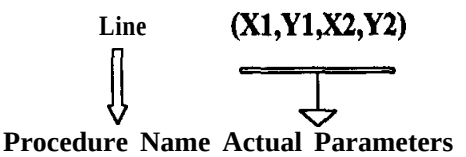
บทที่ 12
การออกแบบโปรแกรมย่อย (Modular Programming)

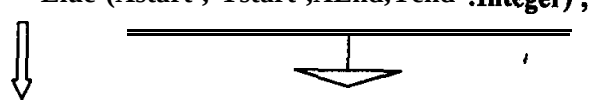
บทนำ ด้วยเหตุผลและความจำเป็นที่เราจะต้องแบ่งโปรแกรมออกเป็นส่วนๆเพื่อสะดวกในการแบ่งงานกันทำระหว่างทีมงานเขียนโปรแกรม นอกเหนือจากนั้นการแบ่งโปรแกรมออกมาเป็นโปรแกรมย่อยๆก็จะทำให้ง่ายในการออกแบบเพราะเราจะสร้างโปรแกรมย่อยนั้นเพื่อบรรลุวัตถุประสงค์เพียงประการเดียว และการเขียนโปรแกรมที่มีขนาดเล็กจะช่วยให้การตรวจสอบ/ทดสอบ (Debug and Testing) กระทำได้ง่าย โดยเฉพาะอยู่ในวิสัยที่มนุษย์สามารถทำการตรวจสอบได้ง่าย (Tracing Program) นอกเหนือจากการออกแบบและเขียนโปรแกรมย่อยนั้นแล้วสิ่งที่สำคัญยิ่งอีกประการหนึ่งที่จะต้องดำเนินการก็คือการเชื่อมต่อระหว่างโปรแกรมย่อยๆเหล่านั้นเข้าด้วยกันให้เป็นระบบโปรแกรม

Actual and Formal Parameter การเชื่อมต่อระหว่างโปรแกรมย่อยนั้นจะมีรูปแบบดังนี้คือจะมีโปรแกรมที่เป็นผู้เรียกใช้ (Caller) และโปรแกรมที่รับใช้ การจะเรียกโปรแกรมย่อยมาใช้งานนั้นจะต้องระบุ Procedure Name และบรรดา Actual Parameters ดังโครงสร้างต่อไปนี้



คำสั่งที่ปรากฏนี้จะเป็นผู้เรียกโปรแกรมย่อยชื่อ Line มาใช้งานโดยการส่งวัตถุดิบ (Actual Parameters) อันประกอบด้วย (X1,Y1,X2,Y2) ไปให้กับโปรแกรมย่อยชื่อ Line มาใช้งาน ภายหลังเมื่อโปรแกรมย่อยชื่อว่า Line ถูกเรียกโดยการส่งจุด 2 จุดคือ (X1,Y1) และ (X2,Y2) มาให้ก็จะปฏิบัติการโดยการลากเป็นเส้นผ่านจุดทั้งสองที่กำหนดให้ เราเรียกคำสั่ง**Line (X1,Y1,X2,Y2)** ว่าเป็น Call Statement โดยที่โปรแกรมย่อยที่ถูกเรียกจะมี Header Line ดังนี้คือ

Procedure Liue (Xstart , Ystart ,XEnd,Yend :Integer) ;



การส่งข้อมูลระหว่า Actual Parameter กับ Formal Parameter จะต้องกำหนด ตำแหน่ง ลำดับ และประเภทของพารามิเตอร์ทั้งสอง ให้ตรงกันมิฉะนั้นการติดต่อระหว่างสองโปรแกรมย่อยนั้นจะล้มเหลว จากตัวอย่างโปรแกรมย่อย Line การส่งต่อพารามิเตอร์จะมีรูปแบบดังนี้

Actual Parameters	Correspond	Formal Parameters
X1		Xstart
Y1		Ystart
X2		XEnd
Y2		Yend

ตัวอย่างของการเรียกโปรแกรมย่อยมาใช้งาน

```
program Main (Input, Output);
  var
    X1, X2. Y1, Y2, X3, Y3 : Integer:
  procedure Line (XStart, YStart, XEnd, YEnd : Integer);
  . . .
  end; {Line}
begin {Main}
  . .
  line (X1, Y1, X2, Y2);
  . .
end. {Main}
```

ตัวแปรแบบท้องถิ่น (Local Variable)

