บทที่ 11 การโปรแกรม (Programming)

11.1 โปรแกรมคืออะไร

เมื่อกล่าวถึงโปรแกรมที่เกี่ยวข้องกับคอมพิวเตอร์ จะมีความหมายดังต่อไปนี้

1. ชุดคำสั่งที่เหมาะที่จะประมวลผลโดยคอมพิวเตอร์ การประมวลผลในที่นี้ จะหมายถึง การใช้ตัวแปลภาษา (Assembler, Compiler หรือ Interpreter) ในการเตรียมที่จะประมวลผล และการประมวลผล

2. ในภาษาสำหรับเขียนโปรแกรม โปรแกรม หมายถึง สิ่งที่รวมเอา Module หรือ โปรแกรมย่อย (Subprogram) (Subroutine ในภาษา FORTRAN, Function ในภาษา C, Procedure ในภาษา Pascal เป็นต้น) หลาย ๆ อันเข้ามารวมเป็นอันเดียวกัน เพื่อใช้งาน

3. (โปรแกรมคอมพิวเตอร์) ส่วนที่เขียนตามหลักไวยกรณ์ของภาษาในการเขียนโปรแกรมภาษาใดภาษาหนึ่ง ประกอบด้วยส่วนของการประกาศ (ตัวแปร ฟังก์ชัน หรืออื่น ๆ) และประโยคคำสั่ง หรือชุดคำสั่งที่จำเป็นสำหรับการแก้ปัญหาใดปัญหาหนึ่ง เช่น การแก้สมการทางคณิตศาสตร์ เป็นต้น การทำงานใดงานหนึ่ง เช่น การจัดระบบฐานข้อมูล เป็นต้น หรือการทำให้มีความสามารถต่าง ๆ เช่น การทำให้คอมพิวเตอร์มีความสามารถเหมือนกับ CD Player เป็นต้น

11.2 การโปรแกรมคืออะไร

กล่าวโดยสรุปแล้ว การโปรแกรม หมายถึง การออกแบบ เขียน และทดสอบโปรแกรมคอมพิวเตอร์ ทั้งนี้ การโปรแกรมมีความหมายอื่นแฝงอยู่ อันได้แก่

- <u>การสร้างเครื่องมือที่ใช้ในการแก้ปัญหา</u> โดยเฉพาะปัญหาทางคณิตศาสตร์ และปัญหาที่เกี่ยวกับการใช้ข้อมูลต่างๆ
- <u>การสื่อสารกับคอมพิวเตอร์</u> หรือเป็นการสั่งงานคอมพิวเตอร์
- <u>การสร้างสรรค์งานศิลปะ</u> ดังจะเห็นได้จากการที่โปรแกรมคอมพิวเตอร์จัดเป็นทรัพย์สินทางปัญญาอย่างหนึ่ง เฉกเช่นเดียวกับภาพยนตร์ งานดนตรี งานละคร และอื่นๆ

11.3 ส่วนประกอบของการโปรแกรม

การโปรแกรมไม่ว่าจะใช้ภาษาใดก็ตาม จะต้องมีส่วนประกอบหลัก ๆ ดังต่อไปนี้

3.1 <u>โปรแกรม Text Editor หรือเรียกสั้น ๆ ว่า Editor</u> เป็นโปรแกรมที่ใช้ในการเรียบเรียงและแก้ไขโปรแกรม โดยโปรแกรมที่ถูกเรียบเรียงขึ้นนี้ จะถูกจัดเก็บในรูปที่แบบของ Text File กล่าวคือ จะประกอบด้วยตัวอักษรเท่านั้น ซึ่งเราเรียกไฟล์ดังกล่าวนี้ว่า *program source file* หรือ *source file* และเรียกข้อมูลที่อยู่ในไฟล์นี้ว่า *source code* หรือ *source*

3.2 <u>ภาษาในการเขียนโปรแกรม</u> หรือพูดง่ายๆ คือ ภาษาที่จะใช้สื่อสารกับคอมพิวเตอร์

3.3 <u>ตัวแปลภาษา</u> อันได้แก่ Assembler, Compiler และ Interpreter

ในกรณีของภาษา C จะใช้ Compiler ในการแปลภาษา โดยทั่วไป โปรแกรม Compiler จะประกอบด้วยส่วนต่างๆ ดังนี้

1. <u>*Compiler*</u> ทำหน้าที่แปลง source file (บันทึกคำสั่งที่มนุษย์สามารถทำความเข้าใจได้) ให้เป็น *object file* (บันทึกคำสั่งที่คอมพิวเตอร์สามารถทำความเข้าใจได้ ซึ่งอยู่ในรูปที่เลขฐานสอง) Source Code จะไม่ขึ้นกับเครื่องคอมพิวเตอร์ที่ใช้ เราสามารถนำ Source Code ที่เขียนขึ้นไปใช้ในเครื่องคอมพิวเตอร์หลาย ๆ แบบได้ เช่น นำ

โปรแกรมที่เขียนขึ้นสำหรับเครื่อง UNIX workstation ที่ใช้ UNIX เป็นระบบปฏิบัติการไปใช้กับเครื่อง PC ที่ใช้ Windows หรือ Linux เป็นระบบปฏิบัติการ เป็นต้น แต่ Object File จะขึ้นกับเครื่องคอมพิวเตอร์และระบบปฏิบัติการ กล่าวคือ จะต่างกันไปตามชนิดของเครื่องคอมพิวเตอร์ (กล่าวให้ละเอียดยิ่งขึ้นคือ ชนิดของโปรเซสเซอร์และชุดคำสั่ง) และระบบปฏิบัติงานที่ใช้ และไม่สามารถนำไปใช้ในเครื่องคอมพิวเตอร์ที่มีคุณสมบัติต่างกันได้

โดยสรุปแล้ว Compiler มีหน้าที่ทำการแปลง source code (High-level code) ให้เป็นคำสั่งที่ CPU สามารถทำได้ (Low-level code) Object file โดยทั่วไปมักมี Extension เป็น .obj หรือ .o

2. *Library* เป็น Function มาตรฐานของภาษา ยกตัวอย่าง เช่น printf, sin, exit, get ในภาษา C เป็นต้น ซึ่งเป็นการอำนวยความสะดวกให้กับผู้เขียนโปรแกรม นอกจาก Function มาตรฐานแล้ว ซอฟท์แวร์ Compiler ทั่วไปในปัจจุบันยังได้มี Function สำหรับอำนวยความสะดวกแก่ผู้พัฒนาโปรแกรมเพิ่มเติม อันได้แก่ ฟังก์ชันสำหรับใช้งาน WINDOWS ฟังก์ชันทางด้านกราฟิกส์ เป็นต้น

3. *Linker* มีหน้าที่นำ Object File ที่ได้จาก Compiler มาทำการรวมกับ Library (Function มาตรฐาน) เพื่อสร้างเป็น Executable File (ไฟล์ที่สามารถสั่งงานได้) Linker จะทำการตรวจสอบ Object File เพื่อหา Function มาตรฐานที่จำเป็น แล้วทำการเชื่อมต่อ Function มาตรฐานเข้ากับ Object File Executable File บน Windows จะมี Extension เป็น .EXE

4. *make Utility* เป็นโปรแกรมที่ช่วยเพิ่มประสิทธิภาพในการพัฒนาโปรแกรม โดยทำหน้าที่ตรวจสอบไฟล์ที่มีการเปลี่ยนแปลง แล้วทำการ Compile และสร้าง Executable File ใหม่ ทำให้ไม่มีความจำเป็นต้อง Compile ทุกไฟล์

5. *Debugger* เป็นโปรแกรมสำหรับตรวจสอบตำแหน่งของข้อผิดพลาดในโปรแกรม ซึ่งเรียกกันว่า *bug* Debugger ที่ดีจะต้องมี Feature ดังต่อไปนี้

- **Break Point** สามารถหยุดการรันโปรแกรมที่จุดไหนก็ได้
- **Single Step** สามารถรันโปรแกรมทีละ Step ได้
- **Display** สามารถแสดงค่าของตัวแปรหรือประโยคคำสั่งได้ทุกเวลาที่ต้องการ
- **Browser** สามารถแสดงส่วนของ source file ที่เกี่ยวข้องได้

6. *Wrappers and Integrated Development Environment* Wrapper เป็นโปรแกรมขั้นสูง (High-level Program) ที่มีหน้าที่กำหนดสิ่งที่จำเป็นต้องทำ เช่น โปรแกรม **cc** ใน UNIX จะทำการรันโปรแกรม C Compiler /lib/ccom และ Linker ld เป็นต้น โดยทั่วไป โปรแกรม Compiler บน Windows ก็มีลักษณะเป็น wrapper เพราะสามารถทำการรัน C compiler และ linker โดยต่อเนื่องกันได้

Integrated Development Environment (IDE) เป็นสิ่งที่รวมเอา Editor, Compiler, Linker และ Debugger เข้ามาให้สามารถใช้งานไปควบคู่กันได้ ยกตัวอย่าง เช่น IDE ใน Borland C++ เป็นต้น

3.4 Algorithm เป็นขั้นตอนของวิธีการที่จะใช้ในการเขียนโปรแกรม หรือพูดง่ายๆคือ ขั้นตอนการสั่งงานคอมพิวเตอร์ เพื่อที่จะแก้ปัญหาใดปัญหาหนึ่ง หรือให้ทำงานใดงานหนึ่ง เนื่องจากความสามารถที่แตกต่างกันระหว่างมนุษย์กับคอมพิวเตอร์ จำเป็นที่จะต้องมีวิธีที่เหมาะสมกับคอมพิวเตอร์

11.4 กระบวนการในการโปรแกรม (Programming Process)

อาจกล่าวได้ว่า การเขียนโปรแกรม มีลักษณะคล้ายคลึงกับการแก้ปัญหาทางด้านวิทยาศาสตร์หรือวิศวกรรม ซึ่งโดยทั่วไป จะมีขั้นตอนดังนี้

1. การทำความเข้าใจปัญหา
2. การเก็บรวบรวมข้อเท็จจริง หรือข้อมูลต่าง ๆ
3. การรวบรวมทฤษฎี หรือหลักการต่าง ๆ ที่เกี่ยวข้องกับปัญหา
4. การตั้งสมมุติฐาน
5. การแก้ปัญหา
6. การตรวจสอบผลที่ได้

โดยทั่วไป กระบวนการในการเขียนโปรแกรม จะมีขั้นตอนดังนี้

1. Problem Definition คือ การกำหนดนิยามของปัญหา หรือกำหนดคุณลักษณะ (Specification) ของโปรแกรม รวมไปถึงการศึกษาและวิเคราะห์ปัญหา
2. Algorithm Preparation คือ การเตรียมวิธีการที่จะใช้ในการแก้ปัญหา หรือการคิด Algorithm และการแยกแยะปัญหา
3. Abstraction คือ การออกแบบโปรแกรม การจำลองรูปที่แบบ (Model) ของโปรแกรมและข้อมูล การกำหนดส่วนต่าง ๆ ของโปรแกรม เช่น โครงสร้างของโปรแกรม โครงสร้างของข้อมูล ชนิดของตัวแปร ฟังก์ชัน เป็นต้น กล่าวโดยสรุปแล้ว Abstraction คือการแปลง Algorithm ให้เป็นโปรแกรม ทั้งนี้ Abstraction จะรวมไปถึงการเลือกภาษาที่จะใช้ในการเขียนโปรแกรม และการออกแบบให้สอดคล้องกับภาษาที่เลือก
4. Coding คือ การลงรหัสโปรแกรมตามที่ได้ออกแบบไว้ ทั้งนี้ จะต้องเป็นไปตามหลักไวยกรณ์ของภาษาที่เลือก โดยทั่วไป รหัสโปรแกรมที่ว่านี้ จะมีชื่อเรียกว่า *Program Source Code* หรือ *Source Code*
5. Testing and Verification คือ การทดสอบโปรแกรม ซึ่งเริ่มต้นที่การแปลงให้เป็นชุดคำสั่งที่คอมพิวเตอร์สามารถนำไปประมวลผลได้ การรันโปรแกรม และการตรวจสอบผลลัพธ์ที่ได้จากการรัน
6. Documentation คือ การทำเอกสารประกอบการใช้งานโปรแกรมที่เขียนขึ้น ซึ่งจะต้องประกอบด้วย

- คำอธิบายถึงวัตถุประสงค์ของโปรแกรม
- คำอธิบายวิธีการที่ใช้ในการเขียนโปรแกรม และ Algorithm
- โครงสร้างของโปรแกรม, Structure Chart, โครงสร้างของข้อมูล
- ขั้นตอนการทำงานของโปรแกรม, Flowchart
- วิธีการใช้โปรแกรม
- ตัวอย่างข้อมูลที่ใช้ในการทดสอบ และผลลัพธ์ที่ได้
- ข้อจำกัดของโปรแกรม

