FORTRAN

(FORmula TRANslation)

ประวัติและที่มา

■ เริ่มต้นการคิดค้นครั้งแรกเมื่อปี คศ. 1953 โดยทีมงานของบริษัท IBM เพื่อที่จะใช้สำหรับเครื่องคอมพิวเตอร์ระดับ mainframe ของบริษัท

■ ต่อมาหลังจากนั้นอีก 3 ปี ได้มีการพัฒนาให้ดีขึ้น และเรียกว่า FORTRAN

■ จนกระทั่งปี คศ. 1962 ได้มีการพัฒนามาถึงรุ่น FORTRAN ซึ่งเป็นรุ่นที่มีชื่อเสียงและรู้จักกันอย่างแพร่หลาย

■ หลังจากนั้นยังคงมีการพัฒนาต่อมาเรื่อยๆ จาก FORTRAN 66, FORTRAN 77 และ FORTRAN 90/95

ทำไมต้องเรียนภาษาฟอร์แทรน

ภาษาคอมพิวเตอร์ที่มีโครงสร้างและนิยมใช้กัน มีอยู่ 2 ภาษา คือ

1. COBOL (พัฒนาเมื่อปี คศ. 1960) , COmmon Bussiness Oriented Language)

- ถูกออกแบบโดยเฉพาะสำหรับใช้งานด้าน data processing ในเชิงธุรกิจ
- ไม่เหมาะสำหรับการเขียนเป็นโปรแกรมด้านวิทยาศาสตร์

2. FORTRAN

- ได้รับการคิดค้นจากนักวิทยาศาสตร์และวิศวกร เพื่องานเฉพาะด้านนี้ ซึ่งตัวอย่างของการใช้ภาษาฟอร์แทรนในทางปฏิบัติ ได้แก่

2.1 ขั้นตอนการผลิตโครงสร้างของเครื่องบินโบอิ้ง 747, และ NASA Lunar capsule (ภาษาฟอร์แทรนถูกใช้ในการควบคุมการเคลื่อนที่ของ machine tools เช่น โปรแกรมของสมการการเคลื่อนที่ เป็นต้น)

2.2 การวิเคราะห์โครงสร้างของสะพานและหลังคา

2.3 โปรแกรมส่วนใหญ่ที่ใช้ในการวิเคราะห์ปัญหาทางวิศวกรรม (Fluid, Heat, Solid)

Etc.

องค์ประกอบของภาษาฟอร์แทรน

ประกอบด้วย 3 ส่วน ด้วยกัน คือ

1. ส่วนที่ใช้แจ้งรายละเอียด (Declaration section)

ส่วนนี้จะไม่มีการนำสั่งการทำงานแต่อย่างใด มีการใช้อยู่ 2 ช่วง คือ

1.1 กำหนดชื่อของโปรแกรม (Program statement)

1.2 ชี้แจงรายละเอียดในแต่ละช่วงของโปรแกรม (Comment statement)

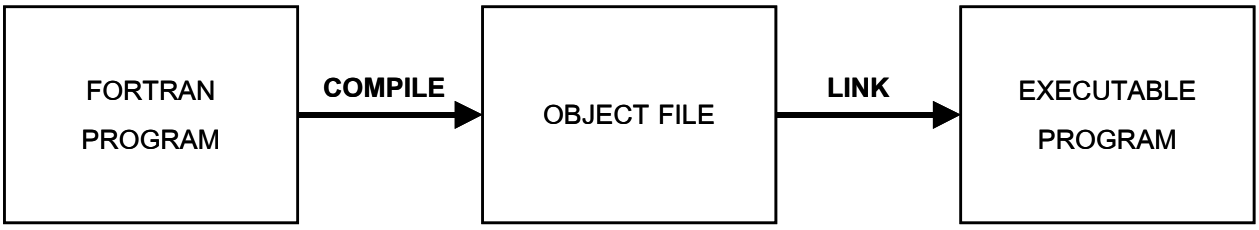
2. ส่วนที่ใช้สั่งการทำงาน (Execution section)
   ส่วนนี้เป็นการสั่งการทำงานในหลายรูปแบบ ได้แก่ การอ่าน/เขียนข้อมูล การคำนวณบวกลบคูณหาร

3. ส่วนที่ใช้หยุดการทำงาน (Termination section)
   ส่วนนี้เป็นการช่วยหยุดการทำงาน ประกอบด้วย 2 คำสั่ง คือ STOP และ END

การแปลงโปรแกรมภาษาฟอร์แทรนเพื่อใช้งาน

ประกอบด้วย 2 ขั้นตอน ภายหลังจากที่เขียนโปรแกรมภาษาฟอร์แทรนเสร็จแล้ว คือ

1. ตรวจสอบความถูกต้องโปรแกรมที่เขียนขึ้นตามหลักไวยกรณ์และแปลงภาษาฟอร์แทรนเป็นภาษาเครื่อง (Compiling) ทำให้ได้ไฟล์ออปเจค (object file) ออกมา
2. เชื่อมรวมคำสั่งต่าง ๆ หรืออื่น ๆ ที่เรียกใช้จาก system libraries (linking) ทำให้ได้ไฟล์ที่สามารถใช้งานได้ทันที (executable program) ส่วนใหญ่จะมีนามสกุลไฟล์ คือ filename.exe



ตัวอย่างโปรแกรม Fortran77

โปรแกรมง่ายๆข้างล่างนี้ จะทำหน้าที่แสดงผลบวกของตัวเลข 2 ชุด

LINE # COLUMN #

	1	2	3	4	5	6	7		72
1								PROGRAM FIRST	
2	C		T	H	I	S		IS OUR FIRST PROGRAM	
3								PRINT *, ‘ TYPE TWO NUMBERS SEPARATED BY A COMMA’	
4								READ *, A, B	
5								SUM = A + B	
6								PRINT *, ‘THE SUM OF’, A,’ AND’, B,’ IS’, SUM	
7								STOP	
8								END	

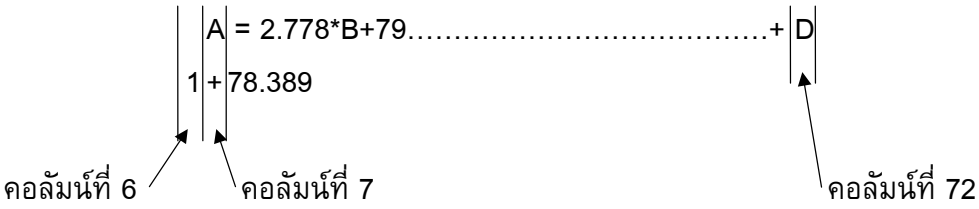
ในอดีตรูปแบบของการป้อนข้อมูลเข้าสู่เครื่องคอมพิวเตอร์จะอาศัยการ์ดเจาะรู (punched card) ซึ่งแต่ละการ์ดจะประกอบด้วย 80 คอลัมน์ และ 12 แถว

■ คอลัมน์ที่ 73 ถึง 80 จะไม่ถูกใช้ในปัจจุบัน เนื่องจากในอดีตมีการใช้ตำแหน่งดังกล่าวสำหรับการแสดงถึงอันดับของการ์ดเจาะรูที่เรียงอยู่ในเครื่องอ่าน