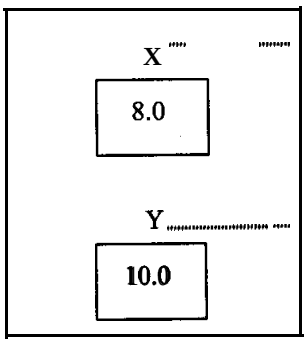
สืบเนื่องมาจากการเชื่อมโยงระหว่างโปรแกรมย่อยซึ่งจะต้องมีส่ง Actual Parameter ไปให้กับโปรแกรมย่อยรับไปในรูปของ Formal Parameter เพื่อนำข้อมูลดังกล่าวไปใช้ในการทำงาน นอกจากข้อมูล Formal Parameter ที่โปรแกรมย่อยได้รับแล้วโปรแกรมย่อยที่ถูกเรียกดังกล่าวอาจจะต้องมีการกำหนดตัวแปรเพิ่มเติมเพื่อใช้ในการประมวลผล ตัวอย่างเช่นเราเรียกโปรแกรมย่อยชื่อ Midrange มาทำงานโดยการส่งค่า X,Y (Actual Parameter) ไปให้โปรแกรมย่อยเพื่อรับไปคำนวณหาค่าผลสอบและค่าเฉลี่ยเสร็จแล้วให้แสดงผลออกมาดังตัวอย่างของโปรแกรมต่อไปนี้

```
procedure ReportSumAve (Numl ,Num2 (input) ;real);
(
        computes and displays the sum and average of Num1 and Nun12 ,
        Pre : Num1 and Num2 are assigned values
Post : The sum and average value of Num1 and Num2 are computed and
displayed.
)
var
        Sum ,
        Average : real ;
Begin
        Sum := num1 + Num2 ;
        Average := Sum / 2.0 ;
        Writeln(' The sum is', Sum :4:2);
        Writeln('The average is ', Average :4:2)
End; (ReportSumAve)
```

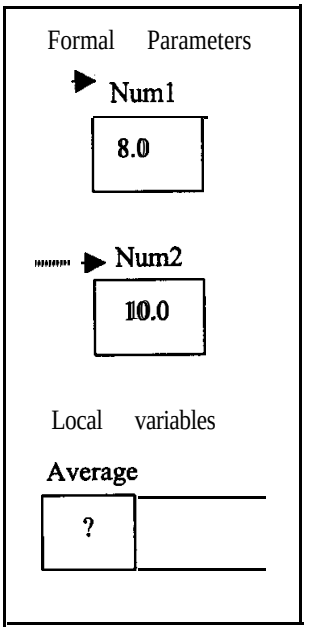
Actual Parameter	Corresponds to	Formal Parameter
X		Num1
Y		Num2

Data Areas After Call of ReportSumAve

Main program data area



ReportSumAve data area



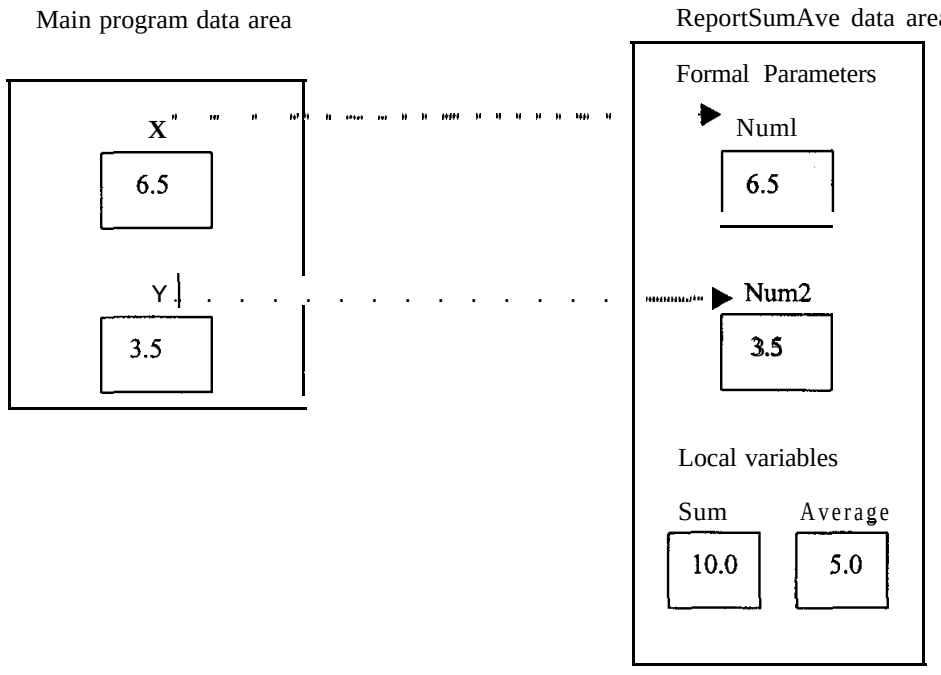
เช่นเรียกโปรแกรมย่อยมาใช้งาน เช่นเรียก ReportSumAve (6.5 ,3.5) ผลที่ได้ก็คือ

