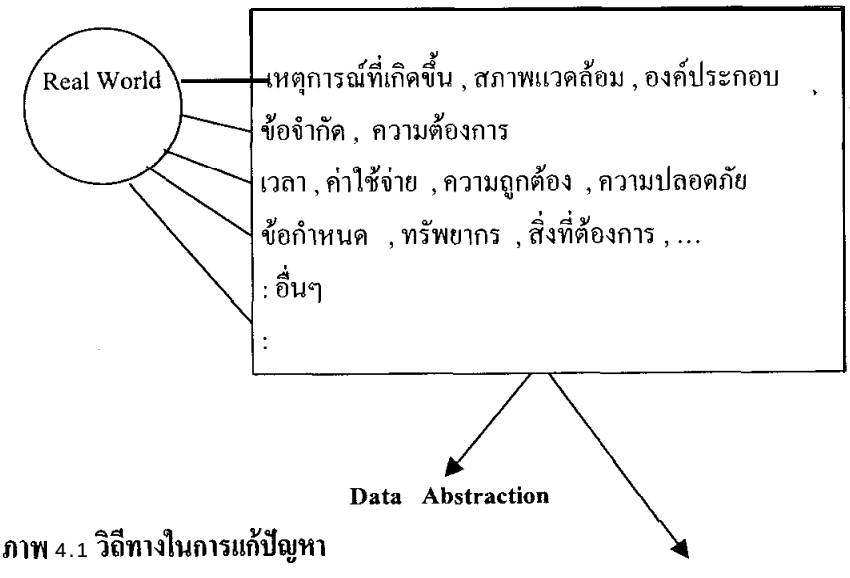
บทที่ 4
การสร้างโปรแกรมคอมพิวเตอร์เพื่อนำไปใช้ในการแก้ปัญหา โดยใช้รูปแบบ Traditional Programming

บทนำ

ถึงแม้ว่าเราจะยอมรับว่าแนวทางการออกแบบโปรแกรมนั้นควรจะใช้รูปแบบของ OOP ก็ตาม แต่สิ่งที่ยังคงต้องตระหนักถึงก็คือ การออกแบบโปรแกรมในรูปแบบของ Procedural Program โดยการใช้ภาษา High Level Language ก็ยังคงมีความจำเป็นอยู่ในกรณีของระบบงานและ/หรือคอมพิวเตอร์บางระบบยังดำเนินการตามรูปแบบของการโปรแกรมแบบเดิม นอกเหนือจากนี้การเรียนรู้และทำความเข้าใจในวิธีการดำเนินงานในลักษณะของ Procedural Style ก็ยังอำนวยประโยชน์ในแง่ของการสร้างความคิดทางตรรกะ การเรียนรู้วิธีเลือกระเบียบวิธีดำเนินงาน หรือที่เราเรียกว่า Algorithm ที่มีประสิทธิภาพได้ นอกจากนี้ยังสามารถเปรียบเทียบกรรมวิธีการดำเนินการออกแบบโปรแกรมในรูปแบบดั้งเดิม คือ Procedural Technique กับรูปแบบของ OO Technique ว่ามีความแตกต่างกันอย่างไรมีข้อดีเสียอย่างไร ในแต่ละวิธี ในบทนี้จะว่าด้วยการออกแบบโปรแกรมโดยใช้ Procedural Technique

แนวทางในการออกแบบโปรแกรม Procedural Program

การแก้ปัญหาของมนุษย์ในกิจกรรมที่สามารถดำเนินการด้วยระบบคอมพิวเตอร์ จะมีวิถีทางในการดำเนินงานดังภาพ



ภาพ 4.1 วิถีทางในการแก้ปัญหา

ภาพ 4.2 การดำเนินงานระมวลผล

เราอาจจะสรุปขั้นตอนในการพัฒนาโปรแกรม (The Software Development Method)

1. การแก้ปัญหาและกำหนดความต้องการ (Specify the problem requirements)
2. วิเคราะห์ปัญหา (Analyze the problems)
3. ออกแบบขั้นตอนการดำเนินงาน (Design the algorithms to solve the problem)
4. ทดสอบการใช้งานของขั้นตอนการดำเนินงาน (Implement the algorithm)
5. ตรวจสอบ และทดสอบ โปรแกรม (Test and verify the completed program)
6. บำรุงดูรักษาและพัฒนาโปรแกรม (Maintain and update the program)

ปัญหาที่เกิดขึ้น (Problem)

การกำหนดความต้องการเพื่อที่จะแก้ปัญหานั้นเราจำเป็นที่จะต้องพิจารณแยกแยะให้เข้าใจอย่างถ่องแท้ถึงขอบเขตของความต้องการที่แท้จริง นอกจากนี้ในการแยกแยะปัญหาเราจะต้องพิจารณากรณีที่อาจจะก่อให้เกิดข้อจำกัด หรือ กรณีที่ดีความคลุมเครือ เราจำเป็นจะต้องวิเคราะห์ให้ละเอียดและถูกต้องโดยการสอบถามจากผู้ใช้ในระบบ ก่อนที่จะนำไปเป็นข้อมูลในการกำหนดเป็น Problem Specificationที่จะดำเนินการต่อไปในขั้นตอนการออกแบบโปรแกรม

การวิเคราะห์ (Analysis)

การวิเคราะห์ปัญหาประกอบด้วยการแยกแยะสิ่งที่ต้องการดำเนินการอันประกอบด้วย

- ข้อมูลที่นำเข้า (Input Data) ว่าจะต้องมีการนำเข้าข้อมูลหรือไม่ ถ้ามี ข้อมูลจะมีโครงสร้างเช่นใด และใช้อุปกรณ์อะไรเป็นเครื่องมือในการส่งข้อมูลเข้าระบบเพื่อนำไปประมวลผลต่อไปตามความต้องการ
- ข้อมูลผลลัพธ์ (Output Data) ว่ามีรูปแบบเป็นอย่างไร จะต้องมีการออกแบบรูปแบบที่นำเสนอเป็นอย่างไร จะใช้สื่อและอุปกรณ์ใดในการแสดงผล
- วิธีการดำเนินการ ตลอดจนข้อจำกัดในวิธีการแก้ปัญหา ในการวิเคราะห์นั้นเราจะต้องระมัดระวังและละเอียดรอบคอบ

- การนิยามข้อมูลให้เป็นนามธรรม (Data Abstraction) หมายถึงการกำหนดโครงสร้างข้อมูลที่จะจัดเก็บในสมองเครื่อง และข้อมูลอื่นๆที่จะต้องจัดเก็บในระหว่างการประมวลผล ซึ่งหมายถึงการกำหนดตัวแปรต่างๆที่จัดเก็บ

การออกแบบโปรแกรม (Design)

