# บทที่ 6

# การพัฒนาโปรแกรม

ในบทที่ผ่านมานั้นเป็นขั้นตอนในการแก้ปัญหาโดยออกแบบระบบและโปรแกรมให้มีคุณลักษณะที่ดีโดยออกเป็นโมดูล(modular) มีระดับของความข้องเกี่ยวกันระหว่างโมดูลต่ำ มีคุณลักษณะการออกแบบที่มีคุณภาพสูง ซึ่งการออกแบบในส่วนของรายละเอียดเหล่านี้เป็นประโยชน์สำหรับโปรแกรมเมอร์ในการเขียนคำสั่งเพื่อให้คอมพิวเตอร์ทำงาน ในรูปภาพที่ 6.1 แสดงถึงขั้นตอนในการพัฒนาระบบ สำหรับขั้นตอนการสร้างโปรแกรม ตามหลักการของวิศวกรรมซอฟต์แวร์นั้น จะอธิบายถึงการเขียนคำสั่งโปรแกรมที่มีคุณภาพซึ่งมีผลสืบเนื่องมาจากการออกแบบโปรแกรมในบทที่แล้ว

## 6.1

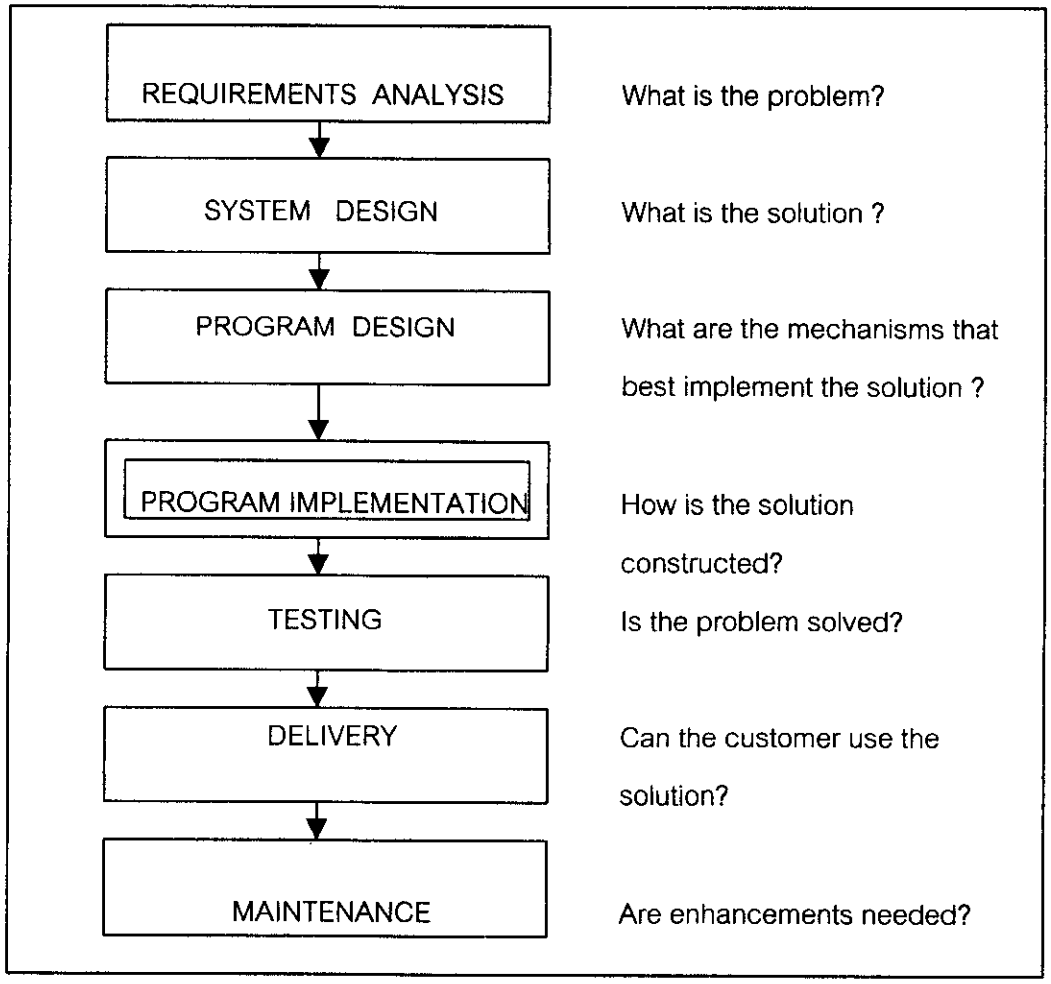
## มาตรฐานและกระบวนการในการเขียนโปรแกรม

การเขียนภาษาโปรแกรมตามที่ได้ออกแบบโปรแกรมไว้นั้น โปรแกรมเมอร์จะต้องทำความเข้าใจถึงการออกแบบ ซึ่งบางครั้งจะมีปัญหาบ้างในการแบ่งระบบให้เป็นโมดูล อย่างไรก็ตามก่อนที่จะเริ่มต้นเขียนคำสั่งโปรแกรมจริง ๆ โปรแกรมเมอร์จะต้องรู้ถึงมาตรฐานและกระบวนการในการจัดการเกี่ยวกับงานที่ต้องรับผิดชอบ เพราะการทำงานเป็นทีมงานที่บางครั้งมีขนาดใหญ่มากๆ การทำงานต้องมีการจัดการทรัพยากรต่างๆที่ดีและเหมาะสม จึงจะเป็นประโยชน์สูงสุด



CT 484 191

รูปภาพที่ 6.1 The System Development Process




## ความจำเป็นสำหรับมาตรฐานและกระบวนการในการเขียนโปรแกรม

อาชีพโปรแกรมเมอร์นั้นต้องเกี่ยวข้องกับการเขียนโปรแกรม ซึ่งได้รับมอบหมายให้พัฒนาในส่วนของโมดูลย่อยต่างๆ บางครั้งโปรแกรมเมอร์เองยังไม่ทราบว่าตนเองเขียนโปรแกรมในส่วนใดของระบบ การมอบหมายงานนั้นจะมีหลายๆโมดูลและในหลายๆระบบขึ้นอยู่กับผู้บริหารโครงงานจะมอบหมายมาให้ ดังนั้นในหลายๆระบบที่มีการออกแบบโปรแกรมที่แตกต่างกันอาจมีข้อผิดพลาดที่เกิดขึ้นต้องมีการปรับปรุงเปลี่ยนแปลงคำสั่ง ซึ่งอาจเป็นการแก้ไขในโมดูลที่ตนเองเคยรับผิดชอบโปรแกรมเมอร์ก็สามารถกระทำได้โดยง่าย แต่บางกรณีต้องช่วยเหลือโปรแกรมเมอร์คนอื่นๆในทีมงาน หรืออาจเป็นการแก้ไขในโมดูลที่ตนเองไม่เคยเห็นมาก่อน ถ้า

การเขียนโปรแกรมไม่เป็นมาตรฐานเดียวกันเขียนไม่มีรูปแบบที่ดี จะก่อให้เกิดปัญหาอย่างมาก ในการทำความเข้าใจและการแก้ไขเป็นอย่างมาก ดังนั้นการเขียนโปรแกรมที่ดีนั้นต้องเขียนคำสั่งโปรแกรมให้ผู้อื่นสามารถเข้าใจได้ด้วย