Actual Parameter	Correspond to	Formal Parameter
6.5		Num1
3.5		Num2

ดังนั้นภายในโปรแกรมย่อย ReportSumAve จะมีตัวแปรท้องถิ่น (Local Variable) สร้างขึ้นเพื่อช่วยในการทำงานคือ Sum , Average

ภาพต่อไปนี้จะแสดงการติดต่อระหว่างโปรแกรมหลักกับโปรแกรมย่อย

Data Areas After Call of ReportSumAve



การส่งพารามิเตอร์ผิดพลาด (Illegral Parameter Substitution Errors) ขั้นตอนการ Compile โปรแกรมนั้น Compiler จะทำการตรวจสอบระหว่าง Actual Parameter List กับ Formal Parameter List ว่ามีความขัดแย้งกันหรือไม่เช่น จำนวนพารามิเตอร์ ต้องเท่ากัน และแต่ละพารามิเตอร์ที่เชื่อมต่อกันจะต้องเป็นประเภทและขนาดเดียวกัน ในกรณีที่เกิดความผิดพลาดการ Compile จะไม่ผ่าน และ Compiler จะแจ้งข้อความผิดพลาดให้ปรากฏ

การส่งข้อสารสนเทศกลับจากโปรแกรมย่อย (Returning Information form Procedure)

การเรียกโปรแกรมย่อยมาทำการประมวลผลข้อมูลให้ผู้เรียกนั้นจะมีการส่งกลับมาหลายลักษณะดังนี้

- เรียกโปรแกรมย่อยมาใช้งานโดยภายหลังที่ดำเนินการในโปรแกรมย่อยเสร็จจะไม่มีผลิตผลใดๆกลับไปยังผู้เรียก ตัวอย่างเช่นการส่งข้อความไปให้โปรแกรมดำเนินการออกเครื่องพิมพ์
- เรียกโปรแกรมย่อยโดยการไม่ส่งข้อมูลใดๆไปให้ปฏิบัติการในโปรแกรมย่อย และโปรแกรมย่อยก็ปฏิบัติงานเอง โดยไม่ส่งผลใดๆกลับมายังผู้เรียก ตัวอย่างเช่นการพิมพ์ หัวเรื่องของงานออกเอกสาร
- เรียกโปรแกรมย่อยโดยการส่งข้อมูลไปให้ปฏิบัติการ ภายหลังการดำเนินงานในโปรแกรมย่อยเสร็จจะส่งสารสนเทศกลับมาเพียงรายการเดียว เช่นการส่ง Actual Parameter List คือ เงินต้น อัตราดอกเบี้ย และจำนวนปีที่ฝากเงิน โปรแกรมย่อยจะนำไปคำนวณหาเงินรวมและส่งกลับมาให้ผู้เรียก ลักษณะนี้ในภาษาปาสคาลจะตั้งชื่อเรียกโปรแกรมย่อยนั้นเป็นชื่อเฉพาะว่า Function (ภาษาซีจะเรียกชื่อโปรแกรมย่อยทุกประเภทว่า Function ในขณะที่ภาษาฟอร์แทรน จะเรียกชื่อโปรแกรมย่อยว่า Subroutine และถ้ามีลักษณะการส่งสารสนเทศกลับไปยังผู้เรียกเพียงหนึ่งรายการจะเรียกโปรแกรมย่อยนั้นว่า Function เช่นเดียวกับภาษาปาสคาล เป็นต้น)
- เรียกโปรแกรมย่อยโดยการส่งข้อมูลไปให้ปฏิบัติการ ภายหลังการดำเนินงานในโปรแกรมย่อยเสร็จจะส่งสารสนเทศกลับมามากว่าหนึ่งรายการ เช่นการส่งอะเรย์ไปให้โปรแกรมย่อยทำการเรียงลำดับ ภายหลังการทำงานเสร็จโปรแกรมย่อยจะส่งอะเรย์ที่เรียงลำดับแล้วกลับมายังผู้เรียก

การติดต่อระหว่างโปรแกรมที่เรียกกับโปรแกรมที่ถูกเรียกในกรณีที่มีผลิตผลส่งกลับมาให้ผู้เรียกนั้น เราจะเขียนรูปแบบได้ดังนี้

Procedure Line (X , Y , Z)

Input Parameters Output Parameter

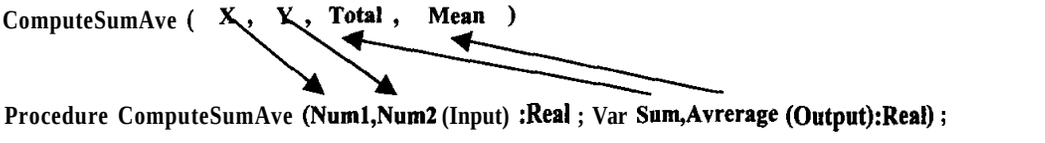
ตัวอย่างการคำสั่งในการติดต่อระหว่างโปรแกรมที่เรียกโปรแกรมย่อยมาใช้งานในภาษาปาคาล