■ คอลัมน์ที่ 1 ถึง 6 ใช้สำหรับวัตถุประสงค์เฉพาะ เช่น

- C ที่ตำแหน่งคอลัมน์ที่ 1 หมายถึง “comment” card
- ตัวอักษรทุกตัว (ยกเว้น ศูนย์) ที่ตำแหน่งคอลัมน์ที่ 6 หมายถึง การต่อเนื่องจากบรรทัดก่อนหน้านี้

เช่น



ซึ่งจะเหมือนกับที่เขียนในรูปแบบอื่น คือ

A = 2.778*B+79.....+D+78.389

<u>บรรทัดที่ 1</u> PROGRAM FIRST

■ บรรทัดนี้แสดงถึงชื่อของโปรแกรมนี้โดยประกอบด้วยคำ “PROGRAM” และตามด้วยชื่อของโปรแกรม

■ ชื่อในโปรแกรมฟอร์แทรนจะมีเงื่อนไขดังนี้

- เริ่มต้นด้วยตัวอักษรเท่านั้น เช่น SUM, SUM1 แต่ไม่ใช่ 1SUM เป็นต้น
- ประกอบไปด้วยตัวอักษรและตัวเลขเท่านั้น เช่น SUM25 แต่ไม่ใช่ SUM% เป็นต้น
- ความยาวของชื่อสูงสุดไม่เกิน 6 ตัว เช่น SUM123 แต่ไม่ใช่ SUM1234 เป็นต้น

■ ช่องว่างระหว่างตัวอักษรไม่มีความหมาย สำหรับ FORTRAN Compiler เช่น

PR OGRA MFIRST จะมีความหมายเดียวกับ PROGRAM FIRST

หรือ SUM = A + B จะมีความหมายเดียวกับ SUM=A+B เป็นต้น

■ ความยาวของชื่อไม่เกิน 32 ตัวอักษร

บรรทัดแรกนี้สามารถที่จะไม่เขียนไว้ในโปรแกรมได้

<u>บรรทัดที่ 2</u> C THIS IS OUR FIRST PROGRAM

การเขียนโปรแกรมที่ดีจะต้องมีการใส่คำอธิบายประกอบโปรแกรม(comment cards)ไว้ให้เพียงพอ

<u>บรรทัดที่ 3</u> PRINT *, ‘TYPE TWO NUMBERS SEPARATED BY A COMMA’

- บรรทัดนี้เป็นการใช้ คำสั่งการพิมพ์ (PRINT)
- ข้อความต่างๆในเครื่องหมาย ‘_____‘ จะแสดงออกบนหน้าจอคอมพิวเตอร์
- คำสั่งการพิมพ์มีอยู่หลายแบบ ดังจะได้เรียนต่อไป

บรรทัดที่ 4 READ *, A, B

- บรรทัดนี้เป็นการใช้ คำสั่งการอ่านข้อมูล (READ) ซึ่งมีรูปแบบทั่วไปดังนี้

READ *, input - list

input - list คือตัวแปรหนึ่งตัวหรือมากกว่า ซึ่งจะถูกแยกด้วยเครื่องหมายจุลภาค(,)

- คำสั่งข้างบนจะทำการอ่านค่าตัวแปร 2 ตัว เข้ามาเก็บไว้เป็นค่า A และ B ตามลำดับ
- ตัวแปรที่เป็นตัวเลขจำนวนจะมีอยู่ 2 แบบ คือ

1. จำนวนเต็ม เช่น 1, 2, 57, 2001 เป็นต้น โดยปกติจำนวนเหล่านี้จะถูกแทนด้วยชื่อตัวแปรที่นำหน้าด้วย I, J, K, L, M, N เช่น

I = 57

IABC = 2001

2. จำนวนจริง เช่น 1.87, 2001.0 เป็นต้น โดยปกติจำนวนเหล่านี้จะถูกแทนด้วยชื่อตัวแปรที่นำหน้าด้วย A - H และ O - Z เช่น

AA = 1.87

ABC = 2001.0

- ดังนั้น บรรทัดที่ 4 นี้ จะต้องการให้มีการป้อนค่าจำนวนจริง ดังเช่น

1.87, 2001. <ENTER>

ซึ่งหมายความว่า

A = 1.87

B = 2001.

บรรทัดที่ 5 SUM = A + B

- บรรทัดนี้จะเป็นคำสั่งที่สั่งให้โปรแกรมทำอย่างใดอย่างหนึ่ง กล่าวคือ สั่งให้นำข้อมูลในตำแหน่ง A และ B มารวมกัน แล้วนำผลที่ได้มาใส่ไว้ที่ตัวแปร SUM

บรรทัดที่ 6 PRINT *, 'THE SUM OF', A, ' AND', B, ' IS', SUM

- บรรทัดนี้จะคล้ายกับบรรทัดที่ 3 แต่จะมีการพิมพ์จำนวนร่วมอยู่ด้วย และได้ผลเป็น

THE SUM OF 1.87 AND 2001. IS 2002.87

- จำนวนจริงเหล่านี้สามารถเขียนอยู่ในรูปแบบของ exponent เช่น

$1.87 = 1.87 \times 10^{0} = 1.87E0$

$= 0.187 \times 10^{1} = 0.187E1$

$= 0.00187 \times 10^{3} = 0.00187E3$

$= 187. \times 10^{-2} = 187.E-2$

- รูปแบบทั่วไปของคำสั่งพิมพ์ข้างบน คือ

PRINT *, output - list

output - list เป็นค่าตัวแปรหนึ่งตัวหรือมากกว่า ซึ่งจะแยกกันด้วยเครื่องหมายจุลภาค (,)

บรรทัดที่ 7 STOP

- เป็นการบอกคอมพิวเตอร์ว่าหยุดการประมวลผลทุกอย่าง

บรรทัดที่ 8 END

- แสดงให้เห็นว่าไม่มีบรรทัดหลังจากนี้แล้ว

ความผิดพลาดในโปรแกรม

ถ้าในบรรทัดที่ 4 เกิดพิมพ์ผิดเป็น

READ A, B (ลืม * หลัง READ)

เมื่อ compiler ทำการแปลงคำสั่งนี้ และพบว่าคำสั่งนี้ไม่สามารถจัดอยู่ในรูปแบบใดๆของคำสั่งการอ่าน ก็จะมีการแสดงข้อความแจ้งบนหน้าจอ เช่น

*** Syntax error

หรือบาง compiler สามารถแจ้งข้อความได้ละเอียดมากกว่า คือ

*** READ not followed by asterisk

แบบหลังนี้เรียกว่า compilation error ซึ่งช่วยให้ง่ายต่อการแก้ไข

แต่ถ้าหากว่าพิมพ์ผิดที่

SUM = A - B (แทนที่จะเป็น SUM = A + B)

