บทที่ 3
ขั้นตอนการเขียนโปรแกรม

คอมพิวเตอร์เป็นอุปกรณ์ทางอิเล็กทรอนิกส์อย่างหนึ่ง ซึ่งไม่สามารถทำงานด้วยตนเองได้ แต่จะสามารถทำงานได้ตามชุดคำสั่งในโปรแกรมที่ป้อนเข้าสู่เครื่อง ซึ่งทำงานในลักษณะที่ละคำสั่ง (Step by Step) ซึ่งชุดคำสั่งที่เขียนขึ้นนี้จะต้องเขียนเป็นภาษาที่เครื่องเข้าใจได้ นั่นคือ ภาษาเครื่อง Machine Language แต่ถ้ามีการเขียนด้วยภาษาอื่น ที่ไม่ใช่ภาษาเครื่องเรียกว่า ภาษาชั้นสูง High Level Language ก็จะต้องมีตัวแปลภาษาหรือเรียกอีกอย่างว่า คอมไพเลอร์ (Compiler) ให้เป็นภาษาเครื่องอีกที

ภาษาคอมพิวเตอร์ชั้นสูงในปัจจุบันนี้มีอยู่ด้วยกันมากมายหลายภาษา แต่จะมีโครงสร้างโดยทั่วไปแล้วคล้ายกัน คือจะเป็นภาษาอังกฤษ ซึ่งทำให้ง่ายต่อการอ่านและทำความเข้าใจ แต่อาจจะมีรูปแบบของประโยคคำสั่งที่แตกต่างกันไปบ้าง แต่ก็ไม่ยากต่อการเรียนรู้

ในการเขียนโปรแกรมหรือภาษาคอมพิวเตอร์นี้ โดยทั่วไปแล้วแต่ละภาษาจะมีหลักเกณฑ์ในการเขียนและการออกแบบโปรแกรมเหมือนกัน

ซึ่งสามารถที่จะแบ่งขั้นตอนการเขียนโปรแกรมออกได้เป็น 7 ขั้นตอนดังนี้

1. ขั้นตอนการวิเคราะห์ปัญหา (Analysis the Problem)
2. ขั้นตอนการออกแบบโปรแกรม (Design a Program)
3. ขั้นตอนการเขียนโปรแกรมโดยใช้ภาษาใดภาษาหนึ่ง (Coding)
4. ขั้นตอนการตรวจสอบข้อผิดพลาดของโปรแกรม (Testing and Debugging)
5. ขั้นตอนการทดสอบความถูกต้องของโปรแกรม (Testing and validating)
6. ขั้นตอนการทำเอกสารประกอบโปรแกรม (Documentation)
7. ขั้นตอนการบำรุงรักษาโปรแกรม (Program Maintenance)

3.1 การวิเคราะห์ปัญหา (Analysis the Problem

สิ่งที่นักเขียนโปรแกรม(Programmer) จะต้องทำในขั้นตอนนี้ได้แก่

1. การระบุข้อมูลออก (Output Specification) จะพิจารณาว่างานที่ทำมีเป้าหมายหรือวัตถุประสงค์อะไร ต้องการผลลัพธ์ที่มีรูปร่าง หน้าตาเป็นอย่างไร จะต้องคำนึงถึงผู้ใช้เป็นหลักในการออกแบบผลลัพธ์

2. การระบุข้อมูลเข้า(Input Specification) ต้องรู้ว่าข้อมูลเข้าที่จะส่งเข้ามาในโปรแกรมนั้นมีอะไรบ้าง เป็นข้อมูลที่จะป้อนเข้าสู่คอมพิวเตอร์ เพื่อให้โปรแกรมทำการ ประมวลผล และออกผลลัพธ์

3. กำหนดวิธีการประมวลผล(Process Specification) ต้องรู้สูตรหรือวิธีการประมวลผลเพื่อให้ผลลัพธ์ตามต้องการ

ตัวอย่าง 1 ต้องการหาค่าคะแนนเฉลี่ยของนักเรียนในชั้น

การวิเคราะห์ปัญหาเริ่มจาก

กำหนดวัตถุประสงค์ : การคำนวณหาค่าคะแนนเฉลี่ย

รูปแบบผลลัพธ์ : รายชื่อ คะแนน

……. …….