11.5 การเขียนโปรแกรมในลักษณะโครงสร้าง (Structured Programming)

วิธีการที่ใช้ในการเขียนโปรแกรม มีมากมายหลายวิธี ได้แก่ Functional Programming Logic Programming Object-Oriented Programming Structured Programming Top-Down Programming Modular Programming เป็นต้น ซึ่งโดยทั่วไป จะใช้ในลักษณะผสมผสานหลาย ๆ วิธีเข้าด้วยกัน ในที่นี้ จะกล่าวถึง Structured Programming ซึ่งถือได้ว่าเป็นพื้นฐานของวิธีการเขียนโปรแกรมในปัจจุบัน

แรงดลใจ การเขียนโปรแกรมในระยะแรก โดยมากจะเขียนโดยภาษา FORTRAN (หรือภาษาอื่น ๆ ซึ่งมีลักษณะคล้าย FORTRAN) อันมีจุดเด่นประการหนึ่งคือ ประโยคคำสั่ง GO TO ซึ่งเป็นคำสั่งสำหรับสั่งให้คอมพิวเตอร์ข้ามไปยังขั้นตอนที่เราต้องการในโปรแกรมนั้นได้ คำสั่งนี้ มีข้อดีคือ ทำให้เกิดความคล่องตัวในการเขียน

โปรแกรม แต่ก็มีข้อเสียที่สำคัญ คือ มีผลทำให้ลำดับขั้นตอนการทำงานของโปรแกรม กับลำดับของ Source Code ไม่ตรงกัน ทำให้ยากต่อการตรวจสอบและแก้ไขโปรแกรม

แรงดลใจอีกประการหนึ่ง คือ เมื่อพิจารณาจากคุณสมบัติของโปรแกรม ซึ่งมีรากฐานมาจากสมการทางคณิตศาสตร์แล้ว จะเห็นว่า มีโครงสร้างหลักที่ใช้ในการควบคุมการทำงาน อยู่เพียง 3 ลักษณะคือ การทำงานตามลำดับ การเลือกหรือตัดสินใจ และการทำงานวนซ้ำหรือเป็นการวนลูป

<u>หลักการ</u> หลักการเขียนโปรแกรมในลักษณะโครงสร้าง สามารถสรุปได้ ดังนี้

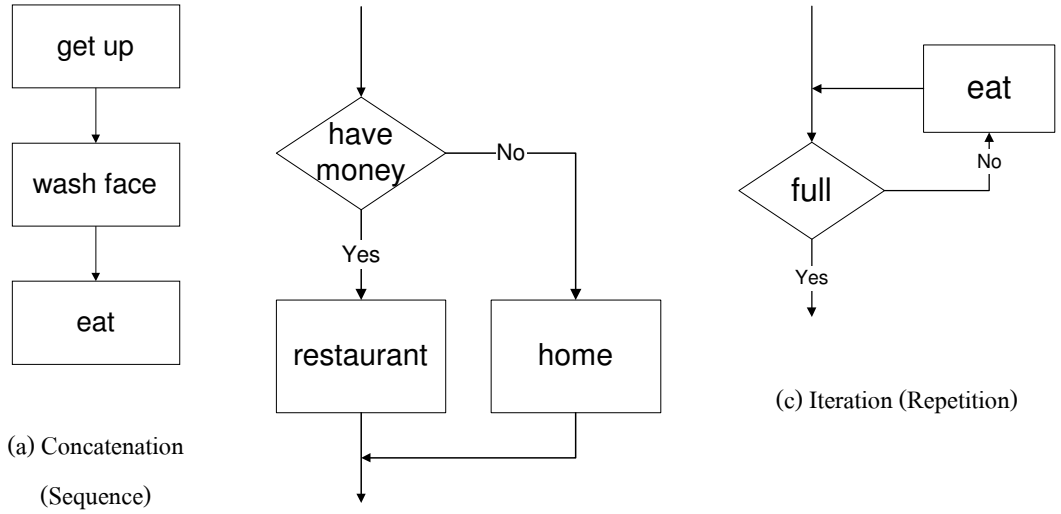
1. ประกอบด้วยโครงสร้างหลัก 3 ลักษณะ คือ

การทำงานตามลำดับ (Concatenation, Sequence) หมายถึง การทำงานแต่ละขั้นตอนตามลำดับ

การเลือกหรือตัดสินใจ (Selection, Decision) หมายถึง การตั้งเงื่อนไขต่าง ๆ ในโปรแกรม ถ้าเป็นจริง ก็จะทำงานตามคำสั่งหรือกลุ่มคำสั่งหนึ่ง แต่ถ้าไม่จริง ก็จะทำงานตามคำสั่งหรือกลุ่มคำสั่งอื่น

การทำงานวนซ้ำหรือการวนลูป (Iteration, Repetition, Looping) หมายถึง การทำงานตามคำสั่ง หรือกลุ่มของคำสั่งซ้ำ ๆ กัน จนกระทั่งเป็นไปตามเงื่อนไขสิ้นสุดการทำซ้ำ จึงจะไปทำงานตามคำสั่งถัดไป

รูปที่ 11.1 เป็นรูปแสดงตัวอย่างโครงสร้างหลักทั้งสามของการเขียนโปรแกรมในลักษณะโครงสร้าง



รูปที่ 11.1 ตัวอย่างโครงสร้าง 3 ลักษณะ (a) การทำงานตามลำดับ (b) การตัดสินใจ (c) การทำงานซ้ำ

2. หลีกเลี่ยงการใช้ประโยคคำสั่ง GO TO หรือ GO TO-less Programming

และเพื่อให้โปรแกรมมีโครงสร้างที่ดี ควรจะนำหลักการของ Top-Down Programming และ Modular Programming มาประกอบในการเขียน กล่าวคือ ควรจะทำการแยกงานที่จะทำออกเป็นส่วนย่อยๆ (Module) แล้วนำมารวมกัน ในลักษณะเดียวกับการผลิตเครื่องมือต่าง ๆ ซึ่งจะผลิตชิ้นส่วนต่าง ๆ ที่จำเป็นก่อน แล้วจึงนำมาประกอบกัน จะมีผลทำให้โปรแกรมแต่ละส่วนไม่ยาวจนเกินไป ทำให้ง่ายต่อการตรวจสอบและแก้ไขโปรแกรม อีกทั้งยังเป็นการใช้โปรแกรมที่มีอยู่เดิมให้เป็นประโยชน์สูงสุดได้ กล่าวคือ สามารถนำ Module ที่มีอยู่เดิม มาใช้ในโปรแกรมที่จะเขียนขึ้นใหม่ได้

ประโยชน์ของการเขียนโปรแกรมในลักษณะโครงสร้าง

กล่าวโดยสรุปแล้ว การเขียนโปรแกรมวิธีนี้ มีข้อดี ดังนี้

1. การทำงานของโปรแกรมเป็นไปตามลำดับของ Source Code ทำให้ง่ายต่อการตรวจสอบ แก้ไข และปรับปรุง
2. สามารถนำ Module ที่สร้างขึ้นไปใช้กับโปรแกรมอื่น ๆ ต่อไปได้

3. สามารถแบ่งงานกันทำและแก้ไขในระหว่างกลุ่มของผู้เขียนโปรแกรมได้

11.6 Top-Down Programming

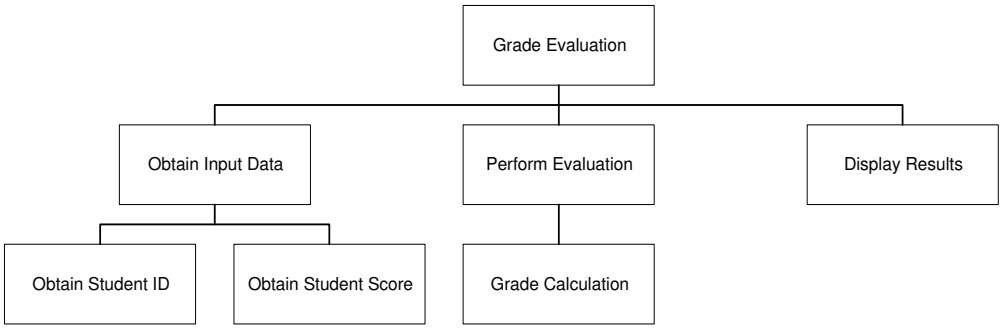
Top-down programming เป็นวิธีการเขียนโปรแกรม โดยทำการแบ่งโปรแกรมออกเป็นโครงสร้างลำดับชั้น (Hierarchy Structure) ตามการทำงาน ซึ่งเริ่มจากการพิจารณา Function หลักของโปรแกรม แล้วทำการแบ่งออกเป็น Function ย่อยๆที่ประกอบกันแล้วสามารถทำงานตามที่ต้องการได้ หรือทำโปรแกรมให้อยู่ในรูปที่ของ Structure Chart แต่ละส่วน จะประกอบด้วยโปรแกรมย่อยที่จะทำงานของส่วนนั้น ๆ ซึ่งอาจจะเป็น 1 โปรแกรมย่อยหรือมากกว่าก็ได้ เปรียบเสมือนการออกแบบอุปกรณ์ต่างๆที่ต้องมีการแบ่งเป็นส่วนประกอบต่างๆ อย่างเช่น สเตอริโอ ก็จะมีการแบ่งเป็นส่วน Amplifier, Deck, Tuner, CD Player และอื่น ๆ เป็นต้น การเขียนโปรแกรมวิธีนี้มีข้อดี ดังนี้

1. สามารถทำให้เรารู้โครงสร้างของโปรแกรม หรือภาพรวมของโปรแกรม
2. ทำให้เราสามารถออกแบบโปรแกรมได้อย่างสมบูรณ์
3. ง่ายต่อการหาจุดบกพร่องของโปรแกรม การแก้ไข หรือการเพิ่มเติมส่วนต่าง ๆ ให้กับโปรแกรม
4. สามารถแบ่งงานกันทำได้
5. สามารถใช้โปรแกรมที่มีอยู่ให้เกิดประโยชน์สูงสุด

โดยทั่วไป จะมีขั้นตอนในการเขียน ดังนี้

1.วิเคราะห์ปัญหา หรือโปรแกรมที่ต้องการจะเขียน
2. แจกแจงปัญหาออกเป็นปัญหาย่อย หรือแบ่งเป็นส่วนย่อย ๆ
3. ทำการเขียน Structure Chart
4. ทำการเขียน Source Code หรือแปลงเป็นโปรแกรมภาษาคอมพิวเตอร์

รูปที่ 11.2 แสดง structure chart สำหรับโปรแกรมตัดเกรด



รูปที่ 11.2 Structure Chart สำหรับโปรแกรมตัดเกรด

11.7 Modular Programming

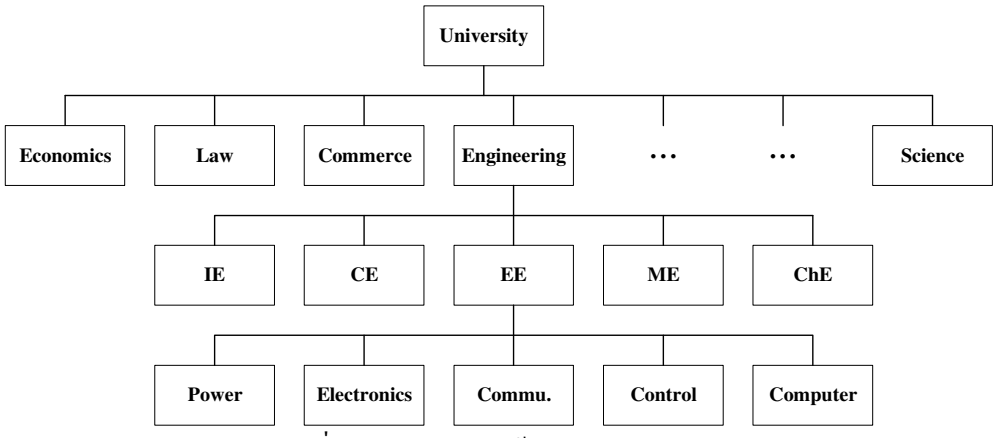
Modular Programming เป็นวิธีการเขียนโปรแกรมที่เน้นการแบ่งโปรแกรมออกเป็นส่วนย่อย ๆ ซึ่งเรียกว่า *Module* มีจุดประสงค์ก็เพื่อให้ง่ายต่อการเขียน และให้เกิดความผิดพลาดในการเขียนน้อยที่สุด โดยเน้นที่การทำให้โปรแกรมมีความซับซ้อนน้อยที่สุด ซึ่งเปรียบเสมือนการเขียนบทความหรือหนังสือต่างที่ต้องมีการแบ่งเป็นหัวข้อต่าง ๆ การเขียนวิธีนี้ มีข้อดี คือ สามารถแบ่งงานกันทำในกรณีที่พัฒนาโปรแกรมเป็นกลุ่มได้ และยังสามารถนำ *Module* มาใช้ร่วมกันได้ อย่างเช่น Module ที่เกี่ยวกับ INPUT/OUTPUT เป็นต้น ในการออกแบบ Module ควรจะ