แบบนี้ compilers จะไม่สามารถตรวจสอบได้ และคำตอบที่ได้ก็จะผิด

สรุป

- โปรแกรมฟอร์แทรนสามารถเขียนตัวอักขระได้สูงสุด 72 ตัวต่อหนึ่งบรรทัด
- แต่ละคำสั่งจะเขียนอยู่ภายในคอลัมน์ที่ 7 - 72, ส่วนคอลัมน์ที่ 1 - 6 จะเก็บไว้สำหรับวัตถุประสงค์เฉพาะ
- การใส่ C หรือ เครื่องหมายดอกจัน (*) ไว้ที่คอลัมน์ที่ 1 จะทำให้ทั้งบรรทัดเป็นคำอธิบายประกอบของโปรแกรม
- โดยทั่วไปอักษรตัวแรกของชื่อตัวแปรจะเป็นตัวกำหนดประเภทของจำนวน (A-H และ O-Z หมายถึง จำนวนจริง, ส่วน I-N หมายถึงจำนวนเต็ม)
- ทุกโปรแกรมจะต้องจบด้วยคำสั่ง END

กฎของการเขียนโปรแกรมได้ดี

- วางแผนการเขียนล่วงหน้า
- เขียน comment cards สำหรับขั้นตอนต่างๆในโปรแกรม
- เขียนโปรแกรมให้ง่ายต่อการเข้าใจ
- ทดสอบทั้งโปรแกรม
- เขียนรายละเอียดว่าโปรแกรมนี้ใช้อย่างไร
- มีความรู้สึกสนุกกับการเขียนโปรแกรม ถึงแม้ว่าจะมีอุปสรรค

คำสั่งต่าง ๆ ที่มีการใช้งานบ่อย ๆ

I/O STATEMENT (READ, PRINT, WRITE & FORMAT STATEMENT)

ถ้าเราต้องการพิมพ์ผลลัพธ์ให้ออกมาอย่างมีระเบียบ เราสามารถใช้คำสั่ง WRITE & FORMAT ร่วมกัน เช่น

เราสามารถแทนคำสั่งนี้

```
PRINT  *, 'STUDENT  NO.', I, ' HAS SCORE ', X
```

ด้วยคำสั่ง

```
      PRINT  5,   I, X
    5 FORMAT(' STUDENT NO. ', I5, '  HAS SCORE ', F6.1)
```

หรือ

```
      WRITE(6,202)  I, X
  202 FORMAT(' STUDENT NO. ', I5, '  HAS SCORE ', F6.1)
```

จะได้ผลลัพธ์ที่เหมือนกันออกมาดังนี้

```
STUDENT  NO.    1   HAS SCORE  91.0
```

จากตัวอย่างข้างบน คำสั่ง PRINT จะใช้กับกรณีแสดงผลลัพธ์ออกทางหน้าจอคอมพิวเตอร์
ส่วนคำสั่ง WRITE สามารถใช้ได้กับการแสดงผลลัพธ์ออกทางหน้าจอคอมพิวเตอร์ ทางไฟล์ข้อมูล หรือทางเครื่องพิมพ์

รูปแบบของคำสั่ง READ & WRITE จะคล้ายคลึงกันคือ

READ (unit number, k) ตัวแปรต่างๆ
เช่น READ (5,*) I, Y

WRITE (unit number, k) นิพจน์ต่างๆ (ได้แก่ ตัวแปร, ตัวอักษรที่ต้องการแสดง)
เช่น WRITE (6,*) I, Y

unit number คือตัวเลขจำนวนเต็มบวก (ตั้งแต่ 1 ขึ้นไป) ที่ใช้แทน Input/Output Device เพื่อสะดวกในการกล่าวอ้างต่อไปในโปรแกรม เช่น

unit number = 5 หรือ * จะหมายถึง การรับข้อมูลเข้ามาจากคีย์บอด จะใช้กับ READ

unit number = 6 หรือ * จะหมายถึง การแสดงข้อมูลออกทางหน้าจอคอมพิวเตอร์ จะใช้กับ WRITE

k จะเป็นได้ทั้ง Label ของคำสั่ง FORMAT หรือ การบ่งบอกประเภทของตัวแปรต่างๆที่อยู่หลังวงเล็บและลักษณะการจัดแสดงอื่นๆที่ต้องการ (เช่น คำสั่ง X ที่กำหนดการเว้นช่องไฟของการพิมพ์ เป็นต้น)

ตัวอย่างของการใช้คำสั่ง READ และ WRITE

1. WRITE (*,*) I, J, S
   k เท่ากับ * หมายถึง ตัวแปรที่ต้องการแสดงเป็นชนิดไหนก็ได้
   สำหรับกรณี I, J เป็นตัวแปรแบบ integer ส่วน S เป็นตัวแปรแบบ real

2. WRITE (*,*) ‘ This is My Test File = ’, NAME
   กรณีนี้ NAME เป็นตัวแปรแบบ character

3. READ (*,’(A)’) FILNAM
   k เท่ากับ ‘(A)’ หมายถึง FILNAM ที่จะอ่านเข้ามาเป็นตัวแปรแบบ character

4. WRITE (*,’(5X, 2I6, 2X, E10.4)’) I, J, X

   จะได้ผลเหมือนกับ

   WRITE (*, 100) I, J, X

   100 FORMAT (5X, 2I6, 2X, E10.4)

   กรณีนี้ 5X หมายถึงเว้นช่องไฟไป 5 ช่อง, 2I6 คือการพิมพ์ค่า I, J เป็น integer 6 หลัก จากนั้นเว้นช่องไฟไปอีก 2 ช่อง จึงค่อยพิมพ์ค่า X ซึ่งเป็นค่าจำนวนจริงในรูปแบบ E

INTEGER, REAL AND CHARACTER DATA

ข้อมูลแบบ Character สามารถประกอบด้วย

1. ตัวอักษรภาษาอังกฤษแบบตัวพิมพ์ใหญ่ 26 ตัว คือ A – Z
2. ตัวอักษรภาษาอังกฤษแบบตัวพิมพ์เล็ก 26 ตัว คือ a – z
3. ตัวเลข 10 ตัว คือ 0 – 9
4. สัญลักษณ์อื่นๆ เช่น “ , ( ) { } [ ] ! @ # $ % ^ & * .
5. ตัวอักษรเฉพาะในแต่ละภาษา เช่น à ç ë

เครื่องคอมพิวเตอร์ PC ในปัจจุบันสามารถเก็บค่าตัวแปรแบบจำนวนเต็มได้ขนาด 1, 2, 4 bytes ดังนั้นจะรองรับค่าจำนวนเต็มที่อยู่ระหว่าง –2,147,483,648 ถึง 2,147,483,647 เมื่อถูกจัดเก็บแบบ 4 bytes

ส่วนจำนวนจริงที่ถูกเก็บแบบ 32 bits (4 bytes) จะมีค่าอยู่ระหว่าง 10^{-38} ถึง 10^{38} และเก็บค่าที่มีเลขนัยสำคัญได้ 7 ตัว ดังนั้นหากตัวเลขจำนวนจริงที่จัดเก็บมีเลขนัยสำคัญมากกว่า 7 จะก่อให้เกิดความผิดพลาดจากการปัดเศษ (round-off error) เช่น ตัวเลขจริง คือ 12345678.9 แต่เครื่องคอมพิวเตอร์จะมองเป็น 12345680.0 เป็นต้น