โปรแกรมก็คือชุดของคำสั่ง (ขั้นตอน) ในการปฏิบัติงาน ดังนั้นการออกแบบโปรแกรมก็คือการออกแบบขั้นตอน (Algorithm) เพื่อให้สนองตอบต่อเจตนาในการนำไปปฏิบัติงาน โดยการใช้รูปแบบของ Top Down Design (Divide and Conquer) เพื่อออกแบบขั้นตอนในการทำงานโดยการแบ่งออกเป็นกิจกรรมย่อยๆที่เรียกว่า Subprogram โดยที่ Subprogram ก็คือโปรแกรมย่อยแต่ละโปรแกรมนั่นเอง เราจะต้องทำการแจงความต้องการของแต่ละโปรแกรมย่อยออกมาแล้วทำการดำเนินงานของแต่ละโปรแกรมย่อยให้สำเร็จ ภายหลังจึงนำโปรแกรมย่อยเหล่านั้นมาเชื่อมต่อ (Interface) เข้าด้วยกัน รูปแบบในการทดสอบและเชื่อมต่อ (Software Interface) ต่อนั้นจะมีลักษณะเป็นแบบจากล่างขึ้นบน (Bottom Up) โปรแกรมย่อยแต่ละโปรแกรมนั้นจะถูกออกแบบและทำการทดสอบเสมือนกับโปรแกรมเดี่ยว เพียงแต่ภายหลังการสร้างเสร็จและทดสอบตัวเอง (Unit Testing) เสร็จจะต้องนำมาทดสอบกับการเชื่อมต่อที่เรียกว่า String Testing ว่าสามารถติดต่อกันได้สำเร็จตามวัตถุประสงค์หรือไม่ กรรมวิธีการออกแบบดังกล่าวนี้สามารถใช้ได้กับภาษาที่เป็นภาษายุคที่3 (Third GL) เช่น COBOL , PASCAL , C และภาษายุคที่ 4 (Fourth GL) เช่น Dbase , SPSS , SAS เพราะจะมีลักษณะคล้ายคลึงกันเพียงแต่การใช้ภาษายุคที่3 (Third GL) จะยุ่งยากมากกว่าในการทำงานบางอย่างเช่นการดำเนินงานกับแฟ้ม หรือการวิเคราะห์ข้อมูล เพราะจะดำเนินการเขียนคำสั่งเองในรูปแบบของ Procedural Language ในขณะที่ภาษา ยุคที่ 4 (Fourth GL) จะมีคำสั่งที่ใช้งานที่เป็นคำสั่งเบ็ดเสร็จให้ใช้งานซึ่งจะใช้คำศัพท์ภาษาอังกฤษ ตัวอย่างเช่นการเปิดแฟ้มข้อมูลในภาษา Dbase จะใช้คำว่า USE “filename” ในขณะที่ภาษาที่เป็น Procedural Language เช่น PASCAL จะต้องเขียนคำสั่งดังนี้

```
Program  Demo;
Type  Rec = Record
          Name :String[30] ;
          Age    : Integer;
          Salary : Longint
        End;
Var
          Fname  : String [20];
          Fi       : File of Rec;
```

```
        Emp : Rec;
Begin
        Write ("Input File Name to be Opened");
        Readl n( Fname) ;
        Assign (Fi,Fname);
        While (not EOF(Fi)) Do
          Begin
              Read (Fi , Emp) ;
              Writeln( "NAME OF EMPLOYEE") ;
              Readln() ; {* Press any Key to Display Next Record*}
        End ;
        Close (Fi) : {* Close File *}
        Readln
End .
```

ในกรณีที่เขียนด้วยภาษา Fourth GL ซึ่งเป็นภาษา Non Procedural เช่น Dbase จะเขียนได้ดังนี้

Set Default to A:	กำหนดให้ระบบใช้ drive A:
Use "data"	ทำการเปิดแฟ้มข้อมูลที่ drive A: ชื่อว่า data
Display	แสดงข้อมูลแต่ละ Record ออกที่จอ
Use	ปิดแฟ้มข้อมูล

เปรียบเทียบรูปแบบการเขียนโปรแกรมทั้ง 2 วิธีจะเห็นได้ว่าทั้งสองโปรแกรมนั้นจะถูกออกแบบมาในลักษณะเดียวกันคือจะลำดับการว่าเมื่อมีข้อมูลส่งเข้ามาโปรแกรมก็จะปฏิบัติการตามที่สั่งไว้ เพียงแต่ว่าการเขียนโปรแกรมในรูปแบบของภาษา Fourth GL จะง่ายกว่าเพราะคำสั่งบางคำจะเป็นสิ่งที่ซ่อนวิธีดำเนินงานหลายๆอย่างเข้าไว้ด้วยกันซึ่งจะช่วยในการทำงานของผู้เขียนโปรแกรมให้ง่ายเข้าแต่เบื้องหลังก็จะต้องไปถอดเป็นคำสั่งย่อยเพื่อดำเนินงานดังเช่นภาษาตระกูล Procedural Language

เนื้อหาในหนังสือเล่มนี้จะเน้นการออกแบบโปรแกรมตามรูปแบบของการโปรแกรมแบบดังเดิมโดยใช้ลักษณะของ Procedural Language

ดังที่กล่าวมาแล้วว่าโปรแกรมหนึ่งโปรแกรมนั้นจะถูกแบ่งออกเป็นส่วนๆในรูปแบบที่เรานิยมใช้คือ Modula Approach ซึ่งเป็นรูปแบบของ Top Down Design เราจึงมองโปรแกรมย่อยหรือโมดุลว่าเปรียบเสมือนโปรแกรมเดี่ยวนั่นเอง

การออกแบบโปรแกรมหรือโปรแกรมย่อย นั้นจะมีขั้นตอนการดำเนินงานดังนี้คือ

- Read Data (Optional)
- Perform the computations
- Display the result or Store data to media

ภายหลังการแจงขั้นตอนซึ่งจะเรียกว่า Algorithm จะต้องนำ Algorithm มาแยกเป็นส่วนย่อยๆเพื่อตรวจสอบความถูกต้องรวมถึงประสิทธิภาพของแต่ละ Algorithm ด้วย องค์ประกอบของประสิทธิภาพก็คือ การใช้พื้นที่ในการดำเนินงาน และความเร็วในการทำงาน นอกจากนี้ยังต้องคำนึงถึงองค์ประกอบอย่างอื่นที่เกี่ยวพันกับการนำ Algorithm ดังกล่าวไปเขียนโปรแกรมต่อไป

องค์ประกอบของโปรแกรมที่ดีจะประกอบด้วย

- ดำเนินงานได้ถูกต้องตามเป้าหมาย (Correctly)
- เป็นโปรแกรมที่เข้าใจง่ายเข้า (Understandable)
- มีความคงทน (Robustness)
- เปลี่ยนแปลงได้ง่ายตามสภาพแวดล้อม (Easy to Change)
- เคลื่อนย้ายได้ง่าย (Portable)
- ใช้พื้นที่น้อย (Storage Efficiently)
- ใช้เวลาในการดำเนินงานน้อย (Execute Efficiently)
- มีความน่าเชื่อถือ (Reliability)
- มีระบบปกป้องตนเอง (Defensive)

- **ดำเนินงานได้ถูกต้องตามเป้าหมาย (Correctly)** หมายความว่าสามารถจะให้ตอบที่ถูกต้องไม่ว่าข้อมูลจะมีลักษณะเช่นใดก็ตาม ตัวอย่างเช่น โปรแกรมคำนวณหาเงินรายได้สุทธิของคนงาน นั้นสามารถจะคำนวณถูกต้องไม่ว่าคนงานจะมีรายได้จะขนาดมากน้อยเท่าไร ก็ตามหรือโปรแกรมคำนวณหาค่าน้ำประปา ซึ่งมีกฎในการคิดค่าน้ำตามขนาดที่ใช้ว่า ถ้าผู้ใช้ใช้น้ำไม่เกิน 20 หน่วยจะคิดราคาหน่วยละ 3.50 บาท แต่ถ้าใช้ตั้งแต่ 20 – 50 หน่วย จะคิดราคาหน่วยละ 5.50 บาท และในกรณีที่ ใช้ตั้งแต่ 50 หน่วย ขึ้นไปจะคิดราคาหน่วยละ 7.50 บาท ดังนั้นไม่ว่าผู้ใช้น้ำจะใช้น้ำปริมาณเท่าไรโปรแกรมก็จะต้องทำการคำนวณได้อย่างถูกต้องเสมอไป
- **เป็นโปรแกรมที่เข้าใจง่ายเข้า (Ease)** ผู้เขียนโปรแกรมบางคนเข้าใจว่าถ้าออกแบบโปรแกรมได้สั้นเท่าไรยิ่งจะทำให้โปรแกรมนั้นเป็นโปรแกรมที่ดี การ