ComputeSumAve (X , Y , Total , Mean)

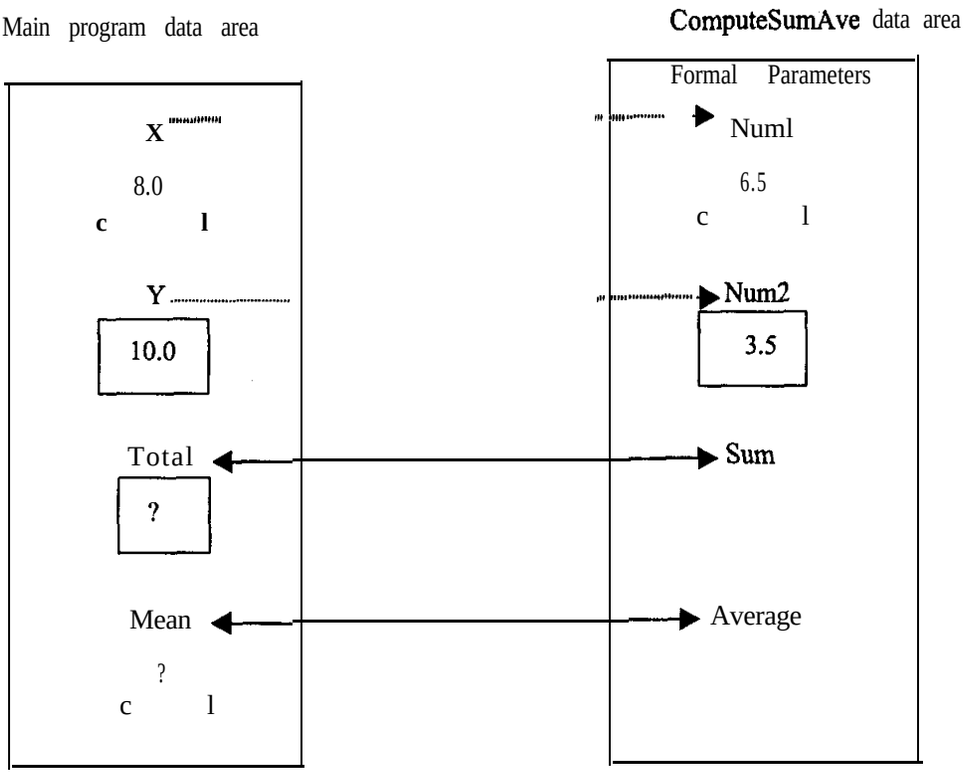
และคำสั่งของโปรแกรมย่อยที่ถูกเรียกจะปรากฏดังนี้

Procedure ComputeSumAve (Num1,Num2 (Input) :Real ; Var Sum,Avrerage (Output):Real) ;

