บทที่ 8
คำสั่งควบคุมลำดับ
(Sequence control statements)

ควบคุมลำดับ หมายถึง ลำดับที่ (order) ซึ่งคำสั่งต่าง ๆ ของโปรแกรมจะถูก execute คำสั่งต่าง ๆ ในโปรแกรม ปกติแล้วจะถูก execute ตามลำดับที่ปรากฏ แต่ถ้าคำสั่งที่ถูก execute นั้นเป็นคำสั่ง ในกลุ่มคำสั่งควบคุมลำดับ จะทำให้ทิศทางปกติ(normal flow) เปลี่ยนไป โดยทั่วไป คำสั่งควบคุมลำดับ ทำให้เกิดการแตกกิ่ง (branching) อย่างมีเงื่อนไข และไม่มีเงื่อนไข และควบคุมการทำงานซ้ำ ๆ กัน (looping) ซึ่งจะได้กล่าวถึงต่อไป Procedure invocations ก็เช่นเดียวกันทำให้ ลำดับของของการ execute เปลี่ยนไป ซึ่งจะได้กล่าวถึงภายหลัง ในหัวข้อเดียวกันนี้

8.1 คำสั่ง GO TO

คำสั่ง GO TO ทำให้มีการย้ายอย่างไม่มีเงื่อนไข ไปยังคำสั่งซึ่งมี labei นั้นกำกับอยู่ โดยมีรูปแบบอย่างใดอย่างหนึ่ง ดังนี้

```
GO TO  label-constant;
GOTO   label-constant;
GO TO  label-variable;
GOTO   label-variable;
```

เมื่อ label-constant เป็น label ที่ปรากฏอยู่ข้างหน้า ของคำสั่งซึ่งมี label และ label-variable เป็น ตัวแปรชนิด label ธรรมดา หรือ ตัวแปรชนิด label มี subscript ซึ่งกำหนดโดยค่าของ label-constant, label-constant ซึ่งได้ประเมินผล ต้องเป็น label ของคำสั่งภายในขอบเขต(scope) ของคำสั่ง GOTO และต้องไม่อยู่ใน embedded DO group ของการเรียงลำดับ (sort) ใด ๆ

ตัวอย่าง

```
GO TO  LAB1;
```

```
GOTO  WHERE;

GOTO  L(J);
```

8.2 คำสั่ง IF

คำสั่ง IF ทำให้การ execute ของคำสั่งหนึ่งคำสั่ง หรือ กลุ่มคำสั่ง มีเงื่อนไข ขึ้นกับว่าค่าของ การทดสอบ boolean นั้นเป็นจริง หรือ เท็จ ถ้าการทดสอบ boolean ให้ค่าเท็จ ส่วนที่เป็น ELSE clause (ซึ่งอาจจะมีหรือไม่มีก็ได้) ที่เป็นคำสั่ง 1 คำสั่ง หรือ กลุ่มคำสั่ง จะได้รับการ execute

รูปแบบทั่วไปของคำสั่ง IF มีดังนี้

```
IF condition THEN group-1; [ELSE group-2];
```

เมื่อ condition เป็น scalar expression ให้ค่าเป็น bit string

group-1 และ group-2 อาจเป็น คำสั่ง simple หรือคำสั่ง compound (ซึ่งประกอบด้วย DO group หรือ Begin group อยู่ภายใน) อย่างใดอย่างหนึ่ง ถ้า group-1 หรือ group-2 เป็นคำสั่ง simple มันต้องไม่ใช่คำสั่ง DECLARE, END, ENTRY,FORMAT หรือคำสั่ง PROCEDURE ถ้าบิทใดบิทหนึ่งของผลลัพธ์ของเงื่อนไขเป็น 1 แล้ว ส่วนของ THEN group-1 จะได้รับการ execute นอกเหนือจากนี้ การควบคุมจะถูกส่งไปยัง group-2 ถ้ามี หรือไปยังคำสั่งถัดไป ในลำดับที่ตามหลังคำสั่ง IF

คำสั่ง IF โดยตัวมันเอง อาจถูกซ้อนกัน ในกรณีนี้ ELSE แต่ละตัว จะคู่กับ IF-THEN อันในสุด ที่ยังไม่ได้จับคู่กับใคร คำสั่งว่าง (empty statements) อาจนำมาใช้กำกับคู่ IF-ELSE ที่ต้องการได้

ตัวอย่าง

```
IF A=Y THEN
   IF Z=X THEN
      IF W>B THEN
         c=0;
      ELSE C=1;
   ELSE;
ELSE A=Y2;
```

ตัวอย่างข้างต้นนี้ คำสั่งว่าง ซึ่งอยู่ก่อนหน้า ELSE ตัวที่ 3 สมนัยกับ การทดสอบ IF-THEN ตัวที่สอง

