# Program = Algorithms + Data Structures

กฤษณะ ชินสาร

Krisana Chinnasarn

Departmant of Computer Science Faculty of Science

Burapha University

Bangsaen Chonburi Thailand, 20131

Tel (6638)745-900 ext 3060-3062

Email: krisana@docsavage.compsci.buu.ac.th

## 1. บทนำ

โลกมนุษย์มีการเปลี่ยนแปลงอย่างต่อเนื่องไม่เว้นแม้แต่วินาทีเดียว เช่นเดียวกับวิทยาการและความรู้ที่มนุษย์เราใช้ศึกษาในแต่ละวันก็จะมีการเปลี่ยนแปลงไปตามเวลา ซึ่งเครื่องคอมพิวเตอร์ที่เรามีใช้อยู่ในปัจจุบันก็คือผลผลิตจากวิทยาการทางด้านคณิตศาสตร์และวิศวกรรม และก็เป็นที่ยอมรับกันโดยทั่วไปว่าเครื่องคอมพิวเตอร์ได้เข้ามามีบทบาทในชีวิตของเรา มันสามารถช่วยในการจัดการสิ่งต่างๆ ในชีวิตอย่างชนิดที่ว่าเรายากที่จะปฏิเสธความสำคัญของมัน แต่ที่กล่าวมานั้นคือผลของการประมวลผลของเครื่องคอมพิวเตอร์ สิ่งที่ต้องการกล่าวถึงในบทความครั้งนี้ ก็คือ พื้นฐานแห่งการประมวลผลของเครื่องคอมพิวเตอร์

ในบทความนี้คณะผู้เขียนจะได้นำเสนอหลักการพื้นฐานของการพัฒนาโปรแกรมเพื่อการทำงานของเครื่องคอมพิวเตอร์เป็นหลัก แต่เพื่อความเข้าใจที่ตรงกันในบทความ ผู้เขียนจึงได้หยิบยกหลักการพื้นฐานของระบบคอมพิวเตอร์ขึ้นมากล่าวเป็นหัวข้อแรกก่อน จากนั้นก็จะกล่าวถึงการพัฒนาโปรแกรม และท้ายสุดของบทความนี้ก็จะเป็นบทสรุปของการพัฒนาโปรแกรม

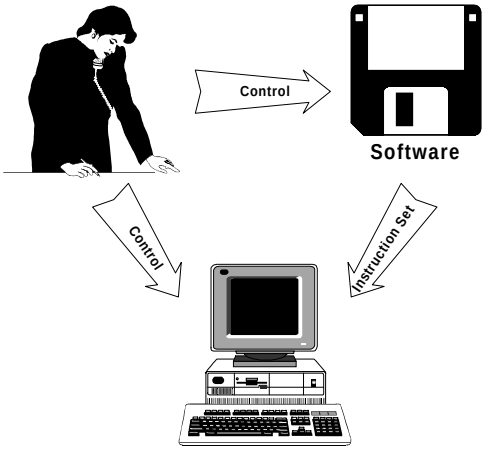
## 2. ระบบคอมพิวเตอร์

ระบบคอมพิวเตอร์จะประกอบด้วยองค์ประกอบ 3 ส่วนหลัก คือ ส่วนตัวเครื่อง (Hardware) ส่วนโปรแกรม (Software) และส่วนของผู้ใช้ (User) ซึ่งทั้ง 3 ส่วนก็จะมีโครงสร้างและหน้าที่ที่แตกต่างกัน แต่เมื่อรวมกันเข้าเป็นระบบเดียวกันแล้วก็คือให้เกิดความสะดวกสบายในการทำงานของคน สามารถทำงานแทนคนในงานที่เสี่ยงอันตรายได้

ส่วนตัวเครื่อง ก็คืออุปกรณ์ต่างๆ ที่ประกอบกันเป็นเครื่องคอมพิวเตอร์ เช่น จอภาพ คีย์บอร์ด เครื่องพิมพ์ และตัวเครื่อง เป็นต้น ซึ่งผู้เขียนคิดว่าท่านผู้อ่านคงทราบถึงหน้าที่ของแต่ละอุปกรณ์ในส่วนของตัวเครื่องนี้เป็นอย่างดีแล้ว ดังนั้นผู้เขียนก็จะไม่ขอกล่าวถึงอีกเพื่อเป็นการประหยัดเวลาและกระดาษ