……. …….

……. …….

คะแนนเฉลี่ย = …..

ข้อมูลเข้า : 1. จำนวนนักเรียนทั้งหมด

2. คะแนนของนักเรียนแต่ละคน

วิธีการประมวลผล : ใช้สูตรในการคำนวณ

$$\text{คะแนนเฉลี่ย} = \frac{\text{ผลรวมของคะแนนทั้งหมด}}{\text{จำนวนนักเรียนทั้งหมด}}$$

3.2 การออกแบบโปรแกรม (Design a Program)

เนื่องจากคอมพิวเตอร์จะต้องมีการทำงานตามคำสั่งของมนุษย์เท่านั้น ดังนั้นจึงต้องมีการบรรจุคำสั่งต่าง ๆ หรือที่เรียกว่าโปรแกรม เพื่อป้อนเข้าสู่เครื่องคอมพิวเตอร์ให้ทำงานได้ ซึ่งในการสั่งงานให้คอมพิวเตอร์ทำงานนั้น คอมพิวเตอร์จะมีการทำงานตามคำสั่งในโปรแกรม เป็นขั้น ๆ ไปอย่างมีลำดับ ดังนั้นจึงต้องเขียนคำสั่งให้ละเอียดชัดเจน และตีความได้อย่างเดียวเท่านั้น ซึ่งจะเรียกลำดับขั้นตอนในการสั่งงานนี้ว่า อัลกอริทึม (Algorithm)

ตัวอย่างของอัลกอริทึมเช่น อัลกอริทึมการสระผม จะต้องเริ่มด้วยการทำผมให้เปียก เมื่อเปียกแล้วจึงใส่แชมพู แล้วขยี้ให้มีฟองเกิดขึ้น หลังจากนั้นก็ล้างออกด้วยน้ำแล้วเริ่มทำใหม่อีกครั้งเป็นต้น

จึงเห็นได้ว่าอัลกอริทึมนี้จะต้องมีการทำงานเป็นลำดับขั้นตอน จะข้ามไปข้ามมาไม่ได้ นอกจากจะต้องเขียนสั่งไว้ต่างหาก ซึ่งการทำงานของคอมพิวเตอร์ก็จะเป็นลักษณะเดียวกันนี้เอง

ในการเขียนอัลกอริทึมนี้แม้จะมีความชัดเจนอยู่ในตัวแล้ว แต่มีจุดอ่อนที่คำอธิบายค่อนข้าง เยิ่นเย้อและถ้าผู้เขียนใช้สำนวนที่อ่านยากก็จะทำให้เป็นชุดคำสั่งที่ไม่ดีนัก ดังนั้นจึงมีการคิดค้นเครื่องมือที่ช่วยอธิบายการทำงาน เพื่อช่วยในการเขียนโปรแกรมดังนี้

ผังงาน (Flowchart)

ซึ่งจะแสดงถึงขั้นตอนการแก้ปัญหาที่ละขั้นตอนในลักษณะ ของรูปภาพทำให้สามารถอ่าน และทำความเข้าใจได้ง่าย

รหัสจำลอง (Pseudo-code)

จะมีรูปแบบเป็นภาษาพูดง่าย ๆ จะเป็นภาษาอังกฤษหรือภาษาไทยก็ได้ โดยจะแสดงขั้นตอนการแก้ปัญหาเป็นขั้นตอนหลัก ๆ แต่ไม่ต้องเจาะเข้าไปในรายละเอียดของการทำงานในแต่ละส่วน

แผนภูมิโครงสร้าง (Structure Charts)

จะมีลักษณะการแบ่งงานใหญ่ออกเป็นโมดูลย่อย ๆ ซึ่งเรียกว่า การออกแบบจากบนลงล่าง (Top-Down Design) และแต่ละโมดูลย่อยก็ยังสามารถแตกออกได้อีกจนถึงระดับที่ล่างสุด ที่สามารถเขียนโปรแกรมได้อย่างง่าย

