บทที่ 4
การออกแบบโปรแกรม

4.1 บทนำ

ในบทนี้จะกล่าวถึงเทคนิคในการสร้างโปรแกรมที่สมบูรณ์ว่าเราจะออกแบบโปรแกรมได้อย่างใด กับได้กล่าวมาแล้วในบทต้นว่ากระบวนการจะสร้างโปรแกรมนั้นมีขั้นตอนอย่างไร ในที่นี้จะสรุปเฉพาะประเด็นที่สำคัญอีกครั้งดังนี้

4.2 ขั้นตอนการสร้างและพัฒนาโปรแกรม

1. แยกแยะปัญหาที่จะแก้ไข (Define the problem)

2. ออกแบบโปรแกรมโดยใช้เครื่องมือต่าง ๆ เช่นเดียวกันกับการออกแบบแปลนสร้างบ้าน (Construct a program design document or 'blueprint')

3. นำคู่มือในการออกแบบโปรแกรมในขั้นที่ 2 มาถอดรหัสเป็นภาษาคอมพิวเตอร์โดยเลือกภาษาที่เหมาะสมกับงาน (Code the program in a suitable programming language)

4. ค้นหาที่ผิดพลาดในโปรแกรม (Debug)

5. ทดสอบโปรแกรมว่าสามารถทำงานตามวัตถุประสงค์ที่วางไว้หรือไม่ (Testing)

6. เขียนเอกสารประกอบการใช้และและอธิบายโปรแกรม เพื่อจะได้ใช้เป็นคู่มือในการพัฒนาโปรแกรมดังกล่าวต่อไปในอนาคต (Prepare Whatever additional documenta-tionin needed to facilitate its conprehension and maintenance over aperiod of time)

Problem definition

ก่อนการสร้างโปรแกรม ผู้เขียนโปรแกรมจำเป็นจะต้องเข้าใจในสิ่งต่อไปนี้ คือ Input, Process, Output

Input

จะต้องทราบว่าเราจะรับข้อมูลจากแหล่งใดใช้อุปกรณ์อะไรลักษณะโครงสร้างของข้อมูล เช่น format เป็นเช่นใด รายละเอียดนี้แยกแยะออกมาได้เป็นประเด็นต่อไปนี้

- ข้อมูลมาจากแหล่งใด เช่น ภายในโปรแกรมสร้างขึ้นมาเอง หรือรับจากแป้นพิมพ์ภายนอก (External File)

- โครงสร้างของข้อมูลมีลักษณะดี คือ

  type เป็น string หรือ numeric

  size ในกรณีที่เป็น string แล้วจะมีขนาดความยาวเท่าใด และในกรณีที่เป็น numeric เป็นชนิดใด เช่น Integer หรือ Real หรือ Rouble

Process

การประมวลผลนั้นเราจะเลือกใช้ algorithm แบบใดจะให้ประสิทธิภาพสูงสุด นอกเหนือจากการประมวลผลข้อมูลแล้ว การประมวลผลยังอาจหมายถึงการแปลงระบบข้อมูลให้อยู่ในลักษณะทำงานได้อย่างมีประสิทธิภาพสูงขึ้น

Output

ข้อมูลที่ได้จากการประมวลผลแล้วจะแสดงผลออกทางใด ใช้อุปกรณ์อะไร มีโครงสร้างของข้อมูลเช่นใด ยกตัวอย่างเช่นถ้าใช้เครื่องพิมพ์จะต้องคำนึงถึงรายละเอียดต่าง ๆ เช่น จำนวนบรรทัดที่พิมพ์ หัวตาราง, และลักษณะโครงสร้างของการพิมพ์ออกมา

4.3 Top Down decomposition

หลักการของการสร้างโปรแกรมก็คือ ให้พยายามแยกรายละเอียดของกิจกรรมที่จะดำเนินการออกมาเป็นกระบวนการย่อยๆโดยจะเรียกแต่ละกระบวนการ (procedure) นี้ว่า Module การแบ่งแยกโปรแกรมให้ออกมาเป็น Module ย่อย ๆ ก็เพื่อเจตนาลดขนาดของโปรแกรมให้สั้นลง และลดความยุ่งยากซับซ้อนของโปรแกรมลง

หลักการแบ่งย่อยจากโปรแกรมเดียวให้ออกมาเป็น module ย่อย ๆ นั้นมีปัจจัยในการแบ่งดังนี้

- module ที่แบ่งแล้วสามารถจะดำเนินการได้ใน 1 กิจกรรม (Task) หรือจะกล่าวได้ว่าการแบ่ง Module ที่เหมาะสมก็คือยึดเอาลักษณะการปฏิบัติการ ชนิด Complete Set Function

- ในแต่ละ Module ควรจะประกอบด้วยกลุ่มคำสั่ง ช่วง 60-120 คำสั่ง การกำหนดขนาดเช่นนี้ เพราะเรายึดถือลักษณะทางกายภาพของมนุษย์เป็นหลักว่ามีความสามารถในการอ่านโปรแกรมได้ยาว 1 หรือ 2 หน้า (หน้าหนึ่งยาวประมาณ 60 คำสั่ง)โดยที่ยังมีความสามารถที่จะจดจำเรื่องราวที่ Module ดังกล่าวกำลังดำเนินกิจกรรมอยู่

วิธีการที่จะแบ่ง module ย่อยได้ก็คือ ตรวจดูจากกิจกรรมแรกที่ดำเนินการจนในที่สุดคำตอบที่ได้ในขั้นสุดท้าย ซึ่งเรียกวิธีนี้ว่า Top down decomposition จะขอยกตัวอย่างของโปรแกรมในงานทำ payroll มาแสดงให้ดูถึงวิธีการแยกโปรแกรมออกมาเป็น moduleย่อย ๆ ได้ดังนี้ คือ ระบบ payroll โดยทั่วไปมักจะมีกิจกรรมที่ทำดังนี้คือ