<u>ส่วนโปรแกรม</u> ก็คือชุดคำสั่งสำหรับที่จะสั่งให้ส่วนตัวเครื่องแต่ละส่วนทำงาน แล้วถูกนำมาเขียนเรียงต่อเนื่องกัน ตามรูปแบบของตัวแปลภาษา (Compiler or Interpreter) แต่ละประเภทที่กำหนดไว้

<u>ส่วนผู้ใช้</u> จะมีหน้าที่ในการควบคุมการสั่งการและการทำงานระบบคอมพิวเตอร์ทั้งระบบ นอกจากนี้ยังมีหน้าที่ในการเฝ้าระวัง และตรวจจับข้อผิดพลาดในการทำงาน รวมทั้งคอยตรวจจับบุคคลไม่พึงประสงค์ที่เมื่อเขาเข้ามาอยู่ในระบบคอมพิวเตอร์ของเราแล้วอาจก่อความเสียหาแก่ระบบของเราได้




ภาพที่ 1 องค์ประกอบของระบบคอมพิวเตอร์

ได้มีอาจารย์ของผู้เขียนท่านหนึ่ง (ผู้เขียนจบ วิชาเอกสถิติ จาก มศว.มหาสารคาม ปัจจุบัน คือ มหาวิทยาลัยมหาสารคาม) ได้กรุณาอธิบายพร้อมยกตัวอย่างถึงองค์ประกอบของระบบคอมพิวเตอร์ไว้อย่างชัดเจนและทำให้เห็นภาพว่า ให้พิจารณาระบบคอมพิวเตอร์หนึ่งระบบเหมือนกับรถยนต์คันหนึ่ง ในส่วนตัวเครื่อง (Hardware) ก็คือ ตัวรถนี้เอง ซึ่งตัวรถก็จะประกอบด้วย ตัวถังรถ ล้อ เครื่องยนต์ พวงมาลัย ช่วงล่างทั้งหมด ฯลฯ ซึ่งอุปกรณ์แต่ละส่วนนั้นก็จะมีหน้าที่ที่แตกต่างกัน อุปกรณ์ทุกอย่างต้องถูกนำมารวมกันจึงจะถือว่าเป็นรถยนต์ จะขาดส่วนใดส่วนหนึ่งไม่ได้ ส่วนโปรแกรม (Software) อาจารย์ท่านได้ยกตัวอย่างไว้ว่ามันก็คือน้ำมันรถยนต์ทั้งหลาย ซึ่งจะคอยเข้าไปหล่อเลี้ยงอุปกรณ์แต่ละส่วนให้มันสามารถทำงานประสานกัน ส่วนผู้ใช้ (User) ก็คือคนขับรถ ซึ่งคนขับนอกจากกว่าเขาจะมีหน้าที่ในการขับรถยนต์แล้ว เขายังมีหน้าที่คอยดูว่าอุปกรณ์แต่ละตัวทำงานปกติหรือไม่ น้ำมันจะหมดหรือยัง เป็นต้น ถ้าหากว่ารถยนต์ขาดส่วนใดส่วนหนึ่งไปใน 3 ส่วน รถยนต์ก็จะวิ่งไม่ได้เป็นแน่แท้

**3. Program = Algorithms + Data Structures**

ในบทความตอนนี้ผู้เขียนจะได้กล่าวถึงเฉพาะในส่วนการสร้างคอมพิวเตอร์โปรแกรม เพื่อสั่งให้เครื่องคอมพิวเตอร์ทำงาน ความจริงแล้วตามคำนิยามของคำว่า “โปรแกรมคอมพิวเตอร์ หมายถึง เซตของชุดคำสั่งสำหรับการสั่งให้คอมพิวเตอร์ประมวลผล” เมื่อเรารู้ว่านิยามเป็นเช่นนี้แล้ว แสดงว่าถ้าเราอยากให้คอมพิวเตอร์ทำงานอะไร เราก็ไปเปิดหนังสือเกี่ยวกับการโปรแกรม (Programming) แล้วก็เลือกหาชุดคำสั่งที่เราพิจารณาเห็นแล้วว่าจะสั่งให้คอมพิวเตอร์ทำงานได้แล้วก็มาวางเรียงต่อเนื่องกันก็น่าจะเพียงพอแล้วใช้ไหม ผู้เขียนขอบอกตามตรงเลยว่า ความคิดของท่านนั้นผิดถนัด ความคิดของท่านนั้นสามารถได้กับเพียงแค่โปรแกรมสำหรับพิมพ์ข้อความ 2-3 บรรทัดออกจอภาพเท่านั้น ไม่สามารถเอาไปใช้กับระบบงานใหญ่ๆ ได้ ยิ่งร้ายไปกว่านั้นถ้าท่านเป็นหัวหน้าโปรเจคแล้วท่านก็เอาแนวคิดที่กล่าวมาข้างต้นไปใช้กับการบริหาร