ออกแบบโปรแกรมที่สั้นโดยที่โปรแกรมนั้นมี Algorithm ที่เข้าใจยากหรือบางครั้งผู้อื่นที่มาอ่านนั้นไม่สามารถทำความเข้าใจได้จะไม่ใช่สิ่งที่ดี เพราะในการที่จะต้องมีการปรับปรุงแก้ไขโปรแกรมนั้น เพื่อสนองตอบต่อสภาพแวดล้อมและความต้องการที่แปรเปลี่ยนไปในอนาคตนั้นจะทำยากมาก บางครั้งผู้พัฒนาอาจจะไม่สามารถพัฒนาได้ ท้ายสุดอาจจะต้องทิ้งไปและสร้างใหม่ขึ้นมาใช้งาน ลักษณะที่เกิดเช่นนี้จะส่งผลให้เราเสียค่าใช้จ่ายมาก ตัวอย่างของการออกแบบเช่นนี้คือ

สมมุติว่า เราจะต้องดำเนินการรับข้อมูลของลูกค้าของบริษัทโดยแบ่งตามอายุ เป็นกลุ่มๆเพื่อจะนำไปเป็นสารสนเทศในการบริหารการขายสินค้า โดยที่ข้อมูลที่นำเข้าจะเป็นอายุจริงของลูกค้าแต่ละคน (เลขบวกจำนวนเต็มสองหลัก มีค่าตั้งแต่ 10 ปีขึ้นไป จนถึง 60 ปี) กลุ่มอายุของลูกค้าจะมีตารางเสนอผลเป็นรูปแบบตารางที่ 1 ดังนี้คือ

กลุ่มอายุ	จำนวน	ร้อยละ	หมายเหตุ
11 - 20	XXX	XXX	
21 – 30	XXX	XXX	
31 – 40	XXX	XXX	
41 – 50	XXX	XXX	
51 – 60	XXX	XXX	

ตารางที่ 4.1

จากตารางที่ต้องการนี้ ถ้าเราใช้ Algorithm หลังจากรับข้อมูลอาย (AGE) ของลูกค้ารายใดรายหนึ่งมาจัดกลุ่มดังนี้คือ

K = (AGE –1) DIV 10

T (K) = T(K) +1

ผลที่ได้จากขั้นตอนดังกล่าวก็คือ K = 1,2,...,5 (สมมุติว่า อายุที่รับมาจะมีค่าตั้งแต่ 11 ถึง 60) คำสั่ง ดังกล่าวจะนำอายุที่รับเข้ามา ลบ ด้วย 1 เพื่อค่าก่อนที่จะนำไปสร้างค่า K ต่อไปโดยใช้คุณสมบัติของการหาร (เครื่องหมาย DIV ในภาษา ปาสคาล สำหรับการนำเลขจำนวนเต็มหารด้วยเลขจำนวนเต็ม โดยปักเศาทิ้ง) แล้วนำค่าที่ได้ไปเก็บในตัวแปร K ที่เป็นตัวแปรแบบ Integer จากนั้นจึงนำไปสะสมในตัวแปร อะเรย์ T(K) ; K =1,2,..,5 ต่อไป การใช้ขั้นตอนการดำเนินงานดังนี้อาจจะเหมาะสมกับผู้เขียนโปรแกรมที่มีพื้นฐานทางคณิตศาสตร์ แต่ถ้า

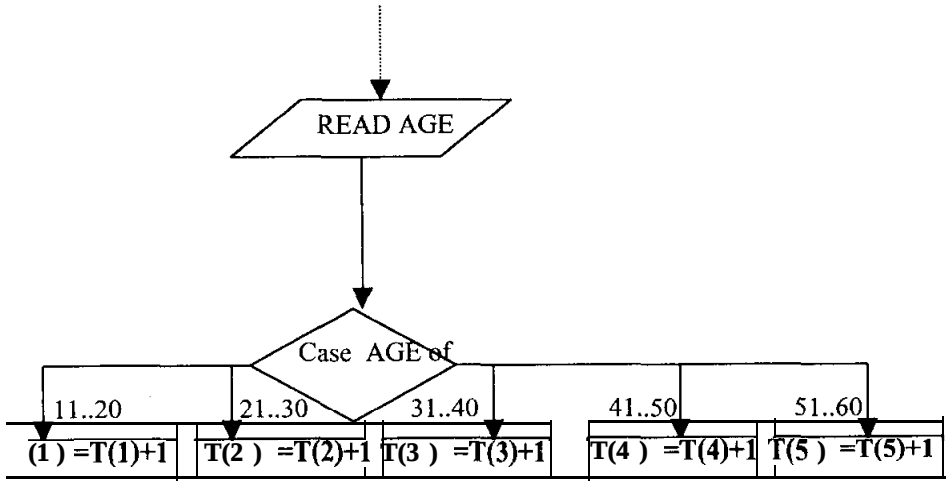
ผู้ที่ไม่มีพื้นฐานมาอาจจะอ่านแล้วไม่เข้าใจ นอกจากนี้ในกรณีที่ต่อไปจะนำไปจะนำโปรแกรมดังกล่าวไปใช้ในการจัดกลุ่มใหม่ที่มีลักษณะเป็นกลุ่มที่แปรเปลี่ยนไปจากเดิม เช่น ตารางที่2

กลุ่มอายุ	จำนวน	ร้อยละ	หมายเหตุ
11 - 25	XXX	XXX	
26 – 35	XXX	XXX	
36 – 45	XXX	XXX	
46 – 60	XXX	XXX	
> 60	XXX	XXX	

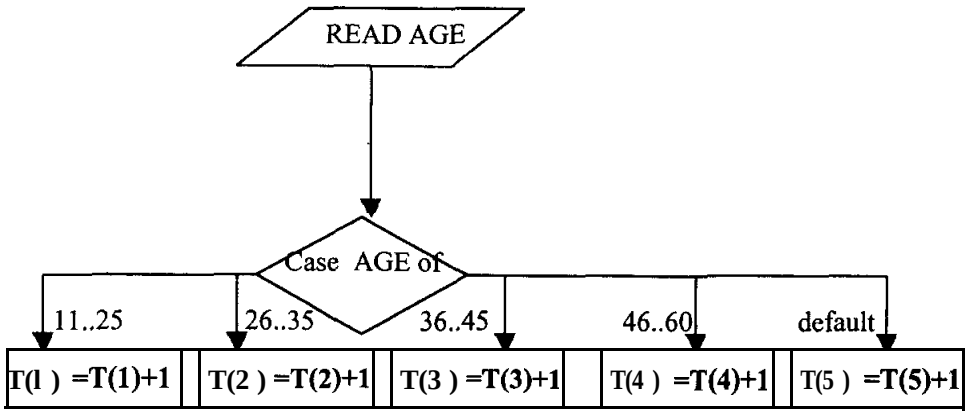
ตารางที่ 4.2

โปรแกรมที่ใช้ขั้นตอนดังกล่าวก็จะต้องดำเนินการแก้ไขใหม่ค่อนข้างมาก ซึ่งอาจจะหมายความว่า การเขียนขึ้นมาใหม่จะง่ายกว่า

ดังนั้นสำหรับในปัญหานี้การออกแบบที่ดีกว่า ก็คือการใช้คุณสมบัติของคำสั่ง CASE เข้ามาช่วยในการดำเนินงาน ดังรูปแบบนี้



การใช้ CASE อาจจะมีจุดบกพร่องตรงที่ว่าค่าที่จะจำแนกนั้น จะต้องเป็นค่าคง หรืออักขระที่คงที่ หรืออาจจะเป็น ช่วงที่ปรากฏ และในกรณีที่เป็นช่วงจะต้องมีค่าข้างต่ำและค่าข้างสูงจะปล่อยให้ว่างไม่ได้ดังนั้น ในบางภาษาก็มักจะสร้าง Default ให้ใช้เป็นกรณีสุดท้ายด้วยดังนั้น จากความต้องการในตารางที่ 2 เราก็อาจจะเลือกทางเลือกต่อไปนี้ในการออกแบบก็ได้