ภาษาฟอร์แทรนจะกำหนดให้ตัวแปรที่ขึ้นต้นด้วยตัวอักษร I ถึง N เป็นจำนวนเต็ม
ส่วนตัวแปรที่ขึ้นต้นด้วยตัวอักษร A - H และ O - Z เป็นจำนวนจริง เช่น

```
NSTART = 1
SALARY  =  8256.25
APPLES  =  20                    (ค่าของ APPLES จะถูกกำหนดให้เท่ากับ 20.0)
```

ซึ่งถ้าต้องการกำหนดให้ APPLES เป็นจำนวนเต็ม ก็สามารถทำได้โดยกำหนด

```
INTEGER  APPLES
```

ที่ช่วงเริ่มต้นของโปรแกรม ซึ่งหลังจากนั้น APPLES จะกลายเป็นจำนวนเต็ม และหากถูกหารด้วยจำนวนเต็มก็จะได้คำตอบเป็นจำนวนเต็มเช่นเดียวกัน เช่น

```
APPLES/5     ==>     20/5  =  4
```

แต่ควรระวัง หากถูกหารแล้วได้ผลลัพธ์ที่ไม่เป็นจำนวนเต็ม เช่น

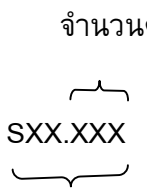
```
APPLES/6     ==>     20/6  =  3.3333     ==>   3
APPLES/7     ==>     20/7  =  2.86       ==>   2
```

จะเกิดการตัดตัวเลขหลังจุดทศนิยมออก แต่ไม่ใช่เป็นการปัดเศษ (**ต้องระวัง**)
แต่ถ้าต้องการกำหนดให้ค่าคงที่ INCOME มีค่าเท่ากับ 8256.25 ซึ่งเป็นจำนวนเต็ม เราสามารถกำหนดได้ว่า

```
REAL  INCOME
```

ที่ช่วงเริ่มต้นของโปรแกรมเช่นเดียวกัน
รูปแบบของค่าจำนวนจริงสามารถถูกระบุในคำสั่ง FORMAT ได้หลายรูปแบบ ดังนี้

1. รูปแบบ Fw.d



S คือตำแหน่งที่ต้องเผื่อไว้สำหรับค่าจำนวนจริงบวก หรือ ลบ (+/-)
เช่น F7.3

2. รูปแบบ E(w+4).d

ตัวเลข 4 ที่บวกเพิ่มมาจาก ESXX จำนวน 4 ตัว ที่จะต้องกำหนดแน่นอนลงไป
เช่น E9.4

3. รูปแบบ D(w+4).d

ลักษณะจะเหมือน รูปแบบ E(w+4).d แต่การใช้งานจะเหมาะกับจำนวนจริงที่ถูกระบุว่าเป็น double precision (หรือ real*8)ตั้งแต่ต้นโปรแกรม และทั้ง 2 รูปแบบต่างก็สามารถแสดงเลขนัยสำคัญได้มากกว่า 7 ตัวเช่นกัน

<u>ตัวอย่าง</u> : ถ้าเรามีค่าอยู่ค่าหนึ่ง คือ 0.0000361764 ดังนั้น

F10.4	จะหมายถึง	0.0000	(ทั้งหมด 10 ตำแหน่ง)
F12.6	จะหมายถึง	0.000036	(ทั้งหมด 12 ตำแหน่ง)
F14.8	จะหมายถึง	0.00003618	(ทั้งหมด 14 ตำแหน่ง)
E10.4	จะหมายถึง	0.3618E-04	(ทั้งหมด 10 ตำแหน่ง)
E12.6	จะหมายถึง	0.361764E-04	(ทั้งหมด 12 ตำแหน่ง)
E14.8	จะหมายถึง	0.36176400E-04	(ทั้งหมด 14 ตำแหน่ง)

สำหรับตัวแปรอื่นๆในภาษาฟอร์แทรนที่มีใช้กัน ได้แก่

LOGICAL data มีอยู่ 2 ค่า คือ True หรือ False เช่น

```
LOGICAL DONE
DONE = .FALSE.  เป็นต้น
```

COMPLEX data ก็คือจำนวนเชิงซ้อน (a + bi) เช่น

```
COMPLEX C
C = (1.5, 4.0)          โดย a = 1.5, b = 4.0 เป็นต้น
```

<u>ตัวอย่าง</u> ต้องการหาพื้นที่ของวงกลม จากสูตร Area = $\pi\frac{D^2}{4}$

เราสามารถเขียนโปรแกรมได้ดังนี้

```
PROGRAM  CIRCLE
PI  =  3.141593
READ  *,  D
AREA  =  PI*D*D/4.
PRINT  *, A
STOP
END
```

เราสามารถกำหนดค่าของ π ได้ด้วยคำสั่ง DATA เช่น

```
PROGRAM  CIRCLE
DATA  PI/3.141593/
READ  *,  D
```

คำสั่ง DATA จะมีประโยชน์มากเมื่อในโปรแกรมมีการใช้ค่าคงที่จำนวนมาก และช่วยทำให้โปรแกรมดูง่ายขึ้น เช่น

```
DATA  PI, PI3BY4/3.141593,2.356195/
```

หรือ `DATA  I, J, K, L, M, N/6*0/` ==> กำหนดค่าศูนย์ 6 ตัว

เราสามารถใช้คำสั่ง PARAMETER ได้เช่นกัน ดังนี้

```
PARAMETER  (PI = 3.141593, PI3BY4 = 2.356195)
```

หรือ PARAMETER (PI = 3.141593, PI3BY4 = 3*PI/4.)

หรือ PARAMETER (PI = 4*ATAN(1.), PI3BY4 = 3*PI/4.)

นิพจน์คณิตศาสตร์

นิพจน์คณิตศาสตร์จะถูกใช้ในการคำนวนเพื่อให้ได้ผลลัพธ์ที่เป็นจำนวนออกมา ตัวอย่างเช่น

SUM = A + B

I = I + 1 เช่น I = 5 ==> I = 6

R = S*T (R = S x T) ==> เป็นนิพจน์การคูณ

Q = S/T (Q = S ÷ T) ==> เป็นนิพจน์การหาร

U = S**2 ($U = S^2$) ==> เป็นนิพจน์เลขยกกำลัง

ถ้ามีนิพจน์ที่ซับซ้อน เช่น

D = 4.0 + 6.0 * 2.0

ซึ่งจะหมายถึง

D = (4.0 + 6.0)*2 = 10.0*2.0 = 20.0

หรือ

D = 4.0 + (6.0*2.0) = 4.0 + 12.0 = 16.0

ดังนั้นจะต้องมีการจัดลำดับว่านิพจน์ไหนจะกระทำก่อน ดังนี้

1. ** เป็นนิพจน์ที่กระทำเป็นอันดับแรก
2. * และ / เป็นนิพจน์ที่กระทำเป็นอันดับที่สอง
3. + และ - เป็นนิพจน์ที่กระทำเป็นอันดับสุดท้าย