ออกแบบให้เป็น Module ขนาดเล็กที่สุดเท่าที่ทำได้ เพื่อมิให้เกิดความซ้ำซ้อนของโปรแกรม และเพื่อให้การใช้ Module ร่วมกันเป็นไปอย่างมีประสิทธิภาพสูงสุด ยกตัวอย่าง เช่น เราควรเขียนเป็น Module สำหรับบวก ลบ คูณ หาร ให้เป็นคนละ Module แทนที่จะเขียนให้รวมอยู่ใน Module เดียวกัน เพราะในบางกรณี อาจจะใช้เพียงแค่การบวกเท่านั้น ซึ่งทำให้ประสิทธิภาพของโปรแกรมด้อยลง เป็นต้น

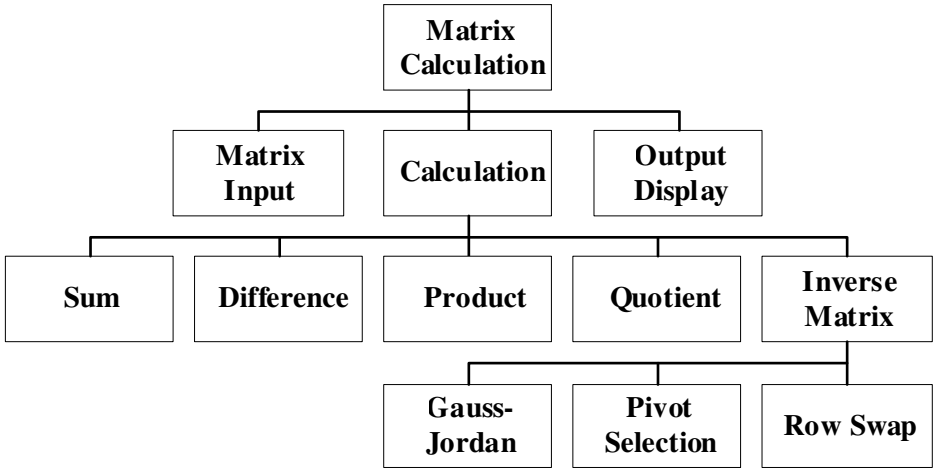
Top-down Programming & Modular Programming โดยสรุปแล้ว วิธีการเขียนโปรแกรมทั้ง 2 วิธีนี้ เป็นการเขียนโปรแกรมที่เหมือนกัน แต่มีมุมมองที่ต่างกัน Top-down Programming จะเน้นที่ขั้นตอนการออกแบบโปรแกรม (Program Design Process) ในขณะที่ Modular Programming จะเน้นที่ขั้นตอนการเขียนโปรแกรม (Program Coding Process) อนึ่ง หลังออกแบบโปรแกรมแบบ Top-down programming เวลาเขียนรหัสโปรแกรมจะมีชื่อเรียกว่า *Bottom-up Implementation* เพราะจะต้องเริ่มเขียนจากระดับล่างสุด

11.8 Structure Chart

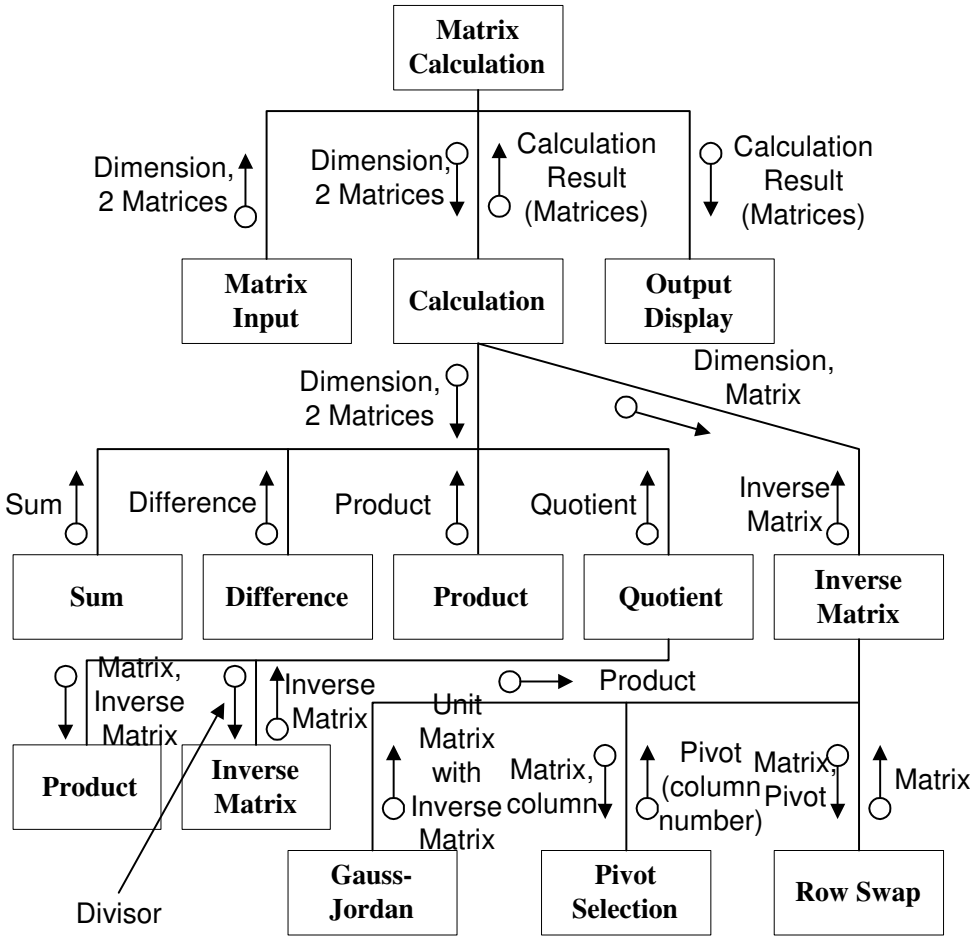
Structure Chart เป็นผังแสดงส่วนประกอบของโปรแกรม มีลักษณะเหมือนต้นไม้ที่แตกกิ่งก้านสาขา หรือผังขององค์กร (รูปที่ 11.3) การแบ่งส่วนประกอบของโปรแกรมจะแบ่งตามหน้าที่ของส่วนประกอบแต่ละส่วน โดยใช้หลักการของ Top-Down Programming กล่าวคือ จะเริ่มต้นพิจารณาจากส่วนประกอบใหญ่ว่า สามารถแบ่งเป็นส่วนประกอบย่อย ๆ ได้กี่ส่วน โดยเรียงตามลำดับความสำคัญ แล้วจึงพิจารณาถึงส่วนประกอบย่อยของส่วนประกอบย่อยอีกทีหนึ่ง โดยทำเป็นขั้น ๆ ไป ส่วนประกอบแต่ละส่วนจะถูกแสดงด้วยสี่เหลี่ยมผืนผ้า สำหรับระบุถึงหน้าที่ใดหน้าที่หนึ่ง และจะเรียงกันตามลำดับชั้น (Hierarchy Structure) นั่นหมายความว่า สี่เหลี่ยมผืนผ้าที่อยู่บนสุดจะแสดงถึงการทำงานหลักของโปรแกรมนั้น สี่เหลี่ยมผืนผ้าที่เรียงลำดับลงมาแสดงส่วนประกอบย่อยของการทำงานหลักนั้น ซึ่งอาจจะถูกแบ่งส่วนประกอบย่อยลงไปได้อีก ผังนี้ จะแสดงถึงหน้าที่หลักของโปรแกรม หน้าที่ย่อยของโปรแกรม และความสัมพันธ์ระหว่างหน้าที่ต่าง ๆ กล่าวคือ จะทำให้เข้าใจถึงโครงสร้างหลักของโปรแกรม รูปที่ 11.4 เป็น structure chart สำหรับแสดงโครงสร้างของโปรแกรมที่ใช้คำนวณเกี่ยวกับ Matrix และรูปที่ 11.5 เป็น structure สำหรับโปรแกรมคำนวณเกี่ยวกับ Matrix ที่เพิ่มรายละเอียดของการรับส่งข้อมูลระหว่างส่วนย่อยด้วย



รูปที่ 11.3 แสดงโครงสร้างของมหาวิทยาลัย



รูปที่ 11.4 แสดง Structure Chart ของโปรแกรมสำหรับคำนวณ Matrix



รูปที่ 11.5 Structure ของโปรแกรมสำหรับคำนวณ Matrix แบบละเอียด มีการแสดงการรับ-ส่งข้อมูล

11.9 Algorithm

Algorithm คือ ลำดับขั้นตอนที่ใช้ในการแก้ปัญหา โดยทั่วไป จะเจาะจงถึงการแก้ปัญหาโดยใช้คอมพิวเตอร์ โดยจะเขียนในลักษณะเป็นคำสั่งของแต่ละขั้นตอน เหมือนๆกับคำอธิบายการทำอาหาร การทดลอง หรือการใช้เครื่องมือต่างๆ โดยทั่วไป Algorithm จะต้องมีคุณสมบัติดังนี้

1. **Exactness** ใช้ข้อความที่ชัดเจน ไม่คลุมเคลือ
2. **Effectiveness** ได้ผลลัพธ์ที่ถูกต้อง

3. **Guaranteed termination** ต้องมีจุดสิ้นสุด

4. **Generality** ต้องใช้ได้กับค่าทั่วๆไป หรือใช้ได้กว้างขวางมากที่สุดเท่าที่ทำได้

คำสั่งในแต่ละขั้นตอนจะต้องเป็นคำสั่งที่คอมพิวเตอร์สามารถทำได้ และต้องสื่อความหมายให้ผู้อ่านสามารถนำไปเขียนเป็นโปรแกรมต่อไปได้ นอกจากนี้ Algorithm ที่ดี ควรมีคุณสมบัติดังนี้

- มีประสิทธิภาพสูง กล่าวคือ ใช้เวลาสั้นในการคำนวณ
- ไม่ซับซ้อน เพื่อให้ง่ายต่อการเขียนโปรแกรม และตรวจสอบข้อผิดพลาด
- ประหยัดทรัพยากร โดยเฉพาะหน่วยความจำ

โดยปกติแล้ว คุณสมบัติ 3 ประการดังกล่าวนี้ค่อนข้างจะมีลักษณะเป็น *Trade-Off* กัน ยกตัวอย่างเช่น

- ถ้าต้องการประสิทธิภาพที่สูง มักจะมีขั้นตอนการคำนวณที่ซับซ้อนขึ้น
- ถ้าต้องการประหยัดทรัพยากร มักจะต้องเพิ่มจำนวนขั้นตอน ส่งผลให้ประสิทธิภาพต่ำลง
- ถ้าต้องการลดความซับซ้อน อาจต้องใช้หน่วยความจำมากขึ้น

ดังนั้น การคิดค้นหา Algorithm ที่ดี จึงเป็นหัวข้อวิจัยสำคัญอันหนึ่งในสาขาวิชาวิศวกรรมคอมพิวเตอร์

<u>หลักการเขียน Algorithm</u>

1. คำสั่งที่ใช้ในแต่ละขั้นตอนต้องเป็นคำสั่งที่คอมพิวเตอร์สามารถทำงานได้ เช่น การรับข้อมูล การคำนวณบวก ลบ คูณ หาร การเปรียบเทียบ เป็นต้น ในกรณีที่เป็นการคำนวณที่ไม่ใช่ Function พื้นฐานของคอมพิวเตอร์ จะต้องเขียน Algorithm อธิบายการคำนวณนั้น
2. ต้องใช้ถ้อยคำที่ชัดเจน และง่ายต่อการทำความเข้าใจ
3. ต้องง่ายต่อการนำไปเขียนเป็นโปรแกรม และสามารถตรวจสอบความถูกต้องได้
4. ใช้สมการทางคณิตศาสตร์แสดงการคำนวณ เฉพาะกรณีที่ยากต่อการอธิบายด้วยคำพูดเท่านั้น