บางระบบนั้นจะใช้คำสั่ง Otherwise แทน Default ตัวอย่างการเขียนโปรแกรม ดังกล่าว โดยใช้ภาษา ปาสคาล

```
Case Day of
        1 , 7 : Gross := Hours * 1.5 * Dailyrate ;
        2,3,4,5,6  Gross := Hours * Dailyrate ;
        Otherwise : Writeln ('Invalid day number' )
End ; { End Case}
```

แต่ถ้าเป็นกรณีที่ในภาษานั้นไม่มีทั้ง Default หรือ Otherwise เราก็สามารถหาทางออกโดยการใช้ If then else เข้าช่วยได้ดังนี้

```
if (Day >=1) and (Day ,=7) then
    Case Day of
        1 , 7 : Gross := Hours * 1.5 * Dailyrate ;
        2,3,4,5,6  :Gross := Hours * Dailyrate
        End ; { End Case}
Else
Writeln ('Invalid day number' ) ;
```

- **เป็นโปรแกรมที่มีความคงทน (Robustness or Durability)** หมายความว่า โปรแกรมสามารถทำงานได้ในทุกสภาวะ เช่นถ้าเกิดมีความผิดพลาดบางอย่างปรากฏ เช่นข้อมูลที่ผิดปกติเข้าไปในระบบโปรแกรมก็สามารถตรวจสอบและรายงานความผิดพลาดได้ โดยไม่เกิดสภาวะของ Abnormal End แล้วทำให้เครื่องทำการ Terminate

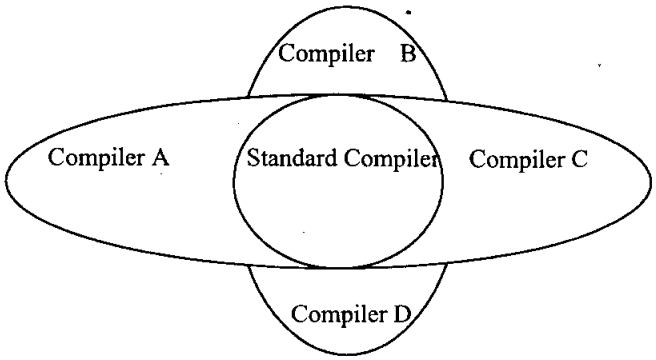
โปรแกรมนั้น ซึ่งส่งผลให้ผู้ใช้ไม่สามารถทราบความผิดพลาดได้ ต้องทำการค้นหาเอง ซึ่งเสียเวลาและยุ่งยาก คุณสมบัติของความคงทนนี้จะมีพื้นฐานเกี่ยวข้องกับคุณสมบัติของการ สร้างระบบปกป้องตนเองของโปรแกรม แต่ถึงอย่างไรก็ตามในกรณีของโปรแกรมขนาดใหญ่นั้นจึงเป้าฯสิ่งที่ยอมรับ กันเกือบจะเป็นทฤษฎีแล้วว่า "เราไม่อาจประกันได้ว่าโปรแกรมนั้นปลอดจากความผิดพลาด (Error free) " เพียงแต่ยอมรับโดยการทดสอบว่าในกรณีของการประมวลผลตามสภาวะปกตินั้น โปรแกรมดังกล่าวสามารถทำงานได้ ส่วนในกรณีที่ทำงานที่ลึกซึ้งหรือซับซ้อนมากๆนั้น อาจจะเกิดความผิดพลาดขึ้นได้ ข้อความดังกล่าวนี้ผู้ใช้โปรแกรมบน Windows คงจะประสบว่าในการทำงานนั้นบางครั้งเกิดความผิดพลาดได้จากการประมวลผล คุณลักษณะของโปรแกรมที่สร้างระบบปกป้องตนเองไม่ให้ล้มเหลวดดยง่ายนั้น จะถูกเรียกว่า Defensive Programming

- การปรับเปลี่ยนให้เข้ากับสภาพแวดล้อม (Easy to Change) เราจะต้องออกแบบโปรแกรมให้มีสภาพเป็น Dynamic ไม่ใช่ Static เพราะว่าโปรแกรมจะต้องมีความสามารถที่จะอยู่บนสภาพแวดล้อมที่แปรเปลี่ยน (Open System) ไปได้โดยไม่ต้องปรับแต่งโปรแกรมมากมายตัวอย่างเช่นโปรแกรมการคำนวณหาค่าราคาสินค้าที่จะต้องบวกค่า Vat เข้าไปด้วยนั้นจะเห็นได้ว่าค่าของVat นั้นจะมีการเปลี่ยนแปลงค่อนข้างจะบ่อยอันสืบเนื่องมาจากสภาวะทางเศรษฐกิจ ดังนั้นถ้าเราออกแบบโปรแกรมโดยการกำหนดค่า Vat ให้ตายตัวลงไป เมื่อมีการเปลี่ยนแปลง Vat ในแต่ละครั้งก็จะเกิดปัญหาในการที่จะต้องปรับเปลี่ยนโปรแกรมให้สอดคล้องกับค่าที่เปลี่ยนไป ซึ่งจะต้องนำโปรแกรมที่แก้ไขแล้วมา Compile , Link/loader ทุกครั้งก่อนที่จะนำไปปฏิบัติงาน ซึ่งจะทำให้เราเสียเวลามาก ดังนั้นในกรณีตัวอย่างดังกล่าวเราควรจะกำหนดค่าของ Vat ให้อยู่ในรูปแบบอื่นแทนที่แก้ไขได้สะดวกกว่า เช่นรับข้อมูลจากแป้นพิมพ์
- โปรแกรมควรจะเคลื่อนย้ายได้ง่าย (Portable) ภาษาโปรแกรมแต่ละภาษาในระดับของ High Level Language นั้นถึงแม้ว่าจะเป็นภาษาเดียวกันตัวอย่างเช่น ภาษาปาสคาลก็จะมีหลายรุ่นขึ้นอยู่กับบริษัทที่ผลิต Software เช่น Turbo Pascal , UCLD Pascal , Borland Pascal , Visual Pascal แต่ละตัวก็จะรูปลักษณ์พิศษเฉพาะแบบบางอย่างเพิ่มขึ้นมาจากภาษาปาสคาลปกติ ดังนั้นการเขียนโปรแกรมภาษาปาสคาลขึ้นมาใช้งานโดยมีความต้องการที่จะปรับเปลี่ยนไปใช้งานกับภาษาปาสคาลของบริษัทใดก็สามารถทำได้โดยไม่ต้องแก้ไขโปรแกรม เพื่อให้บรรลุความต้องการของวัตถุประสงค์นี้จึงมีการกำหนดมาตราฐานของภาษาปาสคาล (Standard Pascal) โดยที่มาตราฐานจะประกอบด้วยกฎเกณฑ์ ต่างๆ เช่น Syntax ของภาษา โครงสร้างของตัวแปร ขึ้นมาเพื่อที่จะได้เป็นการกำหนดให้ภาษาปาสคาลแต่ละตระกูลกำหนดตัวเองให้สามารถนำไปใช้กับภาษาปาสคาล

อื่นๆได้และสามารถใช้งานกับเครื่องคอมพิวเตอร์ต่างตระกูลได้ ซึ่งจะแตกต่างกับภาษาเครื่อง (Machine language) เพราะจะมีลักษณะติดกับระบบคอมพิวเตอร์ (Machine Dependent) เนื่องจากโปรแกรมที่เขียนขึ้นด้วยภาษาเครื่องจะใช้งานกับระบบคอมพิวเตอร์ระบบนั้นได้เพียงระบบเดียวไม่สามารถทำงานกับคอมพิวเตอร์ตระกูลอื่นได้