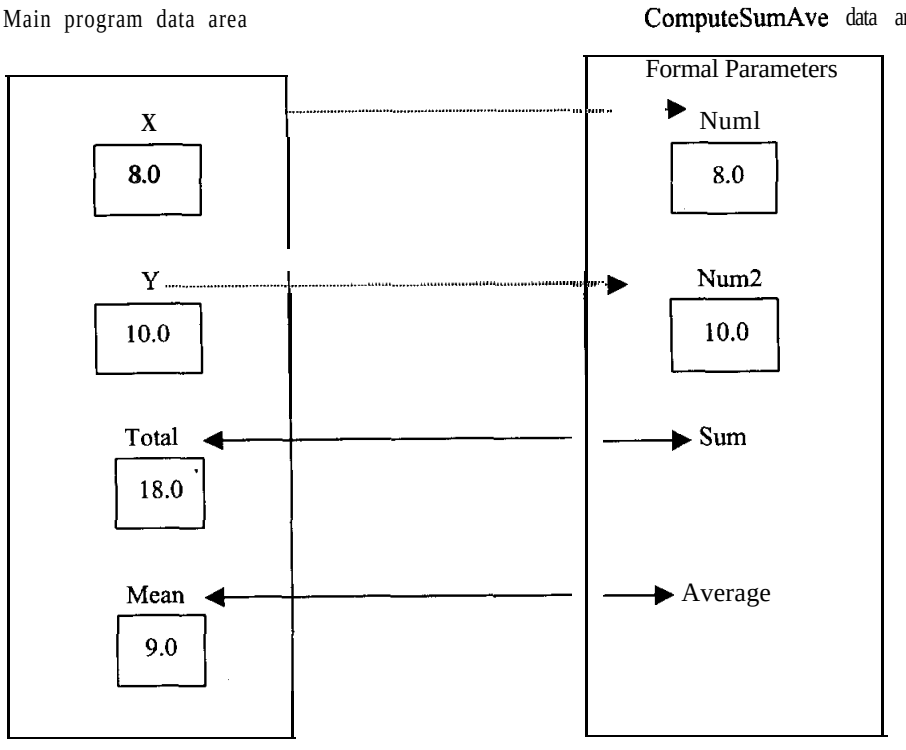
ลักษณะการติดต่อจะปรากฏดังนี้



Data Area After Procedure Call



Data Area After Procedure Execution



ในภาษาปาสคาลนั้น Input Parameters จะเรียกว่า Value Parameter ส่วน Output Parameters นั้นจะเรียกว่า Variable Parameter โดยที่ Variable Parameter จะมีการเปลี่ยนแปลงค่าทุกครั้งที่โปรแกรมย่อยนั้นถูกเรียกให้ดำเนินการ ลักษณะการติดต่อระหว่าง Actual Parameter และ Formal Parameter นั้นจะแบ่งออกเป็น 2 ลักษณะคือ

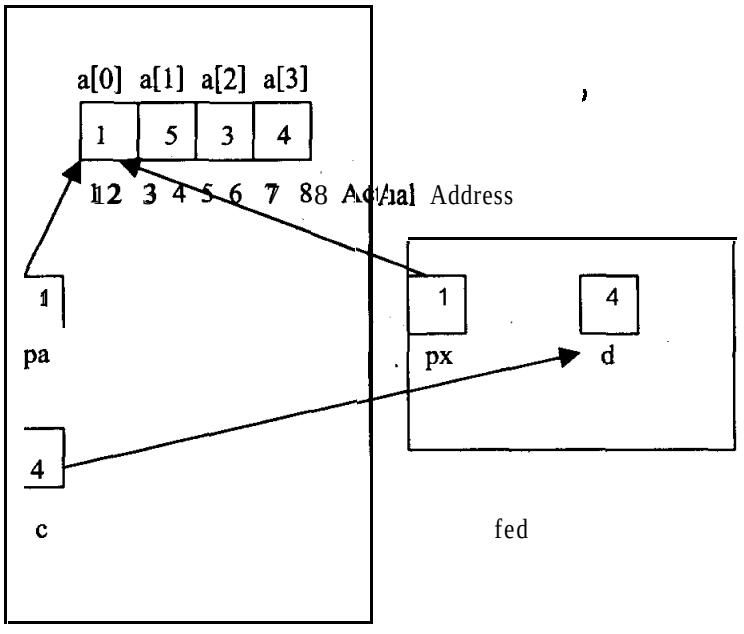
1. การ Pass by Value ซึ่งเป็นการส่งค่าไปให้โปรแกรมย่อยโดยการทำสำเนาไปเก็บในตัวแปรที่โปรแกรมย่อยประกาศไว้ที่ Formal Parameter การส่งข้อมูลโดยวิธีนี้นั้นจะมีข้อดีคือการทำสำเนาส่งไปเป็นการป้องกันความปลอดภัยของตัวแปรที่เป็น Actual Parameter ไม่ให้เสียหายจากการทำงานของโปรแกรมย่อย แต่จะมีข้อเสียในส่วนที่ว่าการทำสำเนาไปเก็บที่ตัวแปรที่เป็น Formal Parameter ทำให้พื้นที่ในสมองเครื่องต้องสูญเสียไปเพื่อทำหน้าที่ในการจัดเก็บ วิธีนี้จะไม่ค่อยเหมาะสมกับการส่งอะเรย์ขนาดใหญ่
2. การ Pass by Reference (Pass by Address) การส่งข้อมูลระหว่าง Actual Parameter และ Formal Parameter นั้นจะมีลักษณะเป็น Logical Pass คือไม่ได้ส่งจริงเพียงแต่แจ้งว่าข้อมูลอยู่ที่ตำแหน่งใด (Actual Address) การดำเนิน

การส่งข้อมูลโดยวิธีนี้ เราจะต้องระมัดระวังไม่ให้การทำงานในโปรแกรมย่อย ก่อให้ข้อมูลที่เป็น Actual Parameter เกิดเสียหาย

ตัวอย่างการติดต่อโดยวิธีที่ 2 โดยใช้ภาษาซี

```
#include < stdio.h>
void main()
{
static int a[] = { 1,5,3,4} ;
int  *pa,  y;
int c = 4;
pa=  a;
y = fed (pa ,c);
printf("%d ",*y) ;
}
fed (px,d)
int *px, d ;
{
        int sum =0 ;
        for (int i =0 ; i < d ; i++)
                { sum+=  *px;
                        px++ ;
                 }
        return  (sum)  ;
{
```