ตัวอย่างเช่น

F = A + B/C

จะหมายถึง $F = A + \frac{B}{C}$

แต่ถ้าต้องการหาค่า

$$F = \frac{A + B}{C}$$

เราสามารถใช้วงเล็บเปิดปิดเข้ามาช่วย ดังนี้

F = (A + B)/C

เพราะนิพจน์ที่อยู่ภายในวงเล็บจะถูกกระทำก่อนเป็นอันดับแรกสุด

เช่น

F = A*((B + C)**D - E) จะหมายถึง $F = ((B + C)^D - E)$

ตัวอย่าง

จงเขียนโปรแกรมเพื่ออ่านค่าอุณหภูมิเป็นองศาเซลเซียส และแปลงให้เป็นองศาฟาเร็นไฮซ์ โดยใช้สูตร

F = C + 32

```
      PROGRAM  CONVERT
C
C     A program to convert a Celsius temperature to Fahrenheit
C
      PRINT  *, 'PLEASE TYPE IN CELSIUS TEMPERATURE'
      READ  *, TEMPC
      TEMPF  =  9.0*TEMPC/5.0 + 32.0
      PRINT  *, 'C=', TEMPC, 'F=', TEMPF
      STOP000
      END
```

หมายเหตุ : INTEGER คูณกับ INTEGER จะได้ INTEGER

REAL คูณกับ REAL จะได้ REAL

INTEGER คูณกับ REAL จะได้ REAL

เช่น เราสามารถใช้

TEMPF = 9*TEMPC/5 + 32

สรุป

- นิพจน์คณิตศาสตร์จะเรียงลำดับดังนี้

  ** (เครื่องหมายยกกำลัง)

  * และ / (เครื่องหมายคูณกับหาร ตามลำดับ)

  + และ - (เครื่องหมายบวกกับลบ ตามลำดับ)

  และถ้ามีวงเล็บเปิดปิด จะทำในวงเล็บก่อนเป็นอันดับแรก

- การหารจำนวนเต็มทำให้เกิดการปัดเศษทิ้ง ช่วยในการกำหนดเลขคู่ หรือคี่ขึ้นมาได้ เช่น

  ถ้า I เป็นเลขคี่ ดังนั้น I - (I/2)*2 = 1

  ถ้า I เป็นเลขคู่ ดังนั้น I - (I/2)*2 = 0

- ค่าจำนวนเต็มจะกลายเป็นจำนวนจริงทันที เมื่อมีการผสมนิพจน์จำนวนเต็มและจำนวนจริง
- คำสั่ง DATA เป็นการกำหนดค่าเริ่มต้นให้กับตัวแปร

- โดยปกติตัวแปรที่เริ่มต้นด้วย A - H และ O - Z คือจำนวนจริง
  และ I - N คือจำนวนเต็ม
  ยกเว้นแต่จะมีการกำหนดลงไปเพิ่มเติม

REAL ____, ____, ____ , ____
INTEGER ____, ____ , ____ , ____

<u>การตัดสินใจ</u> (ใช้ คำสั่ง IF และ GOTO อย่างง่ายๆ)

ตัวอย่าง จงหาค่าของ SIN X (ซึ่ง X มีหน่วยเป็น radians) ซึ่งสามารถแสดงในรูปแบบของอันดับไม่จำกัด คือ

SIN X = X -

ซึ่ง n! = n*(n-1)*(n-2)*……..*2*1

<u>คำถาม</u>
- จะใช้จำนวนเทอมเท่าไหร่ในการคำนวน
- จะหยุดโปรแกรม เมื่อไหร่ และอย่างไร

หมายเหตุ : SIN (45°) = SIN(π/2) = SIN(0.7854) = 0.707 = 1/$\sqrt{2}$

โปรแกรมที่ต้องการอาจเขียนได้ดังนี้

```
      PROGRAM  SINX
      READ  *, X
      SUM  =  0.
      DO  10  N=1,10000              <-- วนรวมทั้งหมด 10000 ครั้ง
      TERM  =                        <-- เช่น  X,



      IF (TERM.LT.0.00001)  GOTO  20
      SUM  =  SUM  +  TERM
   10 CONTINUE
   20 WRITE  *
      STOP
      END
```

- ถ้า TERM น้อยกว่า 0.00001 จะกระโดดออกมาที่ statement label 20 และทำคำสั่ง WRITE ต่อไป
- N อาจจะมีค่าเพียง 15 เมื่อมีการกระโดดออกมาจาก DO Loop

<u>นิพจน์ตรรกะ</u> (Logical Expressions)

```
IF (TERM.LT.0.00001)    GOTO  20
```

.LT.	หมายถึง	$<$	(น้อยกว่า)
.LE.	หมายถึง	$\leq$	(น้อยกว่าหรือเท่ากับ)
.EQ.	หมายถึง	$=$	(เท่ากับ)
.NE.	หมายถึง	$\neq$	(ไม่เท่ากับ)
.GT.	หมายถึง	$>$	(มากกว่า)
.GE.	หมายถึง	$\geq$	(มากกว่าหรือเท่ากับ)

รูปแบบที่ซับซ้อนอื่นๆ เช่น

```
IF ((TERM.LT.0.00001).OR.(N.GE.25))    GOTO  20
```

ถ้าเทอมใดเทอมหนึ่งเป็นจริง ก็จะกระโดดไปที่ 20

```
IF ((TERM.LT.0.00001).AND.(N.GE.25))    GOTO  20
```

ต้องเป็นจริงทั้ง 2 เทอม จึงจะกระโดดไปที่ 20

```
IF (TERM.LT.0.00001)    TERM  =  0.

IF (N.LT.10000)   THEN
   PRINT  *, 'STILL  ADDING"
ELSE
   PRINT  *, 'FINISH  ADDING"
END IF
```

ARRAYS

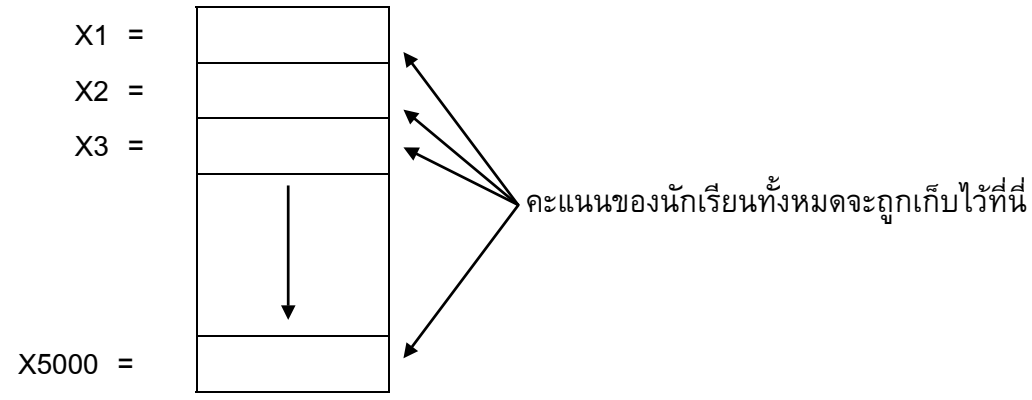
ตัวอย่าง คำนวนหาคะแนนเฉลี่ยของนักเรียนทั้งหมดในห้องเรียน

```
      PROGRAM  AVG
      SUM  =  0.
      N  =  0.
      DO  10  I=1,5000
      READ  *,  X
      IF(X.LT.0.)  GOTO  20          ถ้า คะแนน มีค่าลบ จะกระโดดออกจาก Loop
      SUM  =  SUM + X
      N  =  N + 1
10    CONTINUE
20    CONTINUE
      AVER  =  SUM/N
      PRINT  *, 'AVERAGE  SCORE  IS', AVER
      STOP
      END
```