Standards for You มาตรฐานและขั้นตอนวิธีในการเขียนโปรแกรมสามารถช่วยจัดการความคิด และหลีกเลี่ยงข้อผิดพลาด อทิเช่นการกำหนดถึงระเบียบและวิธีการในการเขียนเอกสารโปรแกรม ซึ่งทีมงานทุกคนต้องปฏิบัติตาม ผลดีคือเอกสารมีความชัดเจนและช่วยให้สามารถเข้าใจและตรวจสอบคำสั่งโปรแกรมเพื่อค้นหาตำแหน่งของความผิดพลาดและแก้ไขคำสั่งโปรแกรมนั้นได้โดยง่าย มาตรฐานและกระบวนการในการเขียนโปรแกรมสามารถช่วยในการเปลี่ยนจากการออกแบบไปเป็นคำสั่งโปรแกรมได้ง่ายขึ้น เมื่อมีการเปลี่ยนแปลงในสิ่งที่ได้ออกแบบไว้จากเดิมอาจเป็นการปรับเปลี่ยนฮาร์ดแวร์หรือรายละเอียดในการติดต่อสื่อสารใหม่ โปรแกรมเมอร์สามารถกระทำได้โดยง่ายและลดความผิดพลาดที่อาจเกิดขึ้นได้

Standard for others มาตรฐานสำหรับคนอื่น ๆ เมื่อเขียนคำสั่งโปรแกรมเสร็จสมบูรณ์ บุคลากรในทีมงานคนอื่นๆอาจต้องการนำไปใช้ ตัวอย่างเช่น ทีมงานทดสอบจะทำการทดสอบคำสั่งโปรแกรมว่ามีความถูกต้องตรงกับความต้องการของลูกค้าหรือไม่ และเมื่อระบบนี้ถูกนำไปใช้งานอาจต้องมีความต้องการที่เปลี่ยนแปลง ลูกค้าอาจต้องการที่จะเปลี่ยนรูปแบบการทำงานของโปรแกรม มีการเพิ่มหน้าที่ที่ต้องการเพิ่มจากระบบเดิม ดังนั้นจึงเป็นสิ่งที่จำเป็นที่ต้องเขียนโปรแกรมคำสั่งให้มีรูปแบบมีโครงสร้างและเอกสารเพื่อให้ง่ายสำหรับคนอื่นในการที่จะเข้าใจ

**ตัวอย่างที่** 6.1 ระเบียบวิธีการในการเขียนคำสั่งโปรแกรม อาจมีการกำหนดให้ทุก ๆ โมดูลที่พัฒนาต้องเริ่มต้นจากส่วนของการอธิบายหน้าที่การทำงานของโมดูลและการติดต่อสื่อสารกับโมดูลอื่น ๆ ดังมีรูปแบบดังนี้

```
**************************************************************************
*
* MODULE TO FIND INTERSECTION OF TWO LINES
*
* MODULE NAME : FINDPT
* PROGRAMMER : K. MERTZ
```

```
* VERSION : 1.0 (12 FEB 87)
*
* PROCEDURE INVOCATION :
*   CALL FINDPT (A1, B1, C1, A2, B2, C2, XS, YS, FLAG)
*
* INPUT PARAMETERS :
*    INPUT LINES ARE OF THE FORM
*         A1* X +B1* Y + C1 = 0 AND
*         A2* X +B2* Y + C2 = 0
*    SO INPUT IS COEFFICIENTS A1, B1, C1 AND
*    A2, B2, C2
* OUTPUT PARAMETERS :
*     IF LINES ARE PARALLEL, FLAG SET TO 1
*    ELSE FLAG = 0 AND POINT OF INTERSECTION
*     IS (XS, YS)
*
* PROCEDURE CALLED BY :
*     DRAW1
*     FINDNEXT
*
* PROCEDURE CALLS :
*     ERRMSG
*
```

ข้อสังเกตุเอกสารโปรแกรมนี้จะทำให้ทราบถึงการข้องเกี่ยวกับโมดูลอื่นๆ ว่ามีการถูกเรียกใช้จากโมดูลใดบ้าง หรือมีการเรียกโมดูลใดบ้างเพื่อทำงาน ซึ่งมีผลดีต่อโปรแกรมเมอร์ที่คอยดูแลบำรุงรักษาสามารถค้นหาโมดูลที่ต้องการและสามารถแก้ได้ง่าย และถ้าการออกแบบ

โปรแกรมมีคุณลักษณะที่ดี ทำให้การแก้ไขคำสั่งโปรแกรมนี้ไม่ส่งผลกระทบกับคำสั่งโปรแกรมในส่วนอื่นๆ ดังนั้นจริงๆแล้วมาตรฐานที่สำคัญของการเขียนคำสั่งโปรแกรมนั้นมีผลมาจากการออกแบบโมดูล รวมทั้งการออกแบบโปรแกรม ซึ่งต้องมีความสอดคล้องกัน เพื่อให้เราสามารถเขียนโปรแกรมคำสั่งที่มีคุณลักษณะที่ดี

## 6.2

# ข้อเสนอแนะในการเขียนโปรแกรม

การเขียนโปรแกรมจะต้องอาศัยความสร้างสรรเป็นอย่างมาก การออกแบบโปรแกรมนั้นเป็นเพียงแนวทางในการที่จะนำไปเขียนโปรแกรมเพื่อพัฒนาชุดคำสั่งในการแก้ปัญหา ซึ่งการเขียนโปรแกรมอย่างไรนั้นโปรแกรมเมอร์จะตัดสินใจเองการแก้ปัญหาชนิดเดียวกันโปรแกรมเมอร์แต่ละคนอาจเขียนจำนวนบรรทัดคำสั่งแตกต่างกันไปรวมทั้งอาจรเลือกใช้ภาษาโปรแกรมในการพัฒนาแตกต่างกัน และไม่ว่าจะเลือกใช้ภาษาโปรแกรมใดก็ตามการแก้ปัญหาโปรแกรมนั้นต้องประกอบด้วยองค์ประกอบหลัก 3 ส่วนคือ โครงสร้างควบคุมทางตรรกศาสตร์(logic control structures) , อัลกอริทึม (algorithm) และโครงสร้างข้อมูล (data structures) สำหรับในหัวข้อนี้จะแนะนำถึงการเขียนโปรแกรมที่ดี

### Control Structure

แนวการเขียนคำสั่งโปรแกรมนั้นต้องยึดหลักของการออกแบบโปรแกรมในบทที่ผ่านมา โดยต้องเขียนให้เป็นโครงสร้างที่ดีซึ่งเขียนคำสั่งโปรแกรมสอดคล้องกับการออกแบบโปรแกรมไปในแนวทางเดียวกัน โดยมีหลักการเขียนคำสั่งดังนี้

Using Fundamental Constructs ใช้โครงสร้างพื้นฐานในการแก้ปัญหาโปรแกรม โดยการเขียนคำสั่งโปรแกรมต้องใช้โครงสร้างควบคุมพื้นฐาน อันประกอบด้วย

Sequence :

BEGIN – END

Selection :

IF – THEN – ELSE

CASE

Repetition :

WHILE – DO

REPEAT – UNTIL

ในการออกแบบโปรแกรมนั้นเราพยายามที่จะหลีกเลี่ยงการใช้คำสั่งกระโดดข้ามแบบไม่มีเงื่อนไขซึ่งการเขียนคำสั่งโปรแกรมสามารถใช้โครงสร้างควบคุมเหล่านี้ทดแทนได้ ซึ่งการใช้กลไกการควบคุมเหล่านี้ทำให้โปรแกรมนั้นง่ายต่อการอ่านและติดตาม และยังช่วยในการจัดการความคิดในขณะเขียนโปรแกรมอีกด้วย นอกจากนี้โปรแกรมเมอร์สามารถพัฒนาฟังก์ชันหรือมาโคร เพื่อแบ่งให้โปรแกรมเป็นโครงสร้างมากขึ้น เช่น ในภาษาแอสแซมบลี พัฒนามาโคร DO เพื่อทำให้โปรแกรมนั้นง่ายต่อการอ่านหรือทำความเข้าใจ ในภาษาโปรแกรมส่วนมากมีส่วนให้ผู้พัฒนาสามารถสร้างโมดูลย่อยๆได้ในหลายๆลักษณะ เพื่อให้นำมาประกอบและใช้ได้ในโปรแกรมได้ตามวัตถุประสงค์