- เก็บรายละเอียดข้อมูลของคนงานในบริษัทในแต่ละเดือน (สัปดาห์)
- นำข้อมูลที่ได้มาตรวจสอบความถูกต้อง และความน่าเชื่อถือ (เช่นคนงานมีการทำงานในบางวันเกิน 24 ชั่วโมง
- ภายหลังการตรวจสอบข้อมูลแล้ว ให้นำข้อมูลดังกล่าวไปคิดเงินเดือนตามเงื่อนไขอัตราค่าจ้าง ซึ่งแตกต่างกันไปตามแต่ละประเภทของงาน
- มีการเก็บประวัติการจ่ายเงินไว้ในแฟ้มประวัติ
- พิมพ์จำนวนเงินที่จะจ่ายให้กับคนงานออกมาทางกระดาษ

จากกิจกรรมดังกล่าวใน payrol system เราสามารถจะแบ่งออกมาเป็น 7 Module ย่อยได้ดังนี้

(1) Obtain Employee's work details from terminal operator

(2) Check employee's details for resonableness

(3) Read employee's history record from main payrolls file (ie the master file)

(4) Calculate the pay and coinage analysis

(5) Update the employee's history record

(6) Print the employee's pay details

(7) Print factory payroll tables

จากทั้ง 7 Module จะเห็นได้ว่า Module 1-6 จะเป็นที่ module ที่วนเวียนทำงานตามจำนวนคนงาน ในขณะที่ module ที่ 7 จะเป็น module ที่ดำเนินการเพียงครั้งเดียวก่อนจะจบงาน

ภายหลังเมื่อเราแยก module ย่อยออกมาได้แล้ว อาจจะขยายรายละเอียดของแต่ละ Module ภายในอีกได้เพื่อความสะดวกในการนำ Module เหล่านี้ไปเขียนโปรแกรมต่อไป

การที่เราแบ่งโปรแกรมเดี่ยวออกมาเป็นโครงสร้างของ Modula Approach นี้จะมีข้อดีหลายประการ คือ

1. ลดขนาดของโปรแกรมที่จะตรวจสอบและค้นหาที่ผิดพลาดให้ลดลงอันจะทำให้งานลดภาระลง เพราะจากการแบ่งเป็น Module จะทำให้แต่ละ Module เสมือนเป็นโปรแกรมย่อยที่เป็นเอกเทศต่างหาก

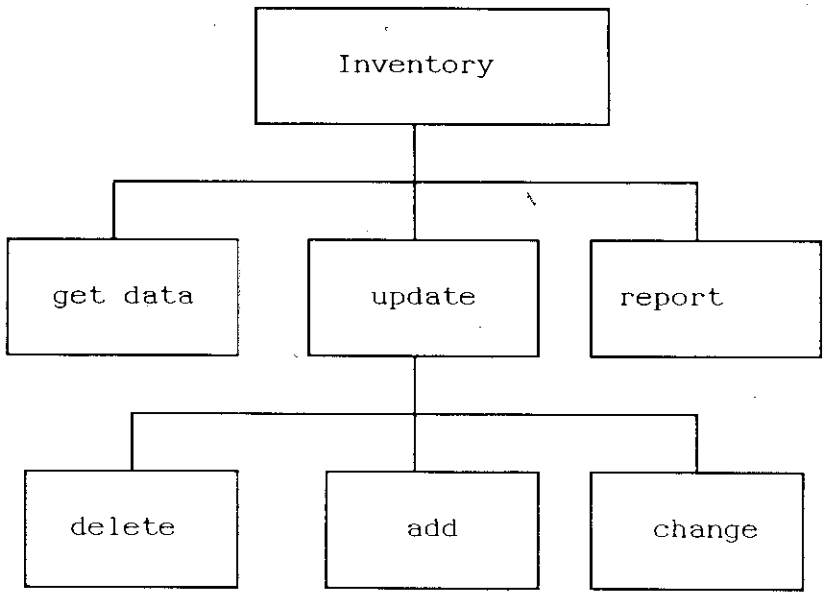
2. การแบ่งเป็น module จะทำให้เราสามารถแบ่งงานการเขียนโปรแกรมให้กับโปรแกรมแถวละคน ได้สะดวกและง่าย

3. แนวคิดของการสร้าง module ในรูปแบบที่เรียกว่า external module จะทำให้เกิดการใช้ทรัพยากรโปรแกรมร่วมกัน ซึ่งอาจจะเรียกอีกนัยหนึ่งว่าการสร้าง Public subprogram ขึ้นมาใช้ ซึ่งในแนวคิดนี้จะทำให้ programmer ต้องสร้าง module ที่เป็นมาตราฐาน ทั้งรูปแบบ โครงสร้าง และการเขียนเอกสารประกอบ Module

4. ในกรณีที่มี library Function ซึ่งก็คือ Module ประเภทหนึ่งนั่นเองปรากฏอยู่ใน Compiler ต่าง ๆ อยู่แล้วเราก็ไม่จำเป็นจะต้องไปสร้าง Module ของตนเองขึ้นมาใช้ เพียงแต่เราเรียก Library Function พวกนี้มาใช้ได้เลยขอเพียงติดต่อให้ถูกต้องภายใต้เงื่อนไขของ Library Function ที่ต้องการเหล่านั้น เราก็สามารถเรียกมาใช้ได้เลยซึ่งเท่ากับเป็นการลดภาระงานการเขียนโปรแกรมไปได้ส่วนหนึ่ง

5. การออกแบบโปรแกรมในลักษณะที่เรียกว่า Modula Approach จะง่ายในการออกแบบและกำหนดทำความเข้าใจในลักษณะการติดต่อของ Module ในการปฏิบัติงานได้

เครื่องมือที่ใช้ในการออกแบบโมดุลนั้น เรามักจะใช้ HIPO ดังรูปที่แสดงในระบบงาน Inventory Control ดังนี้