โปรเจคของท่าน ผู้เขียนขอรับประกันว่า โปรเจคของท่านนั้นพร้อมที่จะล้มตลอดเวลา หรือแม้ว่าจะไม่ล้มก็จะปรากฏว่าโปรแกรมของท่านจะอุดมสมบูรณ์ไปด้วยความผิดพลาดของการเขียนโปรแกรม หรือที่คนคอมพิวเตอร์เขาชอบพูดกันเป็นภาษาอังกฤษว่า “โปรแกรมท่านมีแต่ Bug” ถ้าจะใช้งานได้ท่านต้องหาทางเอาความผิดพลาดออกไปก่อน หรือเรียกว่าการ Debug ผมขอบอกตรงๆ ว่า ผมคนหนึ่งแหล่ะที่จะไม่ขอเข้าร่วมการสังฆกรรมรวมญาติในครั้งนี้กับท่าน

การเขียนโปรแกรมถือว่าเป็นศาสตร์แขนงหนึ่งซึ่งมีความสำคัญอย่างยิ่งต่อระบบคอมพิวเตอร์ ดังจะเห็นได้จากรายวิชาที่ทำการเปิดสอนในระดับปริญญาตรีในสาขาวิทยาการคอมพิวเตอร์ซึ่งส่วนใหญ่แล้วจะเป็นวิชาที่เน้นถึงเทคนิคการโปรแกรมแบบต่างๆ สิ่งหนึ่งที่น่าสนใจมากกว่าเทคนิคในการเขียนโปรแกรม ซึ่งควรแก่การที่โปรแกรมเมอร์มือใหม่ ควรจะพิจารณาและถือปฏิบัติ ก็คือ โครงสร้างของโปรแกรม (Program Structure) ซึ่งจะมีหลักการที่ต้องพิจารณา 3 ประการง่ายๆ คือ

-Sequential คือ ลำดับในการเขียนหรือการวางเซตของชุดคำสั่งในโปรแกรม ซึ่งผู้ที่ทำการโปรแกรมต้องเรียงลำดับของการทำงานของชุดคำสั่งจากบนลงล่างและจากซ้ายไปขวา

-Condition คือ เงื่อนไขในการประมวลผลของโปรแกรม ในเงื่อนไขต้องมีการกำหนดไว้อย่างชัดเจน

-Iteration คือ การกระทำซ้ำของโปรแกรม ลักษณะเด่นของการทำงานด้วยเครื่องคอมพิวเตอร์คือมันสามารถทำงานเดิมได้หลายๆ รอบโดยไม่เคยบ่นและเบื่องาน และทุกรอบที่มันทำงานก็จะให้ผลของการทำงานได้ถูกต้องเหมือนเดิมทุกรอบ (ภายใต้เงื่อนไขที่ว่าข้อมูลที่นำเข้าสู่ระบบคอมพิวเตอร์ต้องเป็นข้อมูลที่ถูกต้องเหมาะแก่การประมวลผลด้วย)