ตัวอย่างของคำสั่งที่ไม่เป็น Standard Pascal ตัวอย่างเช่น X : String[10] ถ้าประกาศตัวแปรเป็น String จะไม่สามารถใช้กับภาษาปาสคาลบางรุ่นบางแบบได้ แต่ถ้าใช้รูปแบบของ Standard Pascal คือ X : ARRAY [1..10] of CHAR ก็จะสามารถทำงานกับภาษาปาสคาลได้ทั่วไป หรือกรณีของภาษาเบสิค นั้นการใช้คำสั่ง NEXT จะถือเป็น Standard Basic ส่วน N. (คำย่อของ NEXT) นั้นจะไม่ใช่ Standard Basic การยึดตามาตราฐานนั้นมิใช่เกิดเฉพาะในส่วนของการเขียนคำสั่งเท่านั้นแต่อย่างรวมความถึงขั้นตอนการ Compile ด้วยเช่นการดำเนินงานที่เรียกว่า Directive Compile จะยอมให้สำหรับ Software บางตัว ซึ่งไม่จัดว่าเป็นรูปแบบของ Standard Basic

สรุปโดยภาพรวมของการออกแบบและเขียนโปรแกรมโดยใช้ภาษาระดับ High Level Language ก็คือให้ยึดหลักของ Standard ของภาษานั้นเป็นเกณฑ์อาจจะดูในรูปของเซตได้ดังนี้



การเขียนหรือออกแบบโปรแกรมให้ยึดเอาเกณฑ์ที่เป็นคุณสมบัติร่วมมาใช้ อย่าใช้คุณสมบัติเฉพาะตัวของ Compiler

- การใช้พื้นที่น้อย (Storage Efficiency) หมายถึง Code ที่ได้จากการแปล (Compile) จะเป็น Code ที่สั้นซึ่งส่งผลให้เราใช้พื้นที่ในการจัดเก็บน้อย และการใช้พื้นที่ในการจัดเก็บข้อมูลน้อย

ประสิทธิภาพของโปรแกรม (Program Efficiency) จะหมายถึงประสิทธิภาพในขณะแปล (Compile) และประสิทธิภาพในขณะที่ประมวลผล (Execution Time) เพราะถ้าหากว่าโปรแกรมที่เราเขียนขึ้นมาใช้เวลาในการแปลนาน เราจะต้องสูญเสียเวลาไปมาก

เพราะการ แปล ไม่ใช่กระทำเพียงครั้งเดียว การพัฒนาโปรแกรมกว่าที่จะทำงานได้บางครั้งต้องกระทำหลายครั้ง ข้อเท็จจริงที่เราประสบก็คือโดยปกติ **Compiler ที่มีความสามารถในการ Compile คือสามารถให้ Object Code มีประสิทธิภาพมักจะต้องใช้เวลาในการ Compile นาน** ในขณะที่ Compiler **ที่ใช้เวลาในการ Compile น้อย**นั้นมักจะให้ **Object Code ที่ไม่ค่อยมีประสิทธิภาพ** ดังนั้นในตัว Compiler แต่ละตัวมักจะกำหนดทางเลือก (Option) สำหรับใช้เลือกในการ Compile โดยที่ถ้าเป็นการพัฒนาเพื่อทดสอบโปรแกรมในระยะต้นๆก็จะเลือกในส่วนของการ Compile ที่ใช้เวลาน้อย แต่ถ้าอยู่ในระยะที่จะพัฒนาออกมาเป็นโปรแกรมที่จะไปใช้งานจริงๆก็จะเลือกในส่วนของการ Compile ซึ่งจะให้ผลผลิตในการ Optimizing Code ตัวอย่างเช่นในสมัยก่อนจะใช้ WATFIV ซึ่งจัดว่าเป็น Compiler ที่เร็วมากในการ Compile และให้ Debug Message ที่ดีมากกับผู้ใช้ แต่จะให้ผลสำหรับ Object Module ที่เป็น Slow Object Code เป็นต้น นอกจากนี้เวลาในการ Compile โปรแกรม ส่วนหนึ่งก็เกิดจากกการเขียนคำสั่งที่ไม่ดีหรือการเลือกใช้ทรัพยากรที่ไม่เหมาะสมอีกด้วย ในกรณีที่เป็นสาเหตุที่เกิดจากการเขียนโปรแกรมผู้เขียนโปรแกรมก็ต้องหลีกเลี่ยงการใช้คำสั่งที่เขียนว่ามีผลเสียต่อการ Compile หรือไม่ ตัวอย่างเช่นการสร้างตัวแปรมากเกินความจำเป็น หรือการเขียน Logical Expression ที่ต้องนำไปตีความใหม่ เช่น IF (X < = 25) Then Process จากตัวอย่างคำสั่ง จะต้องมีการ นำ เงื่อนไข X < =25 ไปแปลงเป็น IF (X < 25 OR X=25) Then Process ซึ่งเราจะเขียนใหม่ได้ดังนี้คือ IF (X < 26) then Process

- โปรแกรมจะต้องใช้เวลาในการทำงานน้อย (Execution Time) หมายความว่า โปรแกรมทำงานตั้งแต่รับข้อมูลเข้าไปจนประมวลผลได้ Output ที่เราต้องการได้โดย ใช้เวลาน้อย การที่จะบรรลุเป้าหมายในข้อนี้ได้ต้องอาศัยองค์ประกอบหลายประการ เช่นการเลือก Algorithm ที่ดี การจัดการในส่วนต่างของระบบโปรแกรม ตัวอย่างเช่น ระบบโปรแกรมหนึ่งได้ถูกออกแบบมาเป็นส่วนๆ (Module ที่เรียกว่า โปรแกรมย่อย (Subroutine or Procedure or Function)) ได้เป็นส่วนๆดังนี้คือ

Module A ใช้เวลาในการทำงาน 5 % ของระบบ

Module B ใช้เวลาในการทำงาน 60 % ของระบบ

Module C ใช้เวลาในการทำงาน 15 % ของระบบ

Module D ใช้เวลาในการทำงาน 20 % ของระบบ

จากข้อมูลดังกล่าวจะพบว่า Module ที่เราควรจะให้การเอาใจใส่ในการพัฒนาก็คือ Module B เพราะ Module B ใช้เวลาของระบบถึง 60 % เพราะถ้าใช้เวลาที่มีจำกัดไปพัฒนา Module A เราจะได้ผลประโยชน์กลับมาน้อยมาก เมื่อเทียบกับ Module B

การที่เราจะใช้ทรัพยากรพื้นที่น้อยๆ นั้นมักจะผกผันกับการใช้ทรัพยากรเรื่องของเวลา กล่าวคือ โปรแกรมที่ประมวลผลได้รวดเร็วมากมักจะใช้ทรัพยากรพื้นที่มากและในขณะ