จากภาพดังกล่าวระบบงาน Inventory จะประกอบด้วย 7 Module ย่อย โดยที่Module Inventory นั้นเราจะเรียกว่า Main Module ซึ่ง Main Module จะเรียก Module Get Data, Module Update และ Module Report และในขณะที่เรียก Module update นั้น Module update จะไปเรียก Module ย่อยอื่นมาอีก 3 Module คือ Module delete, Module Add, และ Module change

4.4 หลักการออกแบบ Module ที่ดี

1. Inter-module communication

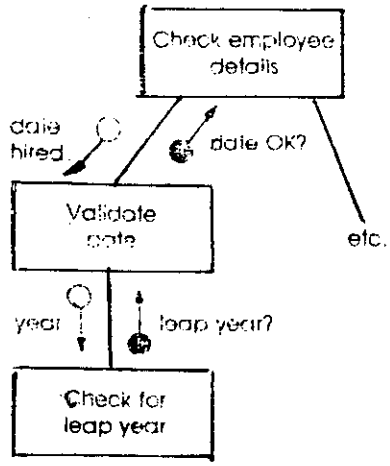
ภายใต้ข้อกำหนดที่ว่า module นั้นก็คือ โปรแกรมที่ดำเนินงานใน 1 ฟังก์ชัน ดังนั้นผลการทำงานใดจะต้องไม่ก่อให้เกิด side effect (ripple effcet) ให้กับ Module อื่นในระบบงานเดียวกันการติดต่อระหว่าง Module จะมีเฉพาะเงื่อนไขของ parameter เท่านั้น ปกติแล้วรูปแบบการ pass parameter ของ module จะอยู่ในรูปแบบใดรูปแบบหนึ่งดังนี้

1. ส่ง information ไปแล้วรับ information กลับ

2. ส่ง information ไปแต่ไม่รับอะไรกลับ เช่น ส่งข้อมูลไปให้ Module ที่ทำหน้าที่ในการพิมพ์

3. ไม่ส่ง information แต่จะรับ information กลับ

ภาพต่อไปนี้จะแสดงวิธีการติดต่อระหว่าง Module



2. Module strength or cohesion

Module strength หรือ cohesion เป็นมาตรการในการวัด การเหนี่ยวนำของคำสั่งใน Module ดังกล่าวว่า คำสั่งเหล่านี้มีความยึดเกาะกันมากเท่าใด ถ้าในกรณีที่มีความยึดเกาะระหว่างคำสั่งต่าง ๆ ภายใน Module มาก ก็แสดงว่า เราออกแบบ Module ดี เราสามารถแบ่งระดับการยึดเกาะของคำสั่งต่าง ๆ ภายใน Module จากระดับสูงสุดไปต่ำสุด ดังนี้

ประเภท	น้ำหนัก
Functional	10 (strong cohesion)
Sequential	9
Communicational	7
Procedural	6
Temporal	3
Logical	1
Concidental	0 (weak cohesion)

Module ที่มีลักษณะเป็น Functional จะหมายความว่า เราไม่สามารถที่จะแบ่งกลุ่มของคำสั่งภายใน Module นั้นให้ออกไปเป็นกลุ่มใหม่ที่ทำหน้าที่หนึ่งได้อีกแล้ว หรืออาจจะกล่าวอีกนัยหนึ่งว่า Module ที่มีลักษณะดังนี้แสดงว่า Module นี้ทำกิจกรรมเพียงอย่างเดียวเท่านั้นตัวอย่างเช่นถ้าเรามี Module หนึ่ง ซึ่งทำหน้าที่ get data และยังทำหน้าที่ตรวจสอบ invalid data ได้อีก แสดงว่า Module นี้ยังไม่ใช่ลักษณะของ Functional แต่ Module ดังกล่าวสามารถแยกเป็นกิจกรรมแต่ละระดับคือ get data และ check invalid data ดังนั้นเราจะเรียก Module ดังกล่าวว่า Sequential ถ้าเราแบ่ง Module ที่มีความเหนี่ยวนำเชิงSequentialออกมาเป็น 2 Module ย่อยซึ่งแต่ละ Module ที่แบ่งแยกออกมาจะมีลักษณะเป็น Module ชนิดที่มีความเหนี่ยวนำเป็น Functional ได้

Module ที่มีความเหนี่ยวนำเป็น Communicational นั้น หมายความว่า module จะประกอบด้วยส่วนที่ทำหน้าที่แยกจากกันแต่ยังใช้โครงสร้างของข้อมูลเหมือนกัน ดังนั้น Module ที่สร้างขึ้นมาเพื่อทำหน้าที่ซ่อนโครงสร้างของข้อมูลนั้นเราจะจัดเป็น Module ที่ยึดเหนี่ยวเชิงCommunicational ตัวอย่างของ Module ที่มีลักษณะการเหนี่ยวนำเชิง Communicational เช่น Module การจัดเก็บข้อมูล, Module การค้นหาข้อมูล เป็นต้น

Module ประเภทที่มีการยึดเหนี่ยวเชิง procedural นั้น มักจะพบในกรณีของการแตกโปรแกรมขนาดยาวให้เป็นส่วน ๆ ไปตามสายการควบคุม โดยไม่คำนึงถึง

กิจกรรมที่ทำ เจตนาเพียงเพื่อจะลดภาวะการดูแลโปรแกรมขนาดยาวให้สั้นลงอันสะดวกในการตรวจสอบโปรแกรมนั้นซึ่งในลักษณะนี้แต่ละ Module ยังคงอยู่ในลักษณะที่ยังมีความสัมพันธ์กันอยู่อย่างมาก ลักษณะนี้เราจะเรียกว่า Module ประเภทนี้มีการยึดเหนี่ยวภายในเชิง procedural ส่วน Module ที่ประกอบด้วยคำสั่งต่างๆที่ทำงานไม่สัมพันธ์กันแต่ต้องประมวลผลในเวลาเดียวกัน เราจะเรียกว่า Module ดังกล่าวมีความเหนี่ยวนำเชิง Temporal หรือ Classical

