บทที่ 9

คำสั่งและส่วนที่เพิ่มเติมจากบทอื่น ๆ

9.1 แฟ้มข้อมูลและการเข้าถึงแฟ้มข้อมูล

การนำข้อมูลเข้า/ออกในภาษาฟอร์แทรน 77 นั้นมีทางเลือกมากมายสำหรับการสร้างและประมวลผลแฟ้มข้อมูล ตอนนี้จะแนะนำแฟ้มข้อมูลและแนวความคิดในการประมวลผลแฟ้มข้อมูล จะเป็นเกี่ยวกับการใช้แฟ้มข้อมูลแบบเข้าถึงโดยลำดับ (sequential files) และแฟ้มข้อมูลที่มีฟอร์แมท (formatted files) ซึ่งมักจะเป็นแฟ้มข้อมูลที่พบว่าถูกใช้บ่อยในภาษาฟอร์แทรน 77

ภาษาฟอร์แทรน 77 มีเทคนิคสำหรับการเก็บข้อมูลไว้ภายในหน่วยความจำหลักหลายวิธีด้วยกัน ความรู้เกี่ยวกับโครงสร้างของแฟ้มข้อมูลและวิธีการที่แฟ้มข้อมูลเหล่านี้จะถูกอ่าน/เขียน (เข้าถึง) เป็นกุญแจของการใช้แฟ้มข้อมูลเหล่านี้อย่างประสบผลสำเร็จ

9.1.1 โครงสร้างของแฟ้มข้อมูลและระเบียน

แฟ้มข้อมูลที่กำหนดโดยภาษาฟอร์แทรน 77 อาจเป็นได้ทั้งแฟ้มข้อมูลภายนอกและแฟ้มข้อมูลภายใน แฟ้มข้อมูลภายนอกหมายถึงการที่ข้อมูลและโปรแกรมถูกเก็บอยู่นอกหน่วยความจำหลัก ซึ่งอาจจะเก็บในเทปแม่เหล็กหรือจานแม่เหล็กก็ได้

แฟ้มข้อมูลภายนอกประกอบด้วยระเบียนหรือบรรทัด ระเบียนเหล่านี้อาจจะมีฟอร์แมทหรือไม่มีฟอร์แมท (unformatted) ก็ได้ เลขจำนวนและตัวอักขระที่อยู่ในเครื่องถูกเก็บในแฟ้มข้อมูลที่มีฟอร์แมทดังเช่นที่มันปรากฏบนจอภาพ (ถ้าใช้เทอร์มินัลเป็นเครื่องนำข้อมูลเข้า) ระเบียนที่ไม่มีฟอร์แมทจะถูกบันทึกหรืออ่านโดยตรงในลักษณะที่มันเก็บอยู่ในคอมพิวเตอร์ ระเบียนที่ไม่มีฟอร์แมทไม่ถูกแปลถ้าเรากำหนดมันผ่านทางจอภาพ

ในภาษาฟอร์แทรน 77 ระเบียนที่มีฟอร์แมทจะถูกอ่านและเขียนภายใต้การควบคุมของ FORMAT ระเบียนที่ไม่มีฟอร์แมทไม่ต้องใช้การกำหนดฟอร์แมท สิ่งที่ถูกเก็บในคอมพิวเตอร์คือตัวข้อมูลและโปรแกรมโดยตรง

9.1.2 วิธีการเข้าถึงข้อมูล (Access Methods)

ข้อมูลและโปรแกรมในหน่วยความจำภายนอกอาจถูกเข้าถึงโดยทางใดทางหนึ่งใน 2 ทาง คือ โดยลำดับหรือโดยสุ่ม ในแฟ้มข้อมูลแบบเข้าถึงโดยลำดับนั้นลำดับของการเข้าถึงระเบียน (อ่านหรือบันทึก) คือลำดับเดียวกับที่ระเบียนเหล่านั้นปรากฏอยู่จริง ๆ ในแฟ้มข้อมูล ในการอ่านหรือ

บันทึกระเบียนที่ห้าในแฟ้มข้อมูลแบบต่อเนื่องไม่มีวิธีการใดที่จะกระโดดข้ามระเบียน 4 ระเบียนแรกไปได้ ถ้าท่านเพิ่งอ่านระเบียนที่ 5 เสร็จ มีวิธีเดียวที่ท่านจะอ่านระเบียนที่ 2 คือท่านต้องกลับไปที่จุดเริ่มต้นของแฟ้มข้อมูลและอ่านระเบียนแรกก่อนแล้วจึงอ่านระเบียนที่ 2 ถ้าท่านสมมุติว่าพจนานุกรมเป็นแฟ้มข้อมูลแบบลำดับ แล้วท่านจะอ่านบันทึกละคำเริ่มต้นจาก A ไปเรื่อย ๆ ถึง Z จนกระทั่งท่านถึงคำที่ท่านต้องการ ดังนั้นถ้าท่านกำลังค้นหาคำว่า zebra ท่านต้องเริ่มอ่านจาก A ถึง Y และหลาย ๆ คำที่ขึ้นต้นด้วย Z ที่อยู่ก่อนจะถึงคำ zebra สำหรับสถานการณ์ของการประมวลผลหลาย ๆ สถานการณ์ที่ท่านต้องการอ่านแฟ้มข้อมูลทั้งแฟ้ม วิธีการเข้าถึงโดยลำดับเป็นวิธีการที่เพียงพอ

วิธีการเข้าถึงโดยสุ่ม (random access) หรือการเข้าถึงโดยตรง (direct access) ทำให้ท่านสามารถอ่านระเบียนในแฟ้มข้อมูลได้โดยตรง นั่นคือไม่ต้องอ่านระเบียนที่อยู่ก่อนมันในแฟ้มข้อมูล ในที่นี้ลำดับของการอ่าน/บันทึกอาจจะไม่เป็นไปตามลำดับจริง ๆ ที่มันเก็บอยู่ในแฟ้มข้อมูล

แฟ้มข้อมูลแบบเข้าถึงโดยลำดับถูกสร้างขึ้นโดยการบันทึกทีละระเบียนในลำดับที่ท่านต้องการใช้โปรแกรม แฟ้มข้อมูลแบบเข้าถึงโดยตรงถูกสร้างโดยโปรแกรมด้วยการบันทึกลงบนจานแม่เหล็กโดยใช้วิธีการเข้าถึงตรง