ฮิโปชาร์ต (HIPO Chart : Hierarchy Input/Output Chart)

จะมีการบอกว่าข้อมูลเข้าคืออะไร มีโปรเซสทำอะไรบ้าง และมีผลลัพธ์อะไรบ้าง แต่จะเห็นภาพได้ไม่ชัดเจนเท่ากับแผนภูมิโครงสร้าง

3.3 การเขียนโปรแกรมโดยใช้ภาษาใดภาษาหนึ่ง (Coding)

หลังจากที่ผ่านขั้นตอนที่สองคือการออกแบบโปรแกรมแล้ว ขั้นต่อไปคือการเขียนโปรแกรม ซึ่งเป็นนำอัลกอริทึมที่สร้างจากขึ้นตอนการออกแบบ มาแปลให้เป็นโปรแกรมคอมพิวเตอร์นั่นเอง ซึ่งในการเขียนโปรแกรมคอมพิวเตอร์นั้น สามารถเลือกใช้ได้หลายภาษา ตัวอย่างของภาษาคอมพิวเตอร์ได้แก่ ภาษาเบสิก(BASIC) ภาษาโคบอล(COBOL) ภาษาปาสคาล (PASCAL) ภาษาซี(C) ฯลฯ แต่ละภาษาก็จะมีรูปแบบ โครงสร้างหรือ ไวยกรณ์ของภาษาที่แตกต่างกันออกไป

ดังนั้นการเขียนโปรแกรมที่ดีนั้น ควรจะต้องทำตามขั้นตอนคือ เริ่มตั้งแต่ วิเคราะห์ปัญหาให้ได้ก่อน แล้วทำการออกแบบโปรแกรม จึงจะเริ่มเขียนโปรแกรม ในการเขียนโปรแกรมนั้นสำหรับผู้ที่ยังไม่มีประสบการณ์การเขียนโปรแกรมเพียงพอ ก็ควรจะทอลองเขียนในกระดาษก่อนแล้วตรวจสอบจนแน่ใจว่าสามารถทำงานได้แล้วจึงทำการคีย์ลงเครื่อง เพื่อเป็นการประหยัดเวลาทำให้สามารถทำงานได้เร็วขึ้น

3.4 การตรวจสอบข้อผิดพลาดของโปรแกรม (Testing and Debugging the Program)

โดยเฉลี่ยแล้วเวลา 50-70 เปอร์เซ็นต์ ของเวลาในการพัฒนาโปรแกรม จะถูกใช้ไปในการหาข้อผิดพลาด (Error) ของโปรแกรมและแก้ไขข้อผิดพลาดนั้น

สามารถแบ่งรูปแบบของข้อผิดพลาด(Error) นี้ออกได้เป็น 3 แบบ

Syntax Error

เป็นข้อผิดพลาดจากการใช้ไวยกรณ์ของภาษาที่ผิดหรือ อาจเกิดจากการสะกดคำผิด เป็นต้น ข้อผิดพลาดที่ง่านที่สุดต่อการหาและการแก้ไข

Run-time Error

เป็นข้อผิดพลาดที่ทำให้เกิดความผิดปกติทางด้านการทำงานของโปรแกรมในระหว่างการปฏิบัติงาน(Execution) โดยทั่วไปมักเกิดจากความรู้เท่าไม่ถึงการณ์ เช่น ทำการเขียนโปรแกรมสั่งงานคอมพิวเตอร์ให้คำนวณหาค่า 1/x โดย x >= 0 และ x<=100

เมื่อคอมพิวเตอร์แทนค่า x ด้วย 0 ก็จะเกิดข้อผิดพลาดประเภท Run-time Error นี้ขึ้นทันที เนื่องจากไม่มีค่าใด ๆ ที่หารด้วย 0 ได้

Logical Error

เกิดจากการตีความหมายของปัญหาผิดไป เป็นข้อผิดพลาดที่หาและแก้ไขยากที่สุด ต้องทำการไล่โปรแกรมที่ละคำสั่งเพื่อหาข้อผิดพลาดนั้น เช่น ให้ทำการคำนวณหาค่า x จากสูตรดังต่อไปนี้ โดยมีการกำหนดค่า a,b และ c มาให้

