บทที่ 8
โปรแกรมย่อย (Subprogram)

แบบฝึกหัดที่ 8

บทที่ 8
โปรแกรมย่อย (Subprogram)

8.1 ฟังก์ชันภายใน (Intrinsic function)

ฟังก์ชันภายในคือโปรแกรมที่ทำการคำนวณหาค่าฟังก์ชันต่าง ๆ และฟังก์ชันเหล่านี้จะให้มาพร้อมกับคอมไพเลอร์ของภาษาฟอร์แทรน ฟังก์ชันภายในนี้บางครั้งอาจถูกเรียกว่า บิลท์อินฟังก์ชัน (Built-in function) หรือฟังก์ชันไลบรารี (Library function)

ตัวอย่างฟังก์ชันภายใน เช่น

ฟังก์ชันทางคณิตศาสตร์

- ฟังก์ชันในการหารากที่ 2 ของเลขจำนวนจริง ซึ่งชื่อฟังก์ชันที่ใช้ทั่ว ๆ ไปคือ SQRT ความหมายของ SQRT(X)คือ $\sqrt{X}$ นั่นเอง
- ฟังก์ชันในการหาค่าสัมบูรณ์ของเลขจำนวนจริง ซึ่งชื่อฟังก์ชันที่ใช้ทั่วไปคือ ABS ความหมายของ ABS(X) คือ $|X|$ นั่นเอง

ฟังก์ชันทางตรีโกณมิติ

- ฟังก์ชันการหา sine ของมุมที่วัดเป็นเรเดียน เช่น SIN(X)
- ฟังก์ชันการหาค่า Cosine ของมุมที่วัดเป็นเรเดียน เช่น COS(X)

ในภาษาฟอร์แทรน 77 นอกจากฟังก์ชันทางคณิตศาสตร์และฟังก์ชันทางตรีโกณมิติแล้ว ยังมีฟังก์ชันของข้อมูลอักขระ เช่น ฟังก์ชัน LEN, INDEX, ICHAR และ CHAR เป็นต้น ซึ่งใช้ในการจัดการกับข้อมูลอักขระ

ตัวอย่างการใช้ฟังก์ชันเลขคณิตเป็นตัวถูกกระทำทางเลขคณิตเช่นเดียวกับตัวแปร

```
Y =   SQRT(X)        อาร์กิวเมนต์ต้องอยู่ในเครื่องหมายวงเล็บ
      ชื่อฟังก์ชัน   อาร์กิวเมนต์ซึ่งอาจเป็นนิพจน์เลขคณิตได้
```

ค่าของฟังก์ชันถูกเก็บในตัวแปร Y

ความหมายทางคณิตศาสตร์คือ $y=\sqrt{x} = x^{1/2}$

```
Z=1.25*SQRT(X)+20.5
```

ใช้ฟังก์ชันเป็นตัวถูกกระทำเลขคณิต

ความหมายทางคณิตศาสตร์คือ $z = 1.25\sqrt{x} + 20.5$

ชนิดของอาร์กิวเมนต์นั้นขึ้นอยู่กับฟังก์ชันที่ใช้ ตัวอย่างเช่นฟังก์ชัน SQRT นั้นใช้อาร์กิวเมนต์ที่เป็นเลขจำนวนจริงหรือนิพจน์เลขคณิตซึ่งมีค่าเป็นเลขจำนวนจริง ฟังก์ชันอาจเป็นส่วนของนิพจน์เลขคณิตได้ นั่นคือเป็นตัวถูกกระทำในนิพจน์เลขคณิตดังตัวอย่างข้างต้น

รูปทั่วไปของการเรียกใช้ฟังก์ชันคือ

```
ชื่อฟังก์ชัน(อาร์กิวเมนต์[,...])
```

โดยที่ ชื่อฟังก์ชัน คือชื่อฟังก์ชันภายในที่เราต้องการเรียกใช้ (ดูจากคู่มือการเขียนภาษาฟอร์แทรนของคอมพิวเตอร์แต่ละระบบ)

อาร์กิวเมนต์ (ต้องมีอย่างน้อย 1 ตัว) ซึ่งอาจเป็นค่าคงที่ ตัวแปร การเรียกใช้ฟังก์ชันอื่น หรือนิพจน์ ในกรณีที่อาร์กิวเมนต์เป็นนิพจน์ คอมพิวเตอร์จะคำนวณหาค่า 1 ค่าของนิพจน์ก่อนแล้วส่งไปเป็นอาร์กิวเมนต์ของฟังก์ชันนั้น ๆ

Function	*Definition (a represents the argument)*	*Number of arguments*	*Name*	*Mode of Argument*	*Value of function*
Absolute value	|a| = a if a ≥ 0 −a if a < 0	1	IABS ABS DABS CABS	Integer Real Double precision Complex	Integer Real Double precision Real
Square root	$\sqrt{a}$, a ≥ 0	1	SQRT DSQRT CSQRT	Real Double precision Complex	Real Double precision Complex
Exponential	e^a note e = 2.718···	1	EXP DEXP CEXP	Real Double precision Complex	Real Double precision Complex
Natural logarithms	ln(a), a > 0	1	ALOG DLOG CLOG	Real Double precision Complex	Real Double precision Complex
Common Logarithms	$\log_{10}$(a), a > 0	1	ALOG10 DLOG10	Real Double precision	Real Double precision
Sine	sin(a) a is expressed in radians	1	SIN DSIN CSIN	Real Double precision Complex	Real Double precision Complex
Cosine	cos(a) a is expressed in radians	1	COS DCOS CCOS	Real Double precision Complex	Real Double precision Complex
Tangent	tan(a) a is expressed in radians	1	TAN DTAN	Real Double precision	Real Double precision

ตารางแสดงฟังก์ชันภายใน (ทางคณิตศาสตร์) ในภาษาฟอร์แทรน

ตัวอย่าง คำนวณค่านิพจน์ $\sqrt{a^2 + b^2}$ เก็บไว้ใน Y ถ้าค่าของ |Y| อยู่ในเรนจ์ -.01<Y<.01 ให้ไปทำคำสั่งที่ 5

```
Y=SQRT(A**2+B**2)
IF(ABS(Y).LT.0.01)GO TO 5
```

ตัวอย่าง คำนวณค่า $\cos^2 x + ce^{-\sin x}$ แล้วปัดเศษทิ้ง I เก็บจำนวนเต็มใน I

```
I=COS(X)**2+C*EXP(-SIN(X))
```

ตัวอย่าง คำนวณค่า ln(R), R ต้องมีค่ามากกว่า 0 แล้วเก็บค่า ln(R) ใน Q

```
IF(R.LE.0)GO TO 8
Q=ALOG(R)
```

ตัวอย่าง การใช้ฟังก์ชันคณิตศาสตร์ที่ไม่ถูกต้อง

```
I=IABS(R)    อาร์กิวเมนต์ของฟังก์ชัน IABS ต้องเป็นชนิด integer
K=SQRT(3)    อาร์กิวเมนต์ของฟังก์ชัน SQRT ต้องเป็นชนิด real
             ดังนั้นควรแก้เป็น  K=SQRT(3.)
```

Function	*Definition (a_1, a_2 represent arguments)*	*Number of arguments*	*Name*	*Mode of Argument*	*Value of function*
Mode conversion	Conversion to integer	1	INT or IFIX IDINT	Real Double precision	Integer Integer
	Conversion lo real	1	FLOAT SNGL REAL	Integer Double precision Complex	Real Real Real
	Conversion to double precision	1	DFLOAT DBLE	Integer Real	Double precision Double precision
Transfer of sign	$\lvert a_1 \rvert$ if $a_2 > 0$ $-\lvert a_1 \rvert$ if $a_2 < 0$	2	ISIGN SIGN DSIGN	Integer Real Double precision	Integer Real Double precision
Positive difference	$a_1 - a_2$ if $a_1 > a_2$ 0 if $a_1 \leq a_2$	2	IDIM DIM DDIM	Integer Real Double precision	Integer Real Double precision
Choose the largest value	The largest of $(a_1, a_2, \cdots)$	≥ 2	MAX0 AMAX1 DMAX1 AMAX0 MAX1	Integer Real Double precision Integer Real	Integer Real Double precision Real Integer
Choose the smallest value	Thr smallest of $(a_1, a_2, \cdots)$	≥ 2	MIN0 AMIN1 DMIN1 AMIN0 MIN1	Integer Real Double precision Integer Real	Integer Real Double precision Real Integer
Remainder function	Remainder of the division of a_1 by a_2 $r = a_1 - INT(a_1/a_2) * a_2$	2	MOD AMOD DMOD	Integer Real Double precision	Integer Real Double precision

ตารางแสดงฟังก์ชันภายในอื่น ๆ ในภาษาฟอร์แทรน

ตัวอย่าง ถ้า A=3. และ B=2.

แล้ว J=IFIX(A/B)+IFIX(A/4.)=1+0=1

ตัวอย่าง ฟังก์ชันในการหาค่าสูงสุดและต่ำสุดนั้นต้องมีอาร์กิวเมนต์อย่างน้อย 2 ตัว

ถ้า J=3 และ K=-2

การใช้ฟังก์ชัน	ค่าของฟังก์ชัน
MIN0(3,9,7,-1,4)	-1
MAX1(3.,9.,7.,-1.,4.)	9
AMIN0(J,K,J*K,J+K)	-6.
MAX0(J,K,J*K,J+K,14)	14

ตัวอย่าง การเปลี่ยนเครื่องหมายและฟังก์ชันหา positive difference นั้นต้องใช้อาร์กิวเมนต์ 2 ตัว

การใช้ฟังก์ชัน	ค่าของฟังก์ชัน
ISIGN(3,-2)	-3
ISIGN(3,2)	3
SIGN(-3.,-2.)	-3.
ISIGN(-3,2)	3
IDIM(3,-2)	5
DIM(3.,2.5)	5
IDIM(-3,-2)	0
IDIM(-3,2)	0

8.2 แนวความคิดเรื่องโปรแกรมย่อย

เมื่อเวลาเราจะเขียนโปรแกรมที่เราต้องการใช้มากกว่า 1 ครั้งในโปรแกรมหนึ่ง หรืออาจจะเป็นประโยชน์ต่อโปรแกรมอื่น ๆ ตัวอย่างเช่นสมมุติว่าเรากำลังเขียนโปรแกรมที่จะประมวลผลข้อมูลเกี่ยวกับบุคลากรของบริษัทแห่งหนึ่ง โดยจะพิมพ์รายงานซึ่งประกอบด้วย วันเกิดของพนักงาน วันที่เข้าทำงาน วันที่ได้ขึ้นเงินเดือนครั้งสุดท้าย วันที่ที่ลาป่วย เราต้องการรายงานที่มีวันที่อยู่ในรูปของ 'MONTH,DAY,YEAR' (เช่น NOVEMBER,23,1952) แต่ข้อมูลที่เราจะอ่านจะอยู่ในรูป 'DDD/YY' โดยที่ DDD เป็นลำดับวันในปีหนึ่งจาก 1 ถึง 366 แต่ละครั้งเราต้องพิมพ์วันที่ในรายงาน เราต้องการโปรแกรมที่จะแปลงลำดับที่ของวันไปเป็นเดือนและวันที่ เราจะต้องทำการแปลงถึง 4 ครั้งในโปรแกรม แต่ละครั้งเป็นการปฏิบัติโดยใช้ตรรกะเดียวกันกับข้อมูลที่ต่างกัน (ในโปรแกรมอื่น ๆ ซึ่งประมวลผลข้อมูลของลูกจ้าง เราอาจต้องการสิ่งที่ทำในลักษณะเดียวกันนี้) จะสะดวกถ้าเราเขียนส่วนนี้ของโปรแกรมเพียงหนเดียว และใช้ หลาย ๆ หนในโปรแกรมเมื่อต้องการ หรืออาจใช้ในโปรแกรมอื่นถ้าจำเป็น ทั้งนี้เราสามารถทำได้โดยการใช้โปรแกรมย่อย

8.2.1 การกำหนดโปรแกรมย่อย

เมื่อใดก็ตามที่เราเขียนโปรแกรม เรากำลังสร้าง "เครื่องมือ" ที่เราอาจใช้เพื่อทำหน้าที่อย่างหนึ่งที่ระบุไว้ เหมือนกับเครื่องมืออื่น ๆ ที่เราใช้ ซึ่งจะปฏิบัติหน้าที่เมื่อเราสั่งให้ทำ และดังนั้นเราสามารถคิดว่ามันไม่เป็นอิสระจากตัวเรา เพราะเราเป็นผู้กำหนดการทำงานนั้น แต่เป็นอิสระกับเครื่องมืออื่น ๆ ที่เราอาจใช้เพื่อวัตถุประสงค์อื่น (เช่นเดียวกับที่ค้อนกับเลื่อยเป็นอิสระต่อกัน อย่างไรก็ดีเราอาจใช้ของทั้งคู่ในงานชิ้นเดียวกัน) ในที่สุดโปรแกรมของเราเป็นโปรแกรมที่มีความสมบูรณ์ในตัวเอง (self-contained) เนื่องจากว่าได้บรรจุข้อมูลและตรรกะทั้งหมดที่จำเป็นในการทำหน้าที่ไว้แล้ว

ในแนวคิดเดียวกัน โปรแกรมย่อยจะทำงานเพียงเมื่อโปรแกรมควบคุม (controlling program) ซึ่งเรามักเรียกว่าโปรแกรมหลัก (Main program) บอกให้ทำเช่นนั้น และเป็นอิสระกับโปรแกรมย่อยอื่นซึ่งอาจถูกใช้โดยโปรแกรมหลักเดียวกัน นอกไปจากนั้นยังสมบูรณ์ในตัวเองและเป็นโปรแกรมที่สมบูรณ์ซึ่งจะกระทำกิจกรรมที่ระบุไว้

8.2.2 เมื่อใดจึงควรใช้โปรแกรมย่อย

ในหัวข้อก่อนได้อธิบายถึงลักษณะของโปรแกรมย่อย เราจะใช้โปรแกรมย่อยเมื่อโปรแกรมหลักที่เรากำลังเขียนอยู่ต้องการการกระทำเหล่านั้น โปรแกรมย่อยเป็นเครื่องมือที่ใช้การได้ทันทีเมื่อเราต้องการการกระทำในโปรแกรมย่อยนั้น ๆ เราต้องมีวัตถุประสงค์เดียว (เช่น แปลงวันที่จากรูปหนึ่งไปอีกรูปหนึ่ง) ซึ่งเราต้องการจะใช้มากกว่า 1 ครั้งในโปรแกรม ข้อกำหนดว่าต้องมีวัตถุประสงค์เดียวนี้มักทำให้โปรแกรมย่อยของเราส่วนมากค่อนข้างสั้น (ปกติ 50 บรรทัดหรือน้อยกว่า)

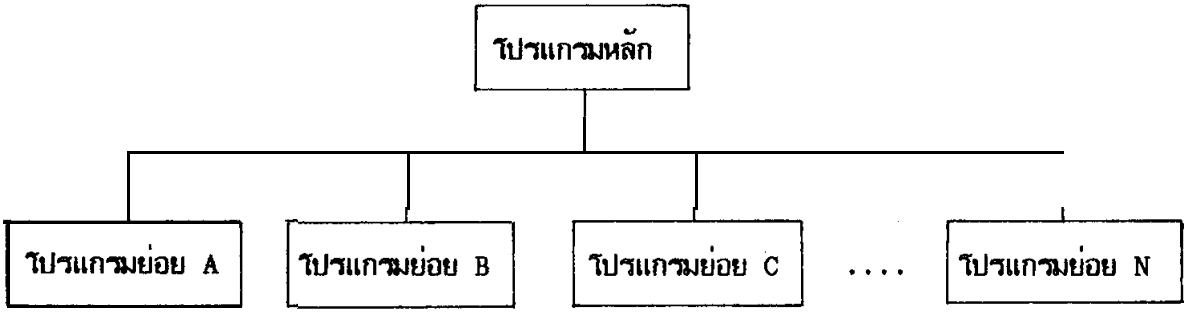
เราจะพบว่ายิ่งเรากำหนดวัตถุประสงค์ของโปรแกรมย่อยได้ชัดเจนเพียงใด การประยุกต์ใช้ในโปรแกรมที่กำลังเขียนอยู่หรือที่จะเขียนขึ้นในอนาคตก็ยิ่งกว้างขวางมากขึ้น