สรุปได้ว่า แฟ้มข้อมูลสามารถเก็บระเบียนที่มีฟอร์แมทหรือไม่มีฟอร์แมทก็ได้ แฟ้มข้อมูลเหล่านี้ (และระเบียนองค์ประกอบของมัน) สามารถถูกเข้าถึงได้ในลักษณะโดยลำดับหรือโดยตรง มีวิธีที่ประกอบกันจาก 4 วิธี ในภาษาฟอร์แทรน 77 แฟ้มข้อมูลมักเป็นแบบต่อเนื่องและมีฟอร์แมทหรือแบบเข้าถึงโดยตรงและไม่มีฟอร์แมท

9.2 การประมวลผลแฟ้มข้อมูล

ท่านต้องกำหนดชนิดของแฟ้มข้อมูลและวิธีการเข้าถึงและให้ข่าวสารที่พอเพียงเพื่อที่การนำข้อมูลเข้า/ออกสามารถเกิดขึ้นได้อย่างถูกต้อง การกำหนดแฟ้มข้อมูลในภาษาฟอร์แทรน 77 ทำได้โดยการใช้คำสั่ง OPEN, READ, WRITE และ CLOSE

9.2.1 คำสั่ง OPEN

รูปทั่วไป

```
OPEN (open-list)
```

โดยที่ open-list ต้องประกอบด้วย

1) ตัวเลขที่แทนแฟ้มข้อมูล (หมายเลขแฟ้ม) ที่เราจะทำการเปิด (เพื่ออ่านหรือบันทึก) เราจะอ้างถึงแฟ้มนี้ด้วยตัวเลขดังกล่าว ซึ่งมีรูปการเขียน

UNIT=integer expression เช่น UNIT=6

หรือ integer expression เช่น 6

integerexpressionต้องมีค่าเป็นเลขจำนวนเต็มบวกเท่านั้น

2) FILE='filename'

filename คือชื่อแฟ้มข้อมูล ตั้งตามหลักการตั้งชื่อตัวแปร

3) STATUS='{OLD, NEW, SCRATCH, UNKNOWN}'

คือบอกสถานภาพของแฟ้มข้อมูลที่ถูกเปิดดังนี้

'OLD' คือแฟ้มนั้นมีอยู่แล้วในระบบ (ก่อนการวิ่งโปรแกรม)

'NEW' คือแฟ้มใหม่ที่สร้างขึ้นขณะที่วิ่งโปรแกรม

'SCRATCH' คือแฟ้มที่เราไม่ต้องการเก็บไว้ในระบบ เมื่อหยุดปฏิบัติงานในโปรแกรมแล้วแฟ้มนี้จะถูกลบทันที กรณีนี้ไม่ต้องใช้ FILE='filename'

'UNKNOWN' คือไม่ทราบสถานภาพของแฟ้ม ในกรณีนี้ระบบจะกำหนดไว้ว่าจะตีความว่าอย่างไร เช่นอาจจะลองหาดูว่าเป็น 'OLD' หรือไม่ ถ้าไม่พบจะถือว่าเป็น 'NEW' เป็นต้น

กรณีที่ไม่ระบุ STATUS เลย ระบบจะถือว่า UNKNOWN

ใน open-list อาจประกอบด้วยสิ่งต่อไปนี้ด้วยก็ได้

4) IOSTAT=status-variable

เป็นสิ่งที่ใช้แสดงว่าการเปิดแฟ้มข้อมูลมีปัญหาหรือไม่อย่างไร status-variable เป็นตัวแปรจำนวนเต็ม และจะมีค่าเป็นศูนย์ถ้าการเปิดแฟ้มข้อมูลไม่มีปัญหาใด ๆ

5) ERR=n (เลขประจำคำสั่งของคำสั่งปฏิบัติการ)

ใช้ระบุคำสั่งที่จะให้ไปทำถ้าการเปิดแฟ้มมีข้อผิดพลาด

6) ACCESS='{SEQUENTIAL}'
{DIRECT }

ใช้ระบุวิธีการเข้าถึงข้อมูล (Access method) ว่าเป็นแบบเข้าถึงโดยลำดับ (SEQUENTIAL) หรือแบบเข้าถึงโดยตรง (DIRECT) ถ้าไม่ระบุจะถือว่าเป็น SEQUENTIAL

7) FORM='{FORMATTED }'
{UNFORMATTED}

ใช้ระบุว่าแฟ้มข้อมูลประกอบด้วยระเบียนที่มีฟอร์แมทหรือไม่มีฟอร์แมท ถ้าไม่ระบุถือว่าเป็น 'FORMATTED'

8) RECL=record-length

ใช้ระบุความยาวของระเบียน ใช้สำหรับแฟ้มข้อมูลแบบเข้าถึงโดยตรงเท่านั้น record--length เป็นนิพจน์ชนิด integer ซึ่งต้องมีค่าเป็นบวก มีความหมายดังนี้

-ในแฟ้มแบบมีฟอร์แมท จะเป็นจำนวนตัวอักขระในแต่ละระเบียน และ

-ในแฟ้มแบบไม่มีฟอร์แมท มันจะเป็นตัววัดความยาวของระเบียนซึ่งวิธีการขึ้นอยู่กับระบบคอมพิวเตอร์

9) BLANK='{ZERO}'
{NULL}

ใช้ระบุว่าสดมภ์ที่เป็นช่องว่าง (blank column) ในรายการข้อมูลตัวเลขหรือฟิลด์ตัวเลข (numeric field) จะถูกแปลเป็นศูนย์ (zero) หรือเพื่อให้ไม่สนใจมัน (null) แต่อย่างไรก็ดีถ้าในฟิลด์ตัวเลขเป็นช่องว่างหมด มันจะแปลเป็นศูนย์

ตัวอย่าง

```
OPEN (UNIT=10,FILE='INFO1',STATUS='OLD')
```

ตัวอย่าง แสดงการอ่านชื่อแฟ้มข้อมูลหรือระบุชื่อแฟ้มในระหว่างวิ่งโปรแกรมได้เช่น

```
CHARACTER*10 INFILE
PRINT *,'ENTER NAME OF INPUT FILE'
READ *,INFILE
OPEN (10,FILE=INFILE,STATUS='OLD')
```