ในปัจจุบันบางองค์กรมีข้อเสนอแนะหรือมาตรฐาน โดยจำลองคำสั่งควบคุมในภาษาต่างๆไว้เป็นมาตรฐาน เพื่อเพิ่มประสิทธิภาพในการพัฒนาโปรแกรมและเป็นวิธีการที่ส่งผลให้ง่ายต่อการติดตามและการพัฒนาโปรแกรมให้เป็นโครงสร้าง

Top-Down Flow การเขียนคำสั่งโปรแกรมที่ดีนั้นควรเขียนขั้นตอนการทำงานจากบนลงล่าง การทำงานเป็นโครงสร้างและไม่ควรมีคำสั่งกระโดดข้ามไปมาในโปรแกรม การทำงานต้องกระทำภายในโครงสร้างที่มีการควบคุมอยู่(Control Flow) ซึ่งการจัดเรียงคำสั่งต่างๆในการแก้ปัญหาเป็นส่วนที่สำคัญ ตัวอย่างต่อไปนี้เป็นโครงสร้างการควบคุมในภาษา PL/I

```
BENEFIT = MINIMUM;
IF (AGE < 75) THEN GO TO A;
GO TO C;
 IF (AGE < 65) THEN GO TO B;
 IF (AGE < 55) THEN GO TO C;
```

```
A: IF (AGE < 65) THEN GO TO B;
   BENEFIT = BENEFIT * 1.5 + BONUS;
   GO TO C;
B: IF (AGE < 55) THEN GO TO C;
    BENEFIT = BENEFIT * 1.5;
C: Next statement
```

การเขียนคำสั่ง PI/1 ข้างต้นนี้ยากต่อการติดตามมีการกระโดดไปมาในคำสั่งมีผลให้ทำความเข้าใจได้ยาก เราสามารถปรับเปลี่ยนรูปแบบโดยจัดเรียงลำดับคำสั่งใหม่ เป็นการทำงานแบบบนลงล่างได้ ดังนี้

```
IF (AGE < 55) THEN BENEFIT = MINIMUM;
ELSE IF (AGE < 65) THEN BENEFIT = MINIMUM + BONUS;
ELSE IF (AGE < 75) THEN BENEFIT = MINIMUM  * 1.5 + BONUS;
ELSE BENEFIT = MAXIMUM;
```

ซึ่งเห็นได้ว่าคำสั่งโปรแกรมหลังจากปรับปรุงแล้ว ทำให้การอ่านหรือติดตามการทำงานของโปรแกรมได้ง่ายและสามารถทำความเข้าใจได้อย่างชัดเจน

**Use Of Submodules** คุณลักษณะที่ดีของการออกแบบโปรแกรมประการหนึ่งคือ modularity เป็นการแบ่งระบบออกเป็นโมดูลย่อยๆซึ่งยังประโยชน์ต่อการเขียนคำสั่งโปรแกรมต่อมาด้วย กล่าวคือการเขียนคำสั่งโปรแกรมนี้เมื่อเขียนไว้แล้วสามารถนำโมดูลต่างๆเหล่านี้มาสนับสนุนหรือนำมาใช้งานได้อีกในภายหลัง นอกจากนี้การออกแบบเป็นลำดับขั้นนั้นการสร้างโปรแกรมจากการออกแบบที่แบ่งไว้เป็นส่วนๆเหล่านี้ยังสามารถซ่อนรายละเอียดการพัฒนาในระดับล่างไว้ ทำให้สามารถเข้าใจระบบได้อย่างง่ายดาย ง่ายต่อการทดสอบและง่ายต่อการบำรุงรักษา โดยทั่วไปเราจะแทนโมดูลด้วยมาโคร หรือฟังก์ชัน หรือโพรซิเจอร์ หรือรูทีนย่อย ซึ่งโมดูลต่างๆเหล่านี้ถูกซ่อนรายละเอียดไว้จากการมองเห็น โดยจะทราบแต่วัตถุประสงค์ของการทำงานของโมดูลเพียงอย่างเดียวขนาดเล็กไปจนถึงโมดูลทั่ว ๆ ไป สามารถใช้ซ้ำกันได้ในหลาย ๆ โปรแกรม แต่ละโมดูลของโปรแกรมควรจะทำงานเพียงอย่างเดียว

การเขียนคำสั่งโปรแกรมนั้นต้องกระทำไปในทิศทางเดียวกับการออกแบบ เพราะจะทำให้ติดตามได้ง่ายและตรงกับความต้องการ กรณีที่มีการเรียกใช้โมดูลต่างๆ ควรมีคำอธิบายถึงชื่อพารามิเตอร์และอธิบายถึงคำสั่งโปรแกรมที่มีความเกี่ยวข้องกันระหว่างโมดูลเพื่อหลีกเลี่ยงจากความสับสนที่อาจเกิดขึ้นได้ เช่น

Reestimate TAX

เป็นคำอธิบายถึงการคิดภาษีแต่มิได้อธิบายถึงที่มาหรือที่ไปว่าเป็นมาอย่างไร

Reestimate TAX based on values of GROSS_INC and DEDUCTS

เป็นคำอธิบายที่ดีกว่าเดิมทำให้ทราบว่าการคิดภาษีนั้นต้องได้รับค่ามาจากโมดูลอื่นจากตัวแปร GROSS_INC และ DEDUCTS

## Algorithms

การออกแบบโปรแกรมที่ผ่านมานั้นเราจะพิจารณาถึงอัลกอริทึมที่เหมาะสมในการแก้ปัญหาอาจมีการนำอัลกอริทึมพิเศษมาใช้ในการเขียนโมดูล ตัวอย่างเช่นการออกแบบแนะนำให้ใช้ควิกซอร์ทในการแก้ปัญหาหรือกำหนดขั้นตอนทางตรรกศาสตร์ของอัลกอริทึมควิกซอร์ท หน้าที่ของโปรแกรมเมอร์คือเขียนคำสั่งโปรแกรมโดยแปลงจากอัลกอริทึมไปเป็นคำสั่งโปรแกรมซึ่งต้องโปรแกรมเมอร์ต้องทราบถึงรูปแบบ ไวยากรณ์ ของโปรแกรมภาษาและสามารถนำไปใช้ได้กับฮาร์ดแวร์ที่ได้ระบุในความต้องการได้

Efficiency บางครั้งโปรแกรมเมอร์พยายามจะเขียนคำสั่งโปรแกรมให้สามารถทำงานได้อย่างรวดเร็วเท่าที่จะเป็นไปได้ เพื่อลดค่าใช้จ่ายในการประมวลผลโปรแกรม ซึ่งจริงๆแล้วค่าใช้จ่ายนั้นยังขึ้นอยู่กับปัจจัยอีกหลายๆอย่างเช่น

1. ค่าใช้จ่ายของเวลาที่ใช้ในการเขียนคำสั่งโปรแกรม
2. ค่าใช้จ่ายของเวลาที่ใช้ในการทดสอบคำสั่งโปรแกรม
3. ค่าใช้จ่ายของเวลาสำหรับผู้ใช้ในการเข้าใจคำสั่งโปรแกรม
4. ค่าใช้จ่ายของเวลาในการปรับปรุงคำสั่งโปรแกรมถ้ามีความจำเป็น