ถึงตอนนี้ผมคิดว่าท่านผู้อ่านคงสามารถเขียนโปรแกรมคอมพิวเตอร์ขนาดเล็กๆ ภายใต้หลักการทั้ง 3 ข้อของการเขียนโปรแกรมแบบโครงสร้างได้บ้างไม่มากก็น้อย (ผมสมมติว่าท่านมีความรู้เกี่ยวกับชุดคำสั่งอยู่บ้างพอสมควรแล้วน่ะครับ) แต่ในมุมมองของผมที่ได้เรียนคณิตศาสตร์ และคอมพิวเตอร์มามากพอสมควร ผมอยากบอกกับท่านว่าท่านยังมีความรู้ไม่เพียงพอต่อการทำงานใหญ่ๆ เพราะท่านยังขาดความรู้เกี่ยวกับการวิเคราะห์ และการทำความเข้าใจถึงหัวใจของปัญหาที่จะมาให้คอมพิวเตอร์ช่วยในการหาคำตอบ บางปัญหาถ้าท่านเขียนโปรแกรมให้ซับซ้อนโดยขาดการวิเคราะห์ให้ลึกซึ้ง ก็จะเสียเวลาของคอมพิวเตอร์ไปโดยเปล่าประโยชน์ เช่น มีโจทย์ข้อหนึ่งบอกว่าให้เขียนโปรแกรมบวกเลข 1 ถึง 100 ถ้าท่านไม่คิดอะไรให้ซับซ้อน ท่านก็จะเขียนโปรแกรมแบบโครงสร้างทุกอย่าง กล่าวคือ ท่านวิเคราะห์แล้วเห็นว่า โปรแกรมนี้ทำการบวกเลข 1 ถึง 100 ก็คือ การเอา 1+2+…+100 จะเห็นว่าจะต้องเกิดการกระทำซ้ำเพื่อทำการบวกเลขตั้งแต่ 1 และจบเงื่อนไขเมื่อเมื่อตัวเลขที่เข้ามาบวกเป็น 100 ซึ่งจะเขียนโปรแกรมด้วยภาษาซีได้ดังนี้

```
#include <stdio.h>
void main(void)
{
    int j, sum=0, n=100;
    for (j=1; j<=n;  j++)       sum+=j;
    printf(“Sum of 1 to %d is %d”,n,sum);
}
```

โปรแกรมที่ 1 แสดงการบวกเลขแบบใช้การกระทำซ้ำ

โปรแกรมที่ผ่านมาสามารถทำงานได้ และก็ให้คำตอบในการบวกที่ถูกต้องด้วย แต่ยังไม่ใช่วิธีการที่ดีที่สุด เพราะในโปรแกรมดังกล่าวจะต้องเกิดรอบการทำงานซ้ำถึง 100 รอบ แต่ถ้าท่านวิเคราะห์ให้ดี โปรแกรมนี้สามารถใช้ความรู้ทางคณิตศาสตร์ในชั้นมัธยมศึกษาตอนปลาย พบว่าการหาผลรวมของตัวเลขจำนวนเต็มจาก 1 ถึง $n$ สามารถสรุปออกมาเป็นสูตร คือ $n(n+1)/2$ เมื่อนำสูตรดังกล่าวมาเขียนเป็นโปรแกรมจะทำให้สามารถตัดการกระทำซ้ำออกไปได้ ซึ่งจะทำให้เขียนโปรแกรมได้ง่าย และกระชับกว่าดังแสดงในโปรแกรมที่ 2

```
#include <stdio.h>
void main(void)
{
   int n=100;
   printf("Sum of 1 to %d is %d",n, n*(n+1)/2);
}
```

โปรแกรมที่ 2 แสดงการบวกเลขแบบไม่ใช้การกระทำซ้ำ

จากที่กล่าวมาในข้างต้นของบทความในตอนนี้ ผู้เขียนต้องการแสดงให้เห็นถึงการทำงานของโปรแกรมคอมพิวเตอร์ 2 โปรแกรมที่ให้ผลการทดลองเหมือนกัน แต่ใช้กระบวนการในการทำงานที่แตกต่างกัน ซึ่งกระบวนการทางความคิดที่แตกต่างกันสำหรับการแก้ปัญหาอย่างหนึ่งอย่างใด บางครั้งเราเรียกว่า อัลกอริธึม (Algorithms) เพื่อการโปรแกรม