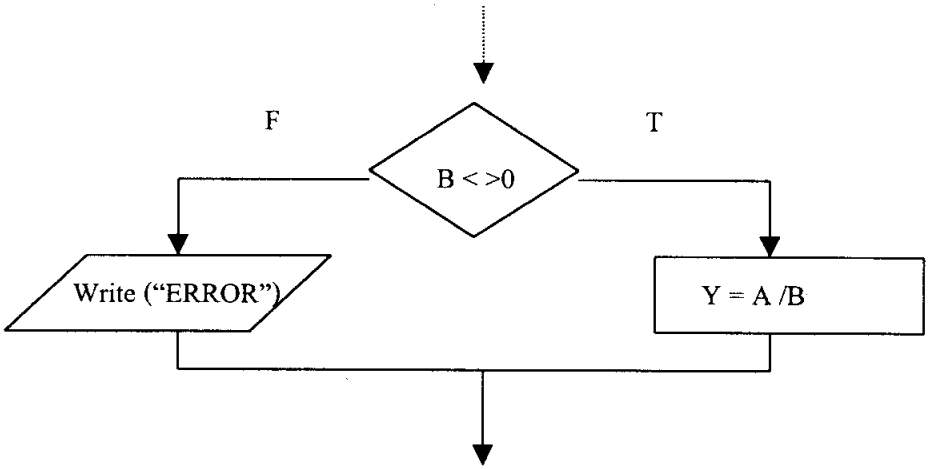
เดียวกันถ้าหากว่าเราใช้ทรัพยากรพื้นที่น้อยก็ส่งผลให้การประมวลผลนั้นใช้เวลามาก การจะเลือกพื้นที่ (Storage) หรือ เวลา (Execute Time) นั้นมักจะขึ้นอยู่กับระบบงานและระบบเครื่องเป็นเกณฑ์ เช่นถ้าเป็นระบบ Online ก็จะเลือกให้ระบบนั้นดำเนินงานได้เร็วที่สุดนั่นหมายความว่าเราจะต้องใช้เครื่องที่มีขนาดใหญ่ แต่ในกรณีที่เป็นงานทั่วๆไปเราก็มักจะเลือกโดยใช้การจัดสรรขนาดที่เป็น Optimum Solution

- โปรแกรมมีความน่าเชื่อถือ (Reliability) มาตราการต่อไปนี้จะเป็นแนวทางในการประเมินคุณสมบัติของโปรแกรมในส่วนของความน่าเชื่อถือ

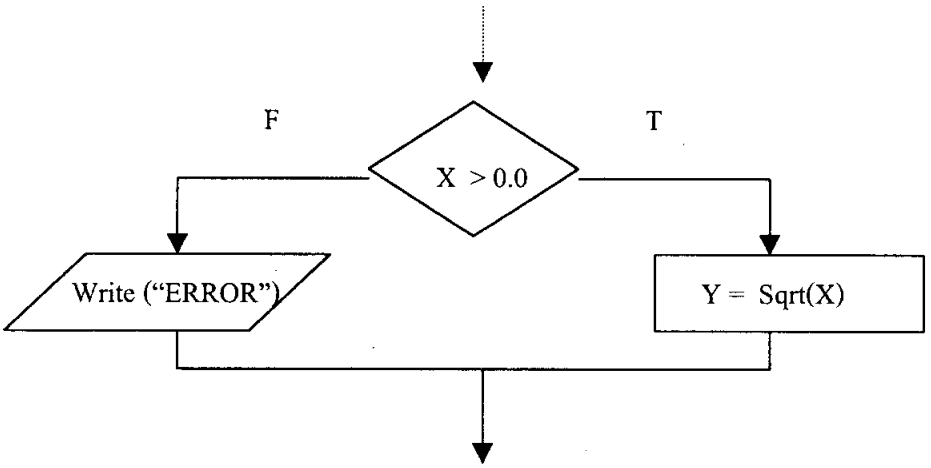
    1. Mean time between error s (MTBE) . The average up times between interruptions to service.
    2. Mean time to repair (MTTR) . After an error is found , how long does it require to get it properly corrected ?
    3. Percent up – time . What percent of the time is the software operational ?
    4. Number of bugs versus calendar months . Plot the number of bugs versus calendar months and use the slope of the curve as a rough measure of progress .

- โปรแกรมมีระบบปกป้องตัวเอง (Defensive Programming) ก็คือการใส่คุณสมบัติในการป้องกันไม่ให้เกิดการล้มเหลวในการทำงาน เช่น จะต้องมีการใส่คำสั่งการตรวจสอบข้อมูลที่มีสถานะที่อาจจะก่อให้เกิดข้อผิดพลาดในการปฏิบัติงานดังตัวอย่างเช่น

    ตัวอย่าง 4.1 Y = A / B ต้องตรวจสอบ ข้อมูล B ว่ามีค่าเป็นหรือไม่ เพราะถ้าค่า B = 0 คำสั่งนี้จะก่อให้เกิดการ Terminate โปรแกรม ดังนั้นเอาจจะปรับปรุงการดำเนินส่วนนี้เสียใหม่ดังนี้คือ



ตัวอย่าง 4.2 Y = Sqrt(X) ต้องตรวจสอบ ข้อมูล B ว่ามีค่าน้อยกว่าหรือเท่ากับ 0 หรือไม่ เพราะถ้าค่า X < 0 .0 คำสั่งนี้จะก่อให้เกิดการ Terminate โปรแกรม ดังนั้นอาจจะปรับปรุงการดำเนินส่วนนี้เสียใหม่ดังนี้คือ



ขั้นตอนถัดไปหลังจากการ Design แล้วก็คือการนำไป Implement Algorithms โดยการเขียนเป็นคำสั่งโดยใช้ภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่งซึ่งเหมาะกับการทำงานนั้นๆ ในขั้นตอนนี้การเขียนโปรแกรมอาจจะเรียกว่าการถอดรหัส (Coding) ก็ได้ การที่เราจะสามารถถอดออกมาเป็นรหัสโดยไม่ยุ่งยากได้นั้นก็จะต้องอาศัยการออกแบบโปรแกรมที่ดีและมีรูปแบบที่เรียกว่าการออกแบบที่เป็น Structure Flow

ลำดับถัดไปจากการถอดรหัสก็คือการตรวจสอบและทดสอบโปรแกรมที่เรียกว่า Debugging and Testing การ Debug หมายถึงการค้นหาที่ผิดพลาดซึ่งจะแบ่งออกเป็น 3 ประเภทดังนี้คือ

- Syntax Error คือความผิดพลาดในส่วนที่เกิดจากการเขียนคำสั่งโปรแกรมที่ขัดกับข้อกำหนดของภาษาโปรแกรมนั้นๆ อาจจะเทียบเคียงได้ง่ายๆคล้ายกับการเขียนคำสั่งที่ผิดไวยกรณของภาษาอังกฤษ เช่นการใช้ประโยคว่า "I is a boy ." แทนที่จะใช้ "I am a boy."

โดยปกติเราจะค้นหาข้อผิดพลาดเบื้องต้นในส่วนของความผิดพลาดในส่วนที่เป็น Syntax error นี้ โดยความช่วยเหลือของ Compiler ที่จะส่ง Message ความผิดพลาดนี้มาให้กับผู้ใช้ทราบเพื่อทำการแก้ไขต่อไป

ตัวอย่างของโปรแกรมที่นำไป Compile แล้วเกิดความผิดพลาด โดยที่ Compiler จะแจ้ง Message ออกมาให้ทราบ

Figure 2.15 Compiler Listing of a Program with Syntax Errors

```
  1 program Metric (Input. Output)
  2 {Converts square meters to square yards.1
  3
  4   const
*****      ^14
  5     MetersToYards : 1.196;          [conversion constant1
*****                   ^16,50
  6
  7   var
  8     SqMeter : Real:          {input - fabric size in meters1
  9
 10 begin
 11   {Head the fabric size in square meters.1
 12   WriteLn ('Enter the fabric size in square meters> ')
 13   ReadLn (SqMeters);
*****       ^6
 14
 15   (Convert the fabric size to square yards.}
 16   MetersToYards * SqMeters := SqYards;
*****                ^59          "104
 17
 18   {Display the fabric size in square yards.1
 19   WriteLn ('The fabric size in square yards is ', SqYards)
*****                                                ^104
 20 end.


  6: illegal symbol
 14:  ";" expected
 16:  "=" expected
 50:  error in constant
 59:  error in variable
104: identifier not declared
```

● Logic Error คือความผิดพลาดที่เกิดจากการทำงานที่ผิดพลาด เช่นการแสดงผล Output ออกมาผิดพลาดทั้งนี้อาจจะเนื่องมาจากผู้เขียนโปรแกรมเขียนคำสั่งผิด หรือ ใช้สูตรผิด ดังตัวอย่างต่อไปนี้