เวลาในการประมวลผล(execution time)เป็นเพียงค่าใช้จ่ายส่วนหนึ่ง ดังนั้นโปรแกรมเมอร์ต้องช่างน้ำหนักถึงเวลาในการประมวลผลกับคุณภาพของการออกแบบ,มาตรฐาน

และ ความต้องการของลูกค้าด้วยในการพัฒนาคำสั่งโปรแกรม เพราะสิ่งที่สำคัญที่สุดคือโปรแกรมต้องทำงานถูกต้องก่อน ก่อนที่จะปรับปรุงประสิทธิภาพให้มีความรวดเร็วขึ้น

Data Structures

การเขียนคำสั่งโปรแกรมนั้นมีความเกี่ยวข้องและมีความสัมพันธ์กับข้อมูลเป็นอย่างมาก ไม่ว่าจะเป็นการเก็บ เรียกใช้ ปฏิบัติการต่างๆกับข้อมูล ดังนั้นการออกแบบรูปแบบของข้อมูลที่ดีจึงเป็นสิ่งสำคัญในการเขียนโปรแกรมคำสั่ง เพราะถ้ามีการจัดการโครงสร้างของข้อมูลต่างๆให้อยู่ในรูปแบบที่เหมาะสมจะช่วยให้การจัดการข้อมูลและการใช้งานได้ง่ายขึ้น

Keeping the Program Simple ในการออกแบบโปรแกรมนั้น การจัดการข้อมูลภายในโมดูลนั้นอาจมีผลต่อทางเลือกของโครงสร้างของข้อมูลในการจัดเก็บ สมมุติต้องการเขียนโปรแกรมในการคิดภาษีรายได้ โดยข้อมูลเข้าคือรายได้ โดยมีอัตราในการคิดดังนี้

1. For the first $10000 of income, the tax is 10%
2. For the first $10000 of income above $10000, the tax is 12%
3. For the first $10000 of income above $20000, the tax is 15%
4. For the first $10000 of income above $30000, the tax is 18%
5. For any income above $40000, the tax is 20%

ดังนั้นผู้มีรายได้ $35,000 จะนำรายได้ $10,000 แรกคิดภาษี 10 % รายได้ $10,000 ถัดไปคิดภาษี 12% รายได้ $10,000 ถัดไปคิดภาษี 15% ส่วนรายได้ที่เหลืออีก $5,000 คิดภาษี 18% ดังนั้นเสียภาษีเท่ากับ $1,000 + $1,200 + $1,500 + $900 ผลรวมคือ $4,600 ซึ่งการคำนวณภาษีนี้สามารถเขียนเป็นอัลกอริทึมได้ดังนี้

```
TAX = 0;
IF TAXABLE_INCOME = 0 ,GOTO EXIT
IF TAXABLE_INCOME > 10,000 , TAX = TAX +1000.0
   ELSE TAX = TAX + 0.10 * TAXABLE_INCOME
   GO TO EXIT
IF TAXABLE_INCOME > 20,000 , TAX = TAX +1200.0
```

```
    ELSE TAX = TAX + 0.12 * (TAXABLE_INCOME – 10000.0)
    GO TO EXIT
IF TAXABLE_INCOME > 30,000 , TAX = TAX +1500.0
    ELSE TAX = TAX + 0.15 * (TAXABLE_INCOME – 20000.0)
    GO TO EXIT
IF TAXABLE_INCOME<40,000,TAX=TAX+1.8*(TAXABLE_INCOME– 30000.0)
    GO TO EXIT
ELSE TAX = TAX + 1800.0 + 0.20 * (TAXABLE_INCOME – 40000.0)
        EXIT: END
```

จากขั้นตอนการประมวลผลข้างต้น เราสามารถกำหนดเงื่อนไขของการคำนวณในรูปแบบของตารางได้ดังนี้

Sample Tax Table

| Bracket | Base | Percent |
|---|---|---|
| 0 | 0.0 | 0.10 |
| 10000 | 1000.0 | 0.12 |
| 20000 | 2200.0 | 0.15 |
| 30000 | 3700.0 | 0.18 |
| 40000 | 5500.0 | 0.20 |

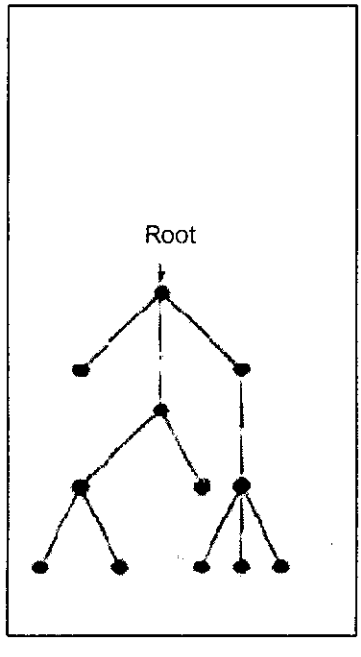
จากการนำเงื่อนไขมาจัดให้อยู่ในรูปแบบของตาราง ทำให้เราสามารถปรับเปลี่ยนอัลกอริทึมของโปรแกรมให้อยู่ในรูปแบบที่ง่ายขึ้นดังนี้

```
I = 1
LEVEL = 1
FOR I = 2 TO 5 DO
        IF TAXABLE_INCOME > BRACKET(I)
        THEN  LEVEL = LEVEL + 1
TAX = BASE(LEVEL) + PERCENT(LEVEL) * (TAXABLE_INCOME – BRACKET(LEVEL))
```

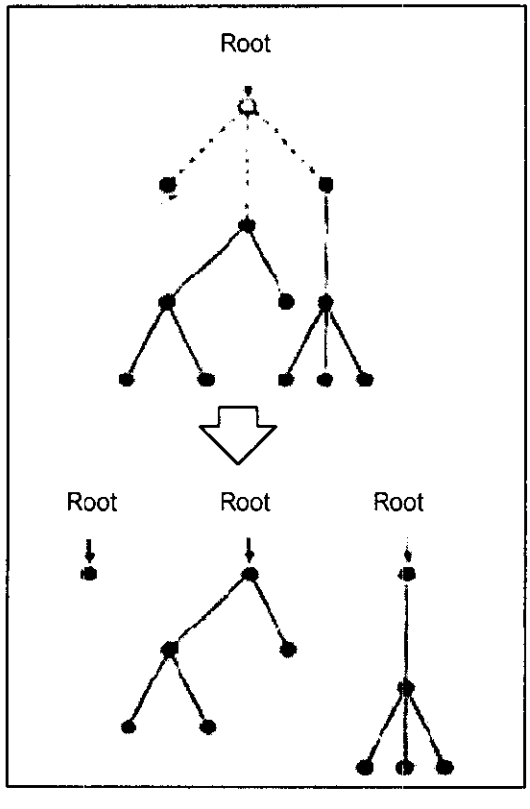
การคำนวณหาค่าของภาษีจะถูกปรับเปลี่ยนให้อยู่ในรูปแบบที่ง่ายขึ้น เพียงแต่เปลี่ยนวิธีในการกำหนดโครงสร้างข้อมูลเสียใหม่เป็นโครงสร้างข้อมูลชนิดแถว(array) ทำให้การเขียนคำสั่งโปรแกรมนั้นง่ายขึ้นสั้นขึ้น สามารถเข้าใจได้ง่ายขึ้น ง่ายต่อการทดสอบและง่ายต่อการแก้ไข