8.2.3 ความสัมพันธ์ระหว่างโปรแกรมหลักและโปรแกรมย่อย

ในตัวอย่างของเราทั้งหมดถึงขณะนี้ โปรแกรมของเราไม่ได้ใช้โปรแกรมย่อยเลย โปรแกรมรวม (overall program) ประกอบด้วยโปรแกรมหลักเท่านั้น เมื่อเราเริ่มใช้โปรแกรมย่อยเราจะมีโปรแกรมรวมซึ่งประกอบด้วยส่วนต่าง ๆ จำนวนหนึ่งซึ่งเป็นอิสระต่อกัน ดังแสดงในรูปที่ 8.1 ถึงแม้ว่าทุกส่วนจะเป็นของโปรแกรมเดียวกัน เราสามารถดูตรรกะของความสัมพันธ์ของส่วนต่าง ๆ ในรูปที่ 8.2 โปรแกรมหลักขณะนี้เป็นตัวควบคุมในโปรแกรมรวมและโปรแกรมย่อยซึ่งจะเป็นผู้รับใช้โปรแกรมหลักทำหน้าที่เพียงเมื่อโปรแกรมหลักร้องขอ

เมื่อเราเขียนโปรแกรมย่อย เราจะรวมคำสั่งของโปรแกรมย่อยในโปรแกรมรวม ซึ่งจะปรากฏเพียงครั้งเดียว โปรแกรมย่อยสามารถถูกเรียกจากโปรแกรมหลักเป็นจำนวนกี่ครั้งก็ได้ เราสามารถใช้โปรแกรมย่อยหลาย ๆ โปรแกรมซึ่งเป็นชนิดบิลท์อิน (built-in) (เช่นฟังก์ชันทางคณิตศาสตร์) เราไม่ต้องการสำเนาเกี่ยวกับรายละเอียดของโปรแกรมเหล่านี้ แต่ต้องการเพียงเรียกใช้เหมือนกับเราได้เขียนขึ้นเอง

รูปที่ 8.1 โครงสร้างของความสัมพันธ์



รูปที่ 8.2 ตรรกะของความสัมพันธ์

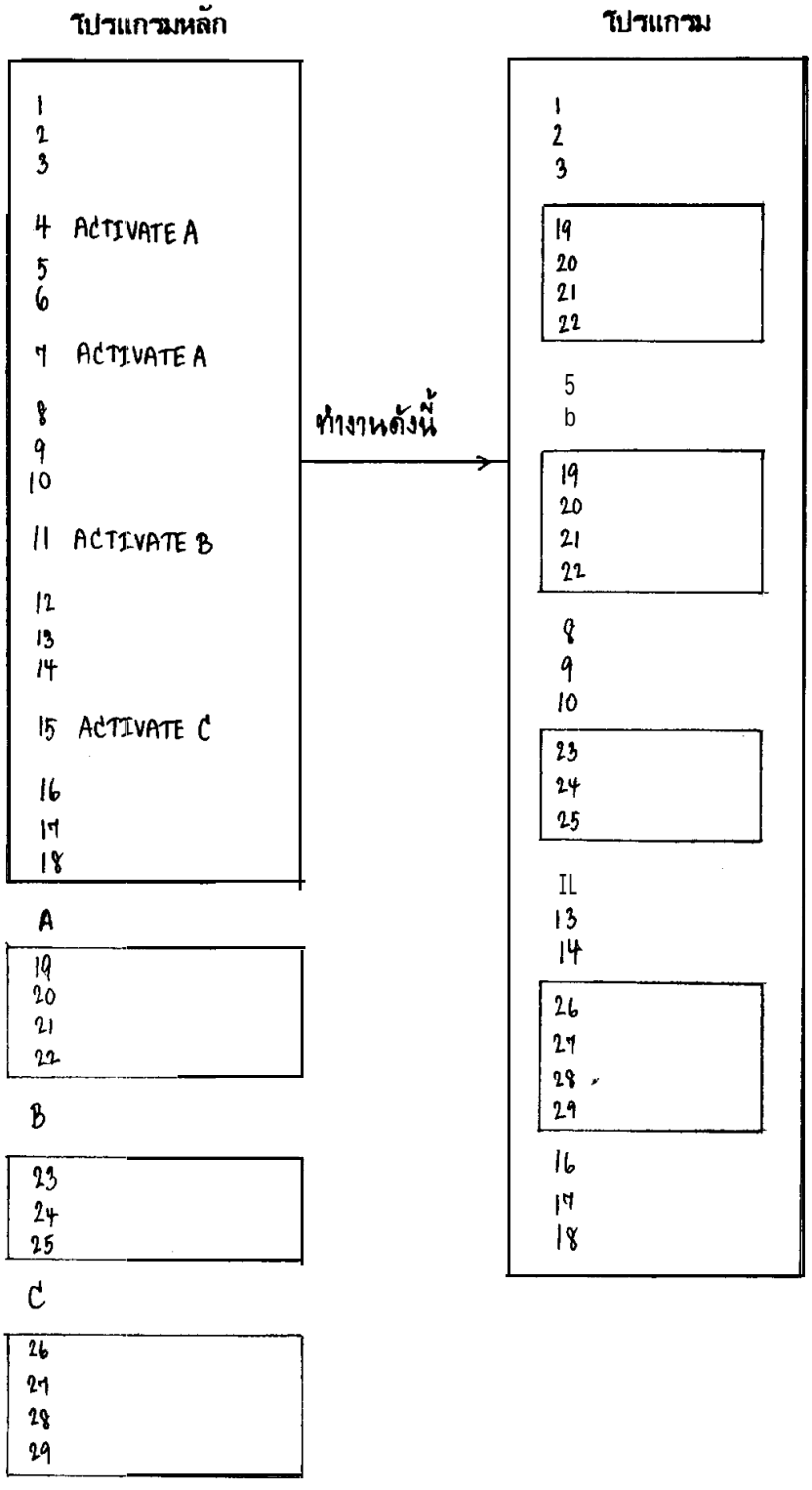
8.2.3.1 ทิศทางของการควบคุมและการปฏิบัติงานอย่างต่อเนื่อง

ถึงแม้ว่าโปรแกรมย่อยส่วนมากปรากฏในโปรแกรมรวม ภาษาฟอร์แทรนจะไม่ปฏิบัติตามคำสั่งใด ๆ ในโปรแกรมย่อยจนกว่าโปรแกรมหลักจะต้องการมัน การปฏิบัติตามคำสั่งในโปรแกรมย่อยไม่สัมพันธ์กับตำแหน่งที่มันอยู่ในโปรแกรมรวม (เรายอมรับรูปแบบของการวางโปรแกรมย่อยไว้ตอนท้ายของโปรแกรมรวม ถึงแม้ว่ามันอาจอยู่ที่ใดก็ได้) เมื่อโปรแกรมหลักต้องการใช้โปรแกรมย่อย ภาษาฟอร์แทรน ย้ายการควบคุมไปที่ตัวโปรแกรมย่อยเอง เพื่อแสดงการทำงานนี้ สมมุติว่าเรากำหนดคำสั่งของเราขึ้นคำสั่งหนึ่งในโปรแกรมหลักซึ่งจะเรียกใช้โปรแกรมย่อยนั่นคือคำสั่ง

ACTIVATE name

ซึ่งจะเป็นคำสั่งที่เราจะใช้ในการเรียกโปรแกรมย่อยชื่อ "name" คำสั่งในโปรแกรมย่อยจะถูกปฏิบัติตามโดยวิธีเดียวกับที่คำสั่งในโปรแกรมหลักถูกปฏิบัติตาม เมื่อการปฏิบัติในโปรแกรมย่อยเสร็จแล้ว ภาษาฟอร์แทรนจะส่งการควบคุมกลับมายังโปรแกรมหลัก โดยมายังคำสั่งที่อยู่ถัดจากคำสั่งซึ่งเรียกใช้โปรแกรมย่อย และลงมือปฏิบัติงานในโปรแกรมหลักต่อไป

เมื่อโปรแกรมย่อยถูกเรียกใช้โดยโปรแกรมหลัก คำสั่งภายในโปรแกรมย่อยถูกปฏิบัติตามคล้ายกับว่ามันปรากฏในโปรแกรมหลักในตำแหน่งของคำสั่งที่เรียกใช้มัน เพื่อให้เข้าใจวิธีการได้ดีขึ้นโปรดดูตัวอย่างในรูปที่ 8.3 โปรแกรมรวมประกอบด้วย 30 คำสั่ง 18 คำสั่งในโปรแกรมหลัก และ 4, 3 และ 4 คำสั่งในโปรแกรมย่อย A, B และ C ตามลำดับ เมื่อเริ่มต้นทำงานในโปรแกรมหลัก เราจะเห็นการต่อเนื่องของเหตุการณ์ต่อไปนี้


โปรแกรมหลัก
โปรแกรม
1
2
3
4 ACTIVATE A
5
6
7 ACTIVATE A
8
9
10
11 ACTIVATE B
12
13
14
15 ACTIVATE C
16
17
18
ทำงานดังนี้
A
19
20
21
22
B
23
24
25
C
26
27
28
29
1
2
3
19
20
21
22
5
b
19
20
21
22
8
9
10
23
24
25
IL
13
14
26
27
28
29
16
17
18

1. คำสั่งที่ 1-3 ถูกปฏิบัติตาม

2. คำสั่งที่ 4 เรียกโปรแกรมย่อย A คำสั่งที่ 19-22 ถูกปฏิบัติตาม และการควบคุมจะกลับมายังโปรแกรมหลักที่คำสั่งที่ 5

3. คำสั่งที่ 5-6 ถูกปฏิบัติตาม

4. คำสั่งที่ 7 เรียกโปรแกรมย่อย A ดังนั้นคำสั่งที่ 19-22 ถูกปฏิบัติตาม การควบคุมกลับมาที่คำสั่งที่ 8 ในโปรแกรมหลัก

5. ปฏิบัติตามคำสั่งที่ 8-10

6. คำสั่งที่ 11 เรียกโปรแกรมย่อย B ดังนั้นคำสั่งที่ 23-25 ถูกปฏิบัติตาม การควบคุมกลับมาที่คำสั่งที่ 12 ในโปรแกรมหลัก

7. ปฏิบัติตามคำสั่งที่ 12-14

8. คำสั่งที่ 15 เรียกโปรแกรมย่อย C ดังนั้นคำสั่งที่ 26-29 ถูกปฏิบัติตาม การควบคุมกลับมาที่คำสั่งที่ 16 ในโปรแกรมหลัก

9. ปฏิบัติตามคำสั่งที่ 16-18 แล้วเป็นอันจบการทำงานของโปรแกรมรวมถึงแม้ว่าจะมีคำสั่งที่ 19-30 ตามมาก็ตาม การปฏิบัติงานของโปรแกรมรวมจะจบที่คำสั่งที่ 18

8.2.3.2 <u>การส่งข้อมูลไปยังโปรแกรมย่อย</u>

พิจารณาพนักงานซึ่งทำงานในสำนักงานที่มีกระดานดำเล็ก ๆ 2 แผ่น แผ่นหนึ่งมีชื่อว่า "ROOM" อีกแผ่นหนึ่งชื่อ "COST" งานของพนักงานผู้นี้คือตรวจสอบห้องต่าง ๆ ในตึกสำนักงานและประมาณราคาค่าซ่อมแซมที่ต้องทำในแต่ละห้องซึ่งราคาขึ้นอยู่กับราคาในการทาสี ซ่อมปูนเป็นต้น ทุกวันเมื่อเขาถึงที่ทำงาน หัวหน้าแผนกคนหนึ่งจะเขียนหมายเลขประจำแผนกของเขาและหมายเลขห้องลงบนกระดานดำชื่อ "ROOM" พนักงานผู้นั้นจะไปที่ห้องดังกล่าว ประมาณรายจ่ายในการซ่อมแซม แล้วก็กลับมายังห้องทำงานของตน เขียนราคาที่ประมาณลงบนกระดานดำชื่อ "COST" แล้วหัวหน้าแผนกจึงอ่านราคาค่าซ่อมจากกระดานดำ แต่ละครั้งที่จะต้องซ่อมแซมห้องใดห้องหนึ่ง ขบวนการนี้จะถูกทำซ้ำ หัวหน้าแผนกทำหน้าที่เช่นเดียวกับโปรแกรมหลัก จะส่งผ่านหมายเลขประจำแผนกและหมายเลขห้อง (ข้อมูลของโปรแกรมหลัก) ไปยังผู้ทำการประมาณค่าซ่อม (โปรแกรมย่อย) โดยการเขียนข้อมูลลงบนกระดานดำชื่อ "ROOM" พนักงานใช้อัตราค่าจ้าง

ที่เขามีอยู่ (ข้อมูลของโปรแกรมย่อย) เพื่อประมาณงานและคิดราคาค่าซ่อม (ข้อมูลของโปรแกรมย่อย) ซึ่งเขาจะส่งกลับไปยังหัวหน้าแผนกโดยการเขียนราคาประมาณลงบนกระดานชื่อ "COST" (เราจะเห็นว่าคนใดก็ตามอาจเขียนหมายเลขห้องลงบนกระดาน และจะได้รับราคาประมาณค่าซ่อมคืนมา)

โปรแกรมหลักและโปรแกรมย่อยจะส่งข้อมูลไปและกลับโดยวิธีการเดียวกัน เมื่อโปรแกรมหลักเรียกโปรแกรมย่อย โปรแกรมหลักจะให้รายชื่อของรายการข้อมูลแก่โปรแกรมย่อย รายการข้อมูลเหล่านี้ซึ่งมีอยู่ในโปรแกรมหลัก เมื่อถูกใช้ในรายการที่เกี่ยวข้องกับการเรียกใช้โปรแกรมย่อย เราเรียกมันว่าแอคชวลอาร์กิวเมนต์ (actual arguments) และเราเรียกรายการดังกล่าวว่าอาร์กิวเมนต์ลิสท์ (argument list) เราสามารถดัดแปลงคำสั่งของเราที่เรียกใช้ซับรูทีน (subroutine) ซึ่งประกอบด้วยอาร์กิวเมนต์ดังนี้

```
ACTIVATE name(arg1, arg2, ..., argN)
```

อาร์กิวเมนต์ทั้งหมดซึ่งไม่ใช่ค่าคงที่จะเป็นชื่อของตัวแปรซึ่งได้ถูกกำหนดในโปรแกรมหลัก โปรแกรมย่อยมีรายการคล้ายคลึงกันและมีจำนวนรายการข้อมูลเท่ากัน โปรแกรมย่อยใช้ชื่อเหล่านี้ข้อมูลภายในของมัน เพื่อที่จะทำการคำนวนการเพื่อให้ได้ผลลัพธ์ซึ่งจะถูกส่งกลับไปยังโปรแกรมหลัก เราเรียกชื่อดังกล่าวว่า ดัมมีอาร์กิวเมนต์ (dummy argument) ต่อไปนี้เป็นคำสั่งที่เราตั้งขึ้นเองเพื่อที่จะใช้กำหนดคำสั่งแรกในโปรแกรมย่อย นั่นคือ

```
SUBPROGRAM name(darg1, darg2, ..., dargN)
```