### 3.1 Algorithms

อัลกอริธึมเป็นเครื่องมือสำหรับการวางโครงสร้างของการเขียนโปรแกรม หรือจะกล่าวอย่างง่ายๆ อัลกอริธึม ก็คือ จุดเริ่มต้นของการเขียนโปรแกรมอย่างเป็นระบบ และมีระเบียบ แม้ว่าตัวผู้เขียนเองจะเรียนมาทางคณิตศาสตร์เป็นส่วนใหญ่ แต่ก็ใช่ว่าเราจะไม่สนใจงานทางด้านสังคมศาสตร์ โดยเฉพาะอย่างยิ่งทางด้านประวัติศาสตร์ หรือความเป็นมาของงานเขียนที่เรากำลังสนใจอยู่ ผู้เขียนได้พยายามค้นคว้าเพื่อหาที่มาที่ไปของรากศัพท์คำว่า Algorithms อยู่นานแสนนานก็ยังหาไม่พบ จนกระทั่งวันหนึ่งผู้เขียนได้ไปพบหนังสือชุด The Art of Computer Program ซึ่งแต่งโดย D.E. Knuth [1] ในเล่มแรกของหนังสือชุดนี้บอกว่าเขาจะเขียนออกมาทั้งหมด 7 เล่ม แต่จริงแล้วในปัจจุบันจะมีเพียง 3 เล่มแรกเท่านั้น ส่วนในอีก 4 เล่มหลังกำลังอยู่ในระหว่างการรวบรวม เมื่อผู้เขียนได้มาเปิดอ่านหน้าที่ 1 ของบทที่ 1 ในหนังสือเล่มที่ 1 จึงได้พบกับความหมาย และที่มาของรากศัพท์คำว่า Algorithms ซึ่งผู้เขียนจะขอสรุปออกมาเพื่อเป็นการเผยแพร่ความรู้แก่ผู้อ่านพอเป็นสังเขป ดังนี้

ในช่วงต้นของการค้นหาที่มาของรากศัพท์คำว่า Algorithms ได้มีคนเข้าใจกันว่ามันอาจมาจากรากศัพท์คำว่า Logarithms ที่มีมาก่อนหน้าอยู่แล้ว โดยเกิดจากการสลับตำแหน่งกันของตัวหนังสือ 4 ตัวแรก แต่เมื่อพิจารณาดูให้ดีแล้วเราก็พบว่าไม่ใช่ เพราะ Logarithms คือ ฟังก์ชันทางคณิตศาสตร์อย่างหนึ่ง ซึ่งถ้าเราสลับตำแหน่งกันแล้วก็ไม่อาจหาความหมายให้กับคำว่า Algorithms ได้อยู่ดี

ก่อนหน้าปี ค.ศ. 1957 นั้นยังไม่ได้มีการบัญญัติความหมายของคำว่า Algorithms ไว้ในดิกชันนารีที่ค่อนข้างจะมีชื่อเสียงมากชื่อว่า Webster's New World Dictionary ดังนั้นถ้าเราอยากรู้ความหมายก็คงทำได้อย่างเดียวคือการหาจากรากศัพท์ของคำที่เขียนใกล้