3. <u>การเชื่อมต่อระหว่างโมดุล (Module Coupling)</u>

การวัดความเป็นอิสระระหว่างโมดุลนั้นจะช่วยในเรื่องแยกแยะว่า โมดุลเหล่านั้นง่ายในการดูแล การเขียนโปรแกรม และการบำรุงดูแลรักษา สิ่งที่เราต้องการในการออกแบบโมดุลก็คือให้โมดุลแต่ละโมดุลเป็นอิสระกันมากที่สุด ความเป็นอิสระจะเกิดขึ้นได้ก็ต่อเมื่อโมดุลแต่ละโมดุลไม่ได้ใช้ข้อมูลร่วมกัน นอกจากนี้การดำเนินภายในโมดุลจะต้องไม่ก่อให้เกิดผลข้างเคียง (side effect) กับโมดุลอื่นในระบบงานเดียวกัน

ตารางต่อไปนี้จะแสดงถึงระดับการเชื่อมโยงระหว่างโมดุล

ประเภท	น้ำหนัก
Independent	0 (weak coupling)
Data	1
Stamp	3
Common	4
Control	5
External	7
Content	9 (strong coupling)

ตารางการเชื่อมโยงระหว่างโมดุลตารางนี้จะมีทิศทางการเลือกใช้ตรงข้ามกับตารางน้ำหนักของการเหนี่ยวนำที่อธิบายมาแล้ว ทั้งนี้เพราะโมดุลที่เราต้องการนั้นจะมีลักษณะเป็น Independent คือมีระดับที่เรียกว่า weak coupling คือโมดุลจะไม่มีส่วนเกี่ยวข้องกันเลยโมดุลที่เชื่อมโยงประเภท data จะหมายความว่าทั้งสองโมดุลที่เชื่อมโยงกันนั้นมีเฉพาะ data ร่วมกันเท่านั้น ซึ่งการติดต่อระหว่าง data ก็จะกระทำโดยการส่งพารามิเตอร์ถึงกันเท่านั้น ดังนั้นโมดุลที่เรียกใช้กันจะรู้เฉพาะชื่อของโมดุลที่เรียกและลักษณะ, จำนวนของพารามิเตอร์เท่านั้น นอกนั้นจะไม่รู้จักโมดุลที่เรียกมาใช้งานว่า โมดุลนั้นนำข้อมูลไปดำเนินงานอย่างใดบ้าง ในลักษณะเช่นนี้ก็หมายความว่าโมดุลนั้นมีสิทธิที่จะเปลี่ยนแปลงปรับปรุงกระบวนการทำงานได้อย่างเป็นอิสระเพียงแต่ขอให้ยังคงพารามิเตอร์ให้มีลักษณะเดิมไว้เท่านั้น ลักษณะของโมดุลประเภทนี้มีข้อควรระวังก็คือการใช้พารามิเตอร์ในลักษณะของการส่งผ่านถ้าเป็นแบบ pass by reference (address) ก็หมายถึงการใช้พื้นที่ข้อมูลร่วมกันกับโมดุลที่เรียกตัวเองนั่นเองโครงสร้างของการ pass by reference ก็คือรูปแบบเดียวกับการใช้คำสั่ง COMMON ในบางภาษาซึ่งทำให้เกิด global variable อันอาจก่อให้เกิดอันตรายกับข้อมูลในส่วนนี้ได้ ทางที่ดีถ้ามีทางเลือกไป การส่งพารามิเตอร์ได้ ควรจะเลือกการ pass by valueจะดีกว่าเพราะจะไม่ก่อให้เกิด side effect ดังที่กล่าวมาแล้วข้างต้น

โมดุลที่เชื่อมกันแบบ Stamp Coupled หมายความว่า พารามิเตอร์ที่ส่งต่อไปนั้นได้รวมโครงสร้างของข้อมูลไว้ด้วย ดังนั้น ในการจะติดต่อระหว่างโมดุลจึงจำเป็นที่จะต้องทราบเข้าใจโครงสร้างของข้อมูลซึ่งกันและกัน ดังนั้นถ้าโครงสร้างของข้อมูลภายในโมดุลหนึ่งเปลี่ยนแปลง เราก็จะต้องไปปรับโครงสร้างของข้อมูลโมดุลที่เชื่อมต่อด้วยลักษณะการเชื่อมต่อแบบนี้ จะทำให้การบำรุงดูแลรักษายุ่งยากเพราะเท่ากับเมื่อบำรุงดูแลรักษาโมดุลหนึ่ง ก็จะต้องดูแลโมดุลอื่นที่เชื่อมโยงด้วย

โมดุลเชื่อมต่อแบบ Common Coupled หมายความถึงโมดุลที่ใช้โครงสร้างข้อมูลร่วมกันเลย ลักษณะเช่นนี้ถ้าโมดุลหนึ่งเปลี่ยนผลกระทบกับอีกโมดุลที่มาเชื่อมโยงจะสูงมาก

โมดุลที่เชื่อมต่อแบบ Control Coupled หมายความว่า การตัดสินใจของโมดุลหนึ่งจะเป็นการควบคุมอีกโมดุลหนึ่ง ตัวอย่างของการเชื่อมต่อแบบนี้ เช่น การส่งสัญญาณการเชื่อมต่อระหว่างประเภทของรับส่งข้อมูล (I/O Module)

โมดุลเชื่อมต่อแบบ External Coupled ก็ต่อเมื่อโมดุลหนึ่งเรียกใช้อีกโมดุลหนึ่งโดยผ่าน External entry point ดังนั้น ผลที่เกิดกับโมดุลหนึ่งจึงเป็นผลทางอ้อม