หมายเหตุ :

1. โปรแกรมข้างบนจะพิมพ์คะแนนเฉลี่ยออกมาเท่านั้น
2. ผู้ใช้จะไม่รู้ว่าตนเองได้พิมพ์ข้อมูลผิดเข้าไป เนื่องจาก
   - ค่า X จะถูกเขียนทับ ในแต่ละครั้งที่มีการพิมพ์
   - ค่า X ที่ถูกพิมพ์เข้าไปก่อนหน้านี้จะไม่สามารถเก็บไว้ได้ จะมีเพียงค่า X ล่าสุดที่อยู่ในหน่วยความจำของคอมพิวเตอร์เท่านั้น
3. ดังนั้นจะทำอย่างไรเพื่อจะเก็บ ค่า X ไว้ได้

   เราสามารถใช้ "ARRAYS" , เช่น ถ้าเรากำหนด X(5000) ที่ตำแหน่งเริ่มต้นของโปรแกรม , เราจะได้



ดังนั้นเราสามารถเขียนโปรแกรมข้างบนใหม่ ได้ดังนี้

```
      PROGRAM  AVG
      DIMENSION  X(5000)
      SUM  =  0.
      N  =  0
      DO  10  I=1,5000
      READ  *,  X(I)
      IF(X(I).LT.0.)   GOTO  20
      SUM  =  SUM  +  X(I)
      N  =  N + 1
   10 CONTINUE
   20 CONTINUE
      AVER  =  SUM/N
      PRINT  *, 'AVERAGE  SCORE  IS', AVER
      DO  30  I=1,N
      PRINT  *, 'STUDENT NO.", I ," HAS SCORE ', X(I)
   30 CONTINUE
      STOP
      END
```

ผลลัพธ์ที่ได้ออกมาจะมีลักษณะดังนี้

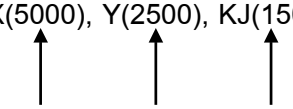
```
AVERAGE  SCORE  IS  82.9275
STUDENT NO.  1  HAS SCORE  91.0000
STUDENT NO.  2  HAS SCORE  80.0000
STUDENT NO.  3  HAS SCORE  87.0000
```

```
STUDENT NO.  32  HAS SCORE  93.0000
```

<-- จะหยุดที่นี่ กรณีมีนักเรียนทั้งหมด 32 คน

หมายเหตุ : การกำหนด DIMENSION

```
DIMENSION    X(5000), Y(2500), KJ(150)
```



real real integer <-- เป็น default

หรือ เราสามารถใช้

```
REAL  X(5000), Y(2500)
INTEGER  KJ(150)
```

<u>คำสั่ง OPEN</u>

ถ้าเราต้องการเก็บคะแนนของนักเรียนไว้ในไฟล์ เราจะต้องเพิ่มคำสั่ง OPEN ไว้ในโปรแกรม ดังนี้

```
      PROGRAM  AVG
      DIMENSION  X(5000)
      OPEN(UNIT=9, FILE='SCORE', STATUS = 'NEW')
                        |
                        |  (เหมือนโปรแกรมที่แล้ว)
                        |
                        ▼
      WRITE(9,202)  I,  X(I)
  202 FORMAT(' STUDENT NO. ', I5,'  HAS SCORE ', F6.1)
```

ถ้าเรามีข้อมูลของคะแนนนักเรียนทั้งหมดอยู่ในไฟล์ชื่อ RAWDAT ซึ่งมีรายละเอียดดังนี้

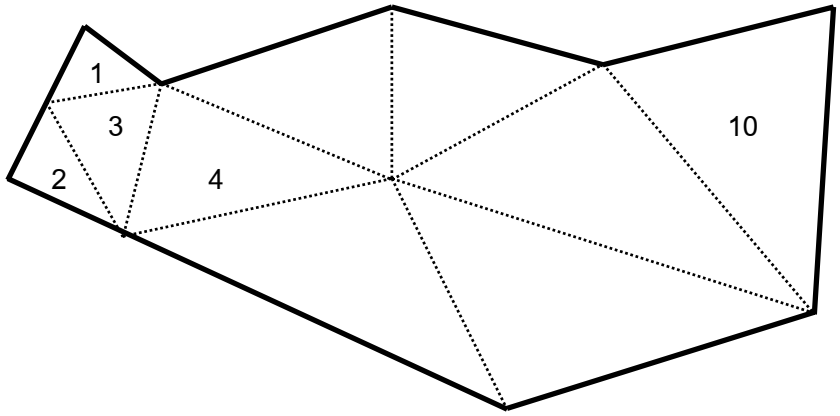
```
1          91.
2          80.
3          87.
|          |
|          |
▼          ▼
32         93.
```

เราสามารถเขียนโปรแกรมให้อ่านข้อมูลจากไฟล์โดยอัตโนมัติ ดังนี้

```
      PROGRAM  AVG
      DIMENSION  X(5000)
      OPEN (UNIT=9, FILE='SCORE', STATUS='NEW')
      OPEN (UNIT=8, FILE='RAWDAT', STATUS='OLD')
      DO  15  I=1,32
      READ(8,*)  N, X(I)
   15 CONTINUE
```

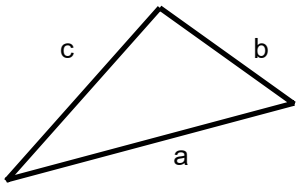
SUBROUTINE และ FUNCTION

ตัวอย่าง คำนวณหาพื้นที่เขตพญาไท ดังแสดงในรูปข้างล่าง



แนวความคิด

1. แบ่งย่อยพื้นที่เขตจากรูปเหลี่ยมใดๆออกเป็นรูปร่างง่ายๆ เช่น รูปสามเหลี่ยมจำนวน 10 รูป
2. ใช้สมการหาพื้นที่รูปสามเหลี่ยม



$$\text{AREA} = \sqrt{s(s-a)(s-b)(s-c)}$$

เมื่อ $s = (a+b+c)/2.$

และ ความยาว a , b , c สามารถวัดได้

3. รวมพื้นที่รูปสามเหลี่ยมทั้ง 10 รูป เข้าด้วยกัน

หมายเหตุ : ลักษณะการป้อนข้อมูลสามารถยกตัวอย่างได้ดังนี้

```
1.6   3.4   2.4   <------  a, b, c  สำหรับรูปสามเหลี่ยมที่ 1
5.4   2.7   3.9   <------  a, b, c  สำหรับรูปสามเหลี่ยมที่ 2
 |     |     |                      |
 |     |     |                      |
 v     v     v                      v
4.7   3.2   2.6   <------  a, b, c  สำหรับรูปสามเหลี่ยมที่ 10
```