แฟ้มข้อมูลหมายเลข 10 จะเป็นแฟ้มข้อมูลแบบเข้าถึงโดยลำดับและมีฟอร์แมทโดยปริยาย หรืออาจระบุให้ชัดเจนโดยเขียนคำสั่ง OPEN ดังนี้

```
OPEN (UNIT=10,FILE=INFILE,STATUS='OLD',
*FORM='FORMATTED',ACCESS='SEQUENTIAL')
```

คำสั่ง OPEN (10,FILE=INFILE,STATUS='OLD',ERR=50) จะทำหน้าที่เช่นเดียวกันแต่ถ้าเกิดข้อผิดพลาดในการเปิดแฟ้ม คอมพิวเตอร์จะไปทำตามคำสั่งเลขที่ 50

ตัวอย่าง

```
OPEN (UNIT=11,FILE='INFO2',STATUS='NEW')
```

เป็นคำสั่งในการสร้างแฟ้มข้อมูลใหม่ชื่อ INFO2

ตัวอย่าง

```
OPEN (12,STATUS='SCRATCH')
```

เป็นคำสั่งในการสร้างแฟ้มข้อมูลชั่วคราว แฟ้มนี้จะถูกลบทิ้งเมื่อถูก CLOSE หรือเมื่อการปฏิบัติงานในโปรแกรมจบลง เราไม่ต้องตั้งชื่อแฟ้ม

9.2.2 คำสั่ง CLOSE

เราใช้คำสั่งนี้เพื่อปิดแฟ้มข้อมูลหรือทำให้แฟ้มข้อมูลและตัวเลขที่แทนแฟ้มข้อมูลไม่เกี่ยวข้องกันอีก

รูปทั่วไปคือ

```
CLOSE (close-list)
```

โดยที่ close-list ต้องประกอบด้วย

1) ตัวเลขแทนแฟ้มข้อมูลที่ระบุไว้ใน open-list คือในข้อความ (UNIT=) นั่นเอง

สิ่งต่อไปนี้อาจมีหรือไม่มีก็ได้

2) IOSTAT เพื่อตรวจจับข้อผิดพลาดที่อาจเกิดจากการพยายามปิดแฟ้มข้อมูล

3) ERR มีรูปแบบเดียวกับในคำสั่ง OPEN

4) STATUS='{KEEP / DELETE}' ข้อความนี้ใช้กับแฟ้มข้อมูลซึ่งถูกเปิดด้วย STATUS='SCRATCH' ไม่ได้

ถ้าไม่ระบุข้อความใด ๆ จะถือว่าเป็น KEEP ยกเว้นสำหรับแฟ้มข้อมูลแบบ SCRATCH

ตัวอย่าง ในการปิดแฟ้มข้อมูล INFO2 ซึ่งถูกสร้างด้วยคำสั่ง OPEN ข้างต้น เราใช้คำสั่ง

```
CLOSE (11)
CLOSE (UNIT=11)
CLOSE (UNIT=11,STATUS='KEEP')
```

ทั้ง 3 คำสั่งเหมือนกัน แฟ้มที่ถูกปิดด้วยคำสั่ง CLOSE แล้ว อาจถูกเปิดใหม่โดยคำสั่ง OPEN อีก ซึ่งอาจใช้หมายเลขแฟ้มตัวเดิมหรือใช้หมายเลขใหม่ แฟ้มที่ไม่ได้ถูกปิดด้วยคำสั่ง CLOSE จะถูกปิดโดยอัตโนมัติเมื่อการทำงานในโปรแกรมสิ้นสุดลง (ยกเว้นการหยุดเนื่องจากเกิดข้อผิดพลาด)

9.2.3 คำสั่ง INQUIRE

เราใช้คำสั่ง INQUIRE เพื่อตรวจสอบคุณสมบัติของแฟ้มข้อมูล หรือความเกี่ยวข้องของมันกับหมายเลขแฟ้ม

รูปทั่วไปคือ `INQUIRE (inquiry-list)`

โดยที่ inquiry-list จะต้องประกอบด้วยหมายเลขแฟ้มหรือชื่อแฟ้ม แต่จะไม่ใช้ทั้งคู่ และอาจมีข้อความ IOSTAT= ____ และ/หรือ ERR= ___

คำสั่ง INQUIRE ทำหน้าที่เป็นคำถามเพื่อสอบถามคุณสมบัติของแฟ้ม เมื่อคำสั่งนี้ถูกทำงาน ตัวแปรในข้อความแต่ละข้อความในคำสั่ง INQUIRE จะถูกกำหนดค่าซึ่งจะเป็นคำตอบของคำถามที่ได้ถามในคำสั่ง INQUIRE

ข้อความและความหมายของมันแสดงไว้ในตารางต่อไปนี้

ข้อความ	ชนิดของ variable	ค่าและความหมายของ variable
IOSTAT=variable	Integer	ค่าเป็นศูนย์ถ้าไม่มีข้อผิดพลาด ค่าเป็นบวกถ้ามีข้อผิดพลาด
EXIST=variable	Logical	ค่าเป็นจริงถ้าแฟ้มที่ระบุไว้ด้วยชื่อหรือหมายเลขแฟ้ม มีอยู่ในระบบ นอกนั้นค่าเป็นเท็จ
OPENED=variable	Logical	ค่าเป็นจริงถ้าตัวเลขที่ระบุนั้นได้ถูกกำหนดให้แฟ้มหรือถ้าแฟ้มมีหมายเลขตามที่ระบุไว้ นอกนั้นค่าเป็นเท็จ
NUMBER=variable	Integer	มีค่าเท่ากับหมายเลขแฟ้มหรือไม่กำหนด
NAMED=variable	Logical	ค่าเป็นจริงถ้าแฟ้มมีชื่อแล้ว นอกจากนั้นเป็นเท็จ
NAME=variable	Character	ค่าเป็นชื่อของแฟ้ม จะไม่กำหนดถ้าแฟ้มไม่มีชื่อ
ACCESS=variable	Character	ค่าเป็น SEQUENTIAL ถ้าสร้างแฟ้ม (เปิดแฟ้ม) โดยระบุ SEQUENTIAL access หรือค่าเป็น DIRECT ถ้าสร้างแฟ้มโดยระบุ DIRECT access นอกนั้นจะไม่กำหนด
SEQUENTIAL= variable	Character	ค่าเป็น YES ถ้าเราเข้าถึงแฟ้มได้โดยลำดับ ค่าเป็น NO ถ้าเราเข้าถึงแฟ้มไม่ได้โดยลำดับ ค่าเป็น UNKNOWN ถ้าแฟ้มไม่เหมาะสมกับการเข้าถึงโดยลำดับ