<u>หมายเหตุ</u> ในกรณีที่ทำการเขียนโปรแกรมขนาดใหญ่ หรือโปรแกรมที่สามารถแบ่งออกเป็นส่วนประกอบย่อย ๆ ได้ การเขียน Algorithm มักจะทำในลักษณะเขียนอธิบายส่วนที่สำคัญ หรือส่วนที่เป็นหัวใจหลักของโปรแกรม เช่น ในโปรแกรมคำนวณหา Inverse Matrix ก็มักจะเขียน Algorithm ที่แสดงวิธีการคำนวณเท่านั้น โดยจะตัดส่วนที่เป็นพื้นฐาน เช่น ส่วน INPUT/OUTPUT เป็นต้น ออก

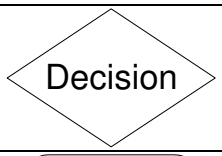
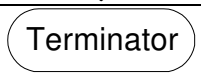
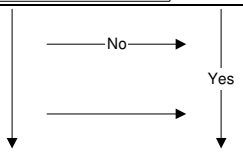
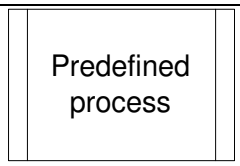
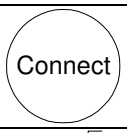
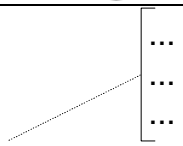
11.10 Flowchart

Flowchart (ผังงาน) เป็นเครื่องมือที่ช่วยในการวางแผนขั้นตอนต่าง ๆ ของโปรแกรม และช่วยในการทำความเข้าใจขั้นตอนการทำงานของโปรแกรม หรือ Algorithm ที่ใช้ในการเขียนโปรแกรมนั้น ประกอบด้วยสัญลักษณ์ต่าง ๆ ซึ่งแสดงความหมายต่าง ๆ กันออกไป

สัญลักษณ์ที่ใช้เป็นประจำ

สัญลักษณ์ที่ใช้ในการเขียน Flowchart โดยมากจะเป็นรูปที่ทรงทางเรขาคณิต เช่น สี่เหลี่ยม วงกลม ลูกศร เป็นต้น ซึ่งมีอยู่ประมาณ 30 สัญลักษณ์ โดยแต่ละสัญลักษณ์จะมีความหมายแตกต่างกันไป แต่สัญลักษณ์ที่ใช้เป็นประจำมีอยู่ไม่มากนัก ดังแสดงในตารางต่อไปนี้

ชื่อสัญลักษณ์	สัญลักษณ์	ความหมาย
Process	Process	แสดงการประมวลผลข้อมูล เช่น การคำนวณ การโยกย้ายข้อมูล เป็นต้น

Decision		แสดงการตัดสินใจ หรือการกำหนดเงื่อนไขต่างๆ
Terminator		แสดงจุดเริ่มต้น หรือจุดสิ้นสุดของโปรแกรม หรือโปรแกรมย่อย (Subprogram)
Data		แสดงการป้อนข้อมูล (Data Input) หรือการแสดงผลลัพธ์จากการประมวลผลข้อมูล (Data Output)
Flowline		แสดงทิศทางของแต่ละ Process ใน Flowchart ลูกศรที่มี Label ต่าง ๆ เช่น Yes, No เป็นต้น จะใช้เฉพาะกับสัญลักษณ์ Decision เท่านั้น
Predefined Process		แสดงการประมวลผลที่ที่ไม่ได้เป็น Process พื้นฐาน (เช่น การบวก ลบ คูณ หาร) และได้รับการกำหนดนิยามไว้แล้ว กล่าวคือ แสดงการเรียกใช้โปรแกรมย่อย (Subprogram) เช่น Function ในภาษา C เป็นต้น
Connector		แสดงจุดเชื่อมต่อของ Flowchart ในกรณีที่มีความยาวมาก ๆ ทั้งนี้ เพื่อให้สะดวกในการอ่าน
Annotation		แสดงข้อความต่าง ๆ สำหรับอธิบายขั้นตอนต่าง ๆ ที่ปรากฏใน Flowchart ทั้งนี้ เพื่อให้ความกระจ่างแก่ผู้อ่านมากขึ้น

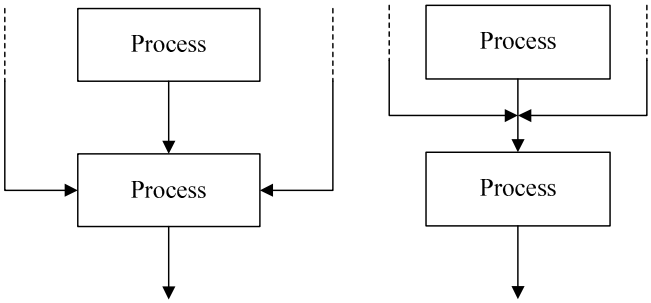
หลักเกณฑ์การเขียน Flowchart

จุดประสงค์หลักของ Flowchart ก็คือ ช่วยให้สามารถทำความเข้าใจโปรแกรมได้ดี และชัดเจนยิ่งขึ้น อันจะเป็นประโยชน์ต่อผู้เขียนโปรแกรม ในการวางแผนการพัฒนาโปรแกรม หรือปรับปรุงโปรแกรมให้ดีขึ้น ในกรณีที่เกิดข้อผิดพลาดขึ้น และเป็นประโยชน์ต่อผู้ใช้ ในการทำความเข้าใจการทำงาน และข้อจำกัดต่าง ๆ ของโปรแกรม ดังนั้น Flowchart ที่ดีจะต้องอยู่ในรูปที่แบบที่ง่ายต่อการทำความเข้าใจ และไม่ทำให้เกิดความสับสนต่อผู้อ่าน ซึ่งมีหลักการเขียนทั่วไปดังต่อไปนี้

1. แบ่ง Flowchart ออกเป็นกลุ่ม ๆ (module) ซึ่งเป็นหลักพื้นฐานของการเขียนโปรแกรมในลักษณะโครงสร้าง
2. Flowchart จะต้องเขียนในลักษณะจากบนลงล่าง หรือจากซ้ายไปขวา เพื่อให้สะดวกในการอ่าน
3. Flowchart สำหรับโปรแกรมหลัก (Main Program) แต่ละอันจะเริ่มต้นจาก สัญลักษณ์ TERMINATOR *START* (หรือ *BEGIN*) และสิ้นสุดที่ TERMINATOR *STOP* (หรือ *END*) ส่วน Flowchart สำหรับโปรแกรมย่อยจะเริ่มจาก TERMINATOR *START* เช่นกัน แต่สัญลักษณ์ที่ใช้แสดงการสิ้นสุดของโปรแกรมย่อย จะเป็นสัญลักษณ์ TERMINATOR *RETURN* (หมายถึง กลับไปที่ Main Program หรือโปรแกรมย่อยที่เรียก)
4. สัญลักษณ์แต่ละอันจะถูกเชื่อมโยงด้วย Flowline (เส้นแสดงทิศทางการทำงาน) หรือเส้นตรงที่มีหัวลูกศรตรงปลาย เพื่อแสดงถึงลำดับและทิศทาง โดยทั่วไปทิศทางของ Flowline จะทำจากบนลงล่าง และจากซ้ายไปขวา

5. สัญลักษณ์แต่ละอันจะมีทางเข้า 1 ทาง และทางออก 1 ทาง ยกเว้นแต่สัญลักษณ์ Decision จะมีทางออกได้มากกว่า 1 ในกรณีที่มีขั้นตอนที่มาสู่สัญลักษณ์นั้นเกิน 1 ให้รวมเป็นทางเดียวกัน ก่อนที่จะมาเข้าสัญลักษณ์นั้น (รูปที่ 11.6)

6. ใช้สัญลักษณ์จุดต่อเนื่อง (Connector) เมื่อมีความจำเป็นเท่านั้น เช่น ใช้จุดต่อเนื่องเมื่อขึ้นหน้าใหม่ หรือเมื่อสัญลักษณ์ที่แสดงถึงลำดับขั้นตอนการทำงานห่างกันจนกระทั่งการลากเส้นทิศทางการทำงานอาจทำให้เกิดความสับสน



รูปที่ 11.6 วิธีการเขียนสัญลักษณ์ลูกศร กรณีที่มีหลายทางเข้า รูปที่ขวาแสดงวิธีที่ถูกต้อง

7. เส้นทิศทางการทำงานต้องไม่ลากผ่านกัน และตำแหน่งเข้า-ออกของเส้นทิศทางการทำงานควรจะอยู่ตรงกึ่งกลางของสัญลักษณ์

8. ใช้สัญลักษณ์ Annotation ในการเขียนคำอธิบายขั้นตอนต่าง ๆ ที่ไม่ชัดเจน หรือยากต่อการเข้าใจ เช่น ความหมายของ Function หรือค่าของตัวแปรต่าง ๆ

9. ในการเขียนรายละเอียดในสัญลักษณ์ ให้ยึดหลักดังต่อไปนี้

9.1ข้อความที่ใช้ในสัญลักษณ์จะต้องเป็นคำที่ง่ายต่อการเข้าใจ เป็นข้อความที่ไม่กำกวม และไม่ควรจะเป็นสัญลักษณ์เฉพาะของภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง

9.2 ชื่อตัวแปรจะต้องสื่อความหมายของตัวแปรนั้น และจะต้องสอดคล้องกันทั้ง Flowchart

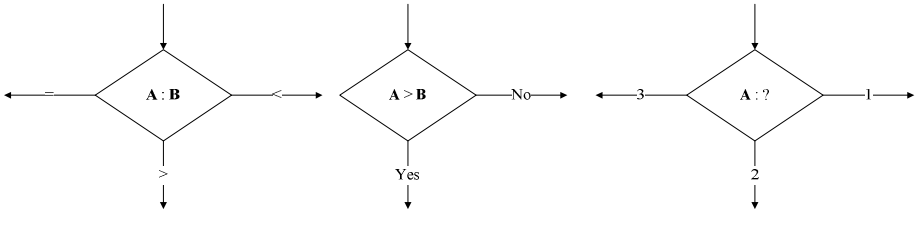
9.3 การเขียนข้อความภายในสัญลักษณ์ Decision จำแนกได้เป็นแบบต่าง ๆ ดังนี้ (รูปที่ 7, 8)

(a) การเปรียบเทียบระหว่างค่าของตัวแปร 2 ค่า เขียนในลักษณะ A : B โดย label ของลูกศรที่ออกจากสัญลักษณ์นี้จะเป็น >, <, =, $\neq$, $\geq$ หรือ $\leq$ (รูปที่ 11.7 (a))

(b) การเปรียบเทียบระหว่างค่าของตัวแปรตัวหนึ่งกับอีกค่าหนึ่ง เขียนในลักษณะ A>B ในกรณีนี้ label ของลูกศรจะเป็น Yes, No (รูปที่ 11.7 (b))

(c) การกำหนดที่ไปโดยใช้ค่าของตัวแปร เขียนในลักษณะ A: ? โดย label ของลูกศรที่ออกจากสัญลักษณ์จะแสดงค่าของตัวแปรตัวนี้ เช่น 1, 2 etc. (รูปที่ 11.7 (c))

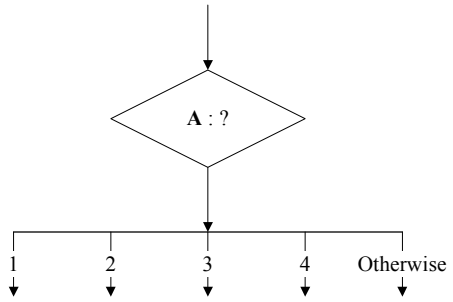
(d) ในกรณีที่มีการกำหนดที่ไปโดยใช้ค่าของตัวแปร ที่มีทางเลือกเป็นจำนวนมาก อย่างในกรณีการใช้ประโยคคำสั่ง Switch ในภาษา C ให้เขียนในลักษณะเหมือนข้อ (c) แต่ให้มีลูกศรออกจากสัญลักษณ์เพียงเส้นเดียว แล้วนำไปแยกเป็นหลาย ๆ เส้น โดยกำหนด label ของลูกศรที่แตกออกไปให้แสดงค่าของตัวแปรตัวนี้ (รูปที่ 11.8)



(a) (b) (c)

รูปที่ 11.7 แสดงวิธีการใช้สัญลักษณ์ Decision

9.4 การเขียนข้อความแสดงการคำนวณ โดยทั่วไปจะเขียนในลักษณะ A = B+C (C or Fortran Style) หรือ A <- B+C (Pascal Style) จะมีความหมายว่า ให้ค่า A เท่ากับ ผลบวกของ B กับ C (หรือพูดตามภาษาคอมพิวเตอร์ว่า หาผลบวกของ B กับ C แล้วนำไปเก็บเป็นค่าของ A เพราะฉะนั้น จะต้องมีค่าของ B และ C ที่จะนำมาคำนวณก่อน)