Using Structure of Data to Determine Structure of Program จากตัวอย่างข้างต้นที่ผ่านมาจะเห็นได้การกำหนดโครงสร้างที่เหมาะสมนั้นจะช่วยในการจัดการและการควบคุมภายในโปรแกรม นอกจากนี้โครงสร้างข้อมูลก็ยังมีอิทธิพลต่อการเลือกภาษาโปรแกรมที่ใช้ในการเขียนคำสั่งโปรแกรมอีกด้วย กล่าวคือถ้าในการแก้ปัญหาโปรแกรมนั้นมีการกระทำซ้ำๆกันในลักษณะของการเรียกตัวเอง อทิเช่นโครงสร้างต้นไม้ สมมุติว่าการปฏิบัติงานในโปรแกรมนั้นเป็นโครงสร้างต้นไม้ที่มีตัวแปรพอยเตอร์ root ชี้ที่โหนดรากต้นไม้นี้มีต้นไม้ย่อยโดยมีเส้นที่เชื่อมต่อระหว่างโหนดที่เป็นราก(root)กับโหนดของต้นไม้ย่อยต่างๆ ถ้ามีการลบราก(root) ออกไปจะส่งผลให้เกิดเป็นกลุ่มของต้นไม้ใหม่ ที่แต่ละกิ่งมีราก(root)ใหม่เป็นของตนเอง ดังภาพที่ 6.2 แสดงถึงรากต้นไม้ ส่วนภาพที่ 6.3 แสดงต้นไม้ย่อยที่เกิดจากรากต้นไม้ในภาพที่ 6.2 ซึ่งมีการลบรากต้นไม้ออก กลายเป็นต้นไม้ใหม่จำนวน 3 ต้น

รูปภาพที่ 6.2 A Rooted Tree




รูปภาพที่ 6.3 Sub-Trees of Rooted Tree



มีภาษาโปรแกรมหลายภาษาที่อนุญาติให้โปรแกรมเมอร์สามารถเขียนคำสั่งโปรแกรมในลักษณะเรียกตัวเองได้ เพื่อทดแทนการเขียนโปรแกรมแบบวนรอบ(loop) เช่นภาษาซี, ภาษาปาสคาล ซึ่งโปรแกรมเมอร์สามารถเลือกภาษาโปรแกรมต่างๆเหล่านี้มาใช้ในการแก้ปัญหา ทำให้การเขียนโปรแกรมสั้นขึ้น ง่ายขึ้น และมีผลให้ทำความเข้าใจการทำงานของโปรแกรมได้ง่ายอีกด้วย

Localizing Input and Output in Separate Modules สำหรับส่วนของโปรแกรมที่ทำหน้าที่อ่านข้อมูลหรือสร้างผลลัพธ์หรือรายงานพิเศษต่างๆนั้นโดยทั่วไปจะขึ้นอยู่กับฮาร์ดแวร์หรือซอฟต์แวร์ที่มีการกำหนดไว้ เมื่อมีการเปลี่ยนฮาร์ดแวร์หรือซอฟต์แวร์ในภายหลังทำให้ยุ่งยากต่อการทดสอบ การพัฒนาคำสั่งโปรแกรมที่ดีนั้นควรกำหนดส่วนของโมดูลต่างๆเหล่านี้แยกออกจากส่วนของคำสั่งในการปฏิบัติการ ซึ่งทำให้เราสามารถเขียนโมดูลในการนำข้อมูลเข้าได้ในหลายวิธีการที่แตกต่างกันได้ ทำให้การทดสอบและบำรุงรักษาทำได้โดยง่ายและรวดเร็ว

## General Guidelines

กลยุทธ์ในการออกแบบเพื่อทำให้โปรแกรมเมอร์สามารถเขียนคำสั่งโปรแกรมให้มีคุณภาพสูงจากการออกแบบโปรแกรมมีด้วยกันหลายวิธี ในที่นี้จะขอกล่าว 2 วิธีการที่ช่วยอนุรักษ์คุณลักษณะที่ดีดังกล่าวต่อไปนี้

**การใช้คำสั่งเทียม** การออกแบบโปรแกรมโดยใช้คำสั่งเทียม(Pseudocode) เป็นเครื่องมือที่ให้ความคล่องตัวมากกว่าเครื่องมือตัวอื่น เพราะโปรแกรมเมอร์สามารถเลือกภาษาโปรแกรมได้อิสระ การเปลี่ยนจากการออกแบบโปรแกรมเป็นคำสั่งโปรแกรมก็สามารถกระทำได้โดยทันทีทันใดรวมทั้งการแทนค่าข้อมูลต่างๆ ซึ่งคำสั่งเทียมนี้สามารถใช้ดัดแปลงหรือปรับปรุงการออกแบบซึ่งไม่ส่งผลกระทบต่อคำสั่งโปรแกรม โปรแกรมเมอร์สามารถทดลองสร้างคำสั่งโปรแกรมที่ดีที่สุดได้โดยสามารถเปลี่ยนแปลงคำสั่งโปรแกรม ปรับเปลี่ยนโครงสร้างโดยเขียนหลายๆวิธีเพื่อหาวิธีการเขียนคำสั่งโปรแกรมที่สั้นที่สุดและดีที่สุด สำหรับตัวอย่างสมมติการออกแบบโปรแกรมสำหรับโมดูลในขั้นตอนของระบบประมวลผลตัวอักษร ดังนี้

```
MODULE  PARSE_LINE :
        Read next eighty characters.
        IF  this is a continuation of the previous line,
```

```
        Call CONTINUE
    ELSE determine command type.
    ENDIF
CASE of COMMAND_TYPE
        COMMAND_TYPE is paragrahp : Call PARAGRAPH.
        COMMAND_TYPE is indent : Call INDENT.
        COMMAND_TYPE is skip line : Call SKIP_LINE.
        COMMAND_TYPE is margin : Call MARGIN.
        COMMAND_TYPE is new page : Call PAGE.
        COMMAND_TYPE is double space : Call DOUBLE_SPACE.
        COMMAND_TYPE is single space : Call SINGLE_SPACE.
        COMMAND_TYPE is break : Call BREAK.
        COMMAND_TYPE is anything else : Call ERROR.
ENDCASE
```

การแปลงเป็นคำสั่งโปรแกรมนั้นมีอยู่ด้วยกันหลายวิธีการ ในขั้นแรกของการแปลงการออกแบบไปเป็นคำสั่งโปรแกรมนั้นต้องตรวจสอบชนิดของคำสั่งจากคำสั่งเทียมที่เกี่ยวพันธ์กัน โดยนำมาเขียนเป็นคำสั่งต่างๆดังนี้

PARAGRAPH:

Break line, flush line buffer. Advance one line between paragraphs. If fewer than 2 lines left on page, eject. Set line pointer to paragraph indent.

INDENT:

Break line, flush line buffer. Get line parameter. Set line pointer to indent parameter, Set left margin to indent.

SKIP_LINE:

Break line, flush line buffer. Get line parameter. Advance (parameter) lines or eject if not enough space left on current page.

MAGIN:

Break line, flush line buffer. Get margin parameter. Set line pointer to left margin.

Set right margin to margin.

PAGE:

Break line, flush line buffer. Eject page. Set line pointer to left margin.

DOUBLE_SPACE:

Set interline space to 2.

SINGLE_SPACE:

Set interline space to 1.

BREAK:

Break line, flush line buffer. Set line pointer to left margin.

จากคำสั่งเทียมข้างต้นเราสามารถนำมาปรับเปลี่ยนโดยจัดกลุ่มของขั้นตอนต่างๆที่เกี่ยวพันธ์กัน ดังนี้

FIRST:

PARAGRAPH, INDENT, SKIP_LINE, MARGIN, BREAK, PAGE:

Break line, flush line buffer

DOUBLE_SPACE, SINGLE_SPACE:

No break line, no flush line buffer

SECOND:

INDENT, SKIP_LINE, MARGIN:

Get parameter

PARAGRAPH, BREAK, PAGE, DOUBLE_SPACE, SINGLE_SPACE:

No parameter needed

THIRD:

PARAGRAPH, INDENT, SKIP_LINE, MARGIN, BREAK, PAGE:

Set new line pointer

DOUBLE_SPACE, SINGLE_SPAGE:

New line pointer unchanged

FOURTH:

Individual actions taken

เมื่อพิจารณาถึงคำสั่งในกลุ่มของ FIRST และ THIRD มีกิจกรรมที่กระทำกับกลุ่มของคำสั่งที่เหมือนกัน เราสามารถปรับเปลี่ยนคำสั่งเทียมให้สละสลวยขึ้นดังนี้

INITIAL:

Get parameter for indent, line skip, margin.