ข้อความ	ชนิดของ variable	ค่าและความหมายของ variable
DIRECT=variable	Character	ค่าเป็น YES ถ้าเราเข้าถึงแฟ้มได้โดยตรง ค่าเป็น NO ถ้าเราเข้าถึงแฟ้มไม่ได้โดยตรง ค่าเป็น UNKNOWN ถ้าแฟ้มไม่เหมาะสมกับการเข้าถึงโดยตรง
FORM=variable	Character	ค่าเป็น FORMATTED ถ้าเปิดแฟ้มแบบมีฟอร์แมท ค่าเป็น UNFORMATTED ถ้าเปิดแฟ้มแบบไม่มีฟอร์แมท จะไม่มีค่าถ้าแฟ้มไม่ถูกเปิด
FORMATTED=variable	Character	ค่าเป็น YES ถ้าแฟ้มเป็นแบบมีฟอร์แมท ค่าเป็น NO ถ้าแฟ้มเป็นแบบไม่มีฟอร์แมท ค่าเป็น UNKNOWN ถ้าไม่สามารถระบุชนิดได้
UNFORMATTED= variable	Character	ค่าเป็น YES ถ้าแฟ้มเป็นแบบไม่มีฟอร์แมท ค่าเป็น NO ถ้าแฟ้มเป็นแบบมีฟอร์แมท ค่าเป็น UNKNOWN ถ้าไม่สามารถระบุชนิดได้
RECL=variable	Integer	ค่าเท่ากับความยาวของระเบียนสำหรับแฟ้มข้อมูลแบบเข้าถึงโดยตรง และไม่มีค่าถ้าเราไม่ได้ระบุว่าแฟ้มข้อมูลเป็นแบบเข้าถึงโดยตรง
NEXTREC= variable	Integer	มีค่าเท่ากับ 1 บวกกับหมายเลขของระเบียนสุดท้ายซึ่งอ่านจากหรือพิมพ์ลงในแฟ้มข้อมูลแบบเข้าถึงโดยตรง และจะไม่กำหนดค่าถ้าไม่ทราบจำนวนระเบียน

ข้อความ	ชนิดของ variable	ค่าและความหมายของ variable
BLANK=variable	Character	ZERO ถ้าช่องว่างในฟิลด์ตัวเลขจะถูกแปลเป็นเลขศูนย์ NULL ถ้าเราไม่สนใจช่องว่างทั้งหลายและไม่ถูกกำหนดค่าถ้าไม่สามารถเข้าถึงแฟ้มได้

9.2.4 คำสั่ง READ

รูปทั่วไปคือ

```
READ     (control-list)input-list
```

โดยที่ input-list อาจเป็น ชื่อตัวแปร ชื่อของสายอักขระย่อย (substring name) ชื่อแถวลำดับ หรือ implied DO ซึ่งใช้เครื่องหมายจุลภาค (,) คั่น

control-list ต้องประกอบด้วย

1) หมายเลขแฟ้มที่ต้องการอ่าน

และอาจมีข้อความต่อไปนี้ตั้งแต่หนึ่งข้อความขึ้นไป

2) format codes ซึ่งอธิบายรูปแบบของข้อมูลที่จะอ่าน

3) END=n เป็นการระบุว่าคำสั่งที่มีเลขที่ประจำคำสั่ง n จะถูกปฏิบัติต่อไปเมื่ออ่านไปถึงจุดจบของแฟ้มข้อมูล (end of file record) แบบเข้าถึงโดยลำดับแล้ว

4) ERR=n เป็นการระบุว่าคำสั่งที่มีเลขที่ประจำคำสั่ง n จะถูกปฏิบัติต่อไปเมื่อเกิดข้อผิดพลาดในการนำข้อมูลเข้าแฟ้มข้อมูล

5) IOSTAT=status-variable เป็นการตรวจสอบสถานะของการนำข้อมูลเข้า

6) REC=นิพจน์ชนิด integer ใช้ระบุจำนวนระเบียนที่จะถูกอ่านสำหรับแฟ้มข้อมูลแบบเข้าถึงโดยตรง ข้อความนี้จะต้องมีถ้าข้อมูลเข้ามาจากแฟ้มข้อมูลแบบเข้าถึงโดยตรง ใน control-list จะมีทั้งข้อความ REC= ___ และ END= ___ ไม่ได้

ตัวอย่าง

```
INTEGER NUMBER
CHARACTERS20 NAME
READ(15,'(I5,A20)',ERR=20)NUMBER,NAME
:
20 PRINT*,'INPUT DATA ERROR'
```

ถ้าข้อมูลเข้าจากแฟ้มหมายเลข 15 คือ

```
123^J|OHN^HENRY^DOE
```

จะเกิดข้อผิดพลาดเมื่อพยายามอ่าน 123 J เพื่อเป็นค่าของ NUMBER ในกรณีนี้ คอมพิวเตอร์จะข้ามไปทำคำสั่ง เลขที่ 20 ตามที่ระบุไว้ในข้อความ ERR=20

ตัวอย่าง

```
INTEGERPARTNO,BADNUM,RECLEN
OPEN(10,FILE=FNAME,STATUS='OLD',
*ACCESS='DIRECT',FORM='FORMATTED',RECL=RECLEN)
READ(10,'(A)',REC=PARTNO,IOSTAT=BADNUM)INFO
```