รูปที่ 11.8 แสดงวิธีการใช้สัญลักษณ์ Decision (Case (d))

9.5 ในการแสดงชุดข้อมูลใน Flowchart ให้เขียนในลักษณะ a_1, a_2, a_3 ... โดยให้ a_n แสดงข้อมูล ตัวที่ n ของชุดข้อมูลที่ใช้ชื่อ a

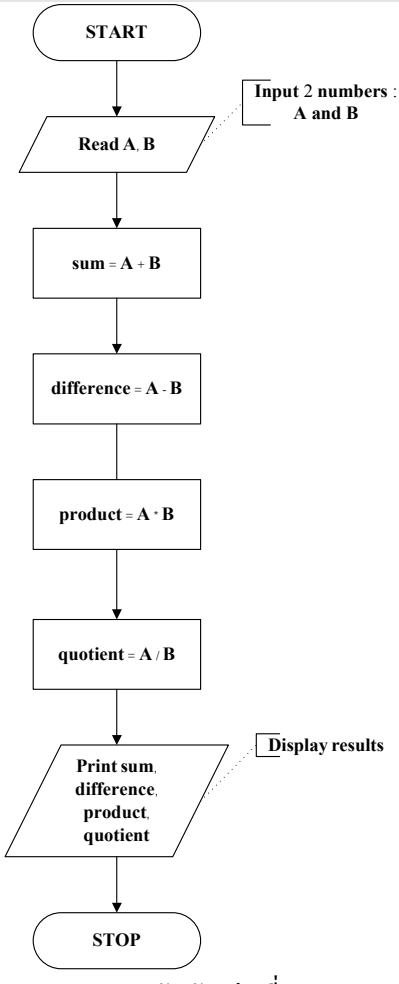
11.11 ตัวอย่าง Algorithm และ Flowchart

<u>ตัวอย่างที่ 1</u> การคำนวณหาผลบวก ลบ คูณ หารของเลข 2 จำนวน

Algorithm

1. รับค่าตัวเลข 2 จำนวน
2. คำนวณหาผลรวมของเลข 2 จำนวน
3. คำนวณหาผลต่างของเลข 2 จำนวน
4. คำนวณหาผลคูณของเลข 2 จำนวน
5. คำนวณหาผลหารของเลข 2 จำนวน
6. แสดงผลลัพธ์ที่ได้

รูปที่ 11.9 แสดง Flowchart สำหรับตัวอย่างนี้



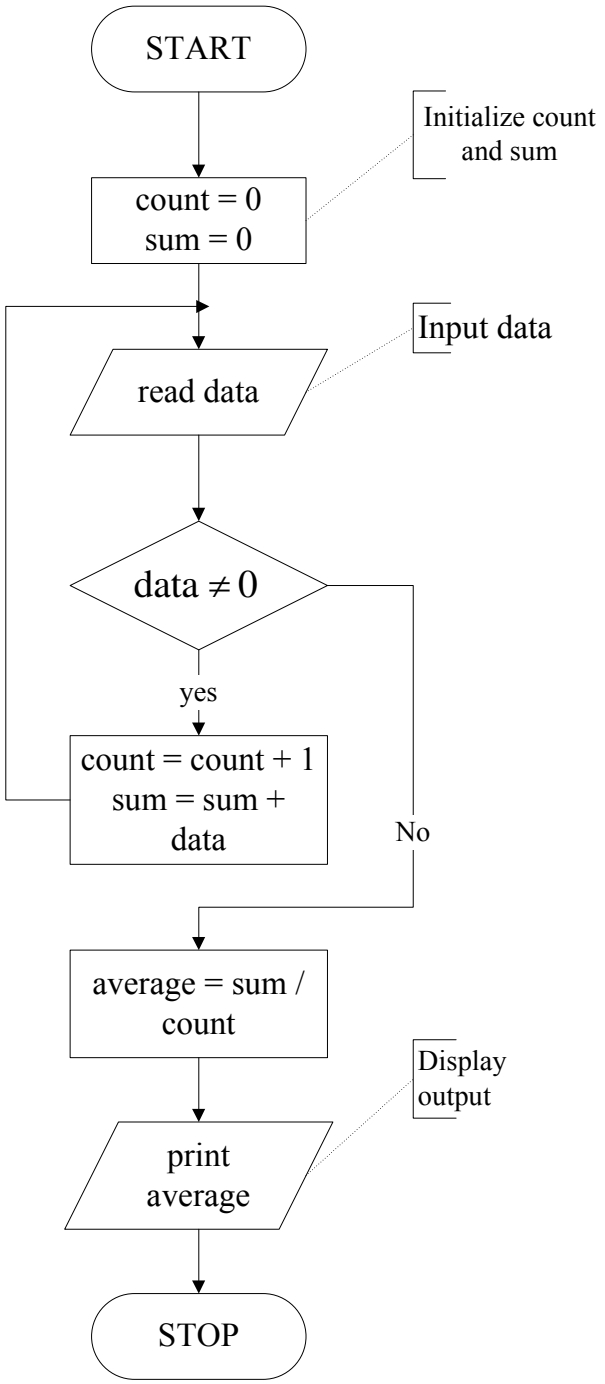
รูปที่ 11.9 Flowchart สำหรับตัวอย่างที่ 1

ตัวอย่างที่ 2 การคำนวณหาค่าเฉลี่ย

Algorithm

1. กำหนดค่าเริ่มต้นของจำนวนค่า (count) และผลรวม (sum) เป็น 0
(ขั้นตอนนี้เรียกว่า Initialization)
2. รับค่าที่จะมาทำการคำนวณ
3. ถ้าค่าที่รับเข้ามาไม่เท่ากับ 0 ให้ทำตามขั้นตอนต่อไปนี้
 3.1 เพิ่มจำนวนค่าทีละ 1
 3.2 ให้ผลรวมเท่ากับ ผลรวมเดิมรวมกับค่าที่รับเข้ามา
 3.3 กลับไปที่ขั้นตอนที่ 2
4. ถ้าค่าที่รับเข้ามาเป็น 0 ให้ทำตามขั้นตอนต่อไปนี้
 4.1 คำนวณค่าเฉลี่ย โดยหาผลหารของผลรวมกับจำนวนค่า
 4.2 แสดงค่าเฉลี่ยที่คำนวณได้
 4.3 จบการทำงาน

รูปที่ 11.10 แสดง Flowchart สำหรับตัวอย่างนี้



รูปที่ 11.10 Flowchart

<u>ตัวอย่างที่ 3</u> โปรแกรมสำหรับหาค่าสูงสุด

Algorithm

1. รับชุดตัวเลขที่ต้องการจะหาค่าสูงสุดเข้ามา โดยมีขั้นตอนดังนี้

1.1 รับค่าจำนวนตัวเลข

1.2 รับจำนวนตัวเลขจนได้ครบตามจำนวนที่ต้องการ

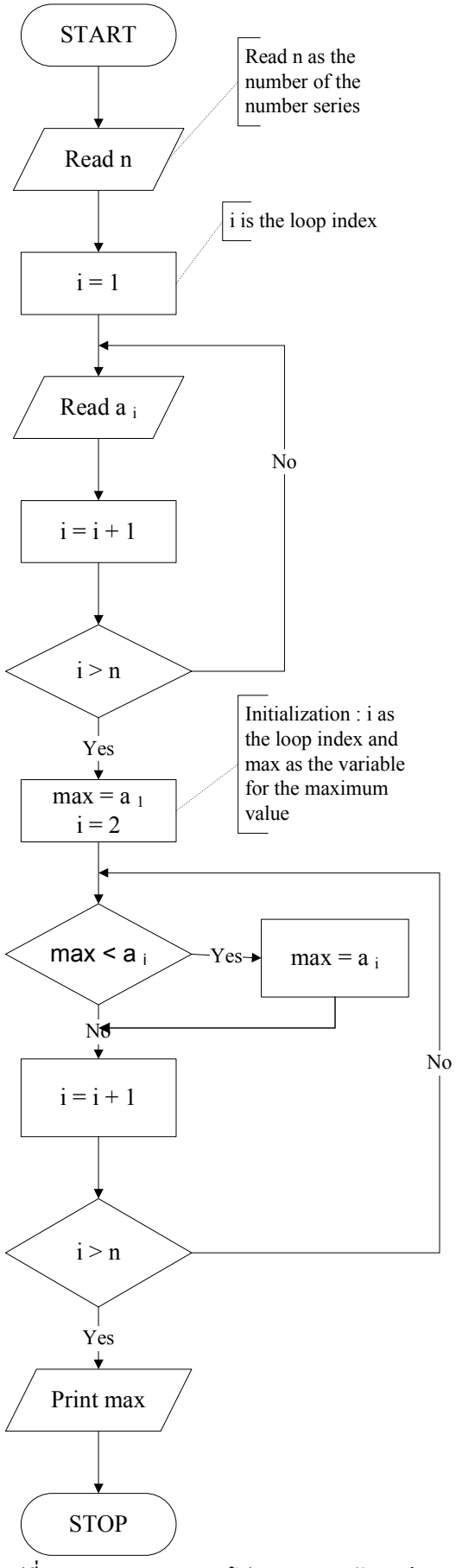
2. กำหนดค่าสูงสุดให้เท่ากับตัวเลขตัวแรก

3. เปรียบเทียบค่าสูงสุดนี้กับตัวเลขตัวถัดไป ถ้าค่าสูงสุดน้อยกว่า ให้กำหนดค่าสูงสุดเท่ากับค่าตัวเลขตัวนี้

4. ทำซ้ำข้อ 3 จนกว่าจะถึงตัวเลขตัวสุดท้ายในชุดตัวเลข

5. แสดงค่าสูงสุดที่หาได้

รูปที่ 11.11 แสดง Flowchart สำหรับตัวอย่างนี้



รูปที่ 11.11 Flowchart ของโปรแกรมสำหรับหาค่าสูงสุด

<u>ตัวอย่างที่ 4</u> โปรแกรมสำหรับหาค่าตัวหารร่วมมาก (ห.ร.ม., Greatest Common Divisor) ของเลขจำนวนเต็ม 2 จำนวน การหาค่า ห.ร.ม. ทำได้โดยใช้วิธีของ Euclid ซึ่งมีวิธีการง่ายๆ คือ นำเลข 2 จำนวนที่ต้องการหามาหารกัน ถ้าหารลงตัว แสดงว่าตัวหารเป็น ห.ร.ม. ถ้าไม่ก็นำตัวหารมาหารด้วยเศษที่ได้จากการหาร ทำเช่นนี้เรื่อยไป จนกว่าเศษที่ได้จากการหารเป็น 0 ซึ่งสามารถแสดงเป็นสมการคณิตศาสตร์ ได้ดังนี้

$$\gcd(x, y) = \begin{cases} x & (when\ y = 0) \\ \gcd(y, x \bmod y) & (when\ y \neq 0) \end{cases}$$

ตัวอย่าง เช่น 720 กับ 256

720 mod 256 = 208

256 mod 208 = 48

208 mod 48 = 16

48 mod 16 = 0

∴ ห.ร.ม. ของ 720 กับ 256 เท่ากับ 16

<u>หมายเหตุ</u> x mod y หมายถึง เศษที่ได้จาก x ÷ y

Algorithm

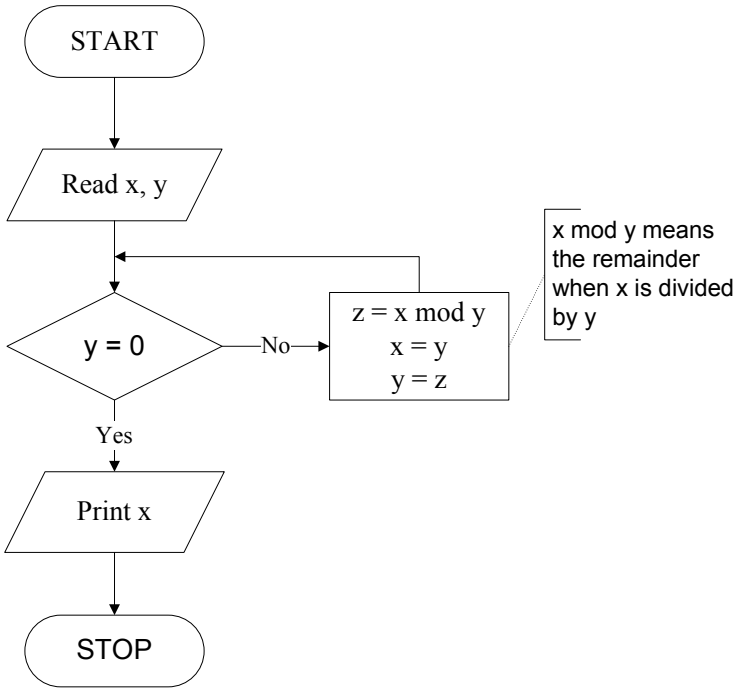
1. รับตัวเลข 2 จำนวนที่ต้องการหา ห.ร.ม. เข้ามา กำหนดให้เป็น x, y
2. ระหว่างที่ค่า y ไม่เป็น 0 ให้ทำตามขั้นตอนต่อไปนี้