Set left margin to parameter for indent.

Set temporary line pointer to left margin for all but paragraph; for paragraph ,set it to paragraph indent.

LINE_BREAKS:

If not (DOUBLE_SPACE or SINGLE_SPACE), break line and flush line buffer and set line pointer to temporary line pointer.

If 0 lines left on page, eject page and print page header.

INVALIDUAL_CASES:

INDENT, BREAK: do nothing.

SKIP-LINE: skip parameter lines or eject.

PARAGRAPH: advance 1 line; if < 2 lines on page, eject.

MARGIN: right_margin = parameter.

DOUBLE_SPACE: interline_space = 2.

SINGLE_SPACE: interline_space = 1.

PAGE: eject page, print page header.

สมมุติว่าโปรแกรมเมอร์เลือกภาษาปาสคาลในการพัฒนา จากคำสั่งเทียมข้างต้นสามารถเปลี่ยนเป็นคำสั่งได้ดังนี้

```
{initial : get parameters}
if (command_type in [indent, line_skip, margin])
   then parm_value := get_parm(input_line);
if (command_type = indent)
   then left_margin := parm_value;
if (command_type = paragraph)
   then temp_line_pointer := paragraph_indent;
   else temp_line_pointer := left_margin
{break current line, begin new line}
if (not ( (command_type=db1_spc) or (command_type=sngl_spc) ) )
    then begin
        call break_and flush_line;
        if (lines_left = 0)
           then call begin_new_page
        line_pointer := temp_line_pointer;
     end;
{actions for individual commands}
case (command_type) of
      line_skip: if (lines_left > parm_value) then
          for I:=1 to parm_value do
              call advance_line
          else call begin_new_page;
       paragraph: begin
            call advance_line;
       if (lines_left < 2) then
            call begin_new_page
  end;
```

```
margin: right_margin := parm_value
dbl_spc: interline_space:= 2;
sngl_spc: interline_space:= 1;
page: call begin_new_page
end; {case of command_type}
```

จะเห็นได้ว่าคำสั่งเทียมเป็นเครื่องมือหนึ่งที่โปรแกรมเมอร์สามารถสร้างคำสั่งโปรแกรมที่มีคุณภาพที่ดีได้ ทั้งนี้ทั้งนั้นเมื่อมีการปรับเปลี่ยนโครงสร้างใดต้องมีเอกสารที่ใช้สำหรับอ้างอิงได้ในภายหลัง สำหรับการบำรุงรักษาในอนาคตด้วย

# 6.3

# Documentation

เอกสารโปรแกรมเป็นเอกสารที่อธิบายรายละเอียดที่เกี่ยวข้องกันของโมดูลโปรแกรมทำให้ผู้อ่านทราบว่าโปรแกรมโมดูลต่างๆทำงานอะไรและปฏิบัติการอย่างไร เอกสารของการพัฒนาระบบในขั้นตอนนี้แบ่งเป็น 2 ชนิดคือ Internal Document เป็นคำอธิบายโดยเขียนอยู่ภายในโมดูลโปรแกรม และ External Document ซึ่งเขียนอธิบายภายนอกโมดูลโปรแกรม

## Internal Document

เป็นเอกสารซึ่งเขียนภายในโมดูลโปรแกรม เป็นรายละเอียดเพื่ออธิบายซอสต์โค๊ดของโมดูลโปรแกรม อทิเช่น การนิยามโมดูลโปรแกรม , คำอธิบายโครงสร้างของข้อมูล, อัลกอริทึม และการควบคุมการไหล(Control Flow) โดยทั่วไปเป็นสารสนเทศแบบสรุปที่รวบรวมไว้เป็นหมายเหตุ(comments) ซึ่งปรากฏที่จุดเริ่มต้นของโปรแกรม เรียกว่า header comment block

Header Comment Block เป็นเอกสารที่ทำให้ผู้อ่านสามารถทราบได้ว่าใคร ? ทำอะไร ? , ที่ไหน ? , เมื่อไร ? , ทำไม ? และอย่างไร ? รายละเอียดในส่วนนี้จะประกอบด้วย

1) What โปรแกรมอะไร

เป็นการกำหนดชื่อของโปรแกรมเพื่อประโยชน์ในการอ้างอิง เมื่อถูกเรียกใช้จากจุดอื่นในระบบ ซึ่งชื่อที่กำหนดต้องง่ายและสื่อความหมายที่ดีด้วย

2) Who ผู้พัฒนาโปรแกรมคือใคร

ระบุชื่อของผู้เขียนโปรแกรม ทำให้ทราบว่ามีใครบ้างที่เป็นผู้รับผิดชอบโปรแกรมในส่วนของการทดสอบและการบำรุงรักษามีสมาชิกในทีมงานใครบ้างเพื่อสามารถตอบคำถามหรืออธิบายในกรณีที่มีปัญหาเกิดขึ้นได้

3) Where โปรแกรมอยู่ในส่วนไหนของการออกแบบระบบ

ในระบบหนึ่งๆประกอบด้วยหลายๆโมดูล ดังนั้นจึงเป็นสิ่งที่จำเป็นที่จะต้องมีเอกสารที่ชี้ให้เห็นว่าโมดูลต่างๆนั้นอยู่ในส่วนใดของระบบ การอธิบายสารสนเทศนี้สามารถกระทำได้หลายกรณี อาจแสดงเป็นไดอะแกรมความสัมพันธ์ของโมดูล ที่มีความสัมพันธ์เป็นลำดับชั้น หรือเขียนเป็นคำอธิบายซึ่งแสดงถึงการเรียกใช้โมดูลและการถูกเรียกใช้จากโมดูลอื่น ดังตัวอย่างที่ 6.1

4) When โปรแกรมถูกเขียนและถูกแก้ไขเมื่อใดบ้าง

ในขณะที่ใช้ระบบอยู่นั้นย่อมมีการแก้ไขหรือปรับเปลี่ยนโมดูลโปรแกรมอันเนื่องมาจากความผิดพลาดที่เกิดขึ้นในการทำงาน หรือความต้องการของลูกค้าเปลี่ยนแปลง ดังนั้นการเปลี่ยนแปลงต่างๆที่เกิดขึ้นต้องมีการเก็บบันทึกไว้ เพื่อให้ทราบถึงสิ่งที่ได้กระทำและติดตามได้ เอกสารในส่วนนี้จะบันทึกถึงวันที่และรายละเอียดต่างๆที่มีการปรับเปลี่ยนในโมดูลโปรแกรม

5) Why ทำไมจึงใช้โปรแกรมนี้

เป็นการอธิบายวัตถุประสงค์โดยทั่วไปของโมดูลโปรแกรมนี้ เพื่อบอกให้ทราบว่าโปรแกรมนี้ทำอะไร

6) How โปรแกรมของคุณใช้โครงสร้างข้อมูลอย่างไร, มีขั้นตอนการทำงานอย่างไร และควบคุมอย่างไร