โมดุลเชื่อมต่อแบบ Content Coupled เป็นกรณีที่เกิดขึ้นเมื่อโมดุลที่มาเชื่อมโยงกันมีรหัสข้อมูลแบบเดียวกัน ซึ่งทำให้การเรียกอีกโมดุลมาทำงานเรียกโดยผ่านจุดเข้าภายใน (Internal entry point) ได้หลายทาง

4.5 การกำหนดลักษณะของโมดุล (Module Specification)

ปัจจัยที่นำมาช่วยในการตัดสินใจกำหนดลักษณะของโมดุล ก็คือ

- ข้อมูลนำเข้ามาได้อย่างไร
- มีกิจกรรมอะไรบ้างที่จะดำเนินการ
- การส่งข้อมูลที่ประมวลผลแล้วออกมา

ลักษณะของโมดุลเพื่อให้สอดคล้องกับการนำไปใช้งานควรจะมีรูปแบบ ดังนี้

- รวมกลุ่มโครงสร้างของข้อมูลเดียวกันไว้ด้วยกัน
- ซ่อนรูปแบบของข้อมูลภายนอก
- ซ่อนการเรียกใช้ข้อมูล
- ซ่อนการดำเนินงาน
- ซ่อนการเลือกใช้ algorithm

นอกจากที่กล่าวมาแล้วถึงประโยชน์ของการใช้ module approach แล้ว แนวคิดการออกแบบนี้ยังช่วยลดความซับซ้อนของระบบงานได้ด้วย

สาเหตุที่เกิดความซับซ้อนของงาน (Complexity) มีอยู่ 3 สาเหตุคือ

1. functional complexity เกิดเพราะภายในโมดุลมีกิจกรรมให้ทำมากเกินไปจนบางครั้งก็มีการตรวจสอบเงื่อนไขมากมายในการแยกแยะวิธีปฏิบัติงานที่จะให้เลือกปฏิบัติ

2. distributed complexity คือเงื่อนไขซึ่ง common function ไม่สามารถแยกแยะได้ ผลก็คือจะไปกระทบกับโปรแกรมอื่น

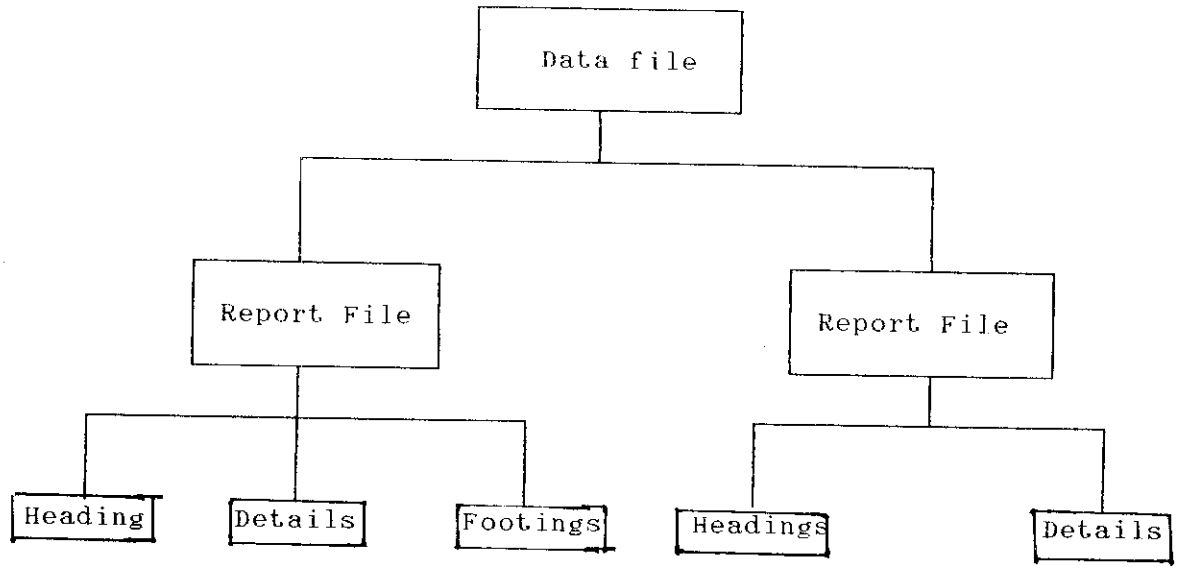
3. connection complexity เกิดจากผลกระทบของแต่ละโมดุลในการใช้ common data ร่วมกันในรูปแบบที่ผิดปกติ

ปกติในการออกแบบโมดุลนั้น เราควรจะเขียนเอกสารกำกับในแต่ละโมดุลประกอบด้วย สิ่งต่อไปนี้ร่วมด้วยนอกจากสิ่งอื่นๆที่จำเป็น

1. An algorithm for solving the problem
2. A domain of permissible input values
3. A range of possible output values
4. A set of side effect

4.7 การออกแบบสายลำดับสำหรับงานออกแบบโมดุล

ปกติเรามักจะออกแบบโดยใช้รูปแบบของ Top down จะใช้รูปแบบของ HIPO (Hierachical Input Plus Output process) วิธีนี้จะเริ่มออกแบบตั้งแต่ระดับบนแล้วแตกแขนงลงมาตามงานที่จะทำต่อในระดับล่างลงไป ดังตัวอย่างต่อไปนี้



ลักษณะการจำแนกและวิธีการเขียนโปรแกรมในลักษณะของโมดุล

ภาษาที่ใช้กันสำหรับการออกแบบโปรแกรมในรูปแบบของโมดุลนั้นแต่ละภาษาก็จะกำหนดแตกต่างกันไป เช่น

ภาษา FORTRAN ก็จะจำแนกโมดุลออกเป็น 2 ประเภทคือ

- Subroutine
- Function

ภาษา PASCAL ก็จะจำแนกโมดุลออกเป็น 2 ประเภทคือ

- Function
- Procedure

ภาษา C ก็จะรวมเรียกโมดุลทุกประเภทเป็นอย่างเดียวกันแล้วเรียกว่า

- Function

(ในหนังสือเล่มนี้จะขออธิบายโดยใช้ ภาษาปาสคาล เป็นหลัก)