8.3 คำสั่ง Iterative DO

รูปแบบที่ง่ายที่สุดของคำสั่งนี้ คือ DO group ซึ่งเป็นการรวม คำสั่งต่าง ๆ และได้รับการ execute เพียงครั้งเดียว รูปแบบ non-iterative ของ DO group ได้กล่าวมาแล้วในบทแรก ๆ, ในหัวข้อนี้ จะกล่าวรายละเอียดต่าง ๆ ของ iterative DO group ซึ่งมีหัวเรื่องเป็น คำสั่ง DO มีรูปแบบอย่างใดอย่างหนึ่ง ในสองแบบข้างล่างนี้

```
DO WHILE (condition);
หรือ DO control-variable = do-specification;
```

เมื่อ control-variable เป็น ตัวแปรที่ไม่มี subscript

condition เป็น boolean expression

และ do-specification มีรูปแบบอย่างใดอย่างหนึ่งข้างล่างนี้

[start-exp[TO end-exp][BY incr-exp]][WHILE(condition)]

[start-up[BY insr-exp][TO end-exp]][WHILE(condition)]

[start-exp [REPEAT(repeat-exp)][WHILE(condition)]

ในรูปแบบทั่วไปเหล่านี้

start-exp เป็น expression หมายถึง ค่าแรกของตัวแปรควบคุม

end-exp เป็น expression หมายถึง ค่าสุดท้ายของตัวแปรควบคุม

incr-exp เป็น expression หมายถึง expression ซึ่งจะเอาค่าไป บวกกับตัวแปรควบคุม หลังการ execute แต่ละครั้งของลูป(loop)

repeat-exp เป็น expression ซึ่งแทนที่ ตัวแปรควบคุม หลังจากกระทำซ้ำแต่ละครั้ง

condition เป็น expression ให้ค่า bit string ซึ่งจะเป็นจริง ถ้าบิทใดบิทหนึ่งใน string นั้นเป็น 1

ถ้ามีรูปแบบของ TO end-exp อยู่ด้วย แต่ ไม่มี BY incr-exp เครื่องจะถือว่า incr-exp มีค่าเท่ากับ 1 ทั้งสองรูปแบบ ที่มีการใช้ TO และ BY execute มีความหมายอย่างเดียวกัน แตกต่างกันก็เพียงแต่ลำดับของอีลิเม้นท์สองตัวนี้เท่านั้น

expression WHILE จะได้รับการประเมินผล ก่อนการ execute ของ DO group ถ้าเงื่อนไขเป็นเท็จ เครื่องจะหยุด execute ลูป และการควบคุมจะถูกส่งไปยังคำสั่งที่ตามหลัง คำสั่ง END ซึ่งคู่กับคำสั่ง DO นั้น

ยกเว้นถ้ามีการใช้ option REPEAT, expression ใน specification ของ DO จะถูกประเมินผลก่อน การ execute ของลูป นั่นคือ การเปลี่ยนแปลงที่เกิดขึ้น กับค่าเริ่มต้น, ค่าสุดท้าย หรือ ค่าเพิ่ม ไม่มีผลต่อจำนวนครั้งที่ลูปนั้น จะได้รับการ execute ในกรณีของ option REPEAT อย่างไรก็ตาม repeat-exp จะถูกคำนวณใหม่ (recomputed) หลังการทำซ้ำ (iteration) แต่ละครั้ง

expression ที่ถูกคำนวณใหม่นี้จะเก็บ ในตัวแปรควบคุม และมีการ execute ตรวจสอบ option WHILE (ถ้ามี)

เพื่อให้การ นิยาม action ของ iterative groups เป็นไปอย่างถูกต้อง ได้แสดง โครงสร้างข้างล่างนี้ พร้อมกับ ชุด (sequence) ของคำสั่ง IF และ GOTO

ที่มีความหมายอย่างเดียวกัน ในโครงร่าง (decomposition) นี้ expression e1, e2, e3 และ e4 หมายถึง start-exp, end-exp, incr-exp, repeat-exp และ condition values, i หมายถึง control-variable ที่ถูกต้อง

เริ่มต้นด้วย

```
DO WHILE(e1);
    :
END;
```

มีความหมายเช่นเดียวกับ ชุดของคำสั่ง ต่อไปนี้

```
LOOP:
    IF ~e1 THEN
        GO TO ENDLOOP;
    :
    GO TO LOOP;
ENDLOOP;
```

ในทำนองเดียวกัน Do-repeat group

```
DO i=e1 REPEAT(e2);
    :
END;
```

มีความหมายอย่างเดียวกับ

```
    i=e1;
LOOP:
    :
    i=e2;
    GO TO loop;
```

หมายเหตุ ในกรณีนี้ ลูปจะถูกกระทำ อย่างไม่มีการจำกัด (indefinitely)