- การเขียนคำสั่งตามความต้องการโดยเข้าใจลำดับการทำงานผิด

$$Y = 3X + \frac{2}{M + 3}$$

เราเขียนคำสั่งดังนี้ในโปรแกรม Y = 3*X + 2/ M +3 โดยความต้องการเราต้องการให้เอาค่าของ M ไปบวก 3 ก่อนแล้วก่อนแล้วค่อยนำไปหาร 2 แต่ด้วยการเขียนตามคำสั่งนี้เครื่องจะทำการ คำนวณดังนี้คือ 3 * X ต่อไป 2/M ต่อไปก็คือ 3*M + 2/M +3 ซึ่งไม่ตรงตามที่เราต้องการ ทางแก้ไขก็คือเขียน Numeric Expression ดังกล่าวเสียใหม่ดังนี้ Y = 3*X + (2/(M+3)) ซึ่งนอกจะเครื่องจะทำงานได้ถูกต้องตามที่ต้องการแล้ว ยังทำให้ผู้ที่เขียนคำสั่งหรืออ่านคำสั่งเข้าใจลำดับการทำงานได้ถูกต้อง ด้วย

- การใช้สูตรคณิตศาสตร์ผิด เช่นการหาเงินรวมจากสูตร

$$Y = A \ (1 + R)^{N}$$

โดยที่ Y หมายถึงเงินรวม A หมายถึงเงินต้น R คืออัตราดอกเบี้ย และ N คือจำนวนปี ในการรับข้อมูล R นั้นเรารับอัตราดอกเบี้ยเป็น 5.5 บาท แล้วนำไปใส่ในสูตรเลย โดยที่ไม่ได้มีการแปลงข้อมูล R ให้เป็นอัตราร้อยละคือหารด้วย 100 ก่อนซึ่งส่งผลให้การคำนวณเงินรวมที่ได้ผิดพลาดจากความเป็นจริง

- การเขียนคำสั่งวน loop ผิดพลาด ทำงานให้การทำงานไม่สามารถออกจาก loop ได้ ตัวอย่างเช่น ให้เขียนโปรแกรมนำข้อมูล เงินต้น อัตราดอกเบี้ย มาคำนวณเพื่อหาว่าภายใต้ข้อกำหนดนั้นเราฝากเงินกี่ปี (N=?) จึงจะได้เงินรวมเป็น 2 เท่าของเงินต้น ซึ่งเราจะเขียน Algorithm ในส่วนของการคำนวณได้ดังนี้คือ

Get Data (A ,R)

Initial N =1 , Y =A

Do Calculate $Y = A \ (1 + R)^{N}$

Increment N with 1

While Not (Y = 2*A)

จากเงื่อนไข While (Y < = 2*A) จะพบว่า เงื่อนไข Not (Y = 2*A) นั้นอาจจะไม่มีโอกาสเกิดขึ้นทั้งนี้เพราะว่าการที่เราต้องการจะต้องได้เงินรวมเท่ากับ 2 เท่าของเงินต้นอาจจะไม่สมารถเป็นจริงได้ เช่นสมมุติว่าตามข้อกำหนดของข้อมูลที่ได้นั้นคือ A = 100 บาท เมื่อโปรแกรมคำนวณถึงปีที่ 13 ได้เงินรวมเท่ากับ 195.75 บาท และพอปีที่ 14 เราจะได้เงินรวมเท่ากับ 225.89 บาท ดังนั้น Not(Y =2*A) ณ.ปีที่13 , 14 ,... จึงยังเป็น T ซึ่งส่งผลให้โปรแกรมที่เขียนนั้นเกิดลักษณะ Infinite Loop

- Run time Error คือความผิดพลาดที่เครื่องไม่สามารถตัดสินใจดำเนินการได้ จึงต้องทำการยุติการทำงาน (Terminate) ณ.คำสั่งนั้นทันทีซึ่งเราเรียกว่า Abnormal End ตัวอย่างที่เกิดคำสั่งคือการ คำนวณโดยการหาค่า X/Y โดยที่ Y มีค่าเป็น 0 หรือใกล้ 0 มากๆ หรือการหาค่า sqrt(X) โดยที่ค่า X < 0.0 หรือการเรียก ตำแหน่งของ Array ที่เกินกว่าที่เราประกาศจองไว้ เช่นเราเขียนคำสั่งในปาสคาลว่า

```
Var
        C : ARRAY[1...10] of Integer ;


Begin
        For I := 1 to 11 Do
          Begin
              C[I] = 5 ;
              Writeln ( " I=" ,I , C[I] ) ;
```

ผลก็คือโปรแกรมจะทำงานวนจนถึงรอบที่ I มีค่าเป็น 11 แล้วเครื่องจะแจ้งว่าผิดพลาด อันเนื่องจากการระบุตัวแปร Array C[I] เกินขนาดที่เราจองไว้

ก่อนที่จะจบในบทนี้ ขอสรุปขั้นตอนในการสร้างโปรแกรม จากตัวอย่างต่อไปนี้

Program ให้นักศึกษาสร้างโปรแกรมเพื่อรับมูลค่าของความยาวของผ้าที่มีหน่วยเป็นตารางเมตรให้ออกมาเป็น Output ที่เป็นความยาวของผ้าแต่จะมีความยาวเป็นหน่วยตารางหลา

Analysis ขั้นตอนนี้คือศึกษาความต้องการว่าจะให้ดำเนินการอะไร ในที่นี้คือการรับข้อมูลเป็นตัวเลขที่เป็นตารางหลาจากแป้นพิมพ์ แล้วนำไปแปลงเป็นตารางเมตรแสดงผลออกทางจอภาพ ซึ่งการแปลงข้อมูลนี้เราจำเป็นจะต้องทราบว่า ตารางหลากับตารางเมตรมีความสัมพันธ์กันอย่างไร สมมุติว่า 1 ตารางเมตรมีค่าเท่ากับ 1.196 ตารางหลา เราจะกำหนดตัวแปรชื่อ SqMete เป็นพื้นที่ในเครื่องสำหรับจัดเก็บข้อมูลที่มีหน่วยเป็นตารางเมตร และ SqYard เป็นพื้นที่ในเครื่องสำหรับจัดเก็บข้อมูลที่มีหน่วยเป็นตารางหลา

Data Requirements

Problem Input

SqMete (the fabric size in square meters)

Problem Output

SqYard (the fabric size in square yards)

Relevant Formular

1 square meter = 1.196 square yards

Design

Formulate the algorithm that solves the problem . Begin by listing the three major step , or subprogram of the algorithm

Algorithms

1. Read the fabric size in square meters
2. Convert the fabric size to square yards
3. Display the fabric size in square yards