2.1 หาเศษที่ได้จากการหาร x ÷ y

2.2 กำหนดให้ x เท่ากับ y และ y เท่ากับเศษที่ได้จากการหาร

3. แสดงค่า x

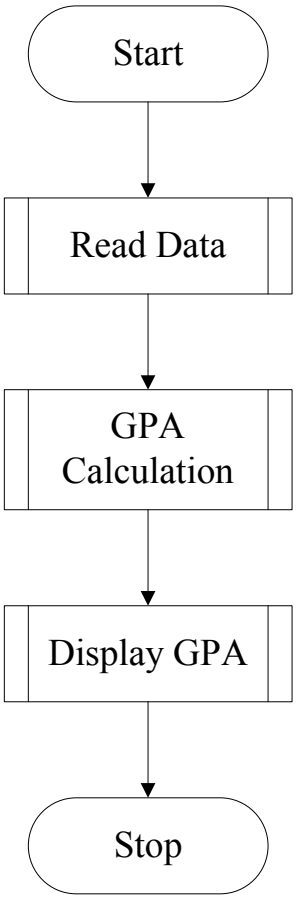
รูปที่ 11.12 แสดง Flowchart สำหรับตัวอย่างนี้



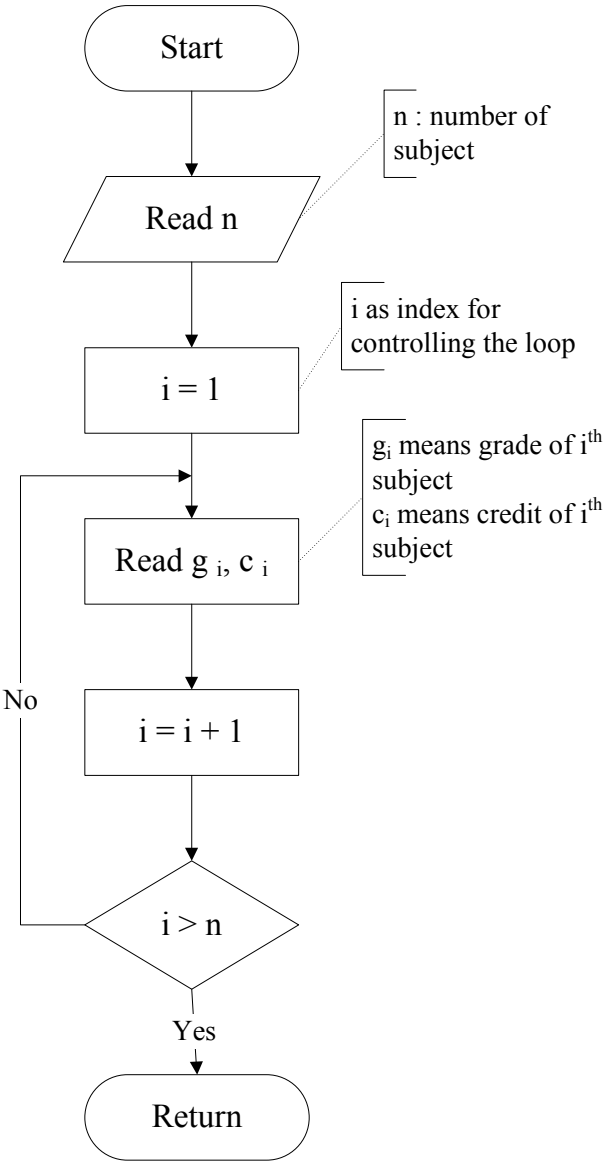
รูปที่ 11.12 Flowchart ของโปรแกรมสำหรับหา ห.ร.ม.

ตัวอย่างที่ 5 โปรแกรมสำหรับหาค่า GPA

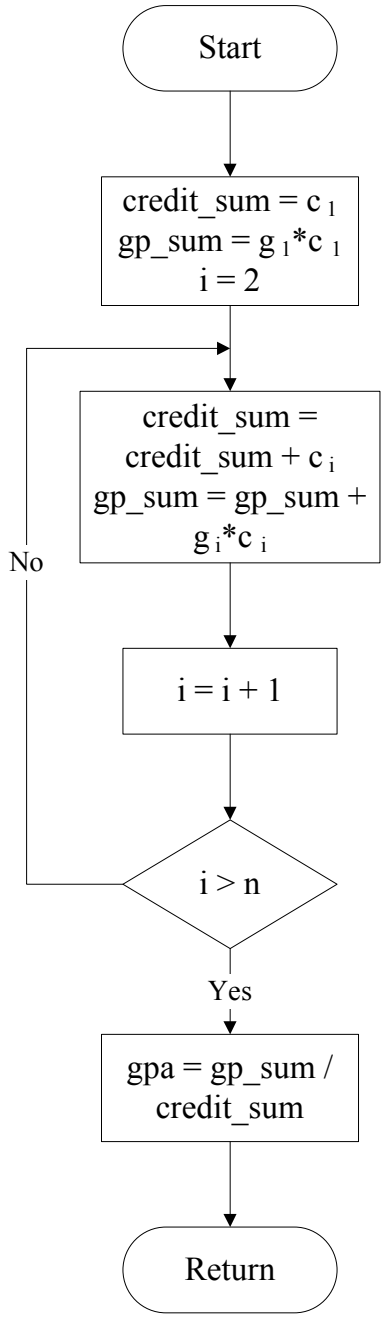
โปรแกรมนี้มีใช้วิธีการคล้ายกับการหาค่าเฉลี่ย (ตัวอย่างที่ 2) จะต่างกันก็แต่เพียงการคำนวณหา Grade Point, การคำนวณหาจำนวนหน่วยกิต และนำมาหารกันเท่านั้น อนึ่ง ในตัวอย่างนี้ จะแสดง Flowchart ในกรณีที่แบ่งเป็นโปรแกรมย่อยหลายอัน (รูปที่ 11.13)



(a) โปรแกรมทั้งหมด



(b) โปรแกรมย่อยสำหรับรับค่าเกรดและหน่วยกิต



(c) โปรแกรมย่อยสำหรับหาค่า GPA

รูปที่ 11.13 Flowchart ของโปรแกรมสำหรับหาค่า GPA

<u>ตัวอย่างที่ 6</u> โปรแกรมสำหรับเรียงลำดับข้อมูล (Sorting)

การเรียงลำดับมีวิธีการหลายวิธี ในตัวอย่างนี้ จะใช้วิธี Bubble Sort ในการเรียงลำดับตัวเลขจากน้อยไปหามาก มีหลักการกล่าวโดยสรุป คือ จะสลับค่าระหว่างจำนวนที่อยู่ติดกัน ซึ่งมีผลทำให้จำนวนน้อยที่สุดลอยขึ้นมาในลักษณะเดียวกับฟองอากาศ กรณีที่มีข้อมูล n ตัว จะต้องทำการสลับที่ในลักษณะดังกล่าวนี้ จำนวน n-1 รอบ

จึงจะเสร็จ สมมุติว่า ข้อมูลที่จะทำการเรียงลำดับ เป็น

7 2 4 3 1 6 5

ข้อมูลจะเปลี่ยนในลักษณะดังนี้

รอบที่ 1	1	7	2	4	3	5	6
รอบที่ 2	1	2	7	3	4	5	6
รอบที่ 3	1	2	3	7	4	5	6
รอบที่ 4	1	2	3	4	7	5	6
รอบที่ 5	1	2	3	4	5	7	6
รอบที่ 6	1	2	3	4	5	6	7

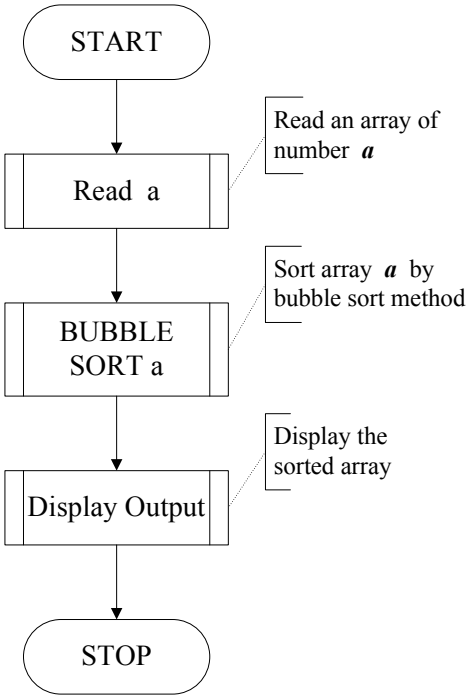
ส่วนที่ทำการเรียงลำดับเรียบร้อยแล้ว

Algorithm

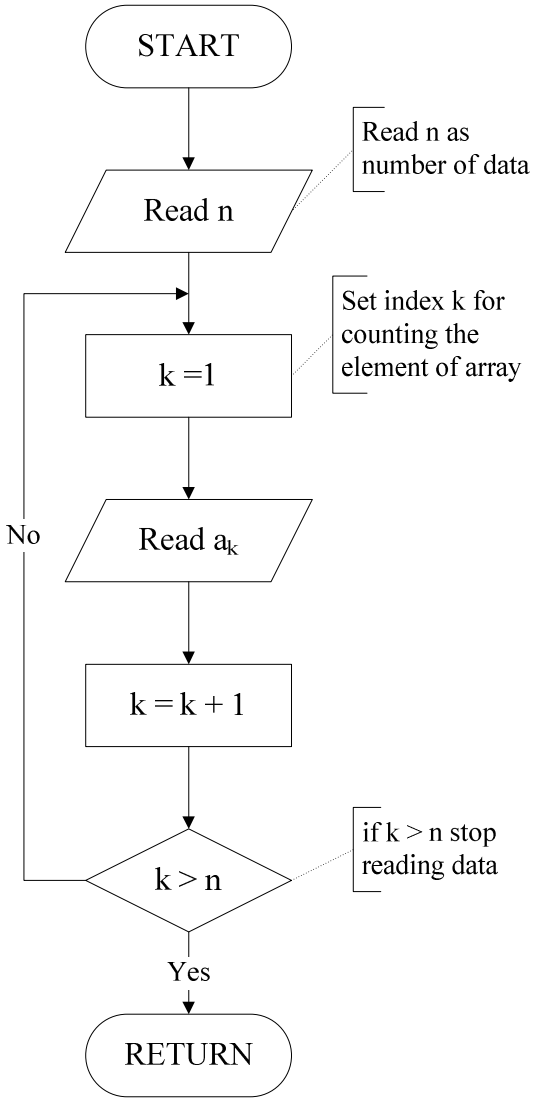
1. รับชุดตัวเลขที่ต้องการจะเรียงลำดับเข้ามา โดยมีขั้นตอนดังนี้
    1.1 รับค่าจำนวนตัวเลข
    1.2 รับจำนวนตัวเลขจนได้ครบตามจำนวนที่ต้องการ
2. เปรียบเทียบเลข 2 ตัวที่อยู่ติดกัน โดยเริ่มจากตัวเลขคู่ที่อยู่หลังสุด
    2.1 ถ้าเลขที่อยู่หลังมีค่าน้อยกว่า ให้สลับตำแหน่งกัน
    2.2 ทำซ้ำจนกว่าจะครบทุกคู่ของตัวเลข
    2.3 ให้ตัดตัวเลขที่อยู่หน้าสุดออกจากชุดตัวเลขที่ต้องการเรียงลำดับ
3. ทำซ้ำขั้นตอนที่ 2 จนกว่าจำนวนชุดตัวเลขที่ยังไม่ได้ทำการเรียงลำดับเหลือ 1
4 แสดงผลการเรียงลำดับ โดยแสดงทีละจำนวนจนกว่าจะครบตามจำนวนของชุดตัวเลขที่ป้อนเข้าไป

Flowchart ของโปรแกรมนี้จะได้ตามรูปที่ 11.14 (a)-(e)