จนกระทั่งจบด้วย คำสั่ง embedded หนึ่งคำสั่ง เช่น คำสั่ง GO TO หรือคำสั่ง STOP

ถ้ามี option WHILE อยู่ด้วย ดังนี้

```
DO i=e1 REPEAT(e2) WHILE(e3);
    :
END;
```

ผลลัพธ์จะมีความหมายอย่างเดียวกับ

```
    i=e1
LOOP:
    IF ~e3  THEN
            GO TO ENDLOOP;
        :
    i=e2;
    GO TO LOOP;
ENDLOOP:;
```

สำหรับ iterative Do-group อย่างง่าย ซึ่งมีรูปแบบดังนี้

```
DO i=e1 TO e2;
    :
END;
```

ถูกปฏิบัติเช่นเดียวกับ

```
DO i=e1 TO e2 BY e3;
    :
END;
```

ซึ่งมีความหมายอย่างเดียวกับชุดของคำสั่งข้างล่างนี้

```
    i=e1;
    LAST=e2;
```

```
INCR=e3;
LOOP:
    IF endtest THEN
        GO TO ENDLOOP;
    :
    i = i + INCR;
    GO TO LOOP;
ENDLOOP:;
```

เมื่อคำสั่ง IF ซึ่งมี endtest เปรียบเทียบ ตัวแปรควบคุมด้วยค่าของ LAST การเปรียบเทียบขึ้นอยู่กับ เครื่องหมายของค่าเพิ่ม INCR ถ้า INCR มีค่าเป็นลบ การทดสอบคือ

```
IF i < LAST THEN
    GO TO ENDLOOP;
```

ถ้าเป็นตัวอย่างอื่น การทดสอบคือ

```
IF I > LAST THEN
    GO TO ENDLOOP;
```

ในท้ายที่สุด ถ้ามี option WHILE เพิ่มดังนี้

```
DO i=el TO e2 BY e3 WHILE(e4);
    :
END;
```

มีความหมายอย่างเดียวกับ

```
    i=el;
    LAST=e2;
    INCR=e3;
LOOP:
```

```
        IF ~e4  THEN
                GO TO ENDLOOP;
        IF  endtest THEN
                GO TO ENDLOOP;
             :
        i = i + INCR;
        GO TO LOOP;
    ENDLOOP;
```

หมายเหตุ ชุดของคำสั่งที่มีความหมายเหมือนกันนี้ ค่าของ LAST และ INCR ขึ้นกับลักษณะเฉพาะของ expressions e2 และ e3 และ อาจมีการเปลี่ยนรูปเลขคณิต, และมีการเปรียบเทียบเกิดขึ้นในแต่ละขั้นตอนขึ้นกับกฎของ PL/I-80 ปกติ

8.4 การประมวลผลเงื่อนไข (Conditional processing)

คำสั่ง ON, REVERT และ SIGNAL จัด run-time facilities ให้กับการเขียนโปรแกรม ซึ่งอาจเกิดการขัดจังหวะของ เงื่อนไขผิดพลาด(error condition) ซึ่งส่วนใหญ่แล้วจะเป็นเหตุให้โปรแกรมนั้นหยุดทำงาน เงื่อนไขต่อไปนี้ ยอมรับใน PL/I-80

เงื่อนไขผิดพลาดทั่วไปได้แก่ ERROR

เงื่อนไขผิดพลาดซึ่งเกิดจากการคำนวณ ได้แก่ FIXEDOVERFLOW, OVERFLOW, UNDERFLOW และ ZERODIVIDE

เงื่อนไขเกี่ยวกับ input/output ได้แก่ ENDFILE, ENDPAGE, KEY และ UNDEFINEDFILE

สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับ ข้อยกเว้นของการประมวลผล จะได้กล่าวถึงต่อไป และให้ดูในหนังสือ "PL/I-80 Application Guide"

8.5 คำสั่ง ON

คำสั่ง ON เป็นการ นิยาม (define) เงื่อนไขเพื่อจะให้กระทำ และจะกระทำ (action) เมื่อ เกิดเงื่อนไขนั้น ระหว่างการ execute โปรแกรม

คำสั่ง ON มีรูปแบบดังนี้

```
ON  condition  ON-unit
```

เมื่อ condition ในที่นี้ อาจจะเป็นอันใดอันหนึ่ง ต่อไปนี้ ERROR, FIXEDOVERFLOW, OVERFLOW, UNDERFLOW, ZERODIVIDE, ENDFILE, ENDPAGE, UNDEFINEDFILE และ key