เคียงกันมากที่สุด ซึ่งแน่นอนว่ามันเป็นเรื่องที่ค่อนข้างยาก แต่ก็ไม่เกินความพยายามของคน ได้มีท่านผู้รู้ทำการค้นคว้าจากดิกชั่นนารีดังกล่าวพบรากศัพท์คำหนึ่งซึ่งน่าสนใจคือคำว่า Algorism ซึ่งได้ให้นิยามไว้ว่า *the process of doing arithmetic using Arabic number* หรือจะทำการแปลเป็นภาษาไทยให้อ่านเข้าใจง่ายๆ ก็คือ กระบวนการทางเลขคณิตสำหรับตัวเลขอาราบิค ซึ่งกระบวนการทางเลขคณิตก็คือการบวก ลบ คูณ หาร การเปรียบเทียบค่า เป็นต้น และก็เป็นที่ยอมรับความหมายของคำว่า Algorithms ว่ามีความหมายเช่นเดียวกับคำว่า Algorism กันมานานพอสมควร ซึ่งก็ยังมีหลายคนที่ยังไม่ค่อยพอใจกับที่มาของคำนี้มากนัก จึงได้ทำการค้นคว้าต่อไปทำให้พบที่มาที่แท้จริงจากหลักฐานชิ้นสำคัญ ซึ่งเป็นประวัติความเป็นมาของวิชาคณิตศาสตร์ ซึ่งโดยแท้จริงแล้วรากศัพท์คำว่า Algorism นั้นจะปรากฏอยู่ในหนังสือคณิตศาสตร์ของชาวเปอร์เซีย [1][2] (Persian textbook) เล่มหนึ่งซึ่งแพร่หลายมาก มีชื่อว่า *Kitab al jabr w'al-muqabala* หรือ *Rules of Restoration and Reduction* ซึ่งเขียนไว้โดย *Abu Ja'far Mohammed ibn Musa* <u>*al-khowarismi*</u> เขาเป็นลูกของ *Moses* เป็นชาวเมือง *khowarism* ปัจจุบัน *khowarism* เป็นเมืองเล็กๆ ในสหภาพโซเวียต นั้นก็คือ ความจริงแล้ว Algorism ก็คือชื่อของนักคณิตศาสตร์ชื่อดังที่ชื่อว่า <u>*al-khowarismi*</u> นั้นเอง ก็มีคนคิดกันไปต่างๆ นานาว่าทำไม Algorism จึงได้เปลี่ยนมาเป็น algorithm บางคนก็คิดว่าเป็นผลเนื่องมาจากความผิดเพี้ยนในการออกเสียงของการใช้ภาษาตัวสุดท้าย –sm และ –thm ซึ่งออกเสียงได้ค่อนข้างจะใกล้เคียงกันมาก จนทำให้ในที่สุดแล้ว Algorism ก็เปลี่ยนมาเป็น Algorithm แต่บางคนก็คิดว่าเวลาที่เปลี่ยนไป ก็อาจมีคนมาบัญญัติคำใหม่ขึ้นมาเพื่อไม่ใช้ไปซ้ำกับชื่ออย่างอื่นก็เป็นไปได้ แต่อย่างไรก็ตามทุกวันนี้พวกเรานักคอมพิวเตอร์ทั้งหลายก็จะรู้จักเพียงแต่คำว่า Algorithm เท่านั้น ส่วน Algorism ก็คงมีเพียงชาวรัสเซียเท่านั้นที่ยังรู้จัก

มีอัลกอริธึมหนึ่งซึ่งถือได้ว่าเป็นอัลกอริธึมที่คลาสสิคมากที่สุดที่ผู้เรียนวิชาโครงสร้างข้อมูลทุกคนต้องมีโอกาสได้ศึกษา เป็นอัลกอริธึม Euclid นักคณิตศาสตร์ชาวกรีกได้นำเสนอไว้เมื่อประมาณ 400 ถึง 300 ปีก่อน ค.ศ. เป็นอัลกอริธึมสำหรับการหาตัวหารร่วมมาก (Greatest Common Divisor, gcd) ของตัวเลขจำนวนเต็ม 2 จำนวน เช่น ตัวหารร่วมมากของ 80 และ 32 ก็คือ 16 ลองดู 2 อัลกอริธึมสำหรับการหาคำตอบให้ตัวเลขทั้ง 2 นี้

<u>อัลกอริธึมที่ 1</u> ใช้วิธีในการเลือกตัวหารที่ละตัวที่สามารถทำการหารตัวเลขทั้ง 2 ได้ลงตัวแล้วมีค่ามากที่สุด ตัวเลขนั้นคือ ตัวหารร่วมมากของตัวเลขทั้งสอง ฉะนั้นเราต้องใช้การกระทำซ้ำของโปรแกรมเพื่อวนหาตัวหารร่วมที่มากที่สุด ซึ่งค่าของตัวหารร่วมต้องไม่มากกว่าตัวเลขที่มีค่าน้อยกว่า ระหว่างตัวเลขทั้ง 2 (ผู้เขียนเคยใช้ทำเป็นการบ้านส่งเมื่อเรียนปริญญาตรี)

```
อัลกอริธึม 1 การหาตัวหารร่วมมากแบบลดค่าตัวหาร
Gcd1_1 [Find Min(m,n) and Set n is minimum]
        if (m<n) then t=n; n=m; m=t;
Gcd1_2 [init divisor] d=n;
Gcd1_3 [Find Remainder of m, n] divide m and n by d
        remainder is r1 and r2 respectively.
Gcd1_4 [Are they zero] If r1=0 and r2=0 then d is answer
Gcd1_5 [Decrement Divisor] Dec(d by 1) and goto Gcd1_3
```