ซึ่งจะเป็นคำสั่งแรกในโปรแกรมย่อยชื่อ name ดังแสดงในรูปที่ 8.4 เมื่อโปรแกรมย่อยถูกเรียก ภาษาฟอร์แทรนจับคู่รายการข้อมูลในดัมมีอาร์กิวเมนต์กับรายการในแอคชวลอาร์กิวเมนต์ และใช้ค่าของตัวแปรที่เป็นแอคชวลอาร์กิวเมนต์เพื่อเป็นค่าของอาร์กิวเมนต์ที่คู่กัน

```
ACTIVATE    (arg1 , arg2, arg3)
                  ↓     ↓     ↓
SUBPROGRAM (darg1, darg2, darg3)
```

รูปที่ 8.4 ความสัมพันธ์ของดัมมีอาร์กิวเมนต์และแอคชวลอาร์กิวเมนต์

8.3 โปรแกรมย่อยฟังก์ชัน (Function subprogram)

8.3.1 โปรแกรมตัวอย่าง

จงเขียนโปรแกรมเพื่อคำนวณ $C(n,m) = \frac{n!}{m!(n-m)!}$

โดยที่ถ้า n เป็นเลขจำนวนเต็มแล้ว

n! (n factorial) = n(n-1)...2.1

C(n,m) แทนจำนวนวิธีในการหยิบของ m ชิ้นจากของที่แตกต่างกันหมด n ชิ้น

$C(6,2) = \frac{6!}{2!\,4!} = \frac{6.5.4.3.2.1}{(2.1)(4.3.2.1)} = 15$

การเขียนโปรแกรมโดยไม่ใช้โปรแกรมย่อยฟังก์ชันเป็นดังนี้

```
      INTEGER COMB
      READ(5,1)N,M
    1 FORMAT(2I2)
      NFACT=1
      DO 10 I=1,N                    }
   10 NFACT=NFACT*I                  } HI n!
      MFACT=1
      DO 20 I=1,M                    }
   20 MFACT=MFACT*I                  } HI m!
      NM=N-M
      NMFACT=1
      DO 30 I=1,NM                   }
   30 NMFACT=NMFACT*I                } HI (n-m)!
      COMB=NFACT/(MFACT*NMFACT)
      WRITE(6,2)N,M,COMB
    2 FORMAT('1','COMBINATIONS OF',I3,'OBJECTS TAKEN',I3,
     *'AT A TIME IS',I4)
```

```
      STOP
      END
```

ในโปรแกรมข้างต้นมีการคำนวณหาค่าแฟคทอเรียลของเลขจำนวนเต็มถึง 3 ครั้ง ในกรณีนี้เราอาจเขียนโปรแกรมย่อยฟังก์ชันขึ้นเพื่อทำการคำนวณหาค่าแฟคทอเรียล และในโปรแกรมหลัก (main program) เราจะเรียกใช้โปรแกรมย่อยนี้เมื่อต้องการ

โปรแกรมหลักและโปรแกรมย่อยฟังก์ชันเพื่อแก้ปัญหาข้างต้นเช่นกัน

```
      INTEGER  COMB,FACT
      READ(5,1)N,M
    1 FORMAT(2I2)
      COMB=FACT(N)/(FACT(M)*FACT(N-M))
      WRITE(6,2)N,M,COMB
    2 FORMAT('1COMBINATIONS OF',I3,'OBJECTS TAKEN',I3,
     *'AT A TIME IS',I4)
      STOP
      END
      INTEGER  FUNCTION  FACT(N)
      FACT=1
      DO 10 I=1,N
   10  FACT=FACTSI
      RETURN
      END
```

8.3.2 คำสั่ง FUNCTION และคำสั่ง RETURN

รูปทั่วไปของคำสั่ง FUNCTION คือ

[type] FUNCTION ชื่อฟังก์ชัน(p_1, p_2, ..., p_n)

โดยที่ ชื่อฟังก์ชันตั้งตามหลักการตั้งชื่อตัวแปร

p_1 , ..., p_n คือดัมมีอาร์กิวเมนต์ ซึ่งใช้สำหรับส่งผ่านข้อมูลไปมากับโปรแกรมเรียก p_i อาจเป็นตัวแปร ชื่อแถวลำดับ หรือชื่อฟังก์ชันอื่น สมาชิกของ แถวลำดับเช่น X(I) ใช้เป็นดัมมีอาร์กิวเมนต์ไม่ได้

type ซึ่งอาจไม่ใช้ก็ได้ เป็นตัวระบุชนิดของค่าที่ฟังก์ชันคำนวณได้และจะส่งไปยังโปรแกรมเรียก ถ้าไม่ระบุ type แล้วชนิดของฟังก์ชันจะเป็นไปตามตัวอักษรตัวแรกของชื่อ type เช่น REAL, INTEGER, DOUBLE PRECISION

ดัมมีอาร์กิวเมนต์ในโปรแกรมย่อยนั้นคือชื่อของแอคชวลอาร์กิวเมนต์ในโปรแกรมหลัก และปรากฏในคำสั่งที่เรียกใช้ฟังก์ชันด้วย เมื่อมีการปฏิบัติตามคำสั่งในฟังก์ชัน ดัมมีอาร์กิวเมนต์จะใช้ค่าของแอคชวลอาร์กิวเมนต์ที่ระบุไว้ในโปรแกรมหลัก

แอคชวลอาร์กิวเมนต์จะมีจำนวนตัว ชนิดและลำดับที่สมนัย (correspond) กับดัมมีอาร์กิวเมนต์ ชื่ออาร์กิวเมนต์ทั้ง 2 ชนิดอาจเหมือนกันหรือต่างกันก็ได้ ถ้าดัมมีอาร์กิวเมนต์เป็นชื่อของแถวลำดับ เราจะต้องกำหนดแถวลำดับโดยใช้คำสั่ง DIMENSION ในโปรแกรมย่อยฟังก์ชันด้วย

รูปทั่วไปของคำสั่ง RETURN คือ

```
RETURN
```

คำสั่ง RETURN จะโยกย้ายการควบคุมการทำงานกลับไปยังโปรแกรมหลัก โปรแกรมย่อยฟังก์ชันอาจมีคำสั่ง RETURN หลายคำสั่งได้ การโยกย้ายการควบคุมจะกลับไปยังที่ ๆ เรียกใช้โปรแกรมย่อยฟังก์ชันนั้น

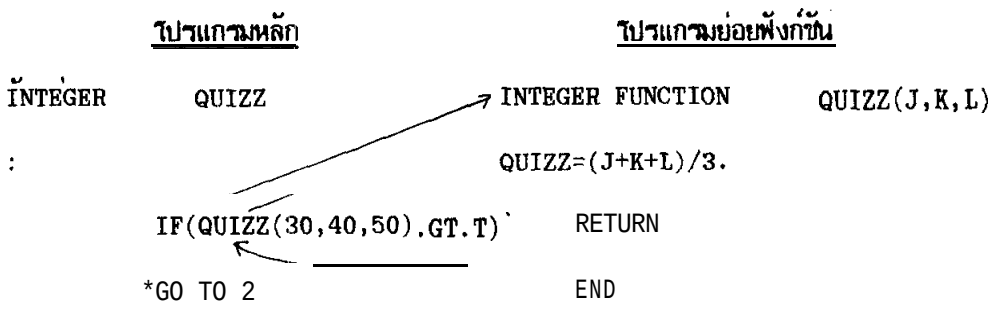
ตัวอย่าง ฟังก์ชัน ALARG จะส่งค่าสูงสุดระหว่างอาร์กิวเมนต์ 2 ตัวใด ๆ

โปรแกรมหลัก

```
    :
Z=ALARG(X,Y)
WRITE(6,3)Z
    :
IF(ALARG(Z,T)-3.1)1,1,2
```

โปรแกรมย่อยฟังก์ชัน

```
FUNCTION ALARG(A,B)
IF(A.GT.B)GO TO 5
ALARG=B
RETURN
5 ALARG=A
RETURN
END
```



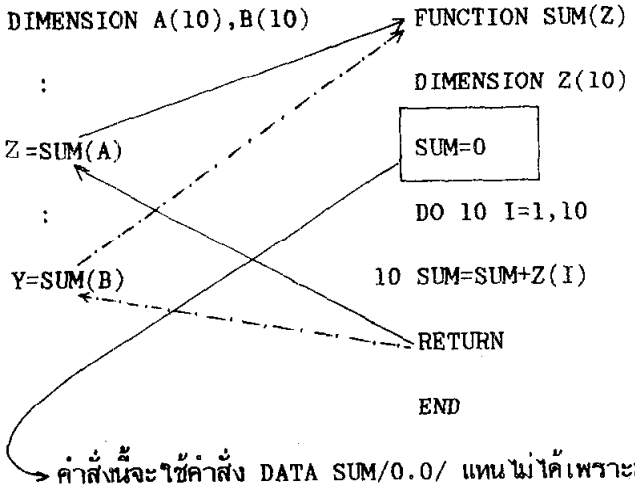
โปรดสังเกตว่าได้กำหนดให้ QUIZZ เป็นชื่อแบบ integer ในทั้ง 2 โปรแกรม ค่าของ QUIZZ ที่ส่งมาจากโปรแกรมย่อยฟังก์ชันเป็นค่าจำนวนเต็ม

ตัวอย่าง

```
I=2                          |  FUNCTION BAKER(X,Y,J)
A=7.0                        |    :
Y=BAKER(3.2,9.8*A,I)         |
```

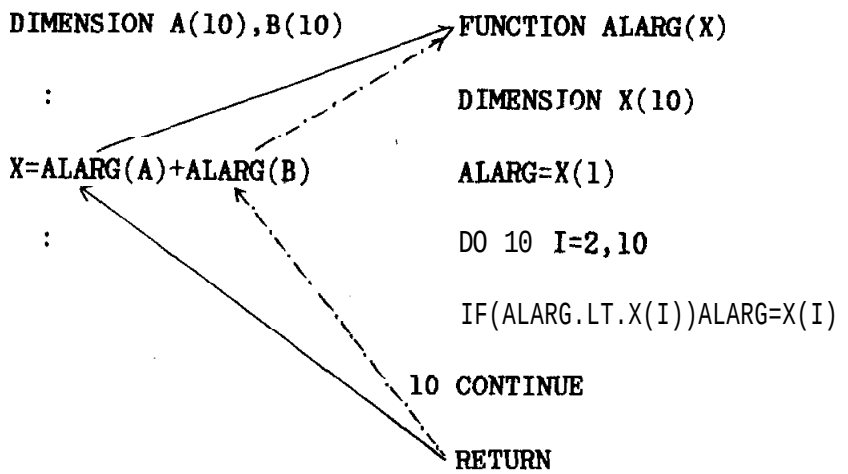
เมื่อมีการเรียกใช้ฟังก์ชัน BAKER การปฏิบัติงานในฟังก์ชัน BAKER จะใช้ค่า X=3.2, Y=9.8*7=68.6 และ J=2

ตัวอย่าง

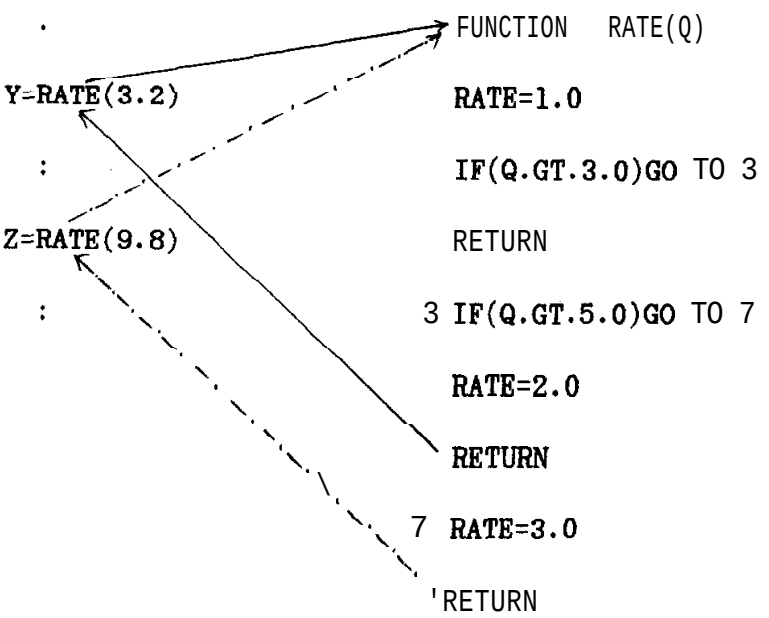


คำสั่งนี้จะใช้คำสั่ง DATA SUM/0.0/ แทนไม่ได้เพราะมันจะไม่กำหนดค่าของ SUM ให้เป็น 0 ใหม่เมื่อฟังก์ชัน SUMถูกเรียกใช้ซ้ำ เช่นในตัวอย่างจากคำสั่ง Y=SUM(B) นั้น แทนที่ $Y = \sum_{i=1}^{10} Bi$ แต่ $Y = \sum_{i=1}^{10} Ai + \sum_{i=1}^{10} Bi$

ตัวอย่าง



ตัวอย่าง



ฟังก์ชัน RATE จะให้ผลลัพธ์

$$RATE(Q)=\begin{cases}1.\ \text{ถ้า } Q\leq 3\\ 2.\ \text{ถ้า } 3<Q\leq 5\\ 3.\ \text{ถ้า } Q>5\end{cases}$$

ดังนั้น Y=2.0, Z=3.0

ในการเขียนโปรแกรมย่อยเพื่อจัดการกับแถวลำดับที่มีขนาดต่าง ๆ กันนั้น เราอาจใช้ตัวแปรชนิด integer เพื่อแทนขนาดของแถวลำดับในคำสั่ง DIMENSION ได้

ตัวอย่าง โปรแกรมหลัก โปรแกรมย่อยฟังก์ชัน

```
DIMENSION CLASS1(10,10)          FUNCTION AVER(ARAY,NR,NC)
DIMENSION CLASS2(5,5)            DIMENSION ARAY(NR,NC)
DIMENSION CLASS3(7,17)           AVER=0
A=AVER(CLASS1,10,10)             DO 5 I=1,NR
B=AVER(CLASS2,5,5)               DO 5 J=1.NC
C=AVER(CLASS3,7,17)            5 AVER-AVER+ARAY(I,J)
TOT=(A+B+C)/3                    AVER=AVER/(NR*NC)
                                 RETURN
                                 END
```

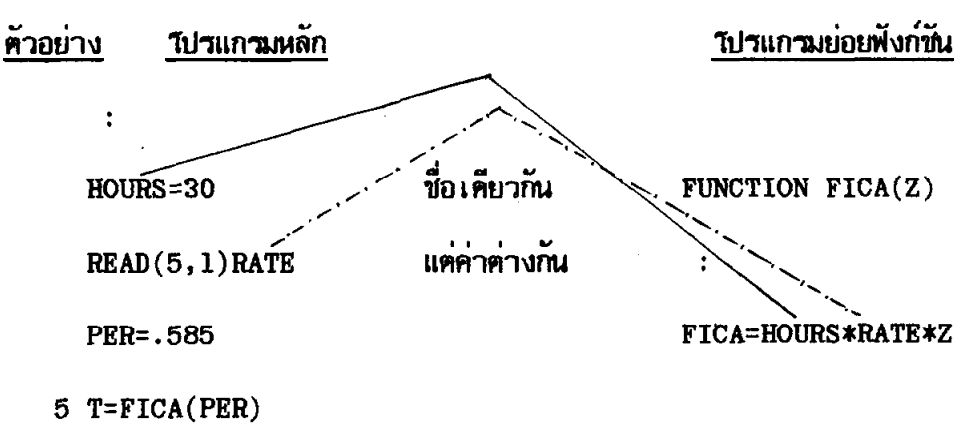
ตัวอย่างต่อไปนี้เป็นการเรียกใช้ฟังก์ชันที่ผิดหรือเป็นการกำหนดฟังก์ชันที่ไม่ถูกต้อง