ON-unit อาจจะเป็นคำสั่งใดคำสั่งหนึ่งของ PL/I-80 หรือ อาจจะเป็น PL/I-80 หลาย ๆ คำสั่ง ซึ่งอยู่ภายใน BEGIN-END block และจะถูก execute เมื่อชื่อเงื่อนไขที่ระบุ ในคำสั่ง ON เกิดขึ้น การออก (exit) จาก BEGIN block ไม่จำเป็นต้องผ่านคำสั่ง RETURN ถึงแม้ข้อจำกัดนี้ ไม่ได้ preclude procedure definition ภายใน BEGIN-END block การออกจาก BEGIN block เกิดขึ้น ผ่านคำสั่ง non-local GO TO, การควบคุม จะยังคงอยู่ที่จุดซึ่งเงื่อนไขนั้นถูกค้นพบ ถ้า คำสั่งทั้งหมดของ BEGIN-END group ถูก execute และไม่มีการย้าย non-local เกิดขึ้น

ON-unit ไม่ได้ทำให้ตัวแปรเป็นอิสระ หมายความว่า การถูกใช้ เกิดขึ้น เมื่อ เกิดเงื่อนไข หรือ ปิดแฟ้มข้อมูล (close file) สำหรับการเกิดของเงื่อนไข เกี่ยวกับ input/output, ON-unit ยังคงมีผลอยู่จนกว่า จบ block ของมัน หรือจนกว่า มันมี การเปลี่ยนรูปกลับ ด้วยคำสั่ง REVERT หรือ ON-unit อันอื่น ซึ่งกำหนดขึ้นภายหลัง ในลำดับ ของการ execute ในกรณีหลังนี้ ON-unit จะถูกเก็บในสแตก(stack) จนกระทั่งมีการ กระทำใหม่ ผ่านคำสั่ง REVERT ซึ่งยกเลิก (cancel) ON-unit อันล่าสุด

หมายเหตุ ในแต่ละจุดที่กำหนดนั้น จะมีเงื่อนไข ON ที่ active ได้มากที่สุดเพียง 16 ครั้ง

8.6 คำสั่ง SIGNAL

คำสั่งนี้ ทำให้ เงื่อนไขอย่างใดอย่างหนึ่ง ซึ่งเกิดขึ้นแล้วในโปรแกรม และเรียก (invoke) ON-unit ที่สมนัยกัน ถ้ายังมีผลอยู่ (active) แต่ถ้าไม่มี ON-unit ที่มีผลอยู่ เครื่องจะ default ให้เอง ส่วนใหญ่แล้ว action ของการ default คือพิมพ์ "backtrace" และจบการ execute โปรแกรม

รูปแบบของคำสั่ง SIGNAL มีดังนี้

```
SIGNAL  condition;
```

เมื่อ condition เป็นเงื่อนไขอย่างใดอย่างหนึ่งที่กล่าวมาแล้วข้างต้น ในคำสั่ง ON

ตัวอย่าง

```
SIGNAL ZERODIVIDE;
```

เป็นการเรียก current ZERODIVIDE ON-unit

8.7 คำสั่ง REVERT

คำสั่งนี้ ใช้ deactive ON-unit ปัจจุบัน แล้วกำหนดใหม่ ให้เป็น อันที่อยู่ก่อนหน้านั้น ถ้ามันมีอยู่จริง รูปแบบของคำสั่ง REVERT มีดังนี้

```
REVERT  condition;
```

เมื่อ condition เป็นเงื่อนไขอันใดอันหนึ่ง ในรายชื่อที่กล่าวมา ในคำสั่ง ON

ตัวอย่าง

```
REVERT  OVERFLOW;
```

หมายถึง deactive ON-unit ปัจจุบัน สำหรับเงื่อนไข OVERFLOW

หมายเหตุ ภายใต้ทางออก ของ PROCEDURE block หรือ BEGIN block คำสั่ง REVERT เกิดขึ้นอัตโนมัติ สำหรับ ON-unit ใด ๆ ที่อยู่ภายใน block

8.8 Default ON-units

ด้วยข้อยกเว้นของ FIXEDOVERFLOW และ ENDPAGE; การ default ON-units ปกติ พิมพ์ ข้อความผิดพลาดที่ตรงกันแล้วจบโปรแกรม

เงื่อนไข FIXEDOVERFLOW ไม่ได้ให้สัญญาณ สำหรับ FIXED BINARY overflow ถึงแม้ว่า มันอาจจะเกิดขึ้นถ้า การคำนวณชนิด FIXED DECIMAL ให้ผลลัพธ์มากกว่าขนาดของเนื้อที่ซึ่งจัดสรรให้ เมื่อเงื่อนไข ENDPAGE เกิดขึ้น เครื่อง default ON-unit โดยการใส่ รูปแบบของ feed character ไว้ใน output file และ set ให้เลขประจำบรรทัดปัจจุบัน เป็น 1

8.9 บิลท์-อิน ฟังก์ชันสำหรับการประมวลผลเงื่อนไข

PL/I-80 มี บิลท์-อิน ฟังก์ชัน หลายตัว ออกแบบมาเพื่อช่วย ในการ handle ข้อยกเว้นต่าง ๆ มีรายชื่อดังนี้

```
ONCODE     ONFILE     ONKEY

PAGENO     LINENO
```