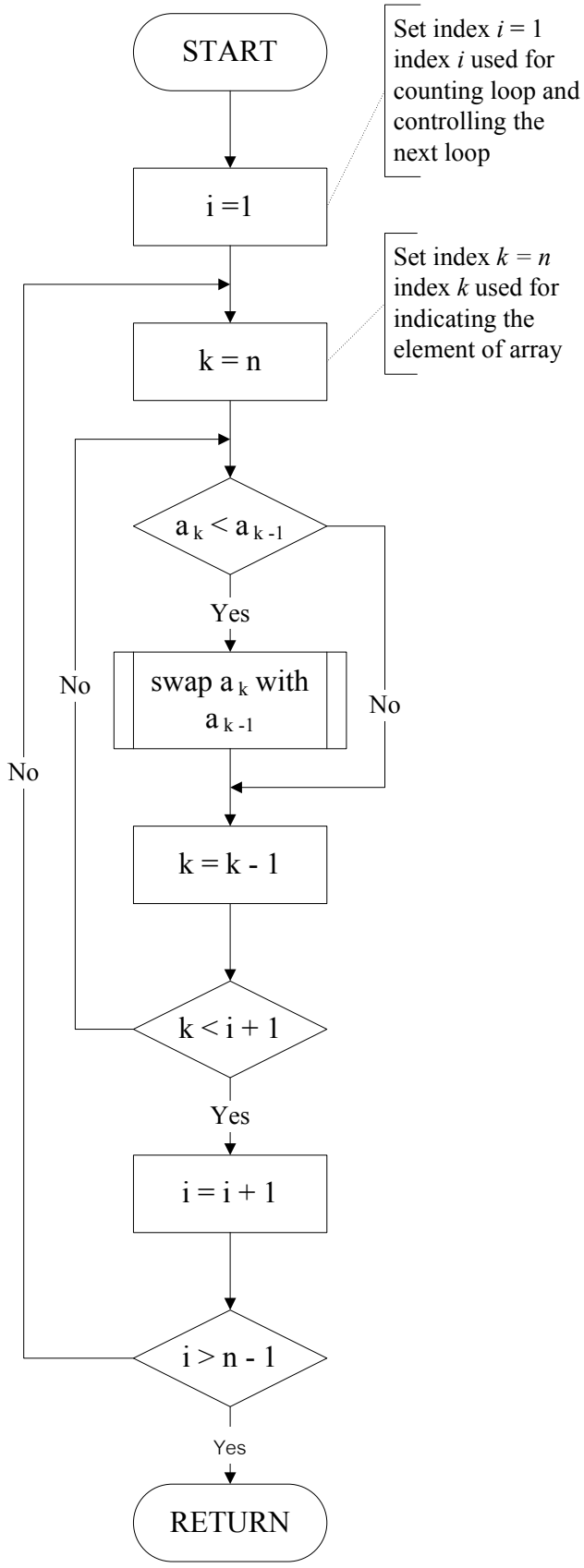
<u>หมายเหตุ</u> ในกรณีที่เขียนคำอธิบายโปรแกรมที่มีขนาดใหญ่ มักจะเขียนเฉพาะ Algorithm ที่สำคัญ ซึ่งในที่นี้คือ Bubble-Sort Algorithm ก็เขียนเฉพาะข้อ 2 และ 3 ส่วน ข้อ 1 และ 4 นั้น เป็นส่วน Input และส่วน Output ซึ่งเป็นส่วนที่คล้ายคลึงกันสำหรับทุกโปรแกรม ดังนั้น จึงไม่มีความจำเป็นต้องเขียนอธิบาย



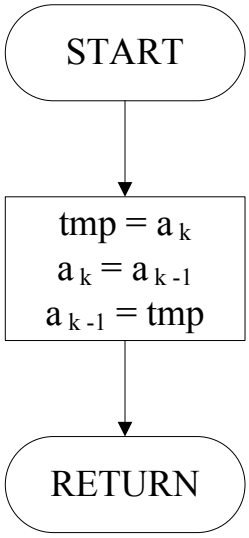
(a) โปรแกรมทั้งหมด



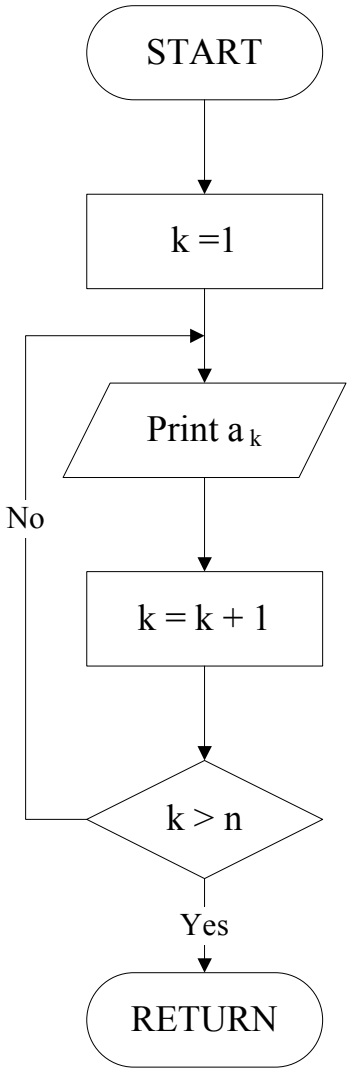
(b) โปรแกรมย่อยส่วนรับข้อมูล



(c) โปรแกรมย่อยส่วน Bubble sort



(d) โปรแกรมย่อยส่วนสลับข้อมูล



(e) โปรแกรมย่อยส่วนแสดงผล

รูปที่ 11.14 Flowchart ของโปรแกรมสำหรับเรียงลำดับข้อมูลด้วยวิธี Bubble Sort

ตัวอย่างที่ 7 การเรียงลำดับโดยวิธี Insertion Sort

การเรียงลำดับด้วยวิธีนี้ มีหลักการคือ จะแบ่งเป็น 2 ส่วน คือ ส่วนที่เรียงลำดับเรียบร้อยแล้ว กับส่วนที่ยังไม่ได้เรียงลำดับ วิธีการเรียงก็คือ จะนำข้อมูลที่ยังไม่ได้รับการเรียงลำดับ มาใส่ในตำแหน่งที่ถูกต้องในส่วนทีเรียงลำดับเรียบร้อยแล้ว สมมุติมีข้อมูล 5 2 6 3 1 4 9 จะได้ผลลัพธ์เป็นดังต่อไปนี้

step 1 2 5 6 3 1 4 9

step 2 2 5 6 3 1 4 9

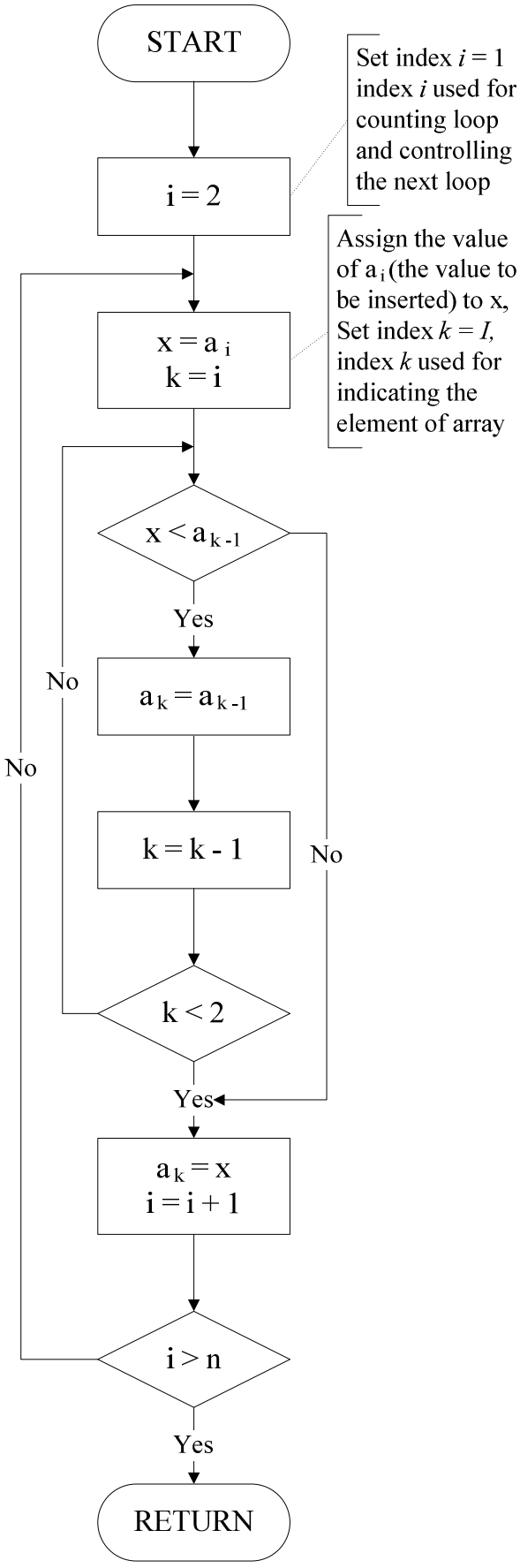
step 3 2 3 5 6 1 4 9

step 4 1 2 3 5 6 4 9

step 5 1 2 3 4 5 6 9

step 6 1 2 3 4 5 6 9

Flowchart ของโปรแกรมนี้จะได้ตามรูปที่ 11.15



รูปที่ 11.15 Flowchart แสดงโปรแกรมย่อยวิธี Insertion Sort

11.12 Pseudocode

Pseudocode (หรือรหัสเทียม รหัสจำลอง รหัสลำลอง) เป็นคำอธิบายขั้นตอนการทำงานของโปรแกรม โดยใช้คำภาษาอังกฤษ และภาษาการเขียนโปรแกรม แบบโครงสร้าง หรืออาจจะใช้ภาษาไทยก็ได้ แต่เนื่องจากภาษาสำหรับเขียนโปรแกรมทั่วไปมีรากฐานจากภาษาอังกฤษ การใช้ภาษาอังกฤษจึงเป็นสิ่งพึงประสงค์ Pseudocode ที่ดีต้องมีความชัดเจน สั้น และได้ใจความ จุดประสงค์หลักของ Pseudocode คือ ช่วยให้ผู้เขียนโปรแกรมสามารถพัฒนาขั้นตอนต่าง ๆให้เป็นโปรแกรมได้ง่ายขึ้น ทำให้มีลักษณะคล้ายคลึงกับโปรแกรมมากกว่าอัลกอริธึม อาจกล่าวได้ว่า Pseudocode เป็นกึ่งกลางระหว่างอัลกอริธึมกับโปรแกรม

ความหมายของ Pseudocode มีผู้ให้ความหมายไว้ต่างๆ กัน เช่น

1. หมายถึง การเขียนโปรแกรมโดยไม่ต้องคำนึงถึงไวยากรณ์ แต่เป็นภาษาที่ผู้เขียนโปรแกรมเข้าใจกันได้ มีลักษณะเป็นภาษาอังกฤษธรรมดาส่วนหนึ่ง กับภาษาโปรแกรม (programming language) อีกส่วนหนึ่ง
2. หมายถึง การทำงานของโปรแกรมที่นำเสนอเป็นภาษาอังกฤษที่ขาดๆหายๆ ซึ่งเป็นส่วนผสมระหว่างภาษาอังกฤษธรรมดากับภาษาโปรแกรม Pseudocode นี้มีไว้ให้ผู้เขียนโปรแกรมเห็นภาพคร่าวๆ โดยไม่ต้องลงไปในรายละเอียด
3. เป็นทั้งวิธีการออกแบบและการแสดงแบบโปรแกรมสำหรับใช้ในการเขียนคำสั่ง ความหมายของคำว่า Pseudoinstruction (คำสั่งลำลอง) ก็คือ คำสั่งที่ไม่ใช่คำสั่งในภาษาคอมพิวเตอร์อย่างแท้จริง แต่เป็นคำสั่งที่เขียนเลียนแบบคำสั่งจริงอย่างย่อ และใช้แนวทางของคำสั่งควบคุมในภาษานั้น นั่น คือ เราอาจจะใช้คำสั่งควบคุม เช่น Do While, If, Else แทนลอจิกของโปรแกรมเพื่อให้เห็นแนวทางการทำงานในโปรแกรม แต่วิธีการทำงานนั้นเราสามารถใช้ภาษาอังกฤษระบุอย่างย่อได้
4. ใช้ข้อความที่เป็นภาษาไทยหรืออังกฤษก็ได้ ในการแสดงขั้นตอนการแก้ปัญหา แต่จะมีคำเฉพาะที่เป็นภาษาในโครงสร้างของโปรแกรมมาช่วยในการเขียน โครงสร้างของ Pseudocode จะมีส่วนคล้ายกับการเขียนโปรแกรมมาก
5. Pseudocode เป็นเครื่องมือที่นิยมใช้กันมากในการออกแบบโปรแกรม ช่วยให้โปรแกรมเมอร์ สามารถเขียนโปรแกรมได้ง่ายขึ้น ภาษาหรือคำที่ใช้เขียน Pseudocode เป็นการผสมผสานระหว่างคำในภาษาอังกฤษทั่วไป กับภาษาคอมพิวเตอร์ โครงสร้างของ Pseudocode จึงมีส่วนที่คล้ายคลึงกันกับการเขียนโปรแกรมมาก