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

ในการสั่งให้คอมพิวเตอร์ทำการคำนวณ ถ้าเขียนประโยคคำสั่งดังนี้

x = -b +(b^2-4*a*c)^0.5/2*a

คอมพิวเตอร์จะตีความเป็น

$$x = -b \pm \frac{\sqrt{b^2 - 4ac}}{2} * a$$

ซึ่งที่ถูกแล้วควรจะเขียนดังนี้

x = (-b +(b^2-4*a*c)^0.5)/(2*a)

โดยทั่วไปแล้ว หลังจากที่เขียนโปรแกรมเสร็จสิ้นแล้ว จะมีวิธีที่จะตรวจสอบข้อผิดพลาดของโปรแกรม เป็นขั้นตอนดังต่อไปนี้

1. ตรวจสอบด้วยตนเอง (Self Checking)

โดยลองเขียนโปรแกรมลงบนกระดาษ แล้วไล่เช็คตรวจสอบการทำงานทีละขั้นด้วยตนเอง ว่าจะมีการทำงานที่ถูกต้องตามความต้องการหรือไม่ เพราะบางครั้งอาจเกิดข้อผิดพลาดแบบที่ 3 ได้ ซึ่งควรระวังเป็นพิเศษ มิฉะนั้นแล้วคอมพิวเตอร์ อาจจะเช็คไม่เจอข้อผิดพลาด แต่ผลลัพธ์ที่ได้ไม่เป็นจริงเป็นต้น

2. ตรวจสอบด้วยการแปลโปรแกรม (Translating)

หลังจากที่เขียนโปรแกรมเสร็จแล้ว และมีการตรวจสอบด้วยตนเองเรียบร้อย ก็จะต้องผ่านโปรแกรมนี้เข้าเครื่องคอมพิวเตอร์ เพื่อทำการแปลให้เป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจ นั่นคือภาษาเครื่อง (Machine Language) นั่นเอง การแปลนี้จะเป็นการตรวจสอบความผิดพลาดของโปรแกรมด้วย ซึ่งในระหว่างที่แปลอาจจะเจอข้อผิดพลาดในแบบที่ 1 หรือแบบที่ 2 ก็ได้ ก็จะต้องทำการแก้ไขในข้อผิดพลาดนั้น

ในการแปลโปรแกรมนั้นจะต้องอาศัยตัวแปรโปรแกรมเรียกว่า คอมไพเลอร์(Compiler) ซึ่งถ้ามีข้อผิดพลาดใด ๆ ก็จะแจ้งให้ทราบ เรียกข้อผิดพลาดใด ๆ ที่แจ้งให้ทราบนี้ว่าเป็นข้อความไดแอคนอสติก (Diagnostic Message) เพื่อให้ทำการแก้ไขให้ถูกต้อง แล้วจึงค่อยสั่งให้แปลใหม่จนกว่าจะถูกต้องแล้วสามารถใช้งานโปรแกรมได้

ตัวอย่างของการแสดงข้อความไดแอคนอสติคเช่น การเขียนโปรแกรมด้วยภาษาฟอร์แทน แล้วสั่งให้คำนวณโดยใช้คำสั่ง ว่า
N = 2*(I+J)) ซึ่งจะเห็นว่า มีวงเล็บเกินมา 1 อัน เมื่อคอมไพเลอร์ของภาษาฟอร์แทรนแปลโปรแกรมแล้วเจอคำสั่งนี้ ก็จะฟ้องข้อความไดแอคนอสติคว่า "unmatcher parentheses" แต่ถ้าเป็นคอมไฟเลอร์ของภาษาอื่น ก็อาจจะฟ้องข้อความที่ต่างกันก็ได้

3.5 การทดสอบความถูกต้องของโปรแกรม (Testing and Validating)