โดยปกติคำอธิบายที่บรรจุในส่วนนี้ได้แก่

1.ชื่อ, ชนิด และวัตถุประสงค์ ของแต่ละโครงสร้างข้อมูลและตัวแปร

2.อธิบายถึงตรรกของโมดูล, รวมถึงอัลกอริทึมหลัก ๆ

3.อธิบายรายละเอียดของการแก้ไขความผิดพลาด

4.ข้อมูลเข้าที่คาดหวัง และผลลัพธ์ที่เป็นไปได้

5.อธิบายเครื่องมือที่ช่วยในการทดสอบ, วิธีการใช้

6.รายละเอียดของความคาดหวังที่สูงขึ้น

เอกสารในส่วนของ header comment block นั้นขึ้นอยู่กับมาตรฐานในการพัฒนาของแต่ละหน่วยงาน สำหรับสารสนเทศที่สำคัญที่สมควรต้องมีนั้นอาจประกอบดังตัวอย่างต่อไปนี้

PROGRAM SCAN – Program to scan a line of text for a given character

PROGRAMMER: Beatrice Clarman (718)345-6789

CALLING SEQUENCE: CALL SCAN(length, char)

Where 'LENGTH' is the length of the line to be scanned, "CHAR" is the character to be sought, line of text passed as array 'NTEXT'

VERSION 1: WRITTEN 1-12-86

REVISION 1.1:2-3-86 improve searching algorithm.

PURPOSE: General- purpose scanning module to be used for each new line of text, no matter the length.

DATA STRUCTURES : Variable LENGTH – INTEGER

Variable CHAR – CHARACTER

ARRAY NTEXT – CHARACTER array of length 'LENGTH'

ALOGRITHM : Reads array NTEXT one character at a time ; if CHAR is found, position in NTEXT returned in variable 'LENGTH' ; else variable 'LENGTH' set to 0

**Other Program Comments** คำอธิบายการทำงานของคำสั่งในโปรแกรมเป็นสิ่งที่จะทำให้เราเข้าใจโครงสร้างการทำงานของโปรแกรมโดยชัดเจน อทิเช่น แยกโปรแกรมเป็นระยะๆ เพื่อแสดงถึงกิจกรรมหลักๆ และภายในแต่ละกิจกรรมจะมีการอธิบายออกเป็นขั้นตอนต่างๆ ตาม

หน้าที่ต่างๆที่เกี่ยวข้อง และเมื่อคำสั่งถูกแก้ไขเราต้องปรับเปลี่ยนคำอธิบายให้เปลี่ยนตามไปด้วย สำหรับตัวอย่างนี้ ถือว่าเป็นคำอธิบายที่ไม่ใช้กัน

```
I3 = I3 +1 ;     INCREMENT I3
```

การเขียนคำอธิบายที่ดีต้องสามารถบอกถึงหน้าที่ได้ ดังเช่น

```
I3 = I3 +1 ;     Set counter to read next case
```

ถ้าจะให้เข้าใจได้ดีขึ้น ควรกำหนดชื่อตัวแปร(parameter) เพื่อช่วยอธิบายถึงกิจกรรมที่กำลังทำได้ จะทำให้เข้าใจได้ชัดเจนกว่า

```
case_counter = case_counter +1 ;
```

**Meaningful variable names and statement labels** เป็นการเลือกชื่อตัวแปร และคำสั่ง ให้สื่อความหมายง่ายต่อการอ่านและเข้าใจ เช่น

```
weekwage = (Hrrate * Hour) + (1.5) * (Hrrate) * (Hours -40)
```

ซึ่งการเลือกชื่อที่สื่อความหมายนี้ไม่จำเป็นต้องมีคำอธิบาย และเมื่อมีข้อผิดพลาดเกิดขึ้น เราสามารถทราบได้ว่าตัวแปรใดผิด ความหมายหรือค่าที่เก็บในตัวแปรเหล่านั้นหมายถึงอะไร แต่ถ้าคำสั่งเดียวกันแต่เขียนอีกลักษณะหนึ่งดังนี้

```
Z = (A + B) + (0.5) * (A) * (B - 40)
```

เป็นประโยคที่กำหนดตัวแปรเป็นตัวอักษรที่ไม่สื่อความหมาย ไม่สามารถบอกได้ว่าทำหน้าที่อะไร

**Formatting to Enhance Understanding** รูปแบบของการเขียนโปรแกรมเป็นส่วนหนึ่งที่ทำให้ผู้อ่านเข้าใจได้ว่าคำสั่งที่กำลังสนใจติดตามอยู่นั้นทำอะไรและกระทำอย่างไรการจัดย่อหน้าและเว้นวรรคประโยคคำสั่งที่เหมาะสม สามารถตอบสนองกับโครงสร้างควบคุมพื้นฐานได้ ดังตัวอย่าง

```
if xcoord < ycoord
then result = -1;
else if xcoord = ycoord
```

```
then if slope1 > slope2
then result = 0;
else result = 1;
else if slope1 > slope2
then result = 2;
else if slope1 < slope2
then result = 3;
else result = 4;
```

เราสามารถปรับเปลี่ยนการย่อหน้าและจัดช่องว่างใหม่ ทำให้มีความชัดเจนขึ้นดังนี้

```
if        xcoord < ycoord    then result = -1;
else      if xcoord = ycoord    then
          if slope1 > slope2    then result = 0;
                                else result = 1;
else      if slope1 > slope2    then result = 2;
else      if slope1< slope2     then result = 3;
else                            result = 4;
```

ในการใช้รูปแบบเพื่อแสดงโครงสร้างการควบคุมนั้น Weinberg([WEI71]) ได้แนะนำรูปแบบประโยคในการอธิบายโดยเขียนที่ด้านหนึ่งของคำสั่งนั้นๆ ตามตัวอย่างต่อไปนี้เป็นคำสั่งภาษาปาสคาล สามารถอ่านคำอธิบายโดยอ่านจากภายในวงเล็บปีกกา

```
procedure skipch;              {updates pointers and skips next single}
var
  k: integer;                  {local pointer}
begin
  for k:= iptr+1 to incount do
```

```
            ch[k-1] := ch[k];          {move each char back one position}
  ch[incount] := " ";
            incount := incount – 1; {shorten string length by 1}
            iptr := iptr – 1;          {move input pointer back; sill mv ahd}
  end; {dkipch}


begin
  incount := endpt;                    {upper bound on chars to edit}
  iptr := 1;                           {begin with start of ch array}
  cr_last := false;
while (iptr < incount) do
  begin
            if (ord([iptr]) < ord (" ")) then
            begin                      {is a control character}
             if (ch[iptr] = chr(cr))
               then                    { character was <CR>}
                     if (cr_last) then skipch   { repeated <cr>s; keep only one}
                       else cr_last := true     { first <cr>;keep it}
                   else                         {char was not <CR>}
                   if (ch{iptr] = char(if))
                     then
                       if (cr_last)
                         then  cr_last := false
                           else   skipch        {skip repeated <LF>s}
                     else                       {neithe <CR> nor <LF>}
                     skipch                     {skip ctrl char. }
                   end                          { if (ch < " " ) }
```

```
                else                          {ch  was printable }
                  ce_last := false ;
                iptr := iptr + 1 ;            { go to next char }
            end;                              {while iptr , incount }
        ch[0] := chr(incount) ;
        endpt := incount ;
    end;
```

**Documenting Data** เป็นการยากที่ผู้อ่านจะเข้าใจถึงโครงสร้างที่ใช้ เนื่องจากการปฏิบัติการในระบบต้องเกี่ยวข้องกับแฟ้มข้อมูลหลายๆแฟ้ม ซึ่งมีการส่งผ่านพารามิเตอร์ระหว่างกัน ดังนั้นจึงมีความจำเป็นอย่างยิ่งที่ต้องมีรายละเอียดของข้อมูลต่างๆ เพื่อช่วยให้ผู้อ่านเข้าใจถึงโครงสร้างข้อมูลที่ใช้ต่างๆ ดังเช่น

```
{ DATA  STRUCTURES:                   }
{    DAMLOC  contains                 }
{      dam locations and height       }
{    DEFAULT VALUES: read from tape   }
{    WATERQUAL to store               }
{      water quality data from sensors}
{    DEFAULT VALUES: initialized to 0 }
```