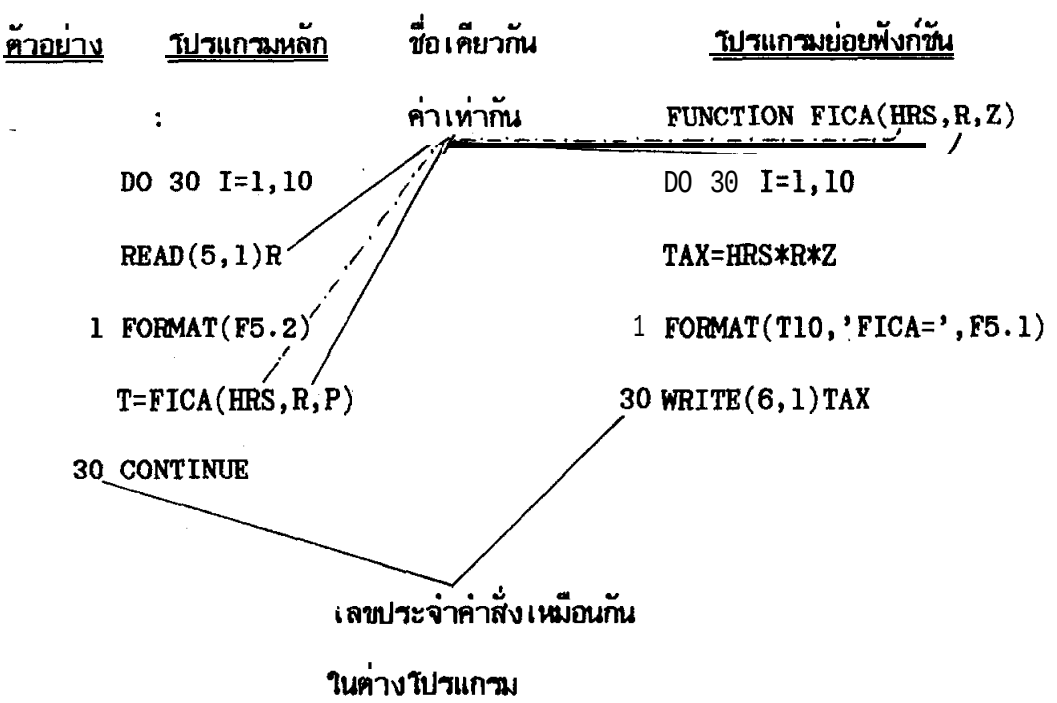
โปรแกรมหลัก	โปรแกรมย่อยฟังก์ชัน	เหตุผล
`X=AMAX(2.1,3,4.)`	`FUNCTION AMAX(X,Y,Z)`	3 เป็นเลขจำนวนเต็ม แต่ Y เป็นชนิด real

`INTEGER SMALL`
`:`
`K=SMALL(3.1,X)`
	`FUNCTION SMALL(T,X)`	ไม่ควรกำหนดชนิด INTEGER ให้ SMALL ในโปรแกรมหลัก

`S=COT(A,B,T(3))`	`FUNCTION COT(X,Y,Z)` `Z=COT(T,Z,P)`	คำสั่งในโปรแกรมย่อยฟังก์ชันเรียกตัวเองไม่ได้
`DIMENSION Q(100)` `C=DAM(Q(1),Q(2),-4)`	`FUNCTION DAM(T(1),T(2),J)` `DIMENSION T(100)`	ตัวแปรอาร์กิวเมนต์เป็นสมาชิกของแถวลำดับไม่ได้
`MIN=SUM(X,2*J,9.8)`	`FUNCTION SUM(X,J)`	จำนวนอาร์กิวเมนต์ไม่เท่ากัน
`DIMENSION CART(10)` `X=TIP(3.9,X,CART)`	`FUNCTION TIP(X,Y,Z)` `TIP=(X+Y)/2`	ต้องกำหนด DIMENSION ให้ Z ด้วย

เนื่องจากคอมไพเลอร์ถือว่าโปรแกรมหลักและโปรแกรมย่อยนั้นเป็นอิสระต่อกัน การใช้ตัวแปรและเลขประจำคำสั่งซ้ำกันใน 2 โปรแกรมนั้นไม่ผิดกฎของภาษา แต่การที่โปรแกรมเมอร์ตั้งตัวแปรเหมือนกันในโปรแกรมทั้ง 2 นั้น ไม่ได้หมายความว่า โปรแกรมย่อยจะสามารถใช้ค่าของตัวแปรเดียวกันจากโปรแกรมหลักได้ วิธีการติดต่อสื่อสารค่าของตัวแปรระหว่างโปรแกรมได้นั้นทำได้ 2 ทางคือ โดยการใช้อาร์กิวเมนต์ และโดยการใช้คำสั่ง COMMON พิจารณาตัวอย่างต่อไปนี้





8.3.3 Statement function

ในโปรแกรมย่อยฟังก์ชันนั้นเราสามารถเขียนคำสั่งหลาย ๆ คำสั่งเพื่อให้ได้คำตอบต่าง ๆ กันตามเงื่อนไขต่าง ๆ (ดูจากคำสั่ง RETURN หลาย ๆ คำสั่ง) แต่อย่างไรก็ดีมันจะส่งค่าเพียง 1 ค่าเท่านั้นไปยังที่ ๆ เรียกมัน ในหลาย ๆ กรณีฟังก์ชันอาจดูง่าย ๆ คือมีเพียง 1 คำสั่งเท่านั้นก็สามารถให้คำตอบที่ต้องการได้ ในกรณีนี้เราจะเขียน statement function แทนที่จะเขียนเป็นโปรแกรมย่อยฟังก์ชัน โดยทั่ว ๆ ไปเรามักใช้ statement function เมื่อเราต้องการหาค่าของนิพจน์คณิตศาสตร์นิพจน์หนึ่ง เมื่อเรามีค่าของตัวแปรหลายชุดและใช้ค่าของนิพจน์ในที่ต่าง ๆ ในโปรแกรม สิ่งหนึ่งซึ่งต่างไปจากโปรแกรมย่อยฟังก์ชันคือ statement function นั้นจะอยู่ในโปรแกรมที่จะเรียกใช้มันนั่นเอง ตัวอย่างเช่น ถ้าต้องการหาโพลิโนเมียลดีกรี 2 ในที่ต่าง ๆ ในโปรแกรมและเมื่อตัวแปรมีค่าต่าง ๆ กัน เราจะแสดงการใช้ statement function POLY เปรียบเทียบกับการไม่ใช้ statement function

โปรแกรมที่ไม่ได้ใช้ statement function	โปรแกรมที่ใช้ statement function
	`POLY(X)=2.1*X**2-3*X+1`
`X=-5.6`	`X=-5.6`
`Y=2.1*X**2-3*X+1`	`Y=POLY(X)`
`X=2.12347`	`IF(POLY(2.12347)-TOT)1,1,3`
`IF(2.1*X**2-3*X+1-TOT)1,1,3`	
`T=10.6`	`T=10.6`
`SUM=SUM+SQRT(2.1*T**2-3*T+1)`	`SUM=SUM+SQRT(POLY(T))`

รูปแบบทั่วไปของการกำหนด statement function

ชื่อฟังก์ชัน($a_1,\ldots,a_n$)=นิพจน์

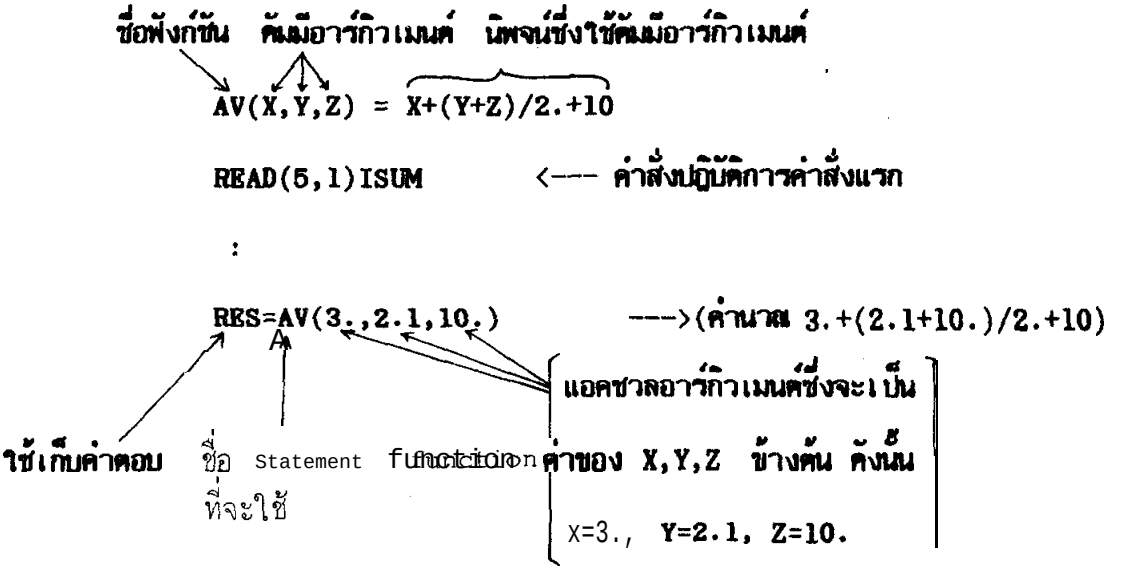
โดยที่ ชื่อฟังก์ชัน ตั้งตามหลักการตั้งชื่อตัวแปร

$a_1,\ldots,a_n$ เป็นดัมมีอาร์กิวเมนต์ ซึ่งต้องไม่เป็นสมาชิกของแถวลำดับ

นิพจน์คือนิพจน์เลขคณิตใด ๆ ที่ต้องไม่มีสมาชิกของแถวลำดับเป็นตัวถูกกระทำในนิพจน์นั้น ๆ นิพจน์นี้จะประกอบด้วย $a_1,\ldots,a_n$ และอาจมีตัวแปรอื่น ๆ ค่าคงที่ ชื่อโปรแกรมย่อยฟังก์ชัน หรือ statement function ที่ได้กำหนดไว้ก่อนแล้ว

Statement function จะต้องอยู่ก่อนคำสั่งปฏิบัติการใด ๆ ในโปรแกรม แต่จะตามหลังคำสั่งเฉพาะเช่นคำสั่ง DIMENSION, คำสั่ง INTEGER เป็นต้น

<u>ตัวอย่าง</u> การกำหนด Statement function และการเรียกใช้



เมื่อปฏิบัติตาม statement function ค่าของแอคชวลอาร์กิวเมนต์จะแทนที่ดัมมีอาร์กิวเมนต์ในนิพจน์ทางขวาของ statement function ถ้าตัวแปรนอกเหนือไปจากดัมมีอาร์กิวเมนต์รวมอยู่ในนิพจน์ ค่าของมันในขณะนั้น ๆ จะถูกใช้ในการหาค่านิพจน์ดังกล่าว

ในนิพจน์ของ statement function สมาชิกของแถวลำดับจะเป็นดัมมีอาร์กิวเมนต์ไม่ได้ แต่ในคำสั่งที่เรียกใช้มัน แอคชวลอาร์กิวเมนต์อาจเป็นสมาชิกของแถวลำดับหรือนิพจน์เลขคณิตได้ และเช่นเดียวกับในโปรแกรมย่อยฟังก์ชัน แอคชวลอาร์กิวเมนต์จะต้องสมนัยกับดัมมีอาร์กิวเมนต์ทั้งจำนวน ชนิดและลำดับที่ ชนิดของค่าที่คำนวณได้ขึ้นอยู่กับชื่อของฟังก์ชัน แต่อาจเปลี่ยนแปลงชนิดได้ด้วยคำสั่งกำหนดชนิด

ตัวอย่าง

1) `F(X)=(X+2)/(X-2)+X**2` -->$f(x)= \frac{X+2}{X-2} + x^2$

`Y=F(4.0)` -->y = 19.0

`Z=F(T)`

2) `BETA(Y)=A*Y**2+B*Y+C`

```
READ(5,1)A,B,C
DO 10 IX=1,10
   Q=BETA(FLOAT(IX))          -->เปลี่ยน IX ให้มีชนิดเดียวกับ Y
10 WRITE(6,2)IX,Q
```

3) `READ(DEGREE)=DEGREE/57.296` -->แปลงองศาเป็นเรเดียน

`Y=SIN(RAD(30.))` ใช้ statement function เป็นอาร์กิวเมนต์ของฟังก์ชันอื่น

4) `B1(Y)=ABS(Y-1.)`

`B2(Y)=LOG(B1(Y))/10` -->เรียกใช้ statement function ที่กำหนดมาก่อนแล้ว

`Z=B2(-5.)`

5) `COMB(N,M)=FACT(N) / (FACT(M)*FACT(N-M))`

`K=COMB(8,2)` -->FACT เป็นฟังก์ชันที่กำหนดมาก่อนแล้ว

6) PAY(HRS,RATE)=HRS*RATE+BONUS

```
BONUS=50.
X=PAY(40.,5.)                    -->X=5*40+50=250
BONUS=BONUS+10
Y=PAY(50.,4.)                    -->Y=4*50+60=260
```

7) FAHR(C)=9./5.*C+32

```
   :
C=0                              -->C ใช้เป็นตัวแปรตามปกติได้ถึงแม้ว่าจะใช้
TEMP=FAHR(100.)                     เป็นตัวมีอาร์กิวเมนต์มาก่อนแล้ว
C=C+1                            -->C มีค่าเท่ากับ 1
```

8)
```
REAL INT
DIMENSION A (10)
INT(R,N)=PRIN* R**N
```

```
DUE=INT(A(3),K*360+J)            -->ใช้สมาชิกของแถวลำดับและนิพจน์เป็น
                                    แอคชาลอาร์กิวเมนต์
```

9)
```
T(X)=SQRT(X**2+9.)+A
A=11
S=T(4.)+SQRT(T(5.))              -->S= √25 + 11 + √(√34 + 11)
```

ตัวอย่าง การกำหนด statement function ที่ไม่ถูกต้องและการเรียกใช้ statement function อย่างไม่ถูกต้อง

1)
```
PAY=HRS*RATE                     PAY ไม่มีอาร์กิวเมนต์
RES=PAY(40.0, 5.0)
```

2) CONS(X,3)=3*X+1 ค่าคงที่เป็นตัวมีอาร์กิวเมนต์ไม่ได้

3) DUE(X,A(1),Z)=A(1)+X+Z A(1) เป็นตัวมีอาร์กิวเมนต์ไม่ได้

```
4)      MERD(X,Y)=(X+Y)**13-A(4)          A(4) อยู่ในนิพจน์ไม่ได้
5)      DOG(X,Y,Z)=X+Y+DOG(C,D,F)         DOG จะเรียกใช้ตัวเองไม่ได้
6)      A(X)=T(X)+1                       ต้องกำหนด T(X) ก่อน
        T(Y)=Y*3+K                        จะใช้ได้ถ้าใส่ T(Y) ไว้ก่อน
7)      TRUE(NCY)=NCY*3/256               คำสั่ง DIMENSION ต้องอยู่ก่อน
        DIMENSION   A(l00)                statement function
8)      SIGN(X)=X
        Y=SIGN(3)                         3 และ X เป็นคนละชนิด
9)      BRA(V0,S)=(V0+S)**3
        Z=BRA(3.12*X)                     BRA ต้องมีอาร์กิวเมนต์ 2 ตัว
```

8.4 โปรแกรมย่อยซับรูทีน (Subroutine subprogram)