บางครั้งโปรแกรมอาจผ่านการคอมไพล์ โดยไม่มีข้อผิดพลาดใด ๆ แจ้งออกมา แต่เมื่อนำโปรแกรมนั้นไปใช้งาน ปรากฏว่าได้ผลลัพธ์ที่ไม่เป็นจริง เนื่องจากอาจเกิดข้อผิดพลาดแบบ Logical Error ขึ้นได้ ดังนั้นจึงควรจะต้องมีขั้นตอนการทดสอบความถูกต้องของโปรแกรมอีกทีด้วย

ซึ่งการทดสอบความถูกต้องของข้อมูลจะมีอยู่หลายวิธีดังนี้

1. กรณีที่ข้อมูลถูกต้อง (Valid case) จะทดสอบโดยใส่ข้อมูลที่ถูกต้องลงในโปรแกรม เพื่อทดสอบผลลัพธ์ที่ได้ว่าตรงกับที่ต้องการหรือไม่

2. การใช้ขอบเขตและความถูกต้องของข้อมูล (Range check and Completeness check) เป็นการเช็คขอบเขตของข้อมูล เช่น วันที่ที่เป็นไปได้จะต้องไม่เกินวันที่ 31 ถ้าเกินจะต้องไม่ผ่าน

หรือความสมบูรณ์ของข้อมูล เช่น การรับข้อมูลที่เป็น วัน/เดือน/ปี จะต้องใส่เป็นตัวเลข 6 ตัวในลักษณะ dd/mm/yy ถ้าใส่น้อยกว่า 6 ตัวจะไม่รับ เป็นต้น

3. การใช้ความสมเหตุสมผล (Consistency Check) เช่น ถ้ามีฟิลด์(Field) ข้อมูลที่เป็นเพศ (หญิง,ชาย) และรายละเอียดส่วนตัวของคน ๆ นั้น

เพศ	วันลาคลอด
m	ต้องไม่มี(ห้ามใส่)
f	อาจมีหรือไม่มีก็ได้

4. ข้อมูลที่เป็นตัวเลขและตัวอักษร (Correct No. and Type character check) เป็นการตรวจสอบว่าถ้าเป็นฟิลด์ที่เป็นตัวเลขอย่างเดียวเช่น จำนวนเงิน ก็ควรจะป้อนข้อมูลได้เฉพาะตัวเลขเท่านั้น ไม่อนุญาตให้ใส่ตัวอักษรในฟิลด์นั้น หรือถ้าเป็นฟิลด์ที่เป็นตัวอักษร ก็จะป้อนตัวเลขไม่ได้ เป็นต้น

5. ข้อมูลเป็นไปตามข้อกำหนด (Existence Check)

ข้อมูลที่ป้อนต้องเป็นไปตามที่กำหนดไว้แน่นอนแล้วเท่านั้น เช่น กำหนดให้ฟิลด์นี้ป้อนข้อมูล เฉพาะตัวเลขที่อยู่ในกลุ่ม 1,2,5,7 ได้เท่านั้น จะป้อนเป็นตัวเลขอื่นที่ไม่อยู่ในกลุ่มนี้ไม่ได้

ในการทดสอบโปรแกรมนี้ยังสามารถแบ่งได้อีก 2 แบบ

1. Program Testing เป็นการทดสอบโปรแกรมแต่ละโปรแกรมต่างหาก แล้วแก้ไขข้อผิดพลาดที่เกิดขึ้น

2. System Testing กรณีที่มีการเขียนโปรแกรมที่เป็นระบบ คือ มีหลายโปรแกรม และจำเป็นที่ต้องใช้โปรแกรมเมอร์หลายคนช่วยกันเขียน หลังจากที่แต่ละคนมีการทำ Program Testing ของตนเองเสร็จแล้ว ก็จะนำโปรแกรมเหล่านั้นมารวมกันให้เป็นระบบเดียว แล้วทำการทดสอบอีกที ซึ่งจะเรียกว่า System Testing โดยทั่วไปแล้ว Program Testing มักจะผ่านแต่ System Testing มักจะไม่ผ่าน เนื่องจากโปรแกรมเมอร์ แต่ละคนอาจมีความเข้าใจในงานไม่ตรงกัน จึงทำงานไม่ประสานกัน ดังนั้น System Testing จึงเป็นเรื่องที่สำคัญ จะต้องทำการแก้ไขจนกว่าจะผ่านให้ได้ และต้องมีการทดสอบข้อมูลนำเข้า เพื่อทดสอบการทำงานของระบบว่าถูกต้องตามต้องการหรือไม่