ภาพการส่งแบบ Pass by Reference จะปรากฏดังนี้



main

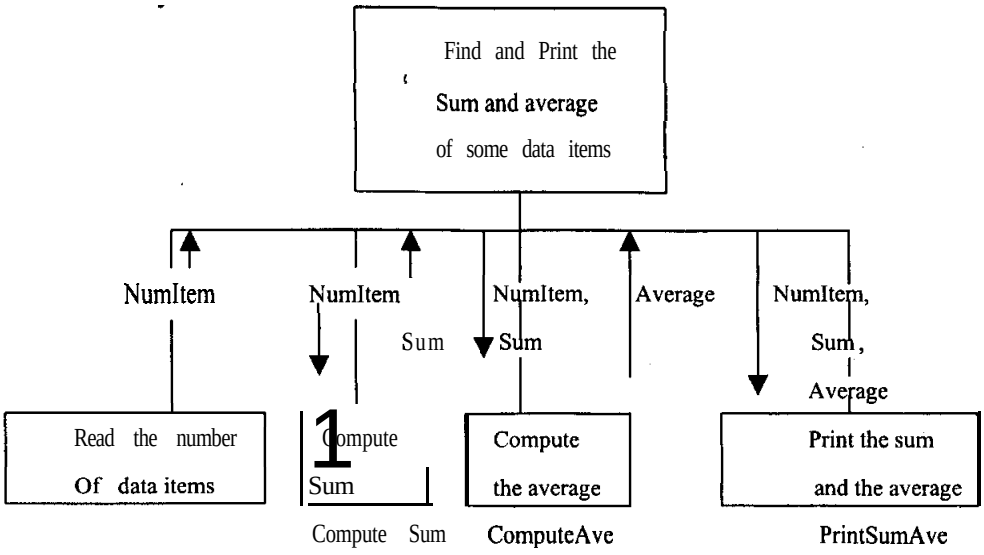
กฎในการติดต่อระหว่าง Actual Parameter กับ Formal Parameter

1. Correspondence between actual and formal parameters is determined by position in their respective parameters lists .These lists must be the same size , although the names of corresponding actual and parameters may differ

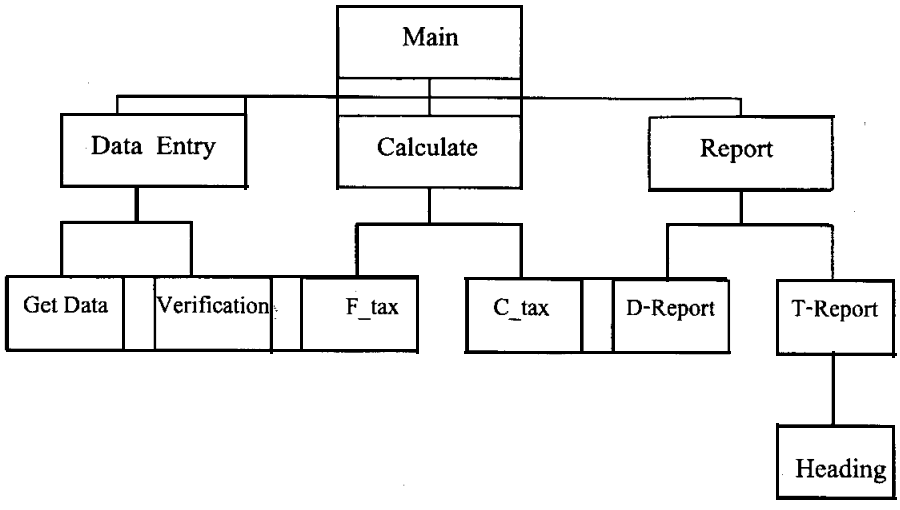
2. For variable parameters , the types of corresponding actual and formal parameters must be identical . For value parameters , the actual parameter must be assignment compatible with its corresponding formal parameter .
3. For variable parameters ,an actual parameter must be a variable . For value parameters , an actual parameter may be a variable , a constant , or an expression.