8.4.1 โปรแกรมตัวอย่าง

อาจารย์ รามาสอนภาษาฟอร์แทรน 2 ห้อง แถวลำดับ FINAL1 เก็บคะแนนสอบไล่ 10 จำนวนของห้องที่ 1 แถวลำดับ FINAL2 เก็บคะแนนสอบไล่ 8 จำนวนของห้องที่ 2 เพื่อให้แน่ใจว่ายุติธรรมทั้ง 2 ห้อง อาจารย์ รามาจึงได้ให้คะแนนสอบใหม่ตามวิธีการดังนี้

1) หาคะแนนเฉลี่ยของแต่ละห้อง ให้ AV1 เก็บคะแนนเฉลี่ยของห้องที่ 1
   AV2 เก็บคะแนนเฉลี่ยของห้องที่ 2
2) ถ้า AV1<AV2 นักศึกษาในห้องที่ 1 แต่ละคนจะได้คะแนนเพิ่มอีกคนละ 5% ของ AV1
3) ถ้า AV1>AV2 นักศึกษาในห้องที่ 2 แต่ละคนจะได้คะแนนเพิ่มอีกคนละ 5% ของ AV2

จงเขียนโปรแกรมเพื่อพิมพ์คะแนนเริ่มต้นและคะแนนที่ปรับแล้ว ข้อมูลเข้าและข้อมูลออกอยู่ในรูปต่อไปนี้

ข้อมูลเข้า

```
10  20  30  40  50  60  70  80                  -->FINAL2
60  70  80  90  50  60  70  80  80  60          -->FINAL1
```

(AV1=70, AV2=45, 5% ของ AV2=2.25 ii)นั่นนักศึกษาในห้องที่ 2 จะได้คะแนนเพิ่มคนละ 2.3 คะแนน)

ข้อมูลออก

```
FINAL1
 60.0    70.0    . . .          60.0
FINAL2
 10.0    20.0    . . .    80.0
BONUS=2.3
ADJUSTED GRADES
 12.3    22.3  32.3  . . .  8 2 . 3
```

โปรแกรมที่ไม่ใช้ซับรูทีน

```
      REAL FINAL1(10),FINAL2(8),FINAL(10)
      READ(5,7)FINAL1,FINAL2
      WRITE(6,4)FINAL1,FINAL2
      SUM=0
      DO 5 I=1,10
      SUM=SUM+FINAL1(I)
    5 CONTINUE
      AV1=SUM/10
      SUM=0
      DO 3 I=1,8
      SUM=SUM+FINAL2(I)
    3 CONTINUE
      AV2=SUM/8
      IF(AV1.GT.AV2)GO TO 20
      CALL ADJUST(FINAL1,10,FINAL,AV1)
      GO TO 2
   20 CALL ADJUST(FINAL2,8,FINAL,AV2)
    2 STOP
    4 FORMAT(1X,'FINAL1'/1X,10F5.1/
     *       1X,'FINAL2'/1X,8F5.1)
    7 FORMAT(10F5.1)
      END
```

โปรแกรมที่ใช้ซับรูทีน

```
      REAL FINAL1(10),FINAL2(8),FINAL(10)
      READ(5,7)FINAL1,FINAL2
      WRITE(6,4)FINAL1,FINAL2
      CALL AVRGE(FINAL1,AV1,10)
      CALL AVRGE(FINAL2,AV2,8)
      IF(AV1.GT.AV2)GO TO 20
      CALL ADJUST(FINAL1,10,FINAL,AV1)
      GO TO 2
   20 CALL ADJUST(FINAL2,8,FINAL,AV2)
    2 STOP
    4 FORMAT(1X,'FINAL1'/1X,10F5.1/
     *       1X,'FINAL2'/1X,8F5.1)
    7 FORMAT(10F5.1)
      END
```

```
      IF(AV1.GT.AV2)GO TO 20
      BONUS=AV1*.05
      DO 8 I=1,10
      FINAL(I)=FINAL1(I)+BONUS
    8 CONTINUK
      WRITE(6,6)BONUS,(FINAL(I),I=1,10)
      GO TO 2
   20 BONUS=AV2*.05
      DO 9 I=1,8
      FINAL(I)=FINAL2(I)+BONUS
    9 CONTINUE
      WRITE(6,6)BONUS,(FINAL(I),I=1,8)
    2 STOP
    4 FORMAT(1X,'FINAL1'/1X,10F5.1/
     *       1X,'FINAL2'/1X,8F5.1)
    8 FORMAT(1X,'BONUS=',F5.1/
     *1X,'ADJUSTED GRADES'/1X,10F5.1)
    7 FORMAT(10F5.1)
      END
```

```
      SUBROUTINK AVRGE(CLASS,AVG,N)
      REAL CLASS(10)
      SUM=0
      DO 5 I=1,N
    5 SUM=SUM+CLASS(I)
      AVG=SUM/N
      RETURN
      END
```

```
      SUBROUTINK ADJUST(CLASS,K,FINAL,AVG)
      REAL CLASS(10),FINAL(10)
      BONUS=AVG*0.05
      DO 9 I=1,K
    9 FINAL(I)=CLASS(I)+BONUS
      WRITE(6,6)BONUS,(FINAL(I),I=1,K)
      RETURN
    6 FORMAT(1X,'BONUS=',F5.1/
     *1X,'ADJUSTED GRADES'/1X,10F5.1)
      END
```

โปรดสังเกตการใช้

1. คำสั่ง CALL ซึ่งใช้เพื่อสั่งให้โปรแกรมย่อยชับรูทีนทำงาน และส่งผ่านอาร์กิวเมนต์ไปยังโปรแกรมย่อยชับรูทีน (ต่อไปจะเรียกสั้น ๆ ว่าชับรูทีน)

2. คำสั่ง SUBROUTINE ซึ่งใช้เพื่อกำหนดว่าโปรแกรมย่อยนั้นเป็นโปรแกรมย่อยชับรูทีน

โปรแกรมที่ใช้ชับรูทีนในตัวอย่างนี้ประกอบด้วยโปรแกรม 3 ส่วนซึ่งเป็นอิสระต่อกัน แต่ละส่วนจะทำงานแต่ละชนิด ส่วนทั้ง 3 คือ

1) โปรแกรมหลัก (Main program) ซึ่งเรียกใช้ซับรูทีน AVRGE และ ADJUST

2) ซับรูทีน AVRGE ทำการคำนวณและส่งค่าเฉลี่ยของคะแนน N คะแนนกลับไปยังโปรแกรมหลัก

3) ซับรูทีน ADJUST ทำการบวกค่าคงที่ (BONUS) ให้คะแนนแต่ละค่าใน K ค่าซึ่งอยู่ในแถวลำดับ CLASS และเก็บคะแนนที่ปรับแล้วในแถวลำดับชื่อ FINAL

โปรแกรมทั้ง 3 จะถูกแปลอย่างเป็นอิสระต่อกัน เราใช้คำสั่ง CALL ในการส่งผ่านข่าว-สารจากโปรแกรมเรียก (calling program) ไปยังโปรแกรมที่ถูกเรียก (called program) โดยผ่านทางอาร์กิวเมนต์ที่ระบุไว้ในคำสั่ง CALL (ซึ่งเรียกว่าแอคชวลอาร์กิวเมนต์) และในคำสั่ง SUBROUTINE (ซึ่งเรียกว่าดัมมีอาร์กิวเมนต์)

8.4.2 <u>คำสั่งในภาษาฟอร์แทรน</u>

8.4.2.1 <u>คำสั่ง CALL</u>

รูปแบบทั่วไป | CALL ชื่อซับรูทีน [($a_1, a_2, \ldots, a_n$)] |

โดยที่ CALL เป็นคีย์เวิร์ด

ชื่อซับรูทีนที่จะใช้

$a_1, \ldots, a_n$ เป็นแอคชวลอาร์กิวเมนต์ คือตัวที่จะใช้สื่อสารกับซับรูทีน อาร์กิวเมนต์อาจเป็นตัวแปร สมาชิกของแถวลำดับ ชื่อของแถวลำดับ ค่าคงที่ นิพจน์และชื่อของฟังก์ชัน เราใช้อาร์กิวเมนต์เพื่อการส่งข้อมูลจากโปรแกรมหลักไปยังซับรูทีนหรือเพื่อรับผลลัพธ์จากซับรูทีน ในซับรูทีนจะไม่มีอาร์กิวเมนต์เลยก็ได้ ($n \geq 0$)

<u>ตัวอย่าง</u>

CALL SUB(X)	ส่ง X ไปยังซับรูทีนชื่อ SUB
CALL XRAY	ไม่ส่งอาร์กิวเมนต์ไปเลย
CALL SI(A,B(1),3*I+K,J)	ส่งอาร์กิวเมนต์ 4 ตัวไปยังซับรูทีน SI ในกรณีนี้จะหาค่าของนิพจน์ 3*I+K ก่อนแล้วส่งไปเป็นค่าของอาร์กิวเมนต์ตัวที่ 3 ของซับรูทีน SI

CALL SUB2(SIN(X),Y) คอมพิวเตอร์จะหาค่า sin(x) ก่อนแล้วส่งไปให้อาร์กิวเมนต์ตัวแรกของซับรูทีน SUB2

8.4.2.2 คำสั่ง SUBROUTINE

คำสั่งนี้จะต้องอยู่เป็นคำสั่งแรกในโปรแกรมย่อยซับรูทีน

รูปทั่วไป

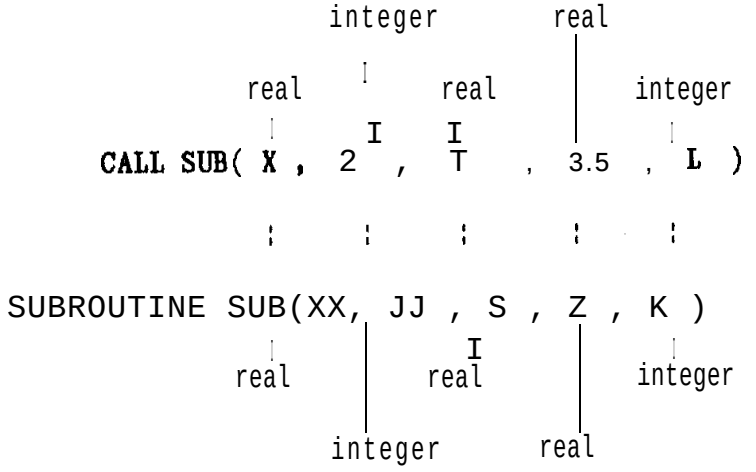
SUBROUTINE ชื่อซับรูทีน[(p_1,p_2,...,p_n)]

โดยที่ SUBROUTINE เป็นคีย์เวิร์ด

ชื่อของซับรูทีน ตั้งตามหลักการตั้งชื่อตัวแปรและไม่ต้องคำนึงถึงชนิด

p_1,p_2,...,p_n เป็นตัวมีอาร์กิวเมนต์ ซึ่งจะใช้ในการสื่อสารกับโปรแกรมหลัก อาร์กิวเมนต์เหล่านี้อาจเป็นตัวแปร ชื่อแถวลำดับ หรือชื่อฟังก์ชัน (ใช้สมาชิกของแถวลำดับและค่าคงที่ไม่ได้)

ถ้าอาร์กิวเมนต์เป็นชื่อของแถวลำดับ เราต้องกำหนดแถวลำดับนั้นในคำสั่ง DIMENSION ในซับรูทีนด้วย ขนาดของแถวลำดับดังกล่าวควรมีขนาดเดียวกับแถวลำดับที่สมนัยกันในโปรแกรมหลัก ซับรูทีนใช้ค่าของอาร์กิวเมนต์จากโปรแกรมหลักผ่านทางตัวมีอาร์กิวเมนต์ แอคชวลอาร์กิวเมนต์ในคำสั่ง CALL จะต้องมีจำนวน อันดับที่ และชนิดเดียวกับตัวมีอาร์กิวเมนต์ในคำสั่ง SUBROUTINE ชื่อที่ใช้เป็นอาร์กิวเมนต์ใน 2 แห่งอาจเหมือนกันหรือต่างกันก็ได้



ตัวอย่าง โปรแกรมหลัก	โปรแกรมย่อย	หมายเหตุ
DATA X,Y/3.,4./	SUBROUTINE ADD(A,B,C)	
CALL ADD(X,Y,R)	C=A+B	
WRITE(6,1)R	RETURN	ค่าของ R คือ 7=X+Y
CALL ADD(X,R,Z)	END	=3+4
WRITE(6,1)Z		ค่าของ Z คือ 10=X+R
		=3+7

ตัวอย่าง แสดงว่าค่าของตัวแปรในโปรแกรมหลักอาจเปลี่ยนค่าไปเนื่องจากซับรูทีน

โปรแกรมหลัก	โปรแกรมย่อย
:	SUBROUTINE TRI(X,C)
A=4	:
CALL TRI(A,B)	X=7
WRITE(6,1)A	C=3.1
:	RETURN
	END

ค่าของ A ตอนที่ CALL ซับรูทีนนั้นมีค่าเท่ากับ 4 ซึ่งจะกลายเป็นค่าของ X ในซับรูทีน แต่เมื่อการทำงานในซับรูทีนเสร็จแล้ว ตอนกลับมายังโปรแกรมหลักนั้น A จะมีค่าเท่ากับ 7 (คือเท่ากับค่าใหม่ของ X ซึ่งเป็นอาร์กิวเมนต์ที่สมนัยกัน)

ตัวอย่าง นิพจน์ซึ่งเป็นอาร์กิวเมนต์ในคำสั่ง CALL จะถูกหาค่าก่อนส่งไปยังซับรูทีน

โปรแกรมหลัก	โปรแกรมย่อย
	SUBROUTINE SUB4(A,B,C)
X=3.	
Y=2.	
CALL SUB4(X,X+Y,3.*Y)	ค่าของ A จะเป็น 3, ค่าของ B จะเป็น 5 และค่าของ C จะเป็น 6

ตัวอย่าง **โปรแกรมหลัก** **โปรแกรมย่อย**

```
DIMENSION A(10),B(10)          SUBROUTINE SUMIT(X,S)
                               DIMENSION  X(10)
CALL SUMIT(A,SUM1)             S=0
                               Do 10 I=1,10
CALL SUMIT(B,SUM2)          10 S=S+X(I)
                               RETURN
                               END
```

หลังจาก CALL แรก SUM1= $\sum_{i=1}^{10} Ai$ (ผลบวกของสมาชิกของแถวลำดับ A)

และหลังจาก CALL ที่ 2 SUM2= $\sum_{i=1}^{10} Bi$ (ผลบวกของสมาชิกของแถวลำดับ B)

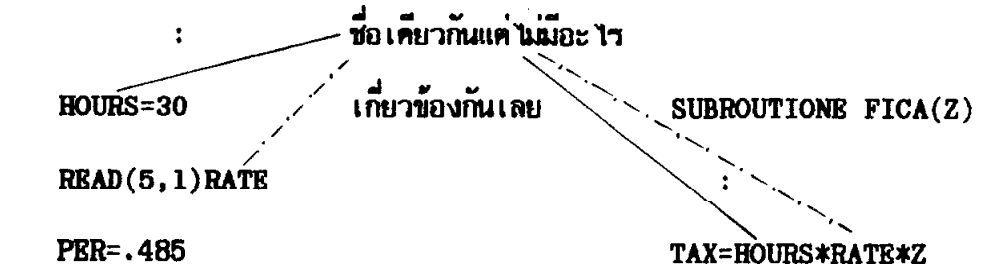
ตัวอย่าง คำสั่ง SUBROUTINE ที่ไม่ถูกต้อง (สมมุติคำสั่ง CALL ถูกต้อง)