กล่าวโดยสรุปคือ รหัสลำลองหรือ pseudocode เป็นคำบรรยายที่เขียนแสดงขั้นตอนวิธี (Algorithm) ของการเขียนโปรแกรม โดยใช้ภาษาที่กะทัดรัด สื่อสารกับโปรแกรมเมอร์ผู้เขียนโปรแกรม โดยอาจใช้ภาษาที่ใช้ทั่วไปและอาจมีภาษาที่ใช้ในการเขียนโปรแกรมประกอบ แต่ไม่มีมาตรฐานแน่นอนในการเขียน pseudocode และไม่สามารถนำไปทำงานบนคอมพิวเตอร์โดยตรง (เพราะไม่ใช่คำสั่งในภาษาคอมพิวเตอร์) และไม่ขึ้นกับภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง ในปัจจุบัน นิยมใช้ pseudocode แสดง Algorithm มากกว่าใช้ผังงาน เพราะผังงานอาจไม่แสดงรายละเอียดมากนักและใช้สัญลักษณ์ซึ่งทำให้ไม่สะดวกในการเขียนโปรแกรมใหญ่ ๆ

ข้อดีของ Pseudocode

1. ลดความซับซ้อนของการพัฒนาโปรแกรม โดยทั่วไป ในการพัฒนาโปรแกรมขนาดใหญ่ จำเป็นต้องแบ่งขั้นตอนการพัฒนาเป็นการเตรียมอัลกอริธึมเพื่อใช้แก้ปัญหา และการเขียนโปรแกรม เพื่อลดความซับซ้อนของแต่ละขั้นตอน โดยในขณะที่ขั้นตอนการเตรียมอัลกอริธึม จะเน้นที่การหาวิธีแก้ปัญหา การเขียน Pseudocode ช่วยในการตรวจสอบความถูกต้องของอัลกอริธึม อีกทั้งยังช่วยให้การเขียนโปรแกรมเป็นไปอย่างมีประสิทธิภาพมากขึ้น

2. เพิ่มความยืดหยุ่นในการเขียนโปรแกรม เนื่องจาก Pseudocode ไม่ขึ้นกับภาษาเขียนโปรแกรมภาษาใดภาษาหนึ่ง จึงสามารถนำไปพัฒนาเป็นโปรแกรมโดยใช้ภาษา C, Fortran, Java หรืออื่นๆ ทำให้สะดวกต่อผู้เขียนโปรแกรม
3. ง่ายต่อการทำความเข้าใจ เนื่องจาก Pseudocode อยู่ในรูปแบบที่เรียบง่าย เรียงลำดับเป็นขั้นตอน และใช้สำนวนกระทัดรัด ชัดเจน และได้ใจความ จึงเป็นสิ่งที่ผู้เขียนโปรแกรมสามารถทำความเข้าใจได้โดยง่าย

โดยทั่วไป Pseudocode มีรูปแบบดังนี้

```
Algorithm <name>
1. ----------------------------
2. ----------------------------
:
5. ----------------------------
End.
```

ตัวอย่างที่ 8 Pseudocode สำหรับการหาค่าผลบวกของเลข 3 จำนวน ที่รับเข้ามาทางแป้นพิมพ์

```
Algorithm Summation of 3 numbers
1. SUM = 0
2. INPUT (value1)
3. INPUT (value2)
4. INPUT (value3)
5. Compute SUM = value1 + value2 + value3
6. OUTPUT (SUM)
End.
```

ตัวอย่างที่ 9 Pseudocode สำหรับการคำนวณหาพื้นที่สามเหลี่ยม

```
Algorithm Triangle area
1. area = 0
2. Read Base
3. Read Height
4. Compute area = 1/2 * Base * Height
5. Print area
End.
```

ตัวอย่างที่ 10 Pseudocode สำหรับคำนวณหาค่าเกรดเฉลี่ย

```
Algorithm Grade point average
1. Set total and grade point to zero
2. Set grade counter to one
3. Input the number of subjects
4. While grade counter is less than or equal to the number of subjects
      1. Input grade and credits
      2. Add credits into total
      3. Add product of credits and grade into grade point
      4. Add one to grade counter
5. Set grade point average to the grade point divided by total
6. Print grade point average
End.
```

<u>หลักเกณฑ์ทั่วไปของการเขียน Pseudocode</u> มีการสรุปหลักเกณฑ์ ในการเขียนไว้หลายแบบ ดังตัวอย่างต่อไปนี้

แบบที่1:

1. สัญลักษณ์ที่ใช้ในการดำเนินการทางคณิตศาสตร์ต่างๆจะถูกใช้งานตามปกติ คือ + สำหรับการบวก – สำหรับการลบ * สำหรับการคูณ และ / สำหรับการหาร
2. ชื่อข้อมูลใช้แทนจำนวนที่จะถูกดำเนินการตามขั้นตอนวิธี
3. การใช้คำอธิบายกำกับขั้นตอนวิธี อาจทำโดยใช้สัญลักษณ์ * หรือ ** กำกับหัวท้ายข้อความคำอธิบาย เพื่อแยกออกมาจากขั้นตอนการทำงาน
4. คำสงวนบางคำที่ใช้ในภาษาระดับสูงทั่วไปอาจนำมาใช้ เช่น Read หรือ Enter สำหรับการรับข้อมูลเข้า และ Write หรือ Print สำหรับการแสดงข้อมูล
5. การเพิ่มหรือลดระยะเยื้องอย่างเหมาะสม เพื่อแสดงระดับของขั้นตอนการทำงานในโครงสร้างควบคุมการทำงานในกลุ่มเดียวกัน

แบบที่ 2:

1. ถ้อยคำหรือประโยคคำสั่ง(statement) ให้เขียนอยู่ในรูปที่แบบของภาษาอังกฤษอย่างง่าย
2.ในหนึ่งบรรทัด ให้เขียนประโยคคำสั่งเพียงคำสั่งเดียว
3.ควรใช้ย่อหน้าให้เป็นประโยชน์เพื่อแยกคำเฉพาะ (Keywords) ได้อย่างชัดเจน รวมถึงจัดโครงสร้างการควบคุมให้เป็นสัดส่วน ซึ่งการกระทำดังกล่าวจะทำให้ง่ายต่อการอ่าน
4. แต่ละประโยคคำสั่งให้เขียนลำดับจากบนลงล่างโดยมีเพียงทางเข้าทางเดียวและมีทางออกทางเดียวเท่านั้น
5.กลุ่มของประโยคคำสั่งต่างๆ อาจจัดรวมกลุ่มเข้าด้วยกันในรูปแบบของโมดูล แต่ต้องกำหนดชื่อโมดูลเหล่านั้นด้วย เพื่อให้สามารถเรียกใช้โมดูลเหล่านั้นได้

<u>สำนวนที่นิยมใช้ในการเขียน Pseudocode</u>

1. การรับข้อมูลเข้า การแสดงผลข้อมูล และการบันทึกข้อมูล
   - การรับข้อมูลเข้า มักจะนิยมใช้คำว่า READ, ENTER หรือ INPUT ตามด้วยชื่อตัวแปรที่ต้องการเก็บข้อมูล ถ้าหากมีหลายตัวใช้สัญลักษณ์ ; (semi-colon) กั้น
   - การแสดงผล ใช้คำว่า OUTPUT หรือ PRINT
   - การบันทึกข้อมูล ใช้คำว่า WRITE หรือ SAVE
2. การคำนวณ
   - ใช้คำว่า Compute ตามด้วยชื่อตัวแปรที่ต้องการเก็บค่าจากการคำนวณ ตามด้วยเครื่องหมายเท่ากับ และนิพจน์ของการคำนวณ เช่น

     Compute area = a × b
   - ใช้ตัวดำเนินการ (operator) และฟังก์ชันทางคณิตศาสตร์ที่เป็นที่เข้าใจกันทั่วไป เช่น + - * / sin(x)
   - การกำหนดค่า (assignment) โดยทั่วไป นิยมใช้เครื่องหมาย = โดย a = b หมายถึง กำหนดค่าของตัวแปร b ให้เป็นค่าของตัวแปร a (ดังนั้น ตัวแปร b จำเป็นต้องมีค่า)
3. การตัดสินใจ และทดสอบทางเลือก
   - มักจะใช้คำว่า IF หรือ IF-THEN-ELSE และ ENDIF

```
IF a>0 THEN
  PRINT POSITION
ELSE PRINT NEGATIVE
ENDIF
```

   - สำหรับในกรณีที่มีมากกว่า 2 ทาง จะใช้คำว่า CASE และ ENDCASE เช่น

```
CASE num OF
  1 : PRINT ONE
  2 : PRINT TWO
  3 : PRINT THREE
ENDCASE
```

4. การกระทำแบบวนซ้ำ (Loop)

- การทำซ้ำจนกว่าเงื่อนไขที่กำหนดจะเป็นจริงจึงหยุดทำ (REPEAT UNTIL) กรณีนี้คล้ายกับประโยคคำสั่ง do…while ในภาษา C

```
REPEAT
      Statement 1
      Statement 2
  ------------------
UNTIL (CONDITION)
```

- มีการตรวจสอบเงื่อนไขก่อนเข้าทำ (Precondition) ถ้าเงื่อนไขเป็นจริง จึงจะเข้าทำคำสั่งภายใน (WHILE … DO)

```
WHILE (CONDITION) DO
      Statement 1
      Statement 2
  ------------------
ENDWHILE
```

- ทำคำสั่งภายในตามจำนวนครั้งที่กำหนด คล้ายกับ for loop ในภาษา C

```
FOR (NUMBER OF TIMES) DO
      Statement 1
      Statement 2
  ------------------
ENDFOR
```

หรือ

```
FOR (each data in the set, i = 1 to 10)
DO
      Statement 1
      Statement 2
  ------------------
ENDFOR
```

5. การกำหนดเงื่อนไข ใช้คำว่า EQUAL AND OR NOT หรือสัญลักษณ์ >, <, >=, <=

6. การเรียกใช้โปรแกรมย่อย ใช้คำว่า CALL ควบคู่กับ with และ returning เช่น CALL sqrt with a returning sqrt_a

7. การกำหนดชื่อตัวแปร ในกรณีที่ต้องการอธิบายค่าของตัวแปรแต่ละตัวเพื่อให้เกิดความชัดเจนยิ่งขึ้น ควรเขียนคำอธิบายในลักษณะดังนี้

* num = the number to the program

* i = index for the looping

และควรอยู่ถัดจากบรรทัดชื่อ Algorithm หรือก่อนขั้นตอนที่ 1

ตัวอย่างที่ 11 Pseudocode ในการบวกเลข 1+2+3+...+100 และพิมพ์ผลลัพธ์ออกมา

```
Algorithm Sum 1 to 100
1. I = 1
2. SUM = 0
3. WHILE ( I<=100 )
      1. Compute SUM = SUM + I
      2. Compute I = I + 1
4. ENDWHILE
5. PRINT SUM
END.
```

ตัวอย่าง pseudocode ของโปรแกรมที่กำหนดให้รับค่าตัวเลขจำนวนเต็ม 3 ค่า แล้วเรียงลำดับค่าจากน้อยไปมาก แล้วแสดงผลลัพธ์ อาจเป็นดังนี้

```
Algorithm Sorting 3 numbers
1. input nbr1, nbr2, nbr3
2. if nbr3 is less than nbr2
      1. swap nbr2 with nbr3
3. endif
4. if nbr2 is less than nbr1
      1. swap nbr1 with nbr2
5. endif
6. if nbr3 is less than nbr2
      1. swap nbr2 with nbr3
7. endif
8. output nbr1, nbr2, nbr3
END.
```

ตัวอย่าง Pseudocode จากส่วนของโปรแกรมต่อไปนี้ (ข้อสอบโอลิมปิก วิชาคอมพิวเตอร์ปี2551 รอบที่1)

```
Algorithm ???
1. sum = 0
2. for i = 1 to n
      1. for j = 1 to i
            1. if (j mod i EQUAL 0)
                  1. for k = 0 to j-1
                        1. sum = sum + 1
                        2. k = k + 1
                  2. endfor
            2. endif
            3. j = j + 1
      2. endfor
      3. i = i + 1
3. endfor
4. print sum
END.
```

ถ้า n มีค่าเท่ากับ 10 หลังจากจบการทำงานแล้ว sum มีค่าเป็นเท่าใด