ข้อแตกต่างระหว่าง Function และ Procedure ในภาษาปาสคาลนั้นก็คือ Function จะให้ผลิตผลกลับไปยังผู้เรียกเพียง 1 อย่างเท่านั้น รูปแบบที่แตกต่างจากนี้ไปเราถือเป็น Procedure ทั้งหมด (ที่จริง Function ก็จัดเป็น Procedure ประเภทหนึ่ง) ตัวอย่างของโครงสร้างการติดต่อและส่งผ่าน พารามิเตอร์ที่ใช้กันในการโปรแกรม

```
Program testfunc(input,output);
var
      x,y  :real   ;
 function f1(x:real) real;
       var
        :
        begin {begin of function}
            :
        end;  {end of function}
```

```
begin { begin of main}
          :
          :
          y := f1(x);
          :
 end.
```

ตัวอย่างที่ 1

```
Program checkorder(input,output);
 var
  c1,c2,c3 : char;
   working boolean ;
      function inorder (x:char, y:char, z:char):boolean ;
         begin
             if x <= y then
                  inorder := y <= z
               else
                    inorder := false
            end;
      begin {begin of main}
          working := true;
```

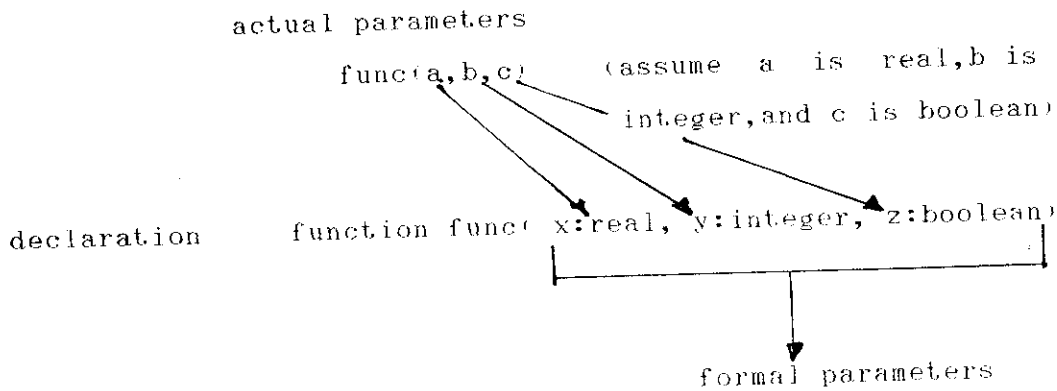
```
        while working do
         begin
           readln(c1,c2,c3);
           write(c1,c2,c3);
            if  (c1  ='*')  and  (c2 ='*') and (c3 ='*') t
              working := false
             else
                 if inorder (c1,c2,c3) then
                     writeln('are in order')
                  else
                     writeln ('are not order')
          end {while loop}
    end.
```

ตัวอย่างที่ 2

```
Program demo(input,output);
var
          a,b,c,i :integer;
function sum(a:integer,b:integer):integer;
   begin
         sum:= a+ b
    end;
 begin
      for i:= 1 to 5 do
          begin
           c := sum(a,b);
           writeln(c)
           end
  end.
```

จากตัวอย่างที่แสดงให้ดูนั้น รูปแบบของการส่งต่อพารามิเตอร์จะปรากฏดังนี้

รูปแสดงความสัมพันธ์ระหว่าง actual parameter และ formal paramete



รูปแบบการเรียก function มาใช้งาน

a. เรียกในฐานะ function เฉยๆ

เช่น

```
—   a := func (b,c) ;
—   writeln('Demonstrate',funct(z);
```

b. เรียกในฐานะ numeric expression

เช่น

```
—     b :=  func1(a:integer) + 5.8  ;
—     c :=  func1(b:integer)/func2(x:integer) ;
```

Procedure เป็น Subprogram Module ประเภทหนึ่งที่เป็นอิสระจาก main program เช่นเดียวกับ funcion แต่จะใช้ได้กว้างขวางกว่าดังที่ได้กล่าวมาแล้ว ลักษณะการเรียกใช้ก็มีโครงรูปคล้ายกับ function ดังตัวแบบต่อปนี้

```
Program main(input,output);
:
:
begin
      module1(list of parameters);
      module2(list of parameters);
      module3(list of parameters);
 :
 :
 end.
```

การ pass parameters ให้กับ Procedure ในกรณีที่มีการส่งผลกลับมายังผู้เรียกนั้น ผลที่เกิดขึ้นจะทำให้ค่าของตัวแปรของผู้เรียก (caller)เปลี่ยน ซึ่งเรียกว่า การ Pass by Reference ในขณะที่ การ Pass อีกวิธีเป็นการ Pass by Value ซึ่งจะไม่ทำให้ค่าของตัวแปรของ caller เปลี่ยนไป ให้ดูจากตัวอย่างต่อไปนี้

ตัวอย่างที่ 3

```
procedure example (x:integer);
begin
        x :=  x+1 ;
        writeln(x)
end;
```

ถ้าผู้เรียกเรียกใช้ดังนี้

```
:
a := 1;
example (a);
writeln(a);
```

ผลที่ปรากฏจะเป็นดังนี้

2 จะเป็นผลจาก procedure

1 จะเป็นผลจาก main

ตัวอย่างที่ 4

```
procedure example (var x:integer);
begin
        x :=  x+1 ;
        writeln(x)
end;
```

ถ้าผู้เรียกเรียกใช้ดังนี้

```
:
a := 1;
example (a);
writeln(a);
```

ผลที่ปรากฏจะเป็นดังนี้

2 จะเป็นผลจาก procedure

2 จะเป็นผลจาก main (เนื่องจาก Procedure เปลี่ยนค่าของ a ไปแล้ว

ตัวอย่างการเรียกใช้ Procedure ที่ถูกต้อง (pass by value)

```
f(x,y)
f(1,a[5]) ;
f(sin(y) + 0.5,z);
f(3.0*z-2.0+sqrt(x),r);
```