คำสั่ง CALL	คำสั่ง SUBROUTINE	เหตุที่ผิด
CALL SUM(A,B,C)	SUBROUTINE SUM (A,B)	อาร์กิวเมนต์ต้องมี 3 ตัว
CALL PROD(A,3,N)	SUBROUTINE PROD(X,Y,I)	ต้องเป็น integer
CALL TOT(X,3.1,2)	SUBROUTINE TOT(X,B(1),J)	B(1) เป็นตัวแปรอาร์กิวเมนต์ไม่ได้
CALL SIS(T(3),M)	SUBROUTINE SIS(S,4)	4 เป็นตัวแปรอาร์กิวเมนต์ไม่ได้
CALL TW(A+B,C)	SUBROUTINE TW(X+Y,D)	X+Y เป็นตัวแปรอาร์กิวเมนต์ไม่ได้

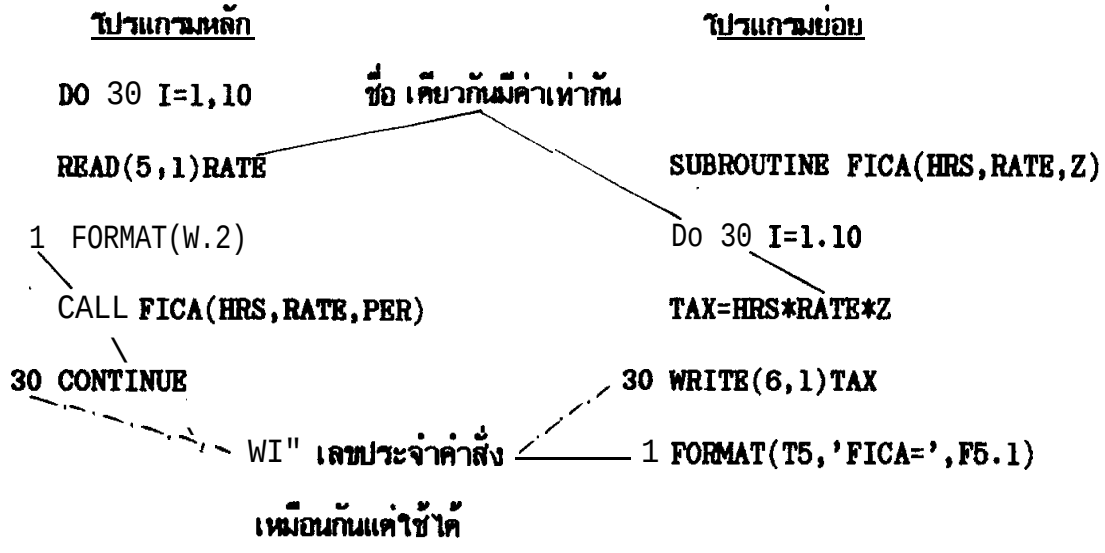
เมื่อ RETURN กลับไปยังโปรแกรมหลักคำสั่งที่อยู่ถัดจากคำสั่ง CALL จะถูกทำงาน การที่จะกลับไปยังจุดใดในโปรแกรมหลักนั้นขึ้นอยู่กับคำสั่ง CALL ที่เรียกซับรูทีน คำสั่งสุดท้ายของโปรแกรมย่อยซับรูทีนคือคำสั่ง END

เนื่องจากคอมไพเลอร์ถือว่าโปรแกรมย่อยและโปรแกรมหลักเป็นอิสระต่อกัน ดังนั้นการใช้ตัวแปรเดียวกันทั้งในโปรแกรมหลักและโปรแกรมย่อยต่าง ๆ จะไม่ทำให้คอมไพเลอร์สับสน ในตัวอย่างต่อไปนี้จะเห็นว่าทั้ง 2 โปรแกรมใช้ตัวแปร HOURS และ RATE แต่มันไม่มีความสัมพันธ์กันเลย

ตัวอย่าง โปรแกรมหลัก โปรแกรมย่อย



ในทำนองเดียวกันหมายเลขประจำคำสั่งนั้นใช้เหมือนกันได้เพราะอยู่คนละโปรแกรม



8.4.2.3 คำสั่ง COMMON

วิธีซึ่งเราใช้ในการส่งผ่านข้อมูลจากโปรแกรมหลักไปยังโปรแกรมย่อยคือการใช้อาร์กิว-เมนต์ อีกวิธีหนึ่งคือการใช้คำสั่ง COMMON ซึ่งจะสะดวกกว่าการที่จะต้องเขียนอาร์กิวเมนต์จำนวนหลายตัวเมื่อเรียกใช้ซับรูทีนหนึ่ง ๆ

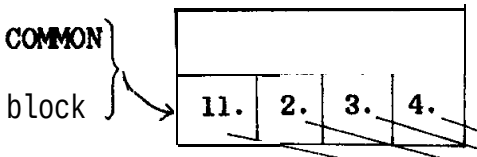
รูปทั่วไปของคำสั่งคือ

COMMON รายชื่อตัวแปร

โดยที่ รายชื่อตัวแปร อาจประกอบด้วยตัวแปรหรือชื่อของแถวลำดับก็ได้ แต่ต้องไม่เป็นดัมมีอาร์-กิวเมนต์และแอดชัวลอาร์กิวเมนต์

ตัวอย่าง

หน่วยความจำหลัก




```
COMMON G1,G2,G3,S                    SUBROUTINE ADDEM
DATA G1,G2,G3,S /11.,2.,3.,4./       COMMON X,Y,Z,W
CALL ADDEM                           Z=X+W*Y
WRITE(6,1)G3                         RETURN
                                     END
```

G1 และ X อ้างถึงสมาชิกตัวแรกใน COMMON block

G2 และ Y อ้างถึงสมาชิกตัวที่ 2 ใน COMMON block

G3 และ Z อ้างถึงสมาชิกตัวที่ 3 ใน COMMON block

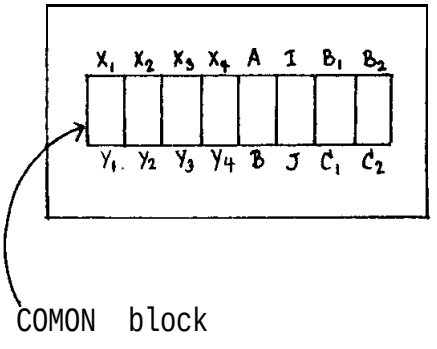
S และ W อ้างถึงสมาชิกตัวที่ 4 ใน COMMON block

ดังนั้นการทำงานในซับรูทีนจะหาค่าใหม่ของ Z=11.+(4.*2.)=11.+8.=19. และเมื่อ RETURN กลับไปยังโปรแกรมหลัก G3 จะมีค่าเท่ากับ 19 ด้วย

ตัวอย่าง

```
COMMON X(4),A,I,B(2)                 SUBROUTINE XYZ(RES)
                                     COMON Y(4),B,J,C(2)
CALL XYZ (TOTAL)                     RES=(Y(1)+B+C(2))/J
WRITE(6,4)TOTAL                      RETURN
                                     END
```

หน่วยความจำหลัก



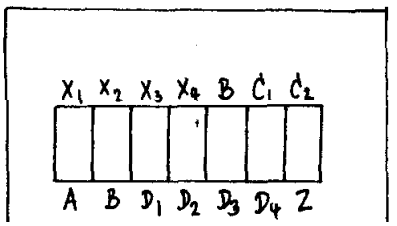
ผลของคำสั่ง CALL คือ TOTAL=(X(1)+A+B(2))/I

ตัวอย่าง

```
COMMON    X(,B,C(2)

          :

CALL COT
```




```
SUBROUTINE COT'

COMMON A,B,D(4),Z

          :
```

เราอาจใช้คำสั่ง COMMON แทนคำสั่ง DIMENSION ได้ดังในตัวอย่าง 3 ตัวอย่างแรก ต่อไปนี้เป็นการใช้อย่างถูกต้อง

```
DIMENSION X(100)          REAL X(100)          COMMON X(100)
COMMON X                  COMMON X
```

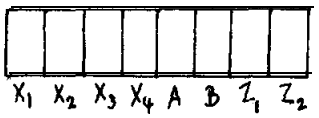
แต่อย่างไรก็ดีการใช้ต่อไปนี้ไม่ถูกต้องคือ

```
DIMENSION X(100)
COMMON X(100)
```

คำสั่ง COMMON นี้ใช้ได้กับโปรแกรมย่อยฟังก์ชันด้วย ในโปรแกรมหนึ่ง ๆ มีคำสั่ง COMMON หลาย ๆ คำสั่งได้ และการเรียงตัวของตัวแปรใน COMMON block ก็จะเรียงต่อเนื่องกันไป เช่น

```
COMMON X(4)
COMMON A,B
COMMON Z(2)
```

COMMON block



8.4.2.4 ลิมิตของมิติในรูปตัวแปร

ในการส่งผ่านแถวลำดับโดยใช้ชื่อของมันเป็นอาร์กิวเมนต์จากโปรแกรมหนึ่งไปยังโปรแกรมอื่น เราใช้ DIMENSION เพื่อกำหนดแถวลำดับ ถ้าขนาดของแถวลำดับในทั้ง 2 โปรแกรม (โปรแกรมหลักและโปรแกรมย่อย) เท่ากันก็ไม่เกิดปัญหา เช่น

โปรแกรมหลัก

```
DIMENSIONA(3,3),B(3,3),C(3,3)
K=3
CALL ADD(A,B,C,K)
```

โปรแกรมย่อย

```
SUBROUTINE ADD(X,Y,Z,K)
DIMENSIONX(3,3),Y(3,3),Z(3,3)
Do 10 I=1,K
DO 10 J=1,K
10 Z(I,J)=X(I,J)+Y(I,J)
RETURN
END
```

ซับรูทีนข้างต้นทำการบวกสมาชิกของแถวลำดับขนาด (3x3) เท่านั้น ดังนั้นเราจึงใช้ซับรูทีนนี้หาผลบวกของแถวลำดับขนาด เช่น (2x2), (5x5) ไม่ได้ พิจารณาตัวอย่างต่อไปนี้

ตัวอย่าง **โปรแกรมหลัก**

```
DIMENSION A(3,3),B(3,3),C(3,3)
DIMENSIONT(2,2),Q(2,2),R(2,2)
K=3
CALL ADD(A,B,C,K)
K=2
CALL ADD(T,Q,R,K) -->ใช้ไม่ได้
```

โปรแกรมย่อย

```
SUBROUTINE ADD(X,Y,Z,K)
DIMENSION X(3,3),Y(3,3),Z(3,3)
DO 10 I=1,K
DO 10 J=1,K
10   Z(I,J)=X(I,J)+Y(I,J)
R E T U R N
END
```

ขนาดไม่เท่ากัน

เราอาจแก้ไขโปรแกรมทั้ง 2 ข้างต้นดังนี้

โปรแกรมหลัก

```
DIMENSIONA(3,3),B(3,3),C(3,3)
DIMENSIONT(2,2),Q(2,2),R(2,2)
M=3
N=3
```

โปรแกรมย่อย

```
SUBROUTINE ADD(X,Y,Z,M,N)
DIMENSIONX(M,N),Y(M,N),Z(M,N)
DO 10 I=1,M
DO 10 J=1,N
```

```
CALL ADD(A,B,C,M,N)                10 Z(I,J)=X(I,J)+Y(I,J)
      :                                RETURN
CALL ADD(T,Q,R,2,2)                   END
```

ซับรูทีน ADD(X,Y,Z,M,N) สามารถบวกแถวลำดับขนาดใด ๆ ก็ได้และไม่จำเป็นต้องเป็นเมตริกซ์จตุรัส (square matrix)

ในกรณีของแถวลำดับแบบ 1 มิติ เราอาจใช้ดังนี้

<u>โปรแกรมหลัก</u> <u>โปรแกรมย่อย</u>

```
DIMENSIONA(10),B(50),C(200)        SUBROUTINE SUB(X,K)
                                   DIMENSION X(1) or X(K)
CALL SUB(A,10)
      :
N=36
CALL SUB(B,N)

CALL SUB(C,N)
```

8.4.2.5 <u>Named COMMON</u>

ในกรณีที่มีตัวแปรจำนวนมากใน COMMON block และโปรแกรมย่อยหลายโปรแกรมไม่ต้องการใช้ตัวแปรทุกตัวใน COMMON named (labeled) COMMON block จะทำให้โปรแกรมย่อยเข้าถึงตัวแปรเพียงบางตัวที่ต้องการเท่านั้น

<u>ตัวอย่าง</u> <u>โปรแกรมหลัก</u> <u>โปรแกรมย่อย</u>

```
COMMON /BLK1/X,Y/BLK2/Q,R          SUBROUTINE SUB1
                                   COMMON /BLK1/X,Y
CALL  SUB1
                                   END
```

```
CALL SUB2                          SUBROUTINE SUB2
                                   COMMON /BLK2/Q,R
                                     :
                                   END
```

โปรแกรมย่อยทั้ง 2 ไม่ได้ใช้ตัวแปรใน COMMON block ร่วมกันเลย แต่มันต่างก็ใช้ร่วมกับโปรแกรมหลัก

รูปทั่วไปของคำสั่ง COMMON คือ

```
COMMON [/ชื่อของ block/]รายชื่อตัวแปร[/ชื่อของ block/]รายชื่อตัวแปร ...
```

โดยที่ ชื่อของ block เป็นชื่อของ COMMON block (ตั้งตามหลักการตั้งชื่อตัวแปร) ถ้าไม่มีชื่อของ block ก็ไม่ต้องใส่เครื่องหมายขีดทับ (/) ทั้ง 2 อัน และคำสั่งก็จะเป็น blank (unnamed) COMMON ดังที่กล่าวมาแล้วใน 8.4.2.3

ตัวอย่าง

```
โปรแกรมหลัก                        โปรแกรมย่อย
COMMON HEAT,X/BLK1/KILO,Q          SUBROUTINE FIGURE
  :                                COMMON /BLK1/LIMA,R//ALFA,BET
CALL FIGURE                          :                  ↑    ↑
  :                                RETURN           HEAT   X
                                   END
```

ตัวอย่าง

```
COMMON /XYZ/X(10),Y(100)/ABC/RR(100)
COMMON A,B,C(10)/BLOCK3/Q,R,S
หรือ
COMMON /BLOCK3/Q,R,S//A,B,C(10)
```

ใส่ // ไว้เพื่อไม่ให้สับสนกับ block BLOCK3 และแสดงว่าเป็น blank COMMON

ในที่นี้มี COMMON block อยู่ 3 block ที่มีชื่อ XYZ, ABC และ BLOCK3 และ blank COMMON ซึ่งมีตัวแปร A, B และสมาชิกของ C อยู่ 10 ตัว

8.4.2.6 Block data

วิธีหนึ่งที่ใช้กำหนดค่าให้แก่ตัวแปรหรือแถวลำดับใน blank COMMON หรือ named COMMON คือใช้โปรแกรมย่อย BLOCK DATA

สมมุติว่าต้องการกำหนดให้สมาชิกแต่ละตัวของแถวลำดับ X ใน blank COMMON มีค่าเป็น 0 และตัวแปร BE ใน block ชื่อ BLK1 มีค่าเป็น 'GOOD' เราจะเขียนดังนี้

โปรแกรมหลัก	โปรแกรมย่อย
CHARACTER*4 BE	BLOCK DATA <------คำสั่งแรก
REAL X(10),Y(4)	COMMON R,S
COMMON X,F/BLK1/M,Y,BE/BLK2/	COMMON /BLK1/L,Z,NE
*R,ITEM	REAL R(10)/10*0./,Z(4)
	CHARACTER*4 NE/'GOOD'/
	END <------คำสั่งสุดท้าย