ภาพที่ 12.1 แสดงการออกแบบโปรแกรมแบบโมดุล

Structure Chart with Data Flow Information



การทดสอบโปรแกรม (System Testing) เนื่องจากรูปแบบของการอออกแบบระบบโปรแกรมขนาดใหญ่นั้นจะเป็นแบบ Top Down Design



การทดสอบนั้นในส่วนของการ Interface เราจะทดสอบตามรูปแบบของ Top Down ส่วนการทดสอบการทำงานของโปรแกรมย่อยแต่ละโมดุลนั้นจะใช้วิธีการทดสอบแบบ Bottom Up

กระบวนการช่วยเหลือในการตรวจสอบหาข้อผิดพลาดของโปรแกรม

(Debugging Tips for Modular Program)

- ให้เขียนข้อความกำกับ (Comment) ร่วมกับการเขียนโปรแกรมด้วย รวมทั้งการเขียนข้อความกำกับในการอธิบายถึงกรรมวิธีการทำงานของโปรแกรมย่อยด้วย เพื่อจะช่วยเหลือผู้ตรวจสอบ
- การกวาดตรวจคำสั่งต่างๆในโปรแกรมนั้น ให้คำเนินการสั่งให้เสนอผลโดยการใช้คำสั่ง Writeln โดยเฉพาะอย่างยิ่งในระหว่างที่มีการเชื่อมต่อระหว่างโปรแกรมหลักกับโปรแกรมย่อยก็ให้แสดงข้อความออกมาให้ผู้ใช้ทราบว่ากำลังอยู่ในการติดต่อกับโปรแกรมย่อย รวมทั้งการแสดงสถานะว่ากำลังทำงานอยู่ในโปรแกรมย่อยด้วย
- การสั่งให้แสดงผลข้อมูลที่เกิดขึ้นจากการทำงาน โดยที่ข้อมูลที่จะแสดงผลก็คือบรรดา input /output parameters โดยการตรวจสอบว่าข้อมูลเหล่านี้ถูกต้องหรือไม่
- ตรวจสอบว่าโปรแกรมย่อยต่างๆที่เรียกมาใช้งานนั้นให้ผลถูกต้องหรือไม่
- ในกรณีที่ ซอฟต์แวร์ที่ใช้นั้นมี ส่วนของ Debugger ก็ให้นำมาใช้ให้เกิดประโยชน์

ตัวอย่างของการตรวจสอบโปรแกรม

```
const
    Debug = True ; (Turn debugging on.)
During debugging mns , and the declaration
const
    Debug = False ; (Turn debugging off.)
During normal runs .
```

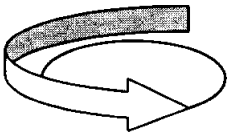
โดยที่โปรแกรมที่จะเขียนในส่วนของการตรวจสอบนั้นจะคำเนินการเขียนดังนี้

```
If Debug then
    Begin
        Writeln ('Procedure ComputeSum entered ');
        Writeln('Input parameter NumItems has value '.NumItems :1)
    End ; (If)
```

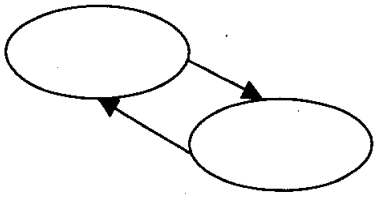
โปรแกรมย่อยชนิดเรียกตัวเอง (Recursive Function)

โปรแกรมย่อยลักษณะนี้อาจจะจำแนกออกเป็น 2 ประเภทคือ

1. **Direct Recursive เรียกผ่านตัวโดยตรง**



2. **Indirect Recursive เรียกผ่านโปรแกรมย่อยผ่านตัวกลางอื่นก่อน**



ตัวอย่างการเขียนโปรแกรมแก้ปัญหาโดยการใช้รูปแบบของ Recursive

- การหาค่าของ N! เช่น

  N! = 1 for N =0 ,1

  5 ! = 5 * 4 * 3 * 2 *1

  เขียนเป็น Recursive Form ได้ดังนี้

  N*(N-1)! ; For N >1

  เขียนเป็นคำสั่งแบบ Recursive ได้ดังนี้

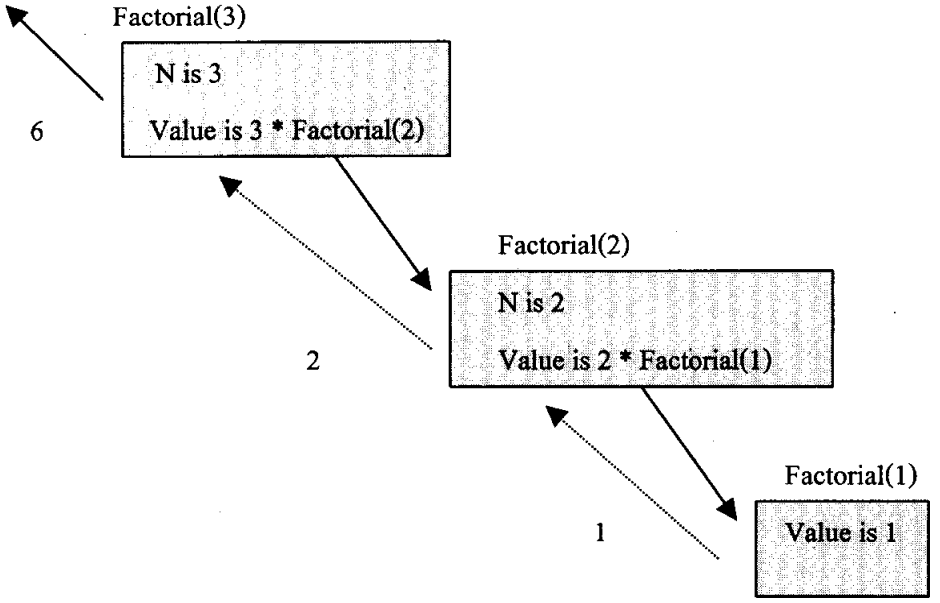
  Factorial := N* Fact(N-1) ;