ตัวอย่างการเรียกใช้ Procedure ที่ไม่ถูกต้อง (pass by reference)

```
f(x,1);          เพราะ1เป็น constant จะ pass เป็น var paramete
                 ไม่ได้
f(2,sin(y)); เพราะ 2, sin(y) ไม่สามารถเป็น var parameter ไ
```

Procedure ที่เห็นปรากฏในหนังสือโดยทั่วไปนั้นมักจะเป็นรูปแบบของ internal Procedure ซึ่งถ้าเป็นลักษณะเช่นนี้แล้ว เรามักจะใช้ในกรณีของการฝึกเขียนโปรแกรมมากกว่าที่จะนำไปใช้ปฏิบัติงานจริง เพราะตามรูปแบบนี้แล้ว Main Program กับ Procedure จะอยู่ file เดียวกันทำให้การใช้งานมีอุปสรรคหลายประการ อาทิเช่น

- ทำให้ไม่มีการมีการ share resource (procedure) เพราะสภาพนี้เป็นสภาพของ private subprogram
- ทำให้การปรับแก้โปรแกรมไม่คล่องตัว เพราะมีส่วนของ Procedure อยู่ร่วม
- ทำให้ Procedure ขาดความเป็นอิสระ

จากเหตุผลที่กล่าวมาแล้ว ในทางปฏิบัติเรามักจะใช้ Procedure แบบ externa ซึ่งกลไลของการใช้ Procedurec ลักษณะนี้ จะทำให้ลดข้อเสียที่กล่าวมาแล้วข้างต้น

รูปแบบทั่วไปของการดำเนินงานแบบนี้จะปรากฏดังนี้

```
program  graph (input,output);
  :
   procedure plot(x,y:real;pen:integer): extern;
begin {main}
   :
   plot(x,y,2);
```

(หมายเหตุ : ภาษา Pascal บาง version อาจจะใช้รูปแบบที่แตกต่างจากนี้ไป ต้อง ศึกษาการใช้เพิ่มเติม)

การใช้ Function และ Procedure ที่กล่าวมานี้นั้นอาจจะเกิด กรณีของการเรียกตัวเองย้อนกลับมาใช้ ซึ่งเราเรียกกระบวนการเช่นนี้ว่า **Recursion** ตัวอย่างของกระบวนการเช่นนี้ก็คือ Function หาค่า fatorial หรือ Function ที่เห็นบ่อยๆ ใกล้ตัวคือ การหาเงินรวมของการคิดจากกรณีของดอกเบี้ยทบต้น

ตัวอย่างที่ 5

```
program main(input,output);
var
   :
function power (x,a:integer);
 var
   :
  begin
        :
```

```
  for i := 1 to a do
      begin
          :
          power := x *power(x,a-1);
          :

      end;
end.
```

แบบฝึกหัด

1. Describe the scope of all the variables used in the following program fragment

```
program outer;
var
     a:integer;
     b:integer;
        .
        .
        .
procedure p1;
var
     b:integer;
     c:integer;
        .
        .
        .
end;{of p1}
procedure p2;
var
     c:integer;
     d:integer;
        .
        .
        .
```

```
            procedure p3;
            var
                 e:integer;
                      .
                      .
                      .
            end;{of p3}
              .
              .
              .
       end;{of p2}
begin
     .
     .
     .
end.{of outer}
```

2. Consider the following two procedure.

```
       procedure swap1(x,y:real);
       var
            t:real;
       begin
            t:=x;
            x:=y;
            y:=t
       end;
```

```
procedure swap2(var x,y:real);
var
     t:real;
begin
     t:=x;
     x:=y;
     y:=t
end;
```

if a has value 1.2 and b has the value 1.5,what is the result of each of the following invocations?

(a) swap1(a,b);
    writeln(a,b)

(b) swap2(a,b);
    writeln(a,b)

3. Given the following procedure:

```
procedure silly(x:integer;var y:integer);
var
     z:integer;
begin
     x:=5;
     y:=6;
     z:=7
end;{of silly}
```

what is the output procedusr by the following three lin

```
     x:=1;y:=2;z:=3;
     silly(y,x);
     writeln(x,y,z)
```

4. Given the following program

```
program mainline(input,output);
var
     a,b,c :integer;
procedure proc1;
var
     a,c :integer;
     procedure proc2;
     var
          b:integer;
     begin
          a:=4;
          b:=5;
          writeln ('inside proc2,value for a,b and c
                    are currently', a,b,c);
     end;{of proc2}
begin{of proc1}
     c:=6;
     writeln('inside proc1, values for a,b,c are
              currently', a,b,c);
      proc2;
      writeln('still inside proc1,values for a,b,c
               are currently', a,b,c)
end;{of proc1}
```

```
{main program begins here}
begin
     a:=1;
     b:=2;
     c:=3;
     proc1;
     writeln('inside main program,values for a,b,c
                    are currently', a,b,c)
end.{of main program}
```

what would be the exact output of the program if it were excuted ? (If any of the writeln commands attempt to print an undefined qquantity, just indicate that with the symbol '***' and keep tracing.)

5. What is the output that will result from the excution of the following program Denote undefined variables by '***'.

```
program block(output);
var
     a,b,c:integer;
     procedure p1(var x:integer;y:integer);
     var
          b,c,d :char;
          procedure p2;
          var
               x,y:integer;
```

```
        begin
            writeln('in p2:',x:3,y:3,a:3,b:3,c:3,d:3);
            a:=ord(b)
        end;{of p2}
    begin{p1}
        d:='x'
        writeln('in p1:',x:3,y:3,a:3,b:3,c:3,d:3);
        x:=y;y:=0;b:=chr(a);c:=chr(x);
        p2;
        writeln('out p1:',x:3,y:3,a:3,b:3,c:3,d:3);
    end;{of p1}
begin {program block}
    a:=1;b:=2;c:=3;
    p1(a,b);
    p1(b,c);
    p1(c,a)
end;{of block]
```

6. Assume that we had a procedure that merged two sorted listed a and b procuced a new master list c containing all the elements of both a and b. The parameters to the procedure are:

(a) a-a 50 element integer array.