เราไม่ใช้คำสั่ง RETURN ในโปรแกรมย่อย BLOCK data คำสั่งที่ใช้ได้ในโปรแกรมย่อย BLOCK data (นอกจากคำสั่ง COMMON) คือ คำสั่ง IMPLICIT, คำสั่งเฉพาะ, คำสั่ง DIMENSION และคำสั่ง DATA ถ้าเราต้องการกำหนดค่าให้แก่ตัวแปรหรือแถวลำดับใน block ใด ๆ แล้วจะต้องกำหนด block นั้นใน BLOCK data ด้วย มิติและการระบุต่าง ๆ จะต้องสอดคล้องกับที่ระบุไว้ในโปรแกรมหลัก ดังนั้นใน BLOCK data จึงมี blank COMMON ด้วย และแถวลำดับแบบ 1 มิติชื่อ R (ซึ่งสมนัยกับแถวลำดับ X ในโปรแกรมหลัก) ซึ่งมีสมาชิก 10 ตัว และแต่ละตัวถูกกำหนดค่าให้เป็นศูนย์ เนื่องจาก BE ปรากฏใน COMMON BLK1 ซึ่งสมนัยกับ NE ซึ่งเรากำหนดให้เป็นตัวแปรอักขระโดยคำสั่ง CHARACTER และมีค่า 'GOOD' เราจึงกำหนด BLK1 ใน BLOCK DATA ด้วย แม้ว่าเราไม่ต้องการกำหนดค่าให้ L และ Z เนื่องจากไม่มีการกำหนดค่าตัวแปรใด ๆ ใน BLK2 ดังนั้นเราจึงไม่กำหนด BLK2 ใน BLOCK data

8.4.2.7 คำสั่ง ENTRY

วิธีที่ทำให้เราสามารถเข้าไปยังจุดต่าง ๆ ของโปรแกรมย่อย (แทนที่จะเข้าที่ต้นโปรแกรมย่อย) ตามต้องการ คือการใช้คำสั่ง ENTRY (เป็นคำสั่งไม่ปฏิบัติการ) ในโปรแกรมย่อย

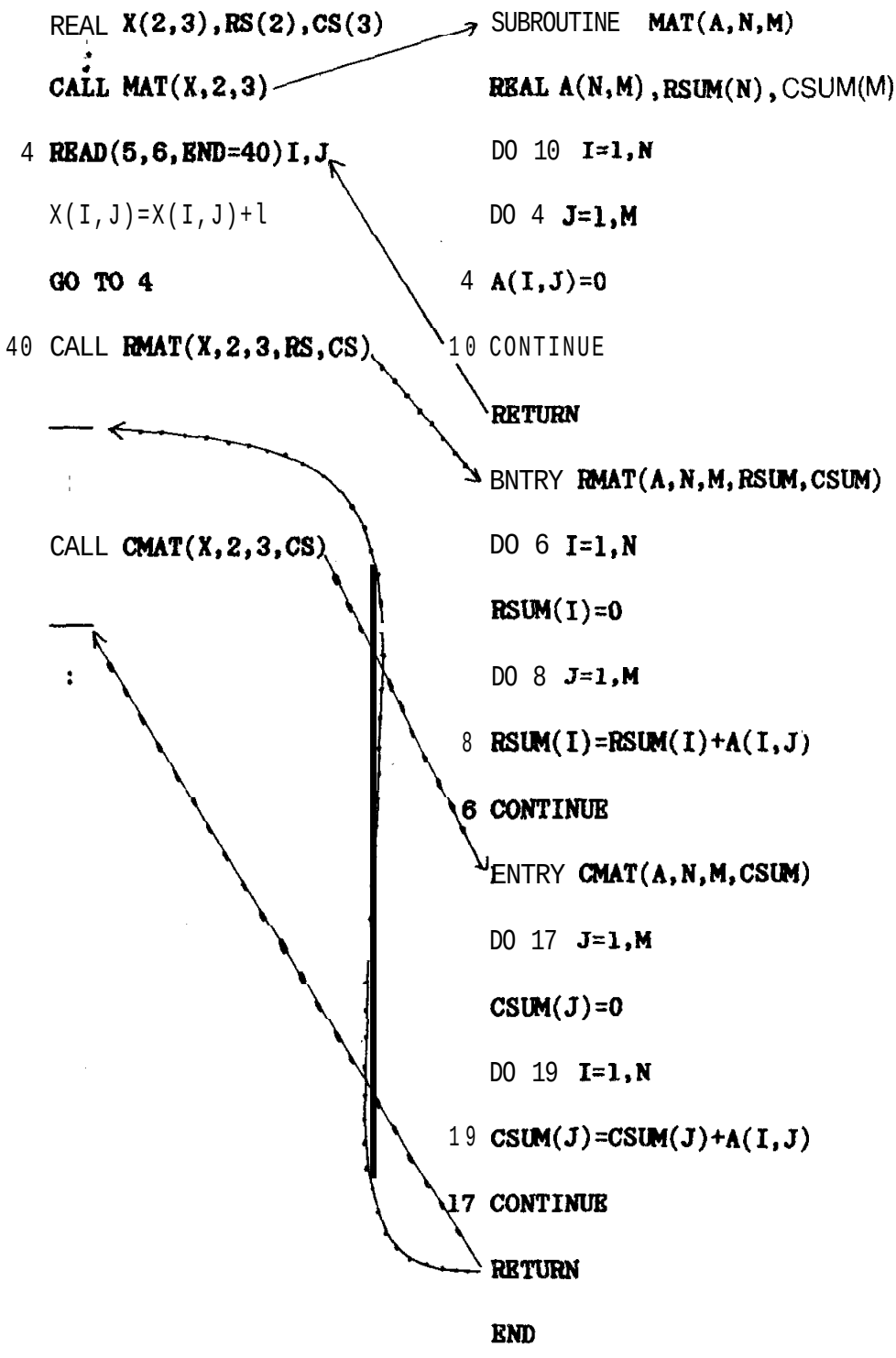
รูปทั่วไป

```
ENTRY ชื่อ(p1,p2,...)
```

โดยที่ ชื่อ เป็นชื่อที่ตั้งตามหลักการตั้งชื่อ (ในโปรแกรมย่อยฟังก์ชันชื่อนี้จะส่งค่ากลับไปยังโปรแกรมหลัก)

$p_1, p_2, \ldots$ เป็นตัวแปรอาร์กิวเมนต์ที่ใช้ในการสื่อสารระหว่างโปรแกรมหลักและโปรแกรมย่อย

ตัวอย่าง โปรแกรมหลัก โปรแกรมย่อย



ลูกศรในตัวอย่างข้างต้นแสดงจุดของการย้ายการควบคุมไปและกลับระหว่างโปรแกรมหลักและโปรแกรมย่อย การทำงานในโปรแกรมย่อยนั้นถ้าไม่มีคำสั่ง RETURN การปฏิบัติจะทำต่อเนื่องไปเรื่อย ๆ เช่นเมื่อ CALL RMAT นั้น จะปฏิบัติตามคำสั่งในโปรแกรมย่อยตั้งแต่คำสั่ง DO 6 I=1,N ถึง 6 CONTINUE แล้วปฏิบัติตามคำสั่ง DO 17 J=1,M เรื่อยไปจน RETURN กลับไปยังโปรแกรมหลัก ส่วนเมื่อ CALL CMAT การปฏิบัติตามคำสั่งในโปรแกรมย่อยจะเริ่มจาก DO 17 J=1,M ไปจนถึงคำสั่ง RETURN

8.4.2.8 คำสั่ง EXTERNAL

คำสั่ง EXTERNAL เป็นคำสั่งในการกำหนดชื่อโปรแกรมย่อยฟังก์ชันที่เราเขียนขึ้นเอง หรือชื่อของโปรแกรมย่อยซับรูทีน ให้เป็นแอคชวลอาร์กิวเมนต์ในโปรแกรมหลัก

รูปทั่วไปคือ

```
ENTERNAL ชื่อ1,ชื่อ2,...
```

โดยที่ ชื่อ1,ชื่อ2,... เป็นชื่อของโปรแกรมย่อยฟังก์ชันหรือชื่อของโปรแกรมย่อยซับรูทีน ซึ่งเราใช้เป็นแอคชวลอาร์กิวเมนต์

ตัวอย่าง สมมุติว่าเราต้องการหาค่าสูงสุดของฟังก์ชัน 3 ฟังก์ชันต่อไปนี้

1) $y = 3x^3 - 2x = -14$, x=2(3)10

2) $y = \sqrt{(x-4)(x+6)}$, x=-4(1)5

3) $y = \sin x$, x=.1(.1)1.1

โปรแกรมสำหรับปัญหาข้อนี้คือ

```
      EXTERNAL POLY,SQPOLY
      INTRINSIC SIN

      CALL MAXI(2.,10.,3.,POLY)
      CALL MAXI(-4.,5.,1.,SQPOLY)
      CALL MAXI(.1,1.1,.1,SIN)
      STOP
      END
-----------------------------------
      SUBROUTINE MAXI(INIT,TER,INC,FUNC)
```

```
      REAL       INIT,TER,INC,MAX
      MAX=FUNC(INIT)
    7 INIT=INIT+INC
      IF(INIT.GT.TER)GO TO 8
      IF(FUNC(INIT).GT.MAX) MAX=FUNC(INIT)
      GO TO 7
    8 WRITE(6,3)MAX
    3 FORMAT(1X,F10.2)
      RETURN
      END
```

```
      FUNCTION POLY(X)
      POLY=3*X**3-2*X**2-14
      RETURN
      END
      FUNCTION SQPOLY(X)
      SQPOLY=SQRT((X-4)*(X+6))
      RETURN
      END
```

8.4.2.9 คำสั่ง INTRINSIC(ในภาษาฟอร์แทรน 77)

คำสั่ง INTRINSIC เป็นคำสั่งที่ทำให้เรากำหนดชื่อฟังก์ชันภายใน (intrinsic function) เป็นแอคชวลอาร์กิวเมนต์ในโปรแกรมหลักได้

รูปทั่วไปคือ

INTRINSIC ชื่อฟังก์ชัน,ชื่อฟังก์ชัน,...

โดยที่ ชื่อฟังก์ชัน นั้นเป็นชื่อฟังก์ชันภายใน

ตัวอย่าง การใช้คำสั่ง EXTERNAL และ INTRINSIC

โปรแกรมหลัก

```
EXTERNAL CTN
INTRINSIC SIN,COS
 :
CALL TRIG(ANGLE,SIN,SINE)
CALL TRIG(ANGLE,COS,COSINE)
CALL TRIG(ANGLE,CTN,COTANG)
```

โปรแกรมย่อย

```
SUBROUTINE TRIG(X,F,Y)
Y=F(X)
RETURN
END
```

```
FUNCTION CTN(X)
CTN=COS(X)/SIN(X)
RETURN
END
```

8.4.2.10 คำสั่ง EQUIVALENCE

คำสั่ง EQUIVALENCE ใช้ในการกำหนดชื่อมากกว่า 1 ชื่อให้แก่ที่ 1 ที่ในหน่วยความจำหลัก เช่น EQUIVALENCE (A,B) จะทำให้ A และ B เป็นชื่อของที่ 1 ที่ในหน่วยความจำหลัก

```
EQUIVALENCE (A,B)
A=1.
B=3.
```

คำสั่ง WRITE(6,11)A จะพิมพ์ค่า 3. ให้

รูปทั่วไปคือ

```
EQUIVALENCE (รายชื่อตัวแปร)[,(รายชื่อตัวแปร),...]
```

โดยที่ รายชื่อตัวแปรในวงเล็บเดียวกันเป็นชื่อของที่ในหน่วยความจำหลักที่เดียวกัน ตัวแปรอาจเป็นสมาชิกของแถวลำดับก็ได้

สำหรับคอมพิวเตอร์บางระบบ คำสั่ง EQUIVALENCE ต้องอยู่หลังคำสั่ง DIMENSION และ COMMON แต่อาจจะมีลำดับต่างกันได้ซึ่งจะไม่กล่าวในที่นี้

ตัวอย่าง

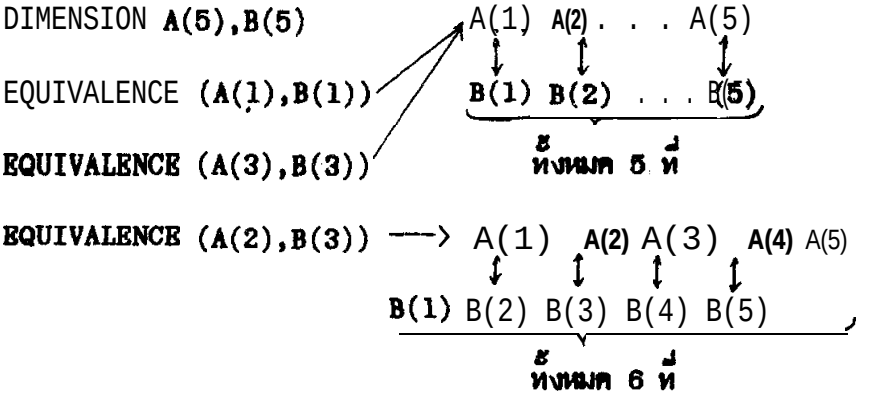
```
EQUIVALENCE (X,Y),(ZZZ,R,STU)
```

ทำให้ตัวแปร X และ Y หมายถึงที่ ๆ เดียวกัน หรือที่ ๆ หนึ่งมี 2 ชื่อคือ X และ Y

เราอาจ EQUIVALENCE ส่วนของแถวลำดับได้โดยการระบุตำแหน่งเริ่มต้นที่จะจับคู่กัน

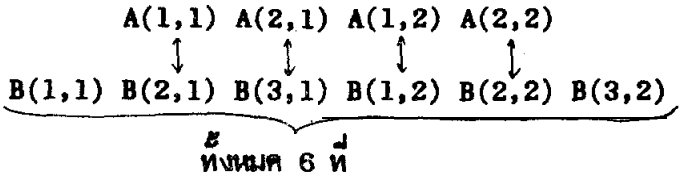
ตัวอย่าง

```
DIMENSION A(5),B(5)
EQUIVALENCE (A(1),B(1))
EQUIVALENCE (A(3),B(3))
EQUIVALENCE (A(2),B(3))
```



ตัวอย่าง

```
DIMENSIONA(2,2),B(3,2)
EQUIVALENCE (A(2,1),B(3,1))
```



คำสั่ง EQUIVALENCE นั้นบางครั้งเราใช้เมื่อโปรแกรมเมอร์ 2 คนหรือมากกว่า 2 คน เขียนส่วนของโปรแกรมอย่างเป็นอิสระต่อกัน และใช้ชื่อต่าง ๆ กันสำหรับรายการข้อมูลเดียวกัน

ดังนั้นแทนที่จะเขียนโปรแกรมใหม่โดยใช้ชื่อตัวแปรเดียวกัน เราใช้คำสั่ง EQUIVALENCE ช่วยได้ นอกจากนั้นคำสั่ง EQUIVALENCE ยังช่วยให้โปรแกรมเมอร์ใช้ที่ของแถวลำดับหนึ่ง (ซึ่งจะใช้เพียงหนเดียวแล้วเสร็จสิ้นไปเลยในช่วงแรกของโปรแกรม) เพื่อเก็บข้อมูลชุดใหม่ แต่การที่จะใช้ชื่อเดิมของแถวลำดับดังกล่าวเพื่อเก็บข้อมูลอย่างอื่นนั้นอาจเกิดการสับสน เราสามารถใช้คำสั่ง EQUIVALENCE เพื่อตั้งชื่อใหม่ให้แก่แถวลำดับได้