อัลกอริธึมที่ 2 ใช้วิธีการหาเศษจากการหารแล้วสลับค่า เป็นหลักการที่ได้เสนอไว้ในหนังสือของ D.E.Knuth [1] เริ่มแรกเขาหาเศษจากการหารของตัวเลขทั้ง 2 ก่อน แล้วพิจารณาดูว่าถ้าเศษของการหารเป็นศูนย์ แสดงว่า n สามารถที่จะหาร m ได้ลงตัว เพราะฉะนั้นตัวหารร่วมมากก็คือ n แล้วจบการทำงาน แต่ถ้าเศษจากการหารไม่เป็นศูนย์ ก็จะใช้การถ่ายค่า n ไปสู่ m และถ่ายค่า r ไปสู่ n แล้ววนกลับไปหาเศษจากการหารระหว่าง m และ n ใหม่

| อัลกอริธึม 2 การหาตัวหารร่วมมากแบบสลับค่า |
|---|
| Gcd2_1 [Find Remainder of m,n] divide m by n then r is remainder<br>Gcd2_2 [Is it zero] If r=0 then n is answer<br>Gcd2_3 [Interchange] Set m is n, n is r goback to Gcd2_1 |

เมื่อเราพิจารณาทั้ง 2 อัลกอริธึม เราก็จะพบว่าอัลกอริธึมที่ 2 ซึ่งใช้ความรู้ทางด้านคณิตศาสตร์ มาช่วยในการวิเคราะห์จะเป็นอัลกอริธึมซึ่งมีความเหมาะสมมากกว่า

จากที่กล่าวมาทั้งหมดจะเห็นว่าความสำคัญของอัลกอริธึมต่อการเขียนโปรแกรมนั้นมีมาก เพราะเลือกอัลกอริธึมที่ดี และเหมาะสมกับงาน แล้วย่อมทำให้โปรแกรมที่เราเขียนขึ้นมานั้นสามารถทำงานได้อย่างเต็มประสิทธิ์ภาพ ตามความคาดหมายของการเขียนโปรแกรมครั้งนั้นๆ นอกจากนั้นอัลกอริธึมที่ดีที่เราเลือกใช้นั้นยังจะช่วยให้โปรแกรมที่เราพัฒนาขึ้นมาสามารถทำงานได้อย่างรวดเร็ว มีการเรียกใช้หน่วยความจำหลัก หน่วยความจำสำรองในประมาณที่เหมาะสมกับงาน เป็นต้น

**3.2 Data Structures**

นอกจากงานวิเคราะห์หาอัลกอริธึมที่เหมาะสมกับงานที่โปรแกรมเมอร์จะมองข้ามไปไม่ได้ ยังมีอีกสิ่งที่มีความสำคัญไม่แพ้กัน ก็คือ การจัดสรรหน่วยความจำหลัก (Main Memory) ให้ตัวแปรต่างๆ ที่เกี่ยวข้อง หน่วยความจำหลักมีความสำคัญต่อระบบคอมพิวเตอร์เป็นอย่างมาก เพราะมันคือที่สำหรับการพักข้อมูล เพื่อรอการประมวลผล หรือรับข้อมูลที่ได้จากการประมวลผลของหน่วยประมวลผลกลาง (CPU: Central Processing Unit) การจัดสรรหน่วยความจำหลักให้ตัวแปรแต่ละตัวนั้น จะต้องพิจารณาให้เหมาะสมกับขนาดและโครงสร้างของข้อมูลที่จำเป็นต้องใช้ ยกตัวอย่างเช่น ต้องการเขียนโปรแกรมเพื่อรวมคะแนนในแต่ละรายวิชาเพื่อตัดเกรด สมมติตั้งชื่อตัวแปรว่า Score ซึ่งเราก็รู้ดีว่าคะแนนของนักเรียนนั้นรวมแล้วต้องไม่เกิน 100 คะแนนอย่างแน่นอน โปรแกรมเมอร์กลับกำหนดประเภทของตัวแปรนั้นเป็น Longint (ให้พิจารณาข้อมูลในตารางที่ 1[3]) ผลก็คือมันสามารถให้คำตอบได้เช่นกัน แต่เราต้องใช้เนื้อที่ในการทำงานถึง 4 ไบต์ ซึ่งในความเป็นจริงแล้วใช้แบบ Byte หรือ ShortInt ก็เพียงพอแล้ว

| Datatype | Bytes | ขอบเขตข้อมูล |
|---|---|---|
| Byte | 1 | 0...255 |
| Word | 2 | 0…65535 |
| ShortInt | 1 | -128...128 |
| Integer | 2 | -32768...32767 |
| Longint | 4 | -2147483648...2147483647 |