ในส่วนของตัวโปรแกรม แสดงถึงการใช้งานโครงสร้างข้อมูลนี้ เช่นใช้เมื่อไรและอย่างไรที่ค่าของข้อมูลเปลี่ยนแปลง

```
var current_dam:damloc;
.
.
.
with current_dam do
read (damnum, damlat, damlon,damhgt); (reset dam record values)
```

External Documentation

สำหรับ internal documentation นั้นเป็นเอกสารที่อยู่ภายในโปรแกรมซึ่งมีเจตนาอธิบายสารสนเทศต่างๆแก่โปรแกรมเมอร์เพื่อให้เข้าใจการทำงานของโปรแกรม แต่สำหรับ external documentation นั้นสำหรับบุคลากรอื่นๆที่ไม่มีหน้าที่ที่เกี่ยวข้องกับการเขียนคำสั่ง อทิเช่น นักออกแบบโปรแกรม เอกสารนี้มีประโยชน์ในการปรับปรุงหรือขยายขีดความสามารถของระบบ เนื่องจากเอกสารนี้จะบ่งบอกถึงรายละเอียดต่างๆที่เกี่ยวข้องกันของโปรแกรมอย่างละเอียด

External Documentation ครอบคลุมถึงภาพรวมของระบบ ซึ่งอาจประกอบด้วยหลายๆ กลุ่มโมดูล เป็นแผนภาพไดอะแกรม ความสัมพันธ์ระหว่างโมดูลต่างๆ แสดงถึงโครงสร้างข้อมูลต่างๆที่ใช้งานร่วมกัน ส่วนสำคัญประกอบด้วย

Describing the Problem ในส่วนแรกของ External Documentation จะอธิบายถึงปัญหาต่างๆ และตำแหน่งที่อยู่ของปัญหาในโปรแกรม รวมทั้งอธิบายถึงทางเลือกในการแก้ปัญหาและข้อสรุปว่าทำไมถึงเลือกวิธีนี้ในการแก้ปัญหา

Describing algorithms เป็นการบรรยายถึงอัลกอริทึมที่ใช้ในโปรแกรมของแต่ละโมดูล ซึ่งอธิบายถึงสูตรที่ใช้ ขอบเขต ขั้นตอนวิธีการในการปฏิบัติงาน เงื่อนไขพิเศษต่างๆ รวมถึงการได้มาโดยอ้างอิงจาก textbook หรือ journal artical หรือ การออกแบบโปรแกรม

Describing the Data สำหรับการบรรยายข้อมูลของเอกสารภายนอกนี้ ผู้ใช้หรือโปรแกรมเมอร์สามารถมองเห็นทิศทางการไหลของข้อมูลผ่านระบบเป็นชั้นของโมดูลโปรแกรม

## 6.4

# ตัวอย่าง

เนื่องจากโปรแกรมที่ทำการพัฒนาขึ้นนั้นมีระดับความยากง่ายแตกต่างกัน ดังนั้นมาตรฐานในการเขียนเอกสารที่ใช้ในการพัฒนาแตกต่างกันไปตามขนาด, ความยากง่าย , ความซับซ้อน ฯลฯ

สำหรับระบบที่มีความซับซ้อนและความยากอยู่ในระดับต่ำนั้น ข้อเสนอแนะในการในการพัฒนาโปรแกรมสรุปได้เป็นดังนี้

DOCUMANTATION:

- Internal document:
  - Header comment block
  - Indentation to exhibit control structure
  - Meaningful variable and data names
  - Definition of interface requirements
  - Description of program invocation

PROCESSING:

- Error handing:
  - Brief error messages

CONTROL FLOW:

- Use structured techniques
- Keep coupling low
- Keep cohesion high

LANGUAGE:

- Avoid unconditional branching where possible
- Use language-specific structures to maximize productivity

# 6.5

# สรุป

ในบทนี้เป็นการให้ข้อเสนอแนะในการเปลี่ยนขั้นตอนการออกแบบโปรแกรมเป็นการสร้างโปรแกรมคำสั่งเพื่อให้คอมพิวเตอร์สามารถปฏิบัติงานได้ ทำให้ทราบว่ามาตรฐานในการพัฒนาเป็นสิ่งที่จำเป็นและเป็นเครื่องมือที่ช่วยให้โปรแกรมเมอร์สามารถทดสอบและแก้ไขคำสั่งโปรแกรมได้ง่าย สำหรับโครงสร้างข้อมูลที่ใช้ในโปรแกรมนั้นควรเก็บในรูปแบบที่ง่ายต่อการทำความเข้าใจ ซึ่งโครงสร้างของข้อมูลสามารถมีอิทธิพลต่อการเลือกภาษาโปรแกรมที่ใช้ในการเขียนคำสั่งโปรแกรม

คำอธิบายโปรแกรมเป็นสิ่งที่เป็นประโยชน์ทั้งในส่วนของการปฏิบัติงานทั้งข้อมูลเข้าและข้อมูลออก โดยทำให้เราสามารถทดสอบความถูกต้องของโมดูลโปรแกรมได้ง่าย รวมทั้งการแก้ไขการปรับเปลี่ยนคำสั่งต่างๆไม่สับสน เอกสารนี้อาจอยู่ในรูปของคำอธิบายภายในโปรแกรมในรูปของ header comment block หรือการใช้ชื่อของคำต่างๆที่สื่อความ หรือการเยื้องหรือจัดรูปแบบของคำสั่งต่างๆ สำหรับเอกสารที่อยู่ภายนอกโปรแกรมนั้นเป็นภาพรวมของโปรแกรมที่จะบอกต่อนักวิเคราะห์หรือนักออกแบบ ว่าระบบมีองค์ประกอบอะไรบ้าง ทำอะไรได้บ้าง และกระทำได้อย่างไร ซึ่งอธิบายถึงตรรกของโปรแกรม หนทางในการแก้ปัญหาต่างๆ อัลกอริทึมและโครงสร้างข้อมูลที่เลือกใช้ในการแก้ปัญหา

เราถือได้ว่าการเขียนคำสั่งเป็นศิลปะอย่างหนึ่งในการประมวลผล โดยข้อเสนอแนะในบทนี้เป็นเพียงวิธีการหนึ่งของวิศวกรรมซอฟต์แวร์ เราสามารถกำหนดถึงมาตรฐานที่เหมาะสมขึ้นมาเพื่อใช้งานและผลิตซอฟต์แวร์ที่มีคุณภาพได้ โดยเน้นที่ความง่ายในการอ่าน ง่ายต่อการทดสอบ และง่ายต่อการบำรุงรักษาคำสั่ง

# 6.6

# แบบฝึกหัด

1. Program Implementation เป็นขั้นตอนที่กระทำอะไร และใครเป็นผู้รับผิดชอบ
2. มาตรฐานในการสร้างโปรแกรมมีความสำคัญอย่างไร
3. โครงสร้างของข้อมูลมีความสำคัญอย่างไรต่อการสร้างโปรแกรม
4. เอกสารสำหรับการสร้างโปรแกรมมีอะไรบ้าง จงอธิบายให้เข้าใจ
5. จงยกตัวอย่างโมดูลโปรแกรมที่มีคุณภาพที่ดี และอธิบายพอเข้าใจว่าดีอย่างไร
6. Header comment block คืออะไร ประกอบด้วยส่วนสำคัญอะไรบ้าง