภาษาเครื่องเรียบร้อยแล้ว
- ส่วนพวกเราต้องศึกษาและเขียนในรูปภาษาโปรแกรมก่อน (คือภาษาซี) จากนั้นจึงแปลงภาษาโปรแกรมไปเป็นภาษาเครื่องทีหลัง
- เนื่องจากภาษาโปรแกรมมีความเคร่งครัดในหลักไวยากรณ์มาก เราจึงต้องอธิบายขั้นตอนต่าง ๆ อย่างชัดเจนเป็นลำดับที่ถูกต้อง
- เราต้องใส่ใจกับการวิเคราะห์ปัญหาและการวางแผนการเขียนโปรแกรม อย่างเป็นระบบ เพื่อทำให้เราสามารถเขียนโปรแกรมได้อย่างถูกต้อง
- ขั้นตอนการทำงานของโปรแกรมมักถูกอธิบายในรูปซูโดโค้ดและโฟลวชาร์ต