Algorithms with Refinement

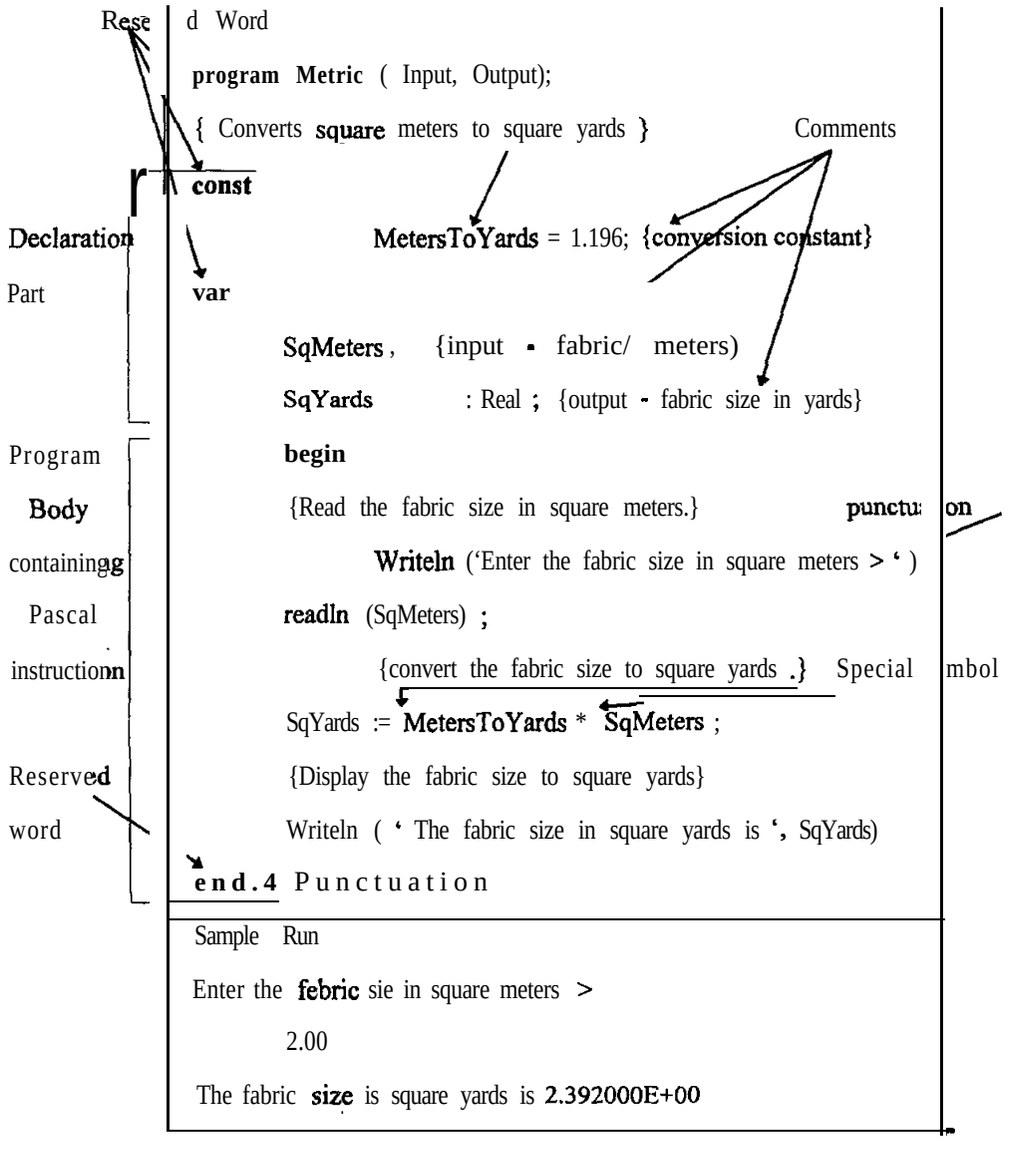
1. Read the fabric size in the square meter from keyboard
2. Convert the square size to square yards

   2.1 The fabric size in square yards is 1.196 times the fabric size in square meters.
3. Display the fabric size in square yards

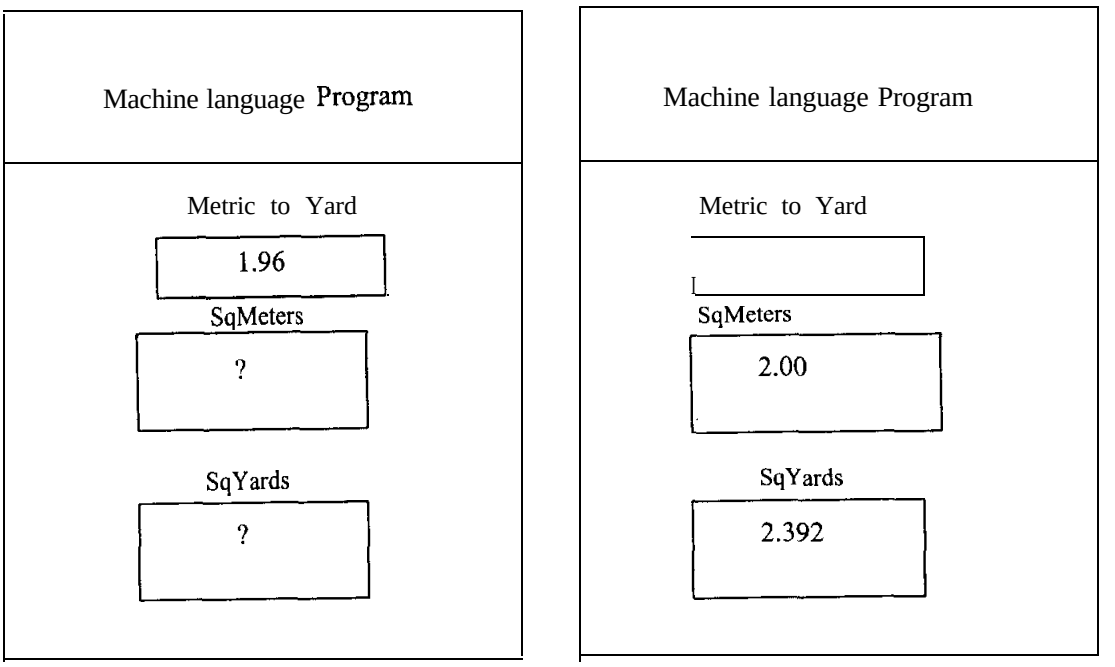
ตรวจสอบการดำเนินงานนี้โดยใช้เครื่องคิดเลขว่าถูกต้องหรือไม่ แล้วจึงนำไปทดสอบเทียบกับการทำงานด้วยโปรแกรมต่อไป

Implementation นำขั้นตอนที่กล่าวมาแล้วนี้ไปถอดรหัส เป็นโปรแกรมภาษาภาษาหนึ่งเช่นภาษาปาสคาลแล้วนำไปแปล (Compile) จากนั้นจึงทำจึงทำการ Link แล้วจึงนำไป Run ต่อไป ตรวจสอบดู Output ได้ว่าถูกต้องหรือไม่

จากตัวอย่างโปรแกรมนี้เมื่อเขียนด้วยภาษาปาสคาลแล้วจะมีลักษณะดังนี้



โดยที่ภายในสมองเครื่องจะมีลักษณะการจัดเก็บข้อมูลดังนี้



คำถามท้ายบท

1. การออกแบบโปรแกรมมีวัตถุประสงค์คืออย่างไร
2. จงอธิบายถึงขั้นตอนการออกแบบโปรแกรมตามรูปแบบของ Procedural Program
3. ทำไมจึงต้องมีการนิยามข้อมูล
4. จงอธิบายถึงความหมายคำว่าประสิทธิภาพของโปรแกรม
5. จงอภิปรายถึงคำกล่าวที่ว่า"โปรแกรมที่สั้นไม่จำเป็นจะต้องเป็นโปรแกรมที่ดี"
6. การออกแบบ นั้นจะต้องคำถึงลักษณะของ สภาพแวดล้อมด้วยเพราะเหตุใด
7. คำว่า Program Correct กับ Program Reliability แตกต่างกันอย่างไร
8. ตามคิดเห็นของท่าน ขั้นตอนใดในการออกแบบโปรแกรมเป็นขั้นตอนที่ยากที่สุด เพราะเหตุใด
9. โปรแกรมที่บางครั้งประมวลผลได้กับข้อมูลชุดหนึ่งๆ บางครั้งก็ไม่สามารถประมวลผลได้กับข้อมูลอีกชุดหนึ่งเป็นเพราะสาเหตุใดได้บ้าง และลักษณะดังกล่าวจัดเข้าข่ายความผิดประเภทใดใน 3 สาเหตุ
10. มาตราฐานของภาษาโปรแกรมหมายความว่าอะไร การทราบมาตราฐานของภาษาโปรแกรมที่เราเขียนนั้นจะส่งผลให้เกิดประโยชน์อย่างไร