3.6 การทำเอกสารประกอบโปรแกรม (Documentation)

การทำเอกสารประกอบโปรแกรม คือการอธิบายในรายละเอียดของโปรแกรมว่าจุดประสงค์ของโปรแกรมคืออะไร ใช้งานในด้านไหน ฯลฯ ซึ่งอาจจะเป็นการสรุปรายละเอียดของโปรแกรม และแสดงผังงาน (Flowchart) หรือ รหัสจำลอง (Pseudo-code) ก็ได้

โปรแกรมเมอร์ที่ดีควรจะมีการทำเอกสารประกอบโปรแกรม ทุกขั้นตอนของการพัฒนาโปรแกรม ไม่ว่าจะเป็นขั้นตอนการออกแบบ การเขียนโปรแกรม หรือขั้นตอนการทดสอบ

โปรแกรม ซึ่งการทำเอกสารนั้นจะประโยชน์อย่างมากต่อหน่วยงาน เนื่องจากบ้างครั้งอาจต้องการเปลี่ยนแปลงแก้ไขโปรแกรมที่ได้มีการทำเสร็จไปนานแล้ว เพื่อให้ตรงกับความต้องการที่เปลี่ยนไป จะทำให้เข้าใจโปรแกรมได้ง่ายขึ้นและจะเป็นการสะดวกต่อผู้ที่ต้องเข้ามารับช่วงงานต่อที่หลัง

เอกสารประกอบโปรแกรมจะมีอยู่ 2 แบบ

1. เอกสารประกอบโปรแกรมสำหรับผู้ใช้ (User Documentation)

จะเหมาะสมสำหรับผู้ใช้ที่ไม่เกี่ยวข้องกับการพัฒนาโปรแกรม แต่เป็นผู้ที่ใช้งานโปรแกรมอย่างเดียว จะอธิบายเกี่ยวกับการใช้โปรแกรม ตัวอย่างเช่น

- โปรแกรมนี้ทำอะไร ใช้งานในด้านไหน
- ข้อมูลเข้ามีลักษณะอย่างไร
- ข้อมูลออกหรือผลลัพธ์มีลักษณะอย่างไร
- การเรียกใช้โปรแกรมทำอย่างไร
- คำสั่งหรือข้อมูล ที่จำเป็นให้โปรแกรมเริ่มทำงาน มีอะไรบ้าง
- อธิบายเกี่ยวกับประสิทธิภาพ และความสามารถของโปรแกรม

2. เอกสารประกอบโปรแกรมสำหรับผู้เขียนโปรแกรม (Technical Documentation) จะแบ่งออกได้เป็น 2 ส่วน

- ส่วนที่เป็นคำอธิบายหรือหมายเหตุในโปรแกรม หรือเรียกอีกอย่างหนึ่งว่าคอมเมนท์ (Comment) ซึ่งส่วนใหญ่มักจะเขียนแทรกอยู่ในโปรแกรม อธิบายการทำงานของโปรแกรมเป็นส่วน ๆ

- ส่วนอธิบายด้าน Technical ซึ่งส่วนนี้มักจะทำเป็นเอกสารแยกต่างหากจากโปรแกรม ซึ่งอธิบายในรายละเอียดที่มากขึ้น เช่นชื่อโปรแกรมย่อยต่าง ๆ มีอะไรบ้าง แต่ละโปรแกรมย่อยทำหน้าที่อะไร และคำอธิบายย่อ ๆ เกี่ยวกับวัตถุประสงค์ของโปรแกรมเป็นต้น

3.7 การบำรุงรักษาโปรแกรม (Program Maintenance)

เมื่อโปรแกรมผ่านการตรวจสอบตามขั้นตอนเรียบร้อยแล้ว และถูกทำมาให้ผู้ใช้(user) ได้ใช้งาน ในช่วงแรกผู้ใช้อาจจะยังไม่คุ้นเคย ก็อาจทำให้เกิดปัญหาขึ้นมาบ้าง ดังนั้นจึงต้องมีผู้คอยควบคุมดูแลและคอยตรวจสอบการทำงาน ซึ่งเมื่อมีการใช้งานไปนาน ๆ ก็อาจจะต้องมีการปรับปรุงแก้ไขโปรแกรมให้เหมาะกับเหตุการณ์ และความต้องการของผู้ใช้ที่เปลี่ยนแปลงไป

3.8 โครงสร้างโปรแกรม

การใช้โครงสร้างโปรแกรม ช่วยให้การเขียนโปรแกรมมีประสิทธิภาพและลดข้อผิดพลาด แนวความคิดโครงสร้างโปรแกรมถูกตีพิมพ์ เมื่อปี ค.ศ. 1964 ในประเทศอิตาลี โดย Bohm และ Jacopini ได้กำหนดทฤษฎีโครงสร้างอันประกอบด้วย โครงสร้างควบคุม 3 แบบ นับแต่นั้นมามีผู้แต่งหลายท่าน เช่น Edsger Dijkstra, Niklaus Wirth, Ed Yourdon และ Michael Jackson ได้รับแนวคิดและพัฒนาการใช้โครงสร้างโปรแกรมซึ่งในปัจจุบันรวมถึงการออกแบบจากบนลงล่าง (Top-Down design) และการออกแบบส่วนจำเพาะ (Modular design)

การออกแบบจากบนลงล่าง (Top-Down Design)

โดยทั่วไปโปรแกรมเมอร์ เขียนโปรแกรมเพื่อแก้ไขปัญหาตั้งแต่เริ่มต้นเรื่อย ๆ ไปจนสิ้นสุด ทำให้เกิดปัญหาการติดขัดและทำความเข้าใจได้ยากในแต่ละส่วนของโปรแกรม จึงควรใช้การออกแบบจากบนลงล่างมากำหนดขอบเขตของปัญหา โดยการแบ่งปัญหาออกเป็นส่วน ๆ ในแต่ละส่วนมีรายละเอียดของ การทำงานอย่างใดอย่างหนึ่งเสร็จสมบูรณ์ในตัว ภายหลังแบ่งหน้าที่การทำงานแล้วจึงทำการเขียนเป็น ภาษาคอมพิวเตอร์ ผลลัพธ์ที่ได้จะมีประสิทธิภาพดีกว่าการเขียนโปรแกรมแบบดั้งเดิม

การออกแบบส่วนจำเพาะ หรือ โมดูล (Modular design)

การเขียนโปรแกรมแบบโครงสร้างต้องอาศัยการออกแบบส่วนจำเพาะ โดยการจับกลุ่มการทำงานที่มีลักษณะการทำงานอย่างเดียวกันเข้าด้วยกันเช่น การคำนวณภาษีหรือการพิมพ์หัวรายงาน การออกแบบส่วนจำเพาะเกี่ยวเนื่องโดยตรงกับการออกแบบจากบนลงล่าง เพราะการกำหนดขอบเขตของปัญหาคือตัวกำหนดคุณสมบัติของส่วนจำเพาะแต่ละส่วน การออกแบบส่วนจำเพาะที่ดีต้องอ่านและเข้าใจง่าย

ทฤษฎีโครงสร้าง (Structure Theorem)

ทฤษฎีโครงสร้างก่อนให้เกิดการปฏิวัติการออกแบบโปรแกรม โดยกำจัดการใช้คำสั่ง GOTO และเริ่มใช้โครงสร้างเป็นกรอบช่วยในการแก้ไขปัญหา ทฤษฎีกล่าวว่า โปรแกรมคอมพิวเตอร์ทุกโปรแกรมสามารถเขียนโดยใช้โครงสร้างควบคุมพื้นฐาน 3 แบบ อันได้แก่ แบบเรียงลำดับ (Sequence) แบบทางเลือก (Selection) และแบบวนซ้ำ (Repetition)

3.9 บทสรุป

ขั้นตอนการพัฒนาโปรแกรมสามารถแบ่งได้ 7 ขั้นตอนคือ (1) ขั้นตอนการวิเคราะห์ปัญหา (Analysis the Problem) (2) ขั้นตอนการออกแบบโปรแกรม (Design a Program) (3) ขั้นตอนการเขียนโปรแกรมโดยใช้ภาษาใดภาษาหนึ่ง (Coding) (4) ขั้นตอนการตรวจสอบข้อผิดพลาดของโปรแกรม (Testing and Debugging) (5) ขั้นตอนการทดสอบความถูกต้องของ

โปรแกรม (Testing and validating) (6) ขั้นตอนการทำเอกสารประกอบโปรแกรม (Documentation) (7)ขั้นตอนการบำรุงรักษาโปรแกรม (Program Maintenance)

3.10 แบบฝึกหัดท้ายบท

โจทย์ต่อไปนี้ให้นักศึกษา วิเคราะห์ปัญหา พร้อมออกแบบโปรแกรม

1. จงเขียนขั้นตอนวิธีแสดงการรับตัวอักษร 3 ตัวจากหน้าจอ และแสดงข้อความตอบรับจากหน้าจอ เช่น “Welcome”

2. โปรแกรมทำการรับตัวเลข 2 จำนวน จากหน้าจอ ให้แสดงผลรวมและผลต่างของเลข 2 จำนวนทางหน้าจอเช่นกัน

3. โปรแกรมทำการอ่านอัตราภาษีเป็นเปอร์เซ็นต์ พร้อมราคาสินค้าที่ขาย 5 ชนิด โปรแกรมทำการหาผลรวมราคาขายก่อนคิดภาษี การคิดภาษีให้นำอัตราภาษีคูณด้วยยอดขายรวม ให้พิมพ์ยอดขายรวม ภาษี และยอดขายรวมภาษี

4. โปรแกรมทำการอ่านยอดคงเหลือในบัญชีต้นเดือน ยอดเงินฝากตลอดเดือนและยอดถอนตลอดเดือน ธนาคารคิดค่าบริการ 1% จากยอดเงินฝาก และถอน จงพิมพ์ยอดคงเหลือสิ้นเดือน โดยบวกยอดเงินฝากหัดยอดเงินถอน และเงินค่าบริการ จากยอดบัญชีต้นเดือน

5. จงหาค่าจ้างทั้งสิ้นต่อสัปดาห์ โดยที่โปรแกรมทำการอ่านจำนวนชั่วโมงทำงานตามปกติ จำนวนชั่วโมงทำงานล่วงเวลาและอัตราค่าจ้าต่อชั่วโมง ค่าจ้างปกติคิดจากชั่วโมงทำงานปกติ กับอัตราค่าจ้างต่อชั่วโมง ส่วนค่าจ้างล่วงเวลาคิดเป็นเท่าครึ่งของอัตราปกติ ค่าจ้างต่อสัปดาห์คิดจากค่าจ้างปกติรวมกับค่าจ้างล่วงเวลา

เอกสารอ้างอิง

ครรชิต มาลัยวงศ์ และโกสันต์ เทพสิทธิทรากรณ์. (2542). **ความรู้พื้นฐานทางคอมพิวเตอร์**. กรุงเทพฯ : ชวนพิมพ์.

งามนิจ อาจอินทร์. (2539). **ความรู้ทั่วไปเกี่ยวกับวิทยาการคอมพิวเตอร์**. มหาวิทยาลัยขอนแก่น

พรรณิกา ไพบูลย์นิมิต. (2539). **เทคนิคการออกแบบโปรแกรมแบบโครงสร้าง.** มหาวิทยาลัยเชียงใหม่.

ลอง, ลารี่. (2543). **เทคโนโลยีคอมพิวเตอร์และสารสนเทศ**. กรุงเทพฯ : เพียร์สัน เอ็ดดูเคชั่น อินโดไชน่า, 2543.