ฟังก์ชัน ONCODE ให้ผลลัพธ์เป็นค่า FIXED BINARY แทนชนิดของข้อผิดพลาด ซึ่งเกิด เงื่อนไข ERROR อันล่าสุด ถ้าไม่มีเงื่อนไขใด ๆ เกิดขึ้น ผลลัพธ์ของฟังก์ชันจะเป็นศูนย์ รหัสของความผิดพลาด (error codes) เปลี่ยนตาม version ให้ดูรายละเอียด ในหนังสือ "PL/I-80 Command Summary"

ส่วนรายละเอียดของ บิลท์-อิน ฟังก์ชัน ONFILE, ONKEY, PAGENO และ

LINENO จะกล่าวถึงในบทต่อไป

8.10 Procedure blocks

Procedure blocks ถูกจำกัดด้วย คำสั่ง PROCEDURE และคำสั่ง END จำนวนเท่ากัน ถูกเรียกโดย คำสั่ง CALL ใน subroutine หรือโดย function calls, Procedure ใช้เพื่อ execute ส่วนของโปรแกรมที่เหมือนกัน (same program segment) เป็นจำนวนตั้งแต่หนึ่งครั้งขึ้นไปโดยไม่ต้องเขียนส่วนนั้นหลาย ๆ ครั้งภายในโปรแกรม ข้อมูลที่สื่อสารกัน ถูกส่งไปยัง procedure โดยการอ้างถึงคือ actual parameters ขณะที่ รายชื่อของตัวแปร ซึ่งทราบโดย procedure และ นิยาม ไว้ในคำสั่ง PROCEDURE หมายถึง รูปแบบของ formal parameters

8.11 การเรียก procedure (Invoking a procedure)

Procedures มีอยู่สองชนิดคือ subroutine procedures และ function procedures ข้อแตกต่างระหว่าง subroutine และ function คือ subroutine ถูกเรียก โดยผ่านคำสั่ง CALL ในขณะที่ function ถูกเรียก โดยการใช้ชื่อฟังก์ชัน แล้วตามด้วย actual parameters ซึ่งอยู่ภายในเครื่องหมายวงเล็บ (ถ้ามี) เมื่อ ข้อความนั้น(context) ต้องมี expression อย่างน้อย 1 ตัว จากนั้น function procedure จะส่งค่า scalar 1 ตัว ไปยังส่วนของโปรแกรมที่เรียก

คำสั่ง CALL ใช้ส่ง การควบคุม(control) ไปยัง subroutine procedure และส่ง ข้อเท็จจริง (ถ้าจำเป็น) ไปยัง procedure โดยมีรูปแบบดังนี้

```
CALL procname [(sub1,...,sub-n)][(arg1,...,arg-m)];
```

เมื่อ procname เป็นชื่อของ procedure ที่กำลังเรียก

sub1 ถึง sub-n หมายถึง รายชื่อของ optional subscripts

ซึ่งจะต้องมี ถ้า procname เป็นตัวแปรชนิด entry แล้วมี subscript

arg1 ถึง arg-m หมายถึง actual parameters ซึ่งจะส่งไปยัง procedure

Actual parameters อาจจะเป็น อันใดอันหนึ่งต่อไปนี้

an arithmetic variable หรือ string variable, an array หรือ structure, a constant, an expression, a label, a file name, หรือ a pointer

แต่ cross-sections of a multi-dimensional array เอามาใช้เป็น actual parameters ไม่ได้

หมายเหตุ จำนวน actual parameters ถูกกำหนดโดย หัวเรื่องของคำสั่ง PROCEDURE ที่สมนัยกัน ถ้าไม่มี parameters ต้องใส่เครื่องหมายวงเล็บคู่ที่ว่าง

ตัวอย่าง คำสั่ง CALL

```
CALL P();

CALL Q(A,(B),B+C);

CALL V(I,J)(A,(B),(B+C);
```

Function ถูกเรียกใช้ในลักษณะอย่างเดียวกัน ยกเว้น ไม่ต้องมีคีย์เวิร์ด CALL และค่าของผลลัพธ์ ต้องใช้เป็น expression

ตัวอย่าง การเรียก function

```
I = F();

IF G(A,(B),B+C) = 5 THEN

     I = H(I,J)(A,(B))+SQRT(X);
```

8.12 โครงสร้างของ procedure definition

ในหัวข้อก่อนนี้ ได้กล่าวถึง วิธีเรียก procedure ส่วนวัตถุประสงค์ของหัวข้อนี้ จะให้รายละเอียดของ โครงสร้างรวมของ subroutine หรือ function definition