BADNUM จะมีค่า + ถ้าการอ่านข้อมูลมีข้อผิดพลาด

จะมีค่า - ถ้าหมดข้อมูลแล้วแต่ไม่มีข้อผิดพลาด

จะมีค่า 0 ถ้าไม่เกิด 2 ลักษณะข้างต้น

PARTNO จะมีค่า + และเป็นตัวระบุจำนวนระเบียนที่ถูกอ่านจากแฟ้มข้อมูลแบบเข้าถึงโดยตรง

ตัวอย่าง โปรแกรมภาษาฟอร์แทรน 77

```
      PROGRAM INVEN
C PROGRAM TO READ A PART NUMBER DURING EXECUTION, ACCESS A
C RECORD IN A DIRECT ACCESS PARTS INVENTORY FILE, AND
C DISPLAY THIS RECORD.
C  VARIABLES  USED  ARE:
C      RECLEN : A PARAMETER SPECIFYING RECORD LENGTH
C      PARTNO : PARTNUMBER
```

```
C     FNAME : NAME OF THE FILE
C     INFO  : A RECORD IN THE FILE
C     BADNUM : 0 IF VALID PART NUMBER, OTHERWISE NONZERO
C*************************************************************
      INTEGER PARTNO,RECLEN,BADNUM
      PARAMETER (RECLEN=30)
      CHARACTER*20 NAME,INFO*(RECLEN)
C GET THE NAME OF THE FILE AND OPEN IT FOR DIRECT ACCESS
      PRINT *,'ENTER NAME OF FILE'
      READ '(A)',FNAME
      OPEN(UNIT=10,FILE=FNAME,STATUS='OLD',ACCESS='DIRECT',
     *FORM='FORMATTED',RECL=RECLEN)
      PRINT *,'ENTER PART NUMBER(0 TO STOP)'
      READ *,PARTNO
C WHILE THERE ARE MORE PART NUMBERS TO ACCESS DO THE FOLLOWING
   10 IF (PARTNO.NE.0)THEN
        READ(10,'(A)',REC=PARTNO,IOSTAT=BADNUM)INFO
        IF (BADNUM.EQ.0)THEN
          PRINT'(1X,''PART'',I3,'':'',A)',PARTNO,INFO
        ELSE
          PRINT'(1X,''INVALID PART NUMBER:'',I3)',PARTNO
        ENDIF
        PRINT*
        PRINT*, 'PART  NUMBER?'
        READ *,PARTNO
```

```
        GO TO 10
      ENDIF
      CLOSE(10)
      STOP
      END
```

ตัวอย่างแฟ้มข้อมูลชื่อ PARTSFILE ที่ใช้ทดสอบโปรแกรม

```
CHROME-BUMPER...$152.95.....15
SPARK-PLUG.....$1.25....125
DISTRIBUTER-CAP..$39.95.....57
FAN-BELT.....$5.80.....32
DOOR-HANDLE.....$18.85.....84
```

ตัวอย่างการวิ่งโปรแกรม

```
ENTER NAME OF FILE
PARTSFILE
ENTER PART NUMBER(0 TO STOP)
4
PART 4 : FAN-BELT.....$5.80.....32
PART NUMBER?
2
PART 2 : SPARK-PLUG.....$1.25....125
PART NUMBER?
10
INVALID PART NUMBER: 10
PART NUMBER?
0
```

ตัวอย่าง การเขียนคำสั่ง READ

วิธีที่ 1 วิธี list-directed เป็นวิธีที่ระบบคอมพิวเตอร์จะกำหนด format ของการนำข้อมูลเข้า/ออกเองโดยอัตโนมัติ

```
READ(5,*)NAME,TIME,RATE
```

หรือแบบอื่นที่เหมือนกันคือ

```
READ(5,FMT=*)NAME,TIME,RATE

READ(UNIT=5,FMT=*)NAME,TIME,RATE

READ(IN,*)NAME,TIME,RATE

READ(UNIT=IN,FMT=*)NAME,TIME,RATE
```

โดยที่ IN มีค่าเท่ากับ 5 ถ้าหน่วยนำข้อมูลเข้าเป็นหน่วยนำข้อมูลเข้ามาตรฐานของระบบคอมพิวเตอร์ เราอาจเขียนคำสั่งได้ดังนี้

```
READ(*,*)NAME,TIME,RATE
```

วิธีที่ 2 วิธีที่เราระบุ format เอง คำสั่งอาจอยู่ในรูป

```
READ(UNIT=*,FMT='(A,2F6.2)')NAME,TIME,RATE
```

หรือ

```
READ(5,10,END=20)NAME,TIME,RATE
10 FORMAT(A,2F6.2)
```

ตัวอย่าง

```
CHARACTER*40 FORM
REAL AMOUNT,RATE

FORM='(/F10.0///F5.0)'

READ FORM,AMOUNT,RATE
```

9.2.5 คำสั่ง WRITE

คำสั่ง WRITE ใช้ในการบันทึกข้อมูลลงแฟ้ม

รูปทั่วไปของคำสั่ง

```
WRITE(control-list)output-list
```

โดยที่ output-list เป็นรายการของนิพจน์ ชื่อแถวลำดับ implied DO โดยใช้เครื่องหมายจุลภาค (,) คั่น

ส่วน control-list จะต้องมี

1) หมายเลขแฟ้มที่ต้องการบันทึก โดยใช้ UNIT=u หรือ u เท่านั้น

และอาจมีข้อความต่อไปนี้ตั้งแต่หนึ่งข้อความขึ้นไปจากสิ่งต่อไปนี้

2) format codes ซึ่งอธิบายรูปแบบของการแสดงผล โปรแกรมเมอร์เป็นผู้เขียนเอง อาจใช้รูป FMT=format codes หรือ format codes เท่านั้น

3) ERR=n ใช้ระบุคำสั่งที่จะให้ทำเมื่อเกิดข้อผิดพลาดในการนำข้อมูลออก

4) IOSTAT=status-variable ใช้ตรวจสอบสถานะของการนำข้อมูลออก

5) REC=นิพจน์ชนิด integer ใช้ระบุจำนวนเรคคอร์ดที่จะบันทึกลงในแฟ้มข้อมูลแบบเข้าถึงโดยตรง ข้อความนี้จะปรากฏไม่ได้ถ้าข้อมูลออกเป็นแบบ list directed (นั่นคือใช้ * ในตำแหน่งของ format codes)