และลักษณะของโปรแกรมเป็นดังนี้

```
      PROGRAM  PYT
C
C     First declare the total area = 0.      We will compute
C     and add each area, one at a time
C
      TTAREA  =  0.
C
C     Compute area of triangle 1 :
C
      READ  *, A, B, C                        A  =  1.6,   B  =  3.4,   C  =  2.4
      S  =  ( A + B + C )/2.                S  =  (1.6+3.4+2.4)/2.  =  3.7
      SABC  =  S*(S - A)*(S - B)*(S - C)      SABC  =  3.7*2.1*0.3*1.3  =  3.03
      AREA  =  SQRT(SABC)                     AREA  =  √3.03  =  1.74
C
C     Add to the total area :
C
      TTAREA  =  TTAREA + AREA                TTAREA  =  0 + 1.74  =  1.74
C
C
C     Compute area of triangle 2 :
C
      READ  *, A, B, C                        A  =  5.4,   B  =  2.7,   C  =  3.9
      S  =  ( A + B + C )/2.                  S  =  (5.4+2.7+3.9)/2.  =  6
      SABC  =  S*(S - A)*(S - B)*(S - C)      SABC  =  6*0.6*3.3*2.1  =  24.95
      AREA  =  SQRT(SABC)                     AREA  =  √24.95  =  4.99
C
C     Add to the total area :
C
      TTAREA  =  TTAREA + AREA                TTAREA  =  1.74 + 4.99  =  6.73
```

ทำการคำนวณหาพื้นที่ซ้ำอีก 8 ครั้ง !! --> โปรแกรมยาว

```
C
      STOP
      END
```

โปรแกรมนี้ยาว เนื่องจากมีการคำนวณหาพื้นที่สามเหลี่ยมซ้ำไปมาถึง 10 ครั้ง ซึ่งสามารถเขียนโปรแกรมให้สั้นด้วยการใช้ subroutine แทน ดังนี้

```
      PROGRAM  PYT
C
      TTAREA  =  0.
C
C     Compute area of triangle 1
C
      READ  *, A, B, C
      CALL  TRI ( A, B, C, AREA )          <-- เรียกใช้ subroutine  TRI
C                                              ซึ่งจะเรียกหาค่า A, B, C และจะให้ค่า AREA กลับมา
C     Add to the total area :
C
      TTAREA  =  TTAREA + AREA
C
C     Compute area of triangle 2
C
      READ  *, A, B, C
      CALL  TRI ( A, B, C, AREA )
C
C     Add to the total area :
C
      TTAREA  =  TTAREA + AREA
```


ทำการคำนวนพื้นที่ซ้ำ 8 ครั้ง แต่สั้นกว่าแบบแรก

```
      STOP
      END
```

```
C
      SUBROUTINE  TRI ( A, B, C, AREA)          <---  Subroutine  TRI
      S  =  ( A + B + C )/2.                          รู้ค่า A, B, C ทำให้สามารถ
      SABC  =  S*(S-A)*(S-b)*(S-C)                    คำนวนหา AREA ได้
      AREA  =  SQRT(SABC)
      RETURN
      END
```

หมายเหตุ การคำนวณหาพื้นที่สามเหลี่ยมจะทำซ้ำกัน 10 ครั้ง เราสามารถทำให้โปรแกรมสั้นลงไปอีกได้ โดยใช้ DO loop (จะอธิบายต่อไป)

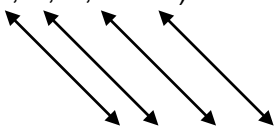
```
      PROGRAM  PYTC
      TTAREA  =  0.
C
      DO 100 I = 1, 100
C
C     Compute area of triangle
C
      READ  *, A, B, C
      CALL  TRI ( A, B, C, AREA )
C
C     Add to the total area :
C
      TTAREA  =  TTAREA + AREA
C
100   CONTINUE
C
      PRINT  *,  TTAREA
C
      STOP
      END
C
      SUBROUTINE  TRI ( A, B, C, AREA )
      S  =  ( A + B + C )/2.
      SABC  =  S*(S - A)*(S - B)*(S - C)
```

```
      AREA  =  SQRT(SABC)
      RETURN
      END
```

หมายเหตุ ทำการพิจารณา SUBROUTINE :

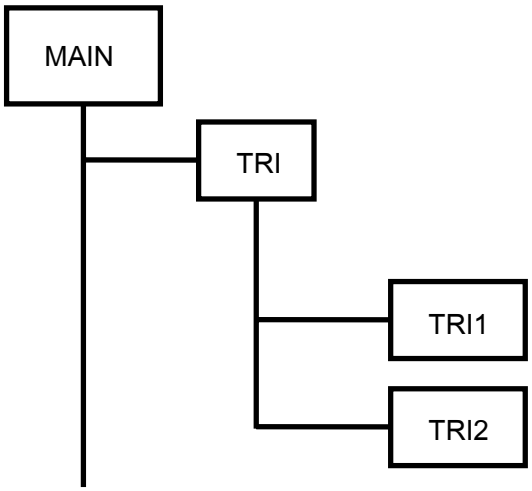
1. จากตัวอย่างข้างบน เรามี SUBROUTINE TRI(A, B, C, AREA)
   โดย A, B, C, AREA เรียกว่า ค่าอาร์กิวเมนต์ (Arguments)
   ชื่อของอาร์กิวเมนต์ที่ใช้ใน SUBROUTINE สามารถแตกต่างจากที่ใช้ใน MAIN PROGRAM ได้ เช่น

```
CALL  TRI ( A, B, C, AREA)
```



```
SUBROUTINE  TRI ( AA, BB, CC, AR)
S  =  (AA+BB+CC)/2.
SABC  =  S*(S-AA)*(S-BB)*(S-CC)
AR  =  SQRT(SABC)
RETURN
END
```

2. SUBROUTINE สามารถเรียกใช้ SUBROUTINE อื่นๆได้



ลักษณะโครงสร้างนี้จะคล้ายกับโครงสร้างของ directory, subdirectory และไฟล์ ใน MS-DOS

หมายเหตุ พิจารณาการใช้ SQRT
โปรแกรมข้างบนมีการใช้ SQRT คือ

```
      AREA  =  SQRT(SABC)
```

SQRT เรียกว่าเป็น ฟังก์ชันแท้จริง (instrinsic function) ซึ่งเป็นฟังก์ชันทั่วไปที่มีอยู่ในภาษาฟอร์แทรนอยู่แล้ว ตัวอย่างของฟังก์ชันแท้จริงอื่นๆ ได้แก่

```
      B  =  SIN(X)              เมื่อ X มีหน่วยเป็น radian
      PI  =  4.*ATAN(1.)
      C  =  LOG(X)              <-- log_e X  หรือ  ln X  (LOG10(X) <-- log_e X)
      D  =  AMAX( A1, A2,….)    <--  หาค่าสูงสุด
      K  =  INT(A)              <--  แปลงจำนวนจริงเป็นจำนวนเต็ม
      E  =  FLOAT(I)            <--  แปลงจำนวนเต็มเป็นจำนวนจริง
      F  =  EXP(A)              <--  e^r
```