ตารางที่ 1 รูปแบบการเก็บเลขจำนวนเต็มบนหน่วยความจำด้วยภาษาปาสคาล

ในทางตรงกันข้าม ถ้าเราต้องการคำนวณค่าก่อสร้างอาคาร และอุปกรณ์ประกอบอาคาร (สมมติหน่วยเป็นจำนวนเต็มบาท) เราจะเห็นว่าราคาของตึกหลายสิบล้านบาท แต่โปรแกรมเมอร์กลับกำหนดประเภทของตัวแปรเป็น Integer หรือ ShortInt ซึ่งจะทำให้ได้ค่าที่ไม่สอดคล้องกับความเป็นจริง ยิ่งไปกว่านั้น ถ้าค่าที่คำนวณได้เกินขอบเขตของข้อมูลชนิดนั้นที่รองรับได้ ผลลัพท์ที่ได้ก็คือขยะจากการคำนวณดีๆ นี่เอง

ดังนั้นสิ่งที่โปรแกรมเมอร์จะต้องพิจารณาในลำดับถัดต่อจากการวิเคราะห์อัลกอริธึม ก็คือ การเลือกโครงสร้างข้อมูลที่เหมาะสมกับขนาดของงาน

**สรุป**

ในการพัฒนาโปรแกรมเพื่อการทำงานอย่างใดอย่างหนึ่งของระบบคอมพิวเตอร์ ไม่ว่าจะเป็นระบบงานที่เล็กกระทัดรัด หรือว่าระบบงานใหญ่ก็ตาม สิ่งที่สำคัญในการเขียนโปรแกรมเพื่อให้เกิดข้อผิดพลาด (Bug) ให้น้อยที่สุด การใช้ความรู้ทางคณิตศาสตร์มาทำการวิเคราะห์หาอัลกอริธึม และการเลือกโครงสร้างข้อมูลสอดคล้องเหมาะสมกับงาน จากประสบการณ์ของผู้เขียนเองที่ได้คลุกคลีกับงานด้านการโปรแกรมมามากกว่า 8 ปี อยากจะข้อแนะนำให้โปรแกรมเมอร์และนักวิเคราะห์ระบบรุ่นน้องว่า งานวิเคราะห์คือหัวใจสำคัญของการโปรแกรม ดังนั้นจึงควรทำให้เป็นนิสัยติดตัวตลอดไป จะได้หากินและอยู่ในวงการนี้ไปได้นานๆ เอาไว้คราวหน้ามีเรื่องดีๆ จะมาเล่าให้ฟังอีก หรือว่าท่านอยากรู้เรื่องอะไรเกี่ยวกับ คอมพิวเตอร์และเทคโนโลยีสารสนเทศ ก็ให้บอกพวกเรามาได้น่ะครับ เรามีหลายสิ่งหลายอย่างที่อยากจะมาเล่าสู่กันฟัง แต่ถ้าเราพูดในเรื่องที่ท่านรู้อยู่แล้ว ทางคณะผู้เขียนเองก็เกรงว่าจะเป็นการเอามะพร้าวห้าวมาขายสวน ท่านจะเลือกส่งมาทาง email มาก็ง่าย หรือจะติดต่อโดยตรงก็สดวกที่ห้อง 509 ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ ชั้น 5 ตึกสิรินธร

**เอกสารอ้างอิง**

[1] D.E. Knuth, *The Art of Computer Program: Vol 1. Fundamental Algorithms,* Addision Wesley, pp.1, 1973.

[2] D. Harel, *Algorithmics: The Spirit of Computing,* 2nd edition, Addison Wesley, pp. 7, 1993.

[3] บุญเลิศ เอี่ยมทัศนา, *เรียนรู้ภาษาปาสคาลด้วยเทอร์โบปาสคาล 4.0-5.0,* บริษัท ซีเอ็ดยูเคชั่น จำกัด, หน้า 70, 2532.