Recursive Function Factorial

```
Function Factorial ( N : Integer ): Integer ';
(
        Computes N!
        Pre : N >=0
        Post : Returns the product 1 * 2* 3*...*N  for N >1 ;
                Returns 1 when N is 1 or 0 .
)
```

```
begin (Factorial)
        If N <= 1 then
                Factorial :=1
        Else
                Factorial := N * Factorial(N-1)
        End ; (Factorial )
```

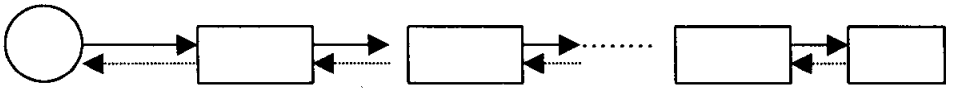
ภาพที่ 12.1 แสดงการคำนวณหาค่า Factorial ตามรูปแบบของ Recursive



- การหาค่าของเงินรวมปีที่ N เมื่อคิดเงินแบบดอกเบี้ยทบต้น

  เงินต้นปีที่ (N-1) + (เงินต้นปีที่ (N-1))* อัตราดอกเบี้ย

เงินรวมปีที่ N เงินรวมปีที่ N-1 เงินรวมปีที่ N-2 .....เงินรวมปีที่ 2 เงินรวมปีที่ 1



■ การแก้ปัญหาของ Polynomial Degree N

$$Y = a + bx + cx^2 + dx^3 + e x^4 + \ldots + zx^N$$

กำหนดให้ N = 3 เพื่อให้เห็นภาพการใช้ Recursive Form

$$Y = a + bx + cx^2 + dx^3$$

$$Y = a + [b + cx + dx^2]x$$

$$Y = a + [\, b + [c + dx]\, x\,]\, x$$

คำถามท้ายบท

1. จงเขียนโปรแกรมย่อย (Procedure) เพื่อรับ พารามิเตอร์ อันประกอบด้วยจำนวน Space เป็นพารามิเตอร์แรกเพื่อทำหน้าที่เป็นช่องว่างในการพิมพ์สำหรับหัวตาราง แรก ส่วนพารามิเตอร์ที่สองจะเป็นข้อความที่จะพิมพ์ ส่วนพารามิเตอร์ตัวที่สามใช้สำหรับสั่งจำนวนครั้งในการพิมพ์
2. จงอธิบายถึงข้อดีข้อเสียของการใช้กรรมวิธีของการส่งผ่านข้อมูลแบบ Pass by Value
3. จงออกแบบโปรแกรมย่อยเพื่อรับ Input Parameter เลขจำนวนเต็ม 2 จำนวนเพื่อหาค่าความแตกต่างแล้วพิมพ์ออกทางจอภาพ
4. ค่ารากที่สองของ N จะประมาณด้วยการใช้วิธีการคำนวณซ้ำๆ โดยใช้สูตรต่อไปนี้

   NG = 0.5 (LG + N/LG)

   โดยที่ NG : ค่าที่เคารอบที่ K

   LG : เป็นค่าที่เคารอบที่ K-1

   LG จะเป็นค่าเริ่มต้นที่กำหนดในการไปใช้งาน การดำเนินการให้เขียนเป็นโปรแกรมย่อยและเลือกค่า LG ที่เหมาะสมเอง
5. จงออกแบบโปรแกรมย่อยในการรับข้อมูลที่เป็นคะแนนดิบ จากผู้เรียกมา แล้วดำเนินการตัดเกรดนักศึกษาตามเกณฑ์ต่อไปนี้คือ

   80 - 100 ได้เกรด 'A'

   75 - 79 ได้เกรด 'B'

   65 - 74 ได้เกรด 'C'

   55 - 64 ได้เกรด 'D'

   < 55 ได้เกรด 'F'