<u>ฟังก์ชันภายนอก</u> (External Functions) <-- ปกติจะไม่ค่อยใช้ แต่จะใช้ subroutine มากกว่า

ถ้าภาษาฟอร์แทรนไม่มีฟังก์ชันที่เราต้องการใช้ เราสามารถเขียนฟังก์ชันขึ้นมาเองได้ เช่น DO Loop ในตัวอย่างที่แล้ว

```
      DO  100  I=1,10
      READ  *,  A, B, C
      AREA  =  AR( A, B, C)
                  |
                  |   (เหมือนตัวอย่างที่แล้ว)
                  v
  100 CONTINUE
      PRINT  *, TTAREA
      STOP
      END
C
      FUNCTION  AR( A, B, C)
      S  =  (A+B+C)/2.
      SABC  =  S*(S-A)*(S-B)*(S-C)
      AR  =  SQRT(SABC)
      RETURN
      STOP
```

DO Loop

- ใช้สำหรับชุดคำสั่งที่มีการทำซ้ำไปมาหลายครั้ง โดยสามารถกำหนดจำนวนครั้งของการทำซ้ำได้ ด้วยรูปแบบของ DO Loop
- ภายใน DO Loop อาจจะประกอบด้วย คำสั่งต่างๆ และ DO Loop อื่นๆ
- บรรทัดสุดท้ายของ DO Loop จะต้องเป็นคำสั่ง CONTINUE

รูปแบบทั่วไปของ DO Loop คือ

```
      DO  10  I=I1,I2,I3
      
      
      
10    CONTINUE
```

10 ถูกเรียกว่า statement label (จะอยู่ในช่วง 1 - 99999)
I ถูกเรียกว่า ตัวแปรของ DO Loop
I1 เป็นตัวแปรเริ่มต้น
I2 เป็นตัวแปรสุดท้าย
I3 เป็นตัวเพิ่มค่า (increments) <-- โดยทั่วไปจะเท่ากับ 1 ถ้าไม่มีการระบุไว้

เช่น

```
      DO  10  I=1,10,1
      PRINT  *, I
   10 CONTINUE
```

การวนอยู่ใน Loop จะหลุดออกมาเมื่อค่า I = 10
ผลของ DO Loop จะพิมพ์ตัวเลข 1,2,3,4,5,6,7,8,9,10

```
      DO  25  J=20,35,5
      PRINT  *, J
   25 CONTINUE
```

ผลของ DO Loop จะพิมพ์ตัวเลข 20,25,30,35

```
      DO  873  KAB=4,17,5
      PRINT  *, KAB
```

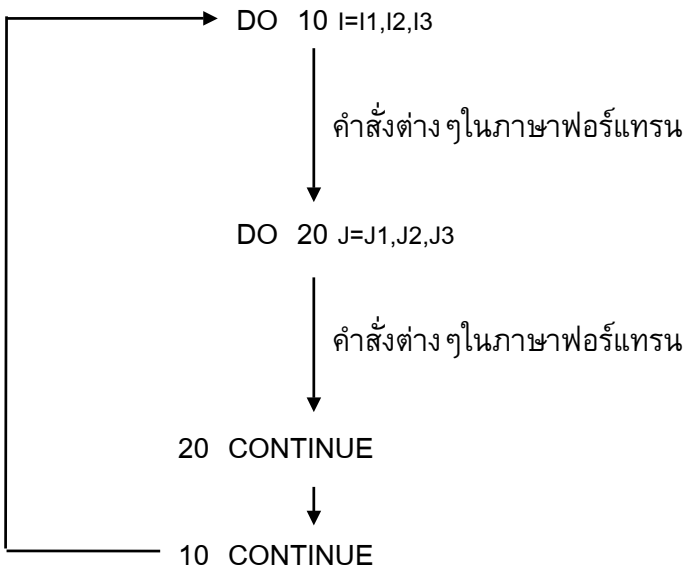
```
 873  CONTINUE
```

ผลของ DO Loop จะพิมพ์ตัวเลข 4,9,14

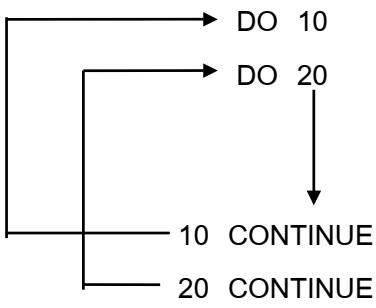
```
      DO  2000  K=-3,6,3
      PRINT  *, K
2000  CONTINUE
```

ผลของ DO Loop จะพิมพ์ตัวเลข -3,0,3,6

<u>Nested DO Loops</u> (การมี DO Loop ซ้อนใน DO Loop)



แต่ไม่สามารถทำดังต่อไปนี้



<u>ตัวอย่าง</u> ทำตาราง “สูตรคูณ” (สูตรคูณ จากแม่ 2 ถึง 13)

```
      DO  10  I = 2,12
      PRINT  *, 'TIMES  TABLE  FOR', I
```

```
      DO  20  J = 1,12
      IJ = I * J
      PRINT  *, I , ' TIMES ', J , ' IS ', IJ
   20 CONTINUE
   10 CONTINUE
```

ผลลัพธ์ที่เกิดขึ้น คือ

```
TIME TABLE FOR 2
 2     TIMES    1    IS    2
 2     TIMES    2    IS    4                    I = 2
 2     TIMES    3    IS    6                    J เปลี่ยนแปลงจาก 1 ถึง 12
                    |
                    |
                    |
 2     TIMES   12   IS   24
TIME TABLE FOR 3
 3     TIMES    1    IS    3
 3     TIMES    2    IS    6                    I = 3
 3     TIMES    3    IS    9                    J เปลี่ยนแปลงจาก 1 ถึง 12
                    |
                    |
                    |
 3     TIMES   12   IS   36
                    |
                    |
                    |
                    |
                    |
                    |
TIME TABLE FOR 12
12     TIMES    1    IS   12
12     TIMES    2    IS   24                    I = 12
12     TIMES    3    IS   36                    J เปลี่ยนแปลงจาก 1 ถึง 12
                    |
                    |
                    |
12     TIMES   12   IS  144
```

PARAMETER STATEMENT

ใช้กำหนดค่าคงที่ตั้งแต่ต้น MAIN PROGRAM เท่านั้น เช่น

```
PROGRAM  TEST
INTEGER  N
REAL  PI
PARAMETER  (PI = 3.141593,  N = 75)
REAL  TEMP(N,N)
```

COMMON STATEMENT

ใช้กำหนดกลุ่มของตัวแปรเพื่อใช้ทั้ง MAIN PROGRAM, SUBROUTINE และ FUNCTION โดยไม่ต้องส่งผ่านค่าด้วย arguments เช่น

```
PROGRAM  TEST1
INTEGER  J
REAL  A,  J,  B
COMMON  A,  J,  B
        •
        •
        •
        •
CALL  ANSWR(X,Y)
        •
        •
        •
        •
END
SUBROUTINE  ANSWR(X,Y)
INTEGER KTOT
REAL TEMP,  SUM,  X,  Y
COMMON  TEMP,  KTOT,  SUM
        •
        •
        •
        •
 END
```