<u>ตัวอย่าง</u>

```
DIMENSIONA(10,10),AINVER(10,10),TEMP(10,10)
EQUIVALENCE (AINVER(1,1),TEMP(1,1))
```

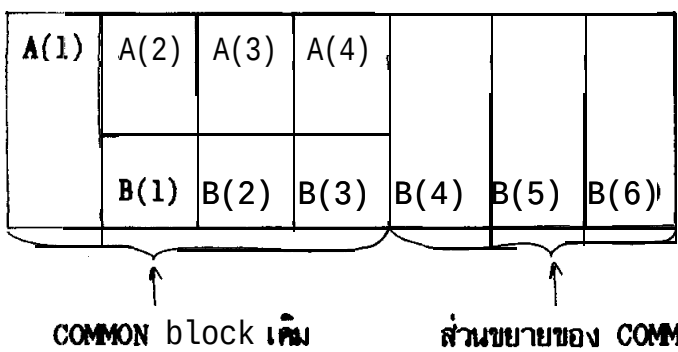
```
WRITE(6,1)AINVER      เมื่อสั่งพิมพ์ค่าของแถวลำดับ AINVER แล้วเราไม่ต้อง
  :                   การใช้ค่าของมันอีกเลย
READ(5,11)TEMP        เราต้องการที่ของ AINVER เพื่อเก็บอุณหภูมิที่จะอ่าน
                      เข้าไป ถ้าเรายังเรียกแถวลำดับนั้นว่า AINVER อาจ
                      เกิดสับสน เราจึงเรียกมันว่า TEMP
```

8.4.2.11 <u>การเกี่ยวข้องกันระหว่างคำสั่ง EQUIVALENCE และคำสั่ง COMMON</u>

เมื่อเราทำการ EQUIVALENCE ตัวแปรใน COMMON block COMMON block จะถูกขยายออกหลังจากสมาชิกตัวสุดท้ายของ block เท่านั้น

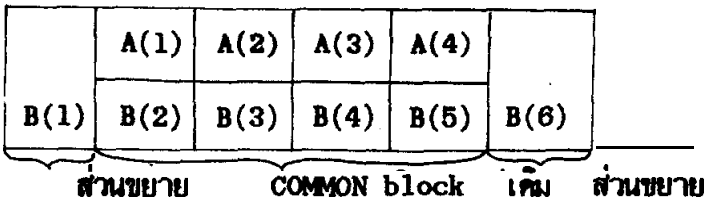
<u>ตัวอย่าง</u> คำสั่งที่ใช้ได้

```
DIMENSION A(4),B(6)
COMMON A
EQUIVALENCE (A(2),B(1))
```



ตัวอย่าง **คำสั่งที่ใช้ไม่ได้**

```
DIMENSION A(4),B(6)
COMMON A
EQUIVALENCE (A(2),B(3))
```



8.4.2.12 **คำสั่ง** SAVE **(ในภาษาฟอร์แทรน** 77)

รูปของคำสั่งคือ

SAVE [รายชื่อตัวแปร]

ตามปกติแล้วเมื่อส่งการควบคุมกลับมายังโปรแกรมหลัก ตัวแปรทั้งหมดในโปรแกรมย่อย **ซึ่งไม่ใช่อาร์กิวเมนต์จะไม่เก็บค่าไว้อีกต่อไป (undefined) (ยกเว้นตัวแปรที่ถูกกำหนดค่าโดยคำสั่ง DATA และตัวแปรใน COMMON block) นั่นหมายความว่าค่าของตัวแปรดังกล่าวจะไม่คงอยู่ในการเรียกใช้โปรแกรมย่อยในครั้งถัดไป ถ้าเราจำเป็นต้องคงค่าเหล่านั้นไว้จากการเรียกใช้ครั้งหนึ่งไว้สำหรับการเรียกใช้ครั้งถัดไป เราทำได้โดยการใช้คำสั่ง SAVE ในโปรแกรมย่อย**

ถ้าเราไม่ใส่รายชื่อตัวแปร แสดงว่าตัวแปรทั้งหมดในโปรแกรมย่อยจะถูกเก็บไว้

แบบฝึกหัดที่ 8

1. พิจารณาการกำหนดคำสั่ง FUNCTION และการเรียกใช้ฟังก์ชัน ถ้าข้อใดผิดจงให้เหตุผล

1.1) `X=MAX(2.1,3.1,4)` — `FUNCTION MAX(X,Y,Z)`

1.2) `IF(LOW(I,J,K))1,2,3` — `FUNCTION LOW(K,I,J)`

1.3)
```
REAL MALL
X=MALL(X,T)
```
— `FUNCTION MALL(X,T)`

1.4)
```
DIMENSION A(5)
INTEGER X,S
Z=A(X,K,3*S)
```
— `FUNCTION A(I,J,T)`

1.6) `T=BAD(1,2+S,3*I)` — `FUNCTION BAD(I,J,K)`

1.6) `M=TUT(SQRT(R),S)` — `FUNCTION TUT(RT,T)`

1.7) `S=MAD(3.,2*S,-J)` — `FUNCTION MAD(X,Y,K)`

1.8) `P=MAT(ABS(K),2,SIN(T))` — `FUNCTION (X,I,T)`

1.9) `S=COT(A,B,COT(3))` — `FUNCTION COT(X,Y,Z)`

1.10)
```
DIMENSION Q(100)
T=DAM(Q(1),Q,-4)
```
— `FUNCTION DAM(T(1),T,J)` / `DIMENSION T(100)`

1.11) `WRITE(6,11)FUNC(1.,2.)` — `FUNCTION FUNC(X,Y)`

1.12) `MIN=SUB(X,2*J,9.8)` — `FUNCTION (X,K,S)`

1.13) `SON=OF(A,GUN)` — `FUNCTION OF (A,NON)`

1.14)
```
DIMENSION B(5)
X=TIP(B(1),X,B(5)
```
— `FUNCTION TIP(A,B,C)` / `DIMENSION A(5)`

1.15)
```
DIMENSION A(5)
Z=CAN(T,3,6/L)
```
— `FUNCTION CAN(T,J,3)`

1.16) A=PAT(L,M,N)

```
FUNCTION PAL(N,ML)
N=l
M=l
L=3
RETURN
```

1.17) S=LARG(2,LARG(3,4))

```
FUNCTION LARG(I,J)
```

1.18) X=COT(X)

```
FUNCTION COT(X)
```

2. คำสั่ง FUNCTION ใดบ้างใช้ไม่ได้ จงบอกเหตุผลด้วย

2.1) FUNCTION AD(A,B,C+D)

2.2) FUNCTION A(A,B,C)

2.3) FUNCTION SORT(X,Y,Z)

2.4) FUNCTION B(A,C(1))

2.5) FUNCTION FUNCTION(X)

2.6) FUNCTION (A,AA,B)

2.7) FUNCTION SQRT(I)

2.8) FUNCTION C(A,B,B)

3. Statement function ใดใช้ไม่ได้ จงบอกเหตุผลด้วย

3.1) SOME(A(I),B)=A(I)*2

3.2) SRT(A,B)=SRT(A)+SRT(B)

3.3) T(Y)=Y**2+2

```
A(X)=T(X)+1
```

3.4) SQR(X)=X**0.5

3.5) ROOT=-B+SQRT(B*B-4*A*C)

```
C(B)=B**2+FUN
```

3.6) LONE(I,J,K)=I*J*K

```
L=LONE(1,2,3)+I*J*K
```

3.7) HI(1,2,3)=A+1+2+3

3.8) A(L)=L+AL+SIM

```
Y=A(K)+AL+SIM
```

3.9) ADD(X,Y,Z)=X+Y+Z

T=ADD(X*Y,T)

3.10) C(X+1,A)=(X+1)*3+A

3.11) MIX(K)=LOG(K+1)

S=MIX(3.1)+3

4. จงกำหนด statement function ซึ่งจะใช้แทนส่วนของคำสั่งที่มีหมายเลขประจำคำสั่ง 5,6,7,8 ได้

```
5 Y=3*X**2+2*X-1
  WRITE(6,1)Y
6 T=I*X**2+7*X-TOT
7 S=3*X**2+K(2)*X-SIN(T)
  SUM=S+T
6 IF(17*X**2+MIN1(A,B)*X-SQRT(A))1,2,3
```

5. จงบอกผลลัพธ์จากโปรแกรมต่อไปนี้

5.1) <u>บัตรข้อมูล</u>

+2+3-1

↑ สดมภ์ที่ 1

```
      IMPLICIT  INTEGER  (A-Z)
      READ(5,7)A,B,C
    7 FORMAT(2I2,I3)
      DO 10  I=1,2
      Y=(10-I)*POL(A,B,C,-I)
   10 WRITE(6,14)Y
   14 FORMAT(1X,I7)
      STOP
      END
```

```
      INTEGER FUNCTION POL(A,B,C,X)
      INTEGER A,B,C,X
      POL=A*X**2+B*X+C
      RETURN
      END
```

5.2) บัตาข้อมูล

บัตาที่ 1 11461733

บัตาที่ 2 1172845

บัตาที่ 3 1166142

บัตาที่ 4 1172

สดมภ์ที่ 1

```
      IMPLICIT INTEGER (A-Z)
      INTEGER ITEM(3)
      REAL PRICE(3)
      DO 10 I=1,3
   10 READ(5,11)ITEM(I),PRICE(I)
   11 FORMAT(I4,F4.2)
      READ(5,12)J
   12 FORMAT(I4)
      L=FIND(ITEM,J)
      IF(L.LT.0)GO TO 8
      WRITE(6,15)ITEM(L),PRICE(L)
   15 FORMAT(1X,I4,2X,F5.2)
      STOP
    8 WRITE(6,13)J
   13 FORMAT(1X,I4,1X,'COULD NOT BE FOUND')
      STOP
      END
```

```
      INTEGER FUNCTION FIND(A,B)
      INTEGER A(3),B
      FIND=-1
      DO 10 I=1,3
      IF(A(I).EQ.B)FIND=I
   10 CONTINUE
      RETURN
      END
```

6. โปรแกรมหลักและโปรแกรมย่อยต่อไปนี้ข้อใดบ้างที่มีข้อผิดพลาด จงอธิบายเหตุผลด้วย

โปรแกรมหลัก	โปรแกรมย่อย
6.1) INTEGER A(10),B(10) CALL SUB(A,B)	SUBROUTINE SUB(I,J) DIMENSION I(10),J(10) DO 10 K=1,10 10 I(K)=J(K) RETURN END
6.2) DIMENSION A(15) CALL SUBD(A,B)	SUBROUTINE SUBD(P,Q) DIMENSION Q(15) DO 10 I=1,15 10 Q(I)=P RETURN END
6.3) DIMENSION X(3,4) CALL SUBE(X)	SUBROUTINE SUBE(X,M,N) DIMENSION X(N,M)

‘6.4)
```
      DIMENSION  X(3.4)
```

```
      SUBROUTINE SUBF(X,A,B)
      DIMENSION X(A,B)
```

```
      CALL SUBF(X,3,4)
```

6.5)
```
      REAL   JSUM(10)
```

```
      SUBROUTINE (JSUM,K,R)
      DIMENSION JSUM(1)
```

```
      CALL SUB(JSUM,N,3.1)
```

```
      REAL JSUM
```

7. กำหนดโปรแกรมย่อยชับรูทีนชื่อ STAT ซึ่งทำการคำนวณสิ่งต่าง ๆ ต่อไปนี้คือ

$$SUMX = \sum_{i=1}^{n} x_i \ , \quad SUMY = \sum_{i=1}^{n} y_i \ , \quad SUMXX = \sum_{i=1}^{n} x_i^2 \ , \quad SUMYY = \sum_{i=1}^{n} y_i^2 \ , \quad SUMXY = \sum_{i=1}^{n} x_i y_i$$

```
      SUBROUTINE STAT(X,Y,N,SUMX,SUMY,SUMXX,SUMYY,SUMXY)
C     N : NUMBER OF ELEMENTS IN ARRAY X AND ARRAY Y
      DIMENSION X(N),Y(N)
      SUMX=0
      SUMY=0
      SUMXX=0
      SUMYY=0
      SUMXY=0
      DO 10 I=1,N
      SUMX=SUMX+X(I)
      SUMY=SUMY+Y(I)
      SUMXX=SUMXX+X(I)*X(I)
      SUMYY=SUMYY+Y(I)*Y(I)
   10 SUMXY=SUMXY+X(I)*Y(I)
      RETURN
```

END

จงเขียนโปรแกรมหลัก (Main program)

1) เพื่ออ่านค่าจากบัตร 2 บัตรโดยที่บัตรแรกมีค่าของ A 20 ตัวและบัตรที่ 2 มีค่าของ B 20 ตัว

2) เรียกใช้ SUBROUTINE STAT

3) คำนวณหาค่าของ $\overline{A} = \sum_{i=1}^{20} Ai/20$, $\overline{B} = \sum_{i=1}^{20} Bi/20$ และ

$$CORR(A,B) = r_{AB} = \frac{n\sum AiBi - \sum Ai \sum Bi}{\sqrt{\left(n\sum A_i^2 - \left(\sum A_i\right)^2\right)\left(n\sum B_i^2 - \left(\sum B_i\right)^2\right)}}$$

4) พิมพ์ผลตามรูปแบบดังนี้

STATISTICAL ANALYSIS

20 OBSERVATIONS

I	A	B
1	XX.X	XX.X
:	:	:
20	XX.X	XX.X

MEAN A = _____

MEAN B = _____

CORR(A,B) = _____

8. จากโจทย์ข้อ 12 แบบฝึกหัดที่ 6 จงใช้ SUBROUTINE STAT ในข้อ 7 ช่วยการเขียนโปรแกรม

9. กำหนดโปรแกรมย่อยชับรูทีนชื่อ ASORT เพื่อใช้ในการเรียงลำดับ (sorting) ข้อมูลเชิงปริมาณ (quantitative data) จากมากไปหาน้อย (descending order) ให้ จงเขียนโปรแกรมหลักเพื่ออ่านค่า x 20 ค่า และ y 30 ค่า แล้วเรียกใช้ ASORT กำหนดให้พิมพ์ผลดังนี้ (พิมพ์บรรทัดละ 10 ค่า)

UNSORTED ARRAY X :

X(1) X(2) . . . X(10)

X(11) X(12) . . . X(20)

SORTED ARRAY X :

__ __ . . .

__ __ . . .

UNSORTED ARRAY Y :

Y(1) Y(2) . . . Y(10)

__ __ . . . __

Y(21) Y(22) . . . Y(30)

SORTED ARRAY Y :

__ __ . . . __

__ __ . . .

__ __ . . . __

```
      SUBROUTINE ASORT(A,N)
C     BUBBLE SORT (DESCENDING ORDER)
C          A : ONE-DIMENSIONAL ARRAY
c          N : NUMBER OF ELEMENTS OF ARRAY A
      DIMENSION A(N)
      K=N-1
      DO 16 I=1,K
      L=N-I
      DO 16 J=1,L
      IF(A(J).GT.A(J+1))GO To 16
      TEMP=A(J)
```

```
      A(J)=A(J+1)
      A(J+1)=TEMP
   16 CONTINUE
      RETURN
      END
```

10. วิธีการเรียงลำดับอีกวิธีหนึ่งซึ่งมีประสิทธิภาพสูงคือวิธีของ Shell-Metzner หรือที่เรียกกันว่า shell sort ได้มีผู้ทำการเปรียบเทียบ shell sort และ bubble sort ไว้ว่า เมื่อทำการ sort เลข 100,000 จำนวน จะใช้เวลา 7.1 วันและ 15 นาที เมื่อใช้วิธี bubble และวิธีของ shell ตามลำดับ

จงเขียนโปรแกรมย่อยซับรูทีน เพื่อทำการ sort ตามวิธีของ Shell-Metzner จากแผนภูมิสายงานที่กำหนดให้ จากนั้นให้นักศึกษาสุ่มตัวเลข 20 จำนวนขึ้นเป็นข้อมูลเข้า แล้วเขียนโปรแกรมหลักเพื่อเรียกใช้โปรแกรมย่อยซับรูทีน แล้วพิมพ์ผลดังนี้

UNSORTED ARRAY :

<----10 ค่า ------->

<------10A-l ------->

SORTED ARRAY :

<----10 ค่า ------->

<------10 ค่า ------>

ผังโปรแกรมเพื่อ sort สมาชิก N ตัวของแถวลำดับ D