Procedures อาจถูก define ตรงตำแหน่งใดตำแหน่งหนึ่ง ในตัวโปรแกรม และถูกข้ามไป (by pass) เมื่อมีการทำงาน ในทิศทางการควบคุมตามลำดับ นั่นคือ procedures จะได้รับกระทำก็ต่อเมื่อผ่านกลไกของการเรียก ตามที่ได้กล่าวมาแล้วในหัวข้อก่อนนี้, โดยปกติ procedure ทั้งหมด ถูก defined เข้าด้วยกัน ใน section เดียว ที่ตอนเริ่มต้น หรือตอนท้าย ของ โปรแกรมหลัก, ขึ้นอยู่กับสไตล์ (style) ของการเขียนโปรแกรม

โปรแกรมหลัก โดยตัวมันเองเป็น procedure definition ชุดหนึ่ง จากนั้น procedures ที่แยกต่างหากจากกัน อาจถูก defined ขึ้นมา และลิงค์เข้าด้วยกัน เพื่อฟอร์ม ให้เป็น หนึ่ง module

โครงสร้างรวมของ procedure definition แต่ละชุด มีดังนี้

```
procname:
        procedure-statement
        procedure-body
        END [procname];
```

เมื่อ procname เป็น ไอเดนติไฟเออร์ 1 ตัว หมายถึง ชื่อของ procedure ชื่อ procedure จะกลายเป็น ค่าคงที่ ชนิด entry ซึ่ง available สำหรับการเข้าถึง (access) ผ่านขอบเขต ซึ่งมันปรากฏอยู่, entry point ของ procedure ถูกกำหนดโดย procedure-statement ส่วน procedure-body ประกอบด้วย ชุดของคำสั่ง, หรือ ไม่มีคำสั่ง (zero statement) และ procedure definition จบด้วยคำสั่งที่สมนัยกัน ซึ่งอาจจะเป็น จุดทางออก (exit point) ของ procedure ด้วย ถึงแม้ว่ากลุ่มของ คำสั่ง RETURN อาจปรากฏอยู่ภายใน procedure-body

procedure-statement บอกเขตการเริ่มต้นของ procedure block, ใช้สำหรับ นิยาม รายชื่อของ formal parameters และกำหนด attributes ของค่าที่ส่งกลับ ของ ฟังก์ชันนั้น โดยมีรูปแบบทั่วไปดังนี้

```
PROCEDURE [(parm1,...,parm-n)]
     [OPTIONS(MAIN)][RETURNS attribute-list][RECURSIVE];
```

เมื่อ parm1 ถึง parm-n เป็น formal parameters ของ procedure, formal parameters ทุกตัว ต้อง declared ภายใน procedure body ที่ระดับของ principal block

Formal parameter แต่ละตัว อาจเป็นตัวใดตัวหนึ่งต่อไปนี้
a non-subscripted variable, an array, a major structure, a label variable, a file name, a entry name หรือ a pointer variable

Formal parameter แต่ละตัว จะต้องไม่มี attribute ต่อไปนี้
STATIC, AUTOMATIC, BASED และ EXTERNAL

OPTIONS(MAIN) หมายถึง procedure นี้ เป็น procedure แรก ที่รับ control เมื่อเริ่มต้นโปรแกรม สำหรับ RETURNS attribute-list ใช้สำหรับ function procedure เมื่อต้องการกำหนด ลักษณะเฉพาะ ของ ค่าที่จะส่งกลับ โดย ฟังก์ชัน ส่วน attribute RECURSIVE บอกว่า procedure นี้ กระทำ(activate) โดยตัวมันเอง อาจจะโดยตรง หรือโดยทางอ้อม ขณะที่ procedure นั้นกำลัง execute

8.13 คำสั่ง RETURN

คำสั่งนี้ ส่งการควบคุม กลับไปยังจุด ใน block เรียก (calling block) ถัดจาก procedure invocation และส่งค่า 1 ค่า กลับไปด้วยเช่นกัน
ถ้า procedure นั้นเป็น function procedure รูปแบบของคำสั่ง RETURN จะเป็นดังนี้

```
RETURN [(ret-exp)];
```

เมื่อ ret-exp เป็นค่าซึ่งจะถูกส่งกลับ ไปยังจุดเรียก ถ้าจำเป็น ค่าที่ส่งกลับนี้ จะถูกเปลี่ยนรูป ให้ตรงกับ attribute ที่กำหนด ใน option RETURNS ของ procedure-statement

ตัวอย่าง คำสั่ง RETURN

```
RETURN;

RETURN (X**2);

RETURN (F(A,(B)));
```

คำสั่ง RETURN จบ (terminate) procedure block ซึ่งมันอยู่ ถ้าคำสั่ง RETURN อยู่ใน main procedure, การควบคุมจะส่งกลับไปยัง โปรแกรมควบคุมระบบ (oprating system)

8.14 คำสั่ง Non-local GO TO

คำสั่งนี้เกิดขึ้น เมื่อ การประเมินผล ค่าคงที่ชนิด label เป้าหมาย เกิดนอก innermost block ซึ่งมีคำสั่ง GO TO โดยทั่วไปแล้ว โปรแกรมเมอร์ควรหลีกเลี่ยง ไม่เขียน non-local GO TO เนื่องจาก ผลลัพธ์ที่ได้บ่อยครั้ง เป็นโปรแกรมที่มีโครงสร้างไม่ดี (poorly-structured program) ก็แล้วแต่โอกาส อย่างไรก็ตาม เมื่อต้องมี non-local GO TO ในกรณีของการจบเงื่อนไขข้อผิดพลาด แต่ มันก็มีประโยชน์เช่นกัน ตรงที่ แตกกิ่ง(branch) โดยตรง ไปยัง a global error recovery block เมื่อ program execution recommences ในกรณีนี้ embedded ON-units ทั้งหมด จะถูก revert โดยอัตโนมัติ และข้อเท็จจริงที่ส่งกลับจาก procedure จะถูกคัดทิ้งไป (discard)

ตัวอย่าง โครงร่างของโปรแกรม แสดง an instance ของ non-local GO TO จากภายใน provedure definition ชุดหนึ่ง

```
P:
  PROCEDURE;
  GO TO L;
  END P;
  CALL P();
  L:;
```

8.15 คำสั่ง STOP

คำสั่งนี้ จบ การ execute ของโปรแกรม ปิดแฟ้มข้อมูลที่เปิดไว้ทั้งหมด แล้ว ส่งการควบคุม ไปยัง โปรแกรมควบคุมระบบ ปกติแล้ว คำสั่ง STOP เกิดขึ้นเฉพาะในระดับ ของ โปรแกรมหลัก เท่านั้น แต่ก็อาจถูก execute ภายใน nested procedure call เพื่อหยุดการ execute ก่อนปกติ

รูปแบบของคำสั่ง มีดังนี้

```
STOP;
```

8.16 Arguments และ parameters

ตามที่ได้กล่าวมาแล้ว ข้อมูล(data items) ส่งโดย block เรียก เรา เรียกว่า actual parameters และ ข้อมูล ซึ่งจะรับใน procedure ถูกเรียก เรียกว่า formal parameters ในการส่งค่านั้น actual parameter แต่ละตัวจะจับคู่กับ formal parameter ที่สมนัยกัน โดยที่ actual parameter และ formal parameter ที่สมนัย กัน จะใช้ หน่วยความจำร่วมกัน actual parameter ถูกส่งโดยการ อ้างถึง ในกรณีนี้ การเปลี่ยนแปลงใด ๆ ของ formal parameter ใน procedure ถูกเรียก จะเปลี่ยน ค่าของ actual parameter ใน block ที่กำลังเรียกด้วย

เมื่อ actual และ formal parameters ไม่ได้ใช้เนื้อที่หน่วยความจำร่วมกัน,

actual parameter จะถูกส่งโดยค่า (value) ในกรณีนี้ ค่าของ actual parameter ตัวนั้นจะถูกก๊อปปี้ ส่งไปยัง procedure ถูกเรียก ดังนั้น การเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นกับ formal parameter จะมีผลเฉพาะค่าก๊อปปี้ เท่านั้น ไม่มีผลกับค่าของ actual parameter แต่อย่างใด

arguments ซึ่งส่ง โดยการอ้างถึง เป็นตัวแปรต่าง ๆ ซึ่ง attribute conform กับ formal parameters, Aggregate expressions ประกอบด้วย อะเรย์ และโครงสร้าง ซึ่งต้อง conform ช่วงของ subscript, type, precision และ scale เสมอ

actual parameter ถูกส่งโดยค่า เมื่อ มันเป็นตัวใดตัวหนึ่งใน : a constant, an entry name, an expression ที่ประกอบด้วย variable references และ operators, variable reference ที่อยู่ในเครื่องหมายวงเล็บ, a function invocation, หรือ an expression ซึ่งมี specification ไม่เหมือนกับ formal parameter ในกรณีหลังนี้ actual parameter จะถูกเปลี่ยนรูป ให้เป็นชนิด, precision และ scale ของ formal parameter

ตัวอย่าง

```
P: PROCEDURE (A,B,C);
   DCL
      A CHAR(10),
      B FIXED,
      C FLOAT;
         :
         :
```

คำสั่ง CALL ข้างล่างนี้ ส่ง actual parameters 3 ตัว ไปยัง procedure P สมนัยกับ formal parameters ทั้ง 3 ตัว คือ A, B, และ C ในตัวอย่างนี้ สมมติว่า X เป็น CHAR(10), Y และ Z เป็น FIXED ทั้งคู่

```
CALL P(X,(Y),Z);
```

actual parameter X ตัวแรก ถูกส่งโดยการอ้างถึง เนื่องจากมันจับคู่กับ formal parameter A, parameter ตัวที่สอง ถูกส่งโดยค่า เนื่องจากมันปรากฏในรูปของ expression และ parameter ตัวที่สาม จะถูกเปลี่ยนรูป ให้เป็นชนิด FLOAT แล้วถูกส่งโดยค่า

8.17 ENTRY attribute

data item ชนิด entry ใช้ในการกำหนด ชื่อ procedure แบ่งออกเป็น 2 ชนิด คือ ค่าคงที่เอนทรี (entry constants) และตัวแปรเอนทรี (entry variables)

ค่าคงที่เอนทรี สมนัยกับ internal procedure หรือ external procedures ที่แยกคอมไพล์ต่างหาก ตัวแปรเอนทรี เป็นข้อมูล ซึ่งมีการกำหนดค่า ให้เป็นค่าคงที่เอนทรี ระหว่างการ execute โปรแกรม

ลักษณะเฉพาะของ formal parameters และค่าที่จะส่งกลับสำหรับ procedure ซึ่งคอมไพล์ ภายนอก ต้อง นิยาม ใน โปรแกรมเรียก ด้วย การ declare เป็น ENTRY เป็นเรื่องจำเป็นที่ โปรแกรมเมอร์ ต้องแน่ใจว่า ได้มีการ declare entry ถูกต้อง จับคู่กับ (match) procedure ซึ่ง นิยาม ภายนอก เพื่อให้มีการลิงค์อย่างถูกต้อง จะเกิดขึ้นเมื่อส่วนของโปรแกรม combined กับ link editor นอกจากนี้แล้ว ตัวแปรซึ่ง จะมีค่าเป็นค่าคงที่เอนทรี จะต้อง นิยาม ด้วยการ declare เป็น ENTRY เช่นเดียวกัน ตัวแปรเอนทรีอาจจะมี subscript ก็ได้ ถ้าจำเป็นต้องใช้, attribute ENTRY ใช้สำหรับ นิยาม ไอเดนติไฟเออร์ ให้เป็นข้อมูลเอนทรี และให้ attribute ของ formal parameters เช่นเดียวกันกับ attribute ของค่าที่จะส่งกลับ ซึ่งอาจจะมีหรือไม่มีก็ได้ ถ้า entry item นั้นเป็นฟังก์ชัน รูปแบบของการ declare ENTRY มีดังนี้

```
DECLARE procname [(sub1, ... ,sub-n)]
        [VARIABLE] [ENTRY [(att1, ... ,att-m)]]
        [RETURNS (ret-att)];
```

เมื่อ attribute เหล่านี้ จะเรียงลำดับอย่างไรก็ได้ แต่ จะต้องมี ENTRY หรือ RETURNS อย่างน้อยหนึ่งตัวอยู่ ไอเดนติไฟเออร์ ที่กำหนดโดย procname เป็นชื่อของ data item ชนิด entry

sub1,...,sub-n เป็นรายชื่อ subscript อาจจะมีหรือไม่มีก็ได้

att1,...,att-m เป็นรายชื่อ ของ formal parameter attributes

ret-att เป็น attribute ของค่าที่จะส่งกลับ สำหรับฟังก์ชัน อาจจะมีหรือไม่มีก็ได้

attribute VARIABLE บอกให้ทราบว่า ข้อมูลนั้น เป็น ตัวแปรเอนทรี ซึ่งต้องมีการกำหนดค่าให้เป็นค่าคงที่เอนทรี ระหว่างการ execute โปรแกรม

หมายเหตุ รายชื่อของ subscript จะถูกต้องก็ต่อเมื่อ item นั้น มี attribute เป็น VARIABLE และ รายชื่อของ formal parameter attributes ไม่ต้องใส่ไว้ ถ้า procedure นั้นไม่ต้องใช้ parameters ในกรณีนี้ ถ้ามี attribute RETURNS ก็ไม่ต้องมี attribute ENTRY ถ้ามี attribute RETURNS

ถ้ากำหนด dimension attribute สำหรับพารามิเตอร์ตัวใดตัวหนึ่ง จะต้องเป็น attribute แรก ที่ถูก declare ถ้า formal parameter เป็น structure ข้อเท็จจริงของ โครงสร้างจะถูกกำหนดโดย level number ข้างหน้า attribute definition สำหรับในรายชื่อของ att1 ถึง att-m จะทำให้อยู่ในรูปของ attribute factoring ไม่ได้

ตัวอย่าง

```
DECLARE  X  ENTRY;
```

```
DECLARE  Y  ENTRY  VARIABLE;

DECLARE  P(0:10) ENTRY (FIXED, FLOAT) VARIABLE;

DECLARE  Q  ENTRY(1, 2 FIXED, 2  FLOAT, (5:10)
            DECIMAL);

DECLARE  R  RETURNS (CHAR(10));
```