ตัวอย่าง การเขียนคำสั่ง WRITE

วิธีที่ 1 list directed

```
WRITE(6,*)BODY,GRAV,WEIGHT
```

หรือแบบอื่นที่เหมือนกันคือ

```
WRITE(6,FMT=*)BODY,GRAV,WEIGHT

WRITE(UNIT=6,FMT=*)BODY,GRAV,WEIGHT

WRITE(NOUT,*)BODY,GRAV,WEIGHT

WRITE(UNIT=NOUT,FMT=*)BODY,GRAV,WEIGHT
```

โดยที่ NOUT เป็นตัวแปรแบบ integer มีค่าเป็น 6 ในการณีที่หน่วยนำข้อมูลออกซึ่งมีหมายเลข 6 เป็นหน่วยนำข้อมูลออกมาตรฐานของระบบ เราอาจใช้เครื่องหมายคอกจัน (*) แทน 6 ได้

ตัวอย่าง

```
WRITE(*,*)BODY,GRAV,STATUS
```

ซึ่งเหมือนกับ

```
PRINT *,BODY,GRAV,STATUS
```

วิธีที่ 2 วิธีกำหนด format เอง คำสั่ง WRITE อาจมีรูปดังนี้

```
WRITE(6,'(1X,A,2F10.2)')BODY,GRAV,STATUS

WRITE(UNIT=6,FMT='(1X,A,2F10.2)')BODY,GRAV,STATUS
```

```
      WRITE(6,20)BODY,GRAV,STATUS
   20 FORMAT(1X,A,2F10.2)
```

9.2.6 คำสั่ง REWIND, BACKSPACE, ENDFILE (ใช้กับ Sequential file)

รูปทั่วไปคือ

REWIND	unit	หรือ	REWIND (position-list)
BACKSPACE	unit	หรือ	BACKSPACE(position-list)
ENDFILE	unit	หรือ	ENDFILK (position-list)

โดยที่ unit คือหมายเลขแฟ้ม

position-list ต้องประกอบด้วย

1) การระบุหมายเลขแฟ้มโดยใช้ UNIT=unit หรือ unit เท่านั้น

และอาจมีข้อความต่อไปนี้คือ

2) ERR=n ถ้าการทำงานเกิดข้อผิดพลาดจะข้ามไปทำคำสั่งที่ n

3) IOSTAT=status-variable ใช้ระบุค่าของตัวแปรในข้อความนี้ ถ้าค่าของมันเป็นศูนย์หมายความว่าการปฏิบัติตามคำสั่งถูกต้อง และค่าจะเป็นบวกถ้าเกิดข้อผิดพลาด

คำสั่ง REWIND เป็นคำสั่งให้กลับไปยังจุดเริ่มต้นของแฟ้ม นั่นคือระเบียนแรกของแฟ้ม

คำสั่ง BACKSPACE เป็นคำสั่งให้กลับไปที่จุดเริ่มต้นของระเบียนที่นำหน้ามา ถ้าแฟ้มข้อมูลอยู่ที่จุดเริ่มต้นแฟ้มอยู่แล้ว คำสั่งนี้จะไม่มีผล

คำสั่ง ENDFILE เป็นคำสั่งให้บันทึกระเบียนสุดท้ายของแฟ้ม (end-of-file record) ลงในแฟ้ม เมื่อคำสั่ง READ (ซึ่งมีข้อความ END=n อยู่) อ่านพบระเบียนนี้ มันจะไปทำคำสั่งที่ n ต่อไป หลังจากคำสั่ง ENDFILE ถูกทำแล้ว เราจะอ่านหรือบันทึกแฟ้มนี้ได้อีกเมื่อแฟ้มถูกกำหนดตำแหน่งใหม่ ซึ่งอาจจะให้ถอยกลับไปที่ระเบียนใด ๆ ก่อนระเบียนสุดท้ายของแฟ้ม โดยใช้ 2 คำสั่งข้างต้น

9.3 คำสั่ง PARAMETER

ใช้ในการกำหนดชื่อให้แก่ค่าคงที่

รูปทั่วไปคือ

```
PARAMETER (p=c[,p=c]...)
```

โดยที่ p เป็นชื่อที่ตั้งตามหลักการตั้งชื่อ

c เป็นค่าคงที่

ชื่อแต่ละชื่อ (p) จะเป็นค่าคงที่ และถูกกำหนดให้มีค่าเท่ากับ c ซึ่งระบุในคำสั่ง PARAMETER เราอาจใช้ชื่อค่าคงที่จากคำสั่ง PARAMETER ในการกำหนดแถวลำดับได้

ตัวอย่าง
```
REAL PI
PARAMETER (PI=3.1416)
```

ตัวอย่าง
```
INTEGER LIMIT
REAL PI
PARAMETER(LIMIT=50,PI=3.1416)
```

ตัวอย่าง
```
INTEGER LIMIT
PARAMETER (LIMIT=25)
INTEGER TEMP(LIMIT),I,COUNT
REAL SUM,TMEAN
PRINT*,'ENTER TEMPERATURES:'
READ(*,*,END=10)(TEMP(I),I=1,LIMIT)
:
```

เราใช้คำสั่ง PARAMETER เพื่อระบุความยาวของค่าคงที่อักขระ ดังตัวอย่างต่อไปนี้

ตัวอย่าง
```
INTEGER LENGTH
PARAMETER (LENGTH=10)
CHARACTER*(LENGTH) ALPHA,BETA*(2*LENGTH)
```

ซึ่งเป็นการกำหนดให้ ALPHA และ BETA เป็นตัวแปรอักขระซึ่งยาว 10 ตัวอักขระและ 20 ตัวอักขระตามลำดับ

ถ้าพารามิเตอร์เป็นชนิดค่าคงที่อักขระ เราไม่จำเป็นต้องระบุขนาดของมันในคำสั่ง CHARACTER เราใช้ * (indefinite length specifier) ในกรณีนี้ค่าของพารามิเตอร์คือความยาวของสายอักขระที่มีชื่อนั้น ๆ

ตัวอย่าง

```
      INTEGER LIMIT
      CHARACTER *(*)TITLE
      PARAMETER (LIMIT=50,TITLE='POPULATION READING')
```

ทำให้พารามิเตอร์ TITLE มีความยาว 18

ตัวอย่าง โปรแกรมเพื่อนับอุณหภูมิที่อ่านเข้าไป แล้วคำนวณอุณหภูมิเฉลี่ย แล้วพิมพ์อุณหภูมิที่มีค่าสูงกว่าอุณหภูมิเฉลี่ยที่หาได้

```
C2345678901234...
      PROGRAM TEMPS
C*****************************************************************
C PROGRAM TO READ A LIST OF TEMPERATURES, COUNT THEM, CALCULATE THE
C WAN TEMPERATURE, AND THEN PRINT A LIST OF TEMPERATURES WHICH ARE
C GREATER THAN THE MEAN. VARIABLES USED ARE:
C      LIMIT : LIMIT ON NUMBER OF ARRAY ELEMENTS
C      TEMP : ONE-DIMENSIONAL ARRAY OF TEMPERATURES
C      I     :  SUBSCRIPT
C      COUNT : NUMBER OF TEMPERATURES
C      SUM   : SUM OF TEMPERATURE
C      TMEAN : MEAN   TEMPERATURES
C*****************************************************************
      INTEGER  LIMIT
      PARAMETER (LIMIT=25)
      INTEGER TEMP(LIMIT),I,COUNT
```

```
      REAL SUM,TMEAN
      PRINT*,'ENTER TEMPERATURES:'
      READ(*,*,END=10)(TEMP(I),I=1,LIMIT)
C SINCE THE INDEX IS ONE MORE THAN THE ACTUAL COUNT, SUBTRACT ONE
   10 COuNT=I-1
c  CALCULATE THE MEAN TEMPERATURE
      SUM=0
      Do 20 I=1,COUNT
      SUM=SUM+TEMP(I)
   20 CONTINUE
      TMEAN=SUM/COUNT
      PRINT 30,COUNT,TMEAN
   30 FORMAT(//1X,I3,'TEMPERATURES WITR MEAN=',F6.1)
C PRINT LIST OF TEMPERATURES GREATER THAN THE MEAN
      PRINT 40
   40 FORMAT(//'LIST OF TEMPS GREATER THAN MEAN:')
      DO 50 I=1,COUNT
      IF(TEMP(I).GT.TMEAN)PRINT*,TEMP(I)
   50 CONTINUE
      STOP
      END
```

0.4 <u>คำสั่ง PROGRAM</u>

คำสั่ง PROGRAM ใช้ในการกำหนดชื่อให้แก่โปรแกรมหลัก

รูปทั่วไป

```
PROGRAM name
```

โดยที่ name เป็นชื่อของโปรแกรมตั้งตามหลักการตั้งชื่อ คำสั่งนี้จะมีหรือไม่มีก็ได้

ถ้ามีต้องใส่เป็นคำสั่งแรกของโปรแกรมหลัก ชื่อ name จะต้องไม่ซ้ำกับตัวแปรใด ๆ ในโปรแกรม ต้องไม่ซ้ำกับชื่อโปรแกรมย่อย ชื่อของ ENTRY หรือชื่อของ BLOCK common ในโปรแกรมของงานเดียวกัน

9.5 Format code

9.5.1 G-format code

รูปทั่วไปคือ

rGw.d

G-format code ใช้กับเลขจำนวนจริง ผลของการแสดงเลขจำนวนจริงโดยใช้ G-format code นั้นจะอยู่ในรูปของเลขจำนวนจริงแบบทั่วไป (ไม่มีเลขชี้กำลัง) นั่นคืออยู่ในรูปเดียวกับการใช้ F-format code หรือจะเป็นแบบมีเลขชี้กำลังก็ได้ ทั้งนี้ขึ้นอยู่กับค่าสัมบูรณ์ (absolute value) ของเลขจำนวนจริงนั้น ๆ

สมมุติเลขจำนวนจริงซึ่งใช้ G-format code นั้นมีค่าในรูป

$$\pm 0.d_1 \ldots d_n \times 10^k$$

ถ้า $0 \le k \le d$ มันจะแสดงผลในรูปเดียวกับการใช้ F-format code ซึ่งมีความกว้างของฟิลด์เป็น w-4 และตามมาด้วย 4 ช่องว่าง

แต่ถ้า k เป็นค่าลบหรือมากกว่า d มันจะแสดงผลในรูปของการใช้ Ew.d

ในทั้ง 2 กรณี จำนวนตัวเลขนัยสำคัญคือ d ตัว

ตัวอย่าง

ค่า	G-format	ผลการแสดง
0.123456	612.6	0.123456^^^^
0.123456E1	011.6	1.23456^^^^
0.123456E5	G11.6	12345.6^^^^
0.123456E6	611.6	123456.^^^^
0.123456E7	612.6	0.123456E+07

ในการอ่านเลขจำนวนจริงด้วย G-format code นั้นจะคล้ายคลึงกับการใช้ F-format Code

9.5.2 Scale factor

เราอาจใส่ Scale factor นำหน้า format code E, F, G และ D ได้

Scale factor อยู่ในรูป | nP |

โดยที่ n เป็นเลขจำนวนเต็ม

ในการนำข้อมูลออก nPFw.d ทำให้ค่าที่แสดงออกมานั้นถูกคูณด้วย 10^n

สำหรับ E (และ D) คือ nPEw.d ทำให้ส่วนที่เป็นเลขทศนิยมที่แสดงออกมานั้นถูกคูณด้วย 10^n และเลขชี้กำลังจะลดลง n

สำหรับ G นั้น scale factor จะมีผลก็ต่อเมื่อค่าที่จะถูกพิมพ์นั้นเป็นการที่จะแสดงด้วยรูป Ew.d เท่านั้น และผลของ scale factor จะเป็นเช่นเดียวกับที่ได้อธิบายแล้วสำหรับ E-format code

ตัวอย่าง N เก็บค่า 27

X เก็บค่า -93.2094

Y เก็บค่า -0.0078

Z เก็บค่า 55.3512

จากคำสั่ง
```
      PRINT 1,N,X,Y,Z
    1 FORMAT(1X,I2,2F11.3,E12.4)
      PRINT 2,N,X,Y,Z
    2 FORMAT(1X,I2,1P2F11.3,3PE12.4)
      PRINT 3,N,X,Y,Z
    3 FORMAT(1X,I2,-1P2F11.3,E12.4)
```

ผลการพิมพ์จะเป็นดังนี้

```
 27  -93.209    -0.008  0.5536E+02
 27  -932.094   -0.070   553.61E-01
 27   -9.321    -0.001   0.0554E+03
```

ในคำสั่ง FORMAT สุดท้ายนั้น เมื่อเราใช้ Scale factor แล้ว มันจะถือว่ามี scale factor นั้นสำหรับ format E, F, G และ D ที่ตามมาทั้งหมด นั่นคือคำสั่ง FORMAT ดังกล่าวเหมือนกับเราใช้

3 FORMAT(1X,I2,-1P2F11.3,-1PE12.4)

ในกรณีที่เราไม่ต้องการให้ E12.4 มี scale factor -lP เราจะใส่เลขศูนย์แทน -1 นั่นคือใส่ 0PE12.4

ในกรณีของการนำข้อมูลเข้านั้น การใช้จะคล้ายกันมาก จะต่างกันตรงที่ถ้าเลขจำนวนจริงในฟิลด์อยู่ในรูปที่มีเลขชี้กำลัง แล้ว scale factor จะไม่มีผล

ตัวอย่าง
```
      REAL A,B,C
      READ 15,A,B,C
   15 FORMAT(2PF6.0,-2PF6.0,F6.0)
```
ถ้าข้อมูลคือ ^^^1.1^^^1.1^^^1.1

ค่าที่อ่านได้คือ A เก็บค่า 110.0

B เก็บค่า .011

c เก็บค่า .011 (-2PF6.0)

ถ้าข้อมูลคือ ^1.1E0^^^1.1^1.1E0

ค่าที่อ่านได้คือ A เก็บค่า 1.1

B เก็บค่า .011

c เก็บค่า 1.1

9.5.3 BN, BZ-format code

BN ย่อมาจาก Blank null

BZ ย่อมาจาก Blank zero

ช่องว่างในฟิลด์ตัวเลขนั้น จะมีค่าหรือไม่มีอย่างไรนั้นขึ้นอยู่กับตัวคอมไพเลอร์ คอมไพเลอร์บางตัวถือว่าเป็นศูนย์ และบางตัวจะไม่สนใจมันเลย ในภาษาฟอร์แทรน 77 เราอาจจะบุความต้องการได้โดยการใช้ BN หรือ BZ-format code

ตัวอย่าง ข้อมูล $537_{\wedge\wedge\lfloor}6.258E3_{\wedge\lfloor}$

คำสั่ง READ'(BZ,I5,F8.0)',NUM,ALPHA

จะทำให้ NUM เก็บค่า 53700

และ ALPHA เก็บค่า 6.258E30

ทั้งนี้เนื่องจาก BZ-format code แปลช่องว่างในฟิลด์ให้เป็นศูนย์

แต่ถ้าเราใช้คำสั่ง READ'(BN,I5,F8.0)',NUM,ALPHA

จะทำให้ NUM เก็บค่า 537

และ ALPHA เก็บค่า 6.258E3

ทั้งนี้เนื่องจาก BN-format code ทำให้ไม่ต้องสนใจช่องว่างดังกล่าวข้างต้น

คำสั่ง READ'(BZ,I5,BN,F8.0)',NUM,ALPHA

จะทำให้ NUM เก็บค่า 53700

และ ALPHA เก็บค่า 6.258E3

9.5.4 S, SP และ SS

เราใช้ S, SP และ SS ในการควบคุมการแสดงเครื่องหมายบวก (+) ในข้อมูลออกที่เป็นเลขจำนวน ถ้า SP (sign positive) ปรากฏในคำสั่ง FORMAT เลขจำนวนบวกทั้งหมดในข้อมูลออกจะแสดงเครื่องหมายบวกไว้ด้วย ในทางตรงข้ามถ้าใช้ SS (Sign Suppress) มันจะลบเครื่องหมายบวกออกหมด ส่วน S นั้นอาจใช้ในการทำให้กลับไปอยู่ในสถานะเดิม ซึ่งอาจเป็นการแสดงหรือไม่แสดงเครื่องหมายบวก

9.5.5 Colon (:) กับการย้อนกลับมาใช้ format ซ้ำ

ตัวอย่าง
```
INTEGER M,N
REAL A,B
CHARACTER*10 ITEM1,ITEM2,FORMA*30,FORMB*30
FORMA='(1X,I5,3I6)'
FORMB='(1X,F5.1,F7.0,F10.4)'
M=1
```

```
      N=2
      A=5.6
      B=7.8
      ITEM1='BUMPER'
      ITEM2='HEADLIGHT'
      PRINT FORMA,N,M
      PRINT FORMB,A,B
      PRINT 10,ITEM1,ITEM2
   10 FORMAT(1X,5(' ITEM IS ',A10))
      PRINT 20,ITEM1,ITEM2
   20 FORMAT(1X,5(': ITEM IS ',A10))
```

ในกรณีที่ตัวแปรมากกว่า format code ที่ระบุไว้ จะเกิดการย้อนกลับไปใช้ format code ซ้ำตั้งแต่วงเล็บเปิดขวาสุดเป็นต้นไป ค่าคงที่อักขระถูกพิมพ์ซ้ำอีก จะหยุดเมื่อมันพบ

1) วงเล็บปิดของ format code

2) format I, F, E, A หรือ L หรือ (G หรือ D)

3) เครื่องหมาย :

ดังนั้นผลการพิมพ์คือ

```
  1    2
6.6   8.
ITEM IS BUMPER       ITEM Is HEADLIGHT   ITEM IS
ITEM IS BUMPER       ITEM IS READLIGRT
```

เช่นเดียวกับการใช้เครื่องหมายขีดทับ (/) เราไม่ต้องใส่เครื่องหมายจุลภาค (,) ก่อนและหลังเครื่องหมาย :

9.5.6 TL, TR-format code

รูปทั่วไป

TLn
TRn

โดยที่ n เป็นเลขจำนวนเต็มบวก

ใช้ในการกำหนดที่ ๆ ตัวอักขระตัวต่อไปว่าจะปรากฏที่ n ตำแหน่งทางซ