(b) asize-an integer value giving the number of element in a (0<= asize <+ 50).

(c) b-a 2000 element integer array.

(d) bsize-an integer value giving the number of element in b (0<= bsize<= 2000).

(e) c-the new master list. It will be a 2050 element integer array.

(f) csize-the number of elements placed into the list c (0<=csize<=2050).

(g) swicth-a signal flag (see Style Clinic 8-5) set totrue if the merge operation was successful, and set to flase otherwise. For each of the seven parameter to the procedure merge, state what parameter passing mechanism (vabriable or value) you would probably use.

7. Given the following recursion function

```
function dunno(m:integer):integer;
  var
       value :integer;
  begin
       if m = 0 then
            value:=3
       else
            value:=dunno(m-1)+5;
       dunno:=value;
       writeln('current value of m and value are ',m,va
  end;{of dunno}
```

What output is produced by the following statement?

writeln(dunno(3));

8. What is the output produced by the following program?

```
program rec(output);
     function p(x:real; n:integer):real;
     begin
          writeln('in p:',x:6:1,n:3);
          if n = 0 then
               p:=1.0
          else
               if odd(n) then
                    p:=x*sqr(p(x,n div 2))
               else
                    p:=x*sqqr(p(x,n div 2));
               writeln('end of p')
          end;{of function p}

begin
     writeln(p(2.0,13))
end.{of program rec}
```

9. Write a subprogram that computes the value of an investment p after years of interest at rate r compounded quarterly. Should your subprogram be a procedure or a function? Should the parameter be passed by value or as variables?

10. Write a procedure that accepts a character string and returns the same character string with all blanks detects, along with an integer that indicates how many blanks existed in the input string. What will be the parameters of the procedure and what will be their type and passing mechanism?

11. Write a complete Pascal function that computes sin(x) using the approximation formula.

$$\sin(x)$$

The number of terms of the series to be uses should be a parameter of the function. For example ,sin(y,3)should compute sin(y) using the first three terms of the series.

12. A student was given the job of writting a Pascal procedure to find the average of a list of up to 200 examination scores in the range 0 to 150. Here is what was produced.

```
procedure avg;
begin
     total:=0.0;average:=0.0;bad:=0;
     if (number<1) or (number>200)
          then writeln('improper number of scores ')
          else begin
          i:=1;
```

```
        while i <= number do
            begin
                if(list[i]<0.0) or
                  (list[i]>150.0)
                     then bad:=bad+1
                     else total:=total+list[i];
                i:=i+1
            end;

        average:=total/(number-bad);
        writeln('the average is',average)
        end
end;{of avg}
```

*(a) Assume that the procedure avg was going to be put into a program library and used by differrent people. Within this conext,discuss the poor programming habits and poor style displayed by the procedure as it is currently written.

(b) Rewrite avg to removedy these problems.

(c) Write the documentation needed for avg so that it be intelligently and properly

13. Write a program to read N; then read N number. Sort the number in ascending order and print the sorted numbers.

14. For I,J,K and L positive integers less then 20, what integers satisfy the following relationship?

$$I^3 + J^3 + K^3 = L^3$$

15. Write a program to read the age of 100 individuals. Count the number of individuals iln each block of 10 years. That is,

years 0-9

years 10-19

years 20-29

etc.

Print the results of your counts in some readable fashion.

16. Write a program for these number problems.

(a) Use the digits 1 through 9 in different combination and the operators plus and minus to obtain the totas 100. You should need a comuter to find some of them.

(b) Do part (a), but restrict the solutions so the digits appear in ascending or discending order. Here are two exanples.

$$123 + 4 - 5 + 67 - 89 = 100$$

$$9 - 8 + 76 - 5 + 4 + 3 + 21 = 100$$

(c) Do the same as part (b). but use hexadecimal digits (123456789ABCDEF) to obtain the hezxadecximal 100.

17. Write a program that reads a source program and punches a new source programwith sequence numbers in columns 73-80. The program should be able to start at any disired number and increment by any disirecd vaule.

18. Write a program that reads a integer number greater than zero in decimal representation of the number.If the input was 7 204 52 0, the output would be

7 SEVEN

204 TWO HUNDRED FOUR

52 FIFTY TWO

The last number is to be a zero, and no number is to be larger than a billion.

19. Develop a set of guidelines for indenting (paragraphing in your programming language. Then circulate the guidelines and see how many people agree or disagree. Modify them as needed until a consensus develops, but send me a copy.

20. FORTRAN or BASIC. Write a program that accept a source program and requences statement numbers. The program should accept any staring value and any increment. A nice number to start with at several laces in the program Thus you would be able to start the program at 10 and then later on you could indicate another area that is to start at 1000.

21. Cobol or PL/I. Write program that will read a source program and provide a new source program where all the paragraphs and procedures have a numerical sufficx.

22. A lazy(and smarter) wayto do the preceding exercise is to locate anaralready writtenprogram to resequence the statement numbers Then check is out, document it, and make ils generally available to users.

23. Write a program that reads an X array and an Y array. Then scale and plot the data on either a line printer or terminal output. If you have a plotter, use the plotter for output.

24. Write a program to read a source program and output some or all of the following.

(a) How many and percentages for each type of statement: Comments, GO TO ,IF DO , etc.

(b) How long are variable names? How many one-character names, two-charater names, three-character names, , , ,? Can you use the preceding information to develop some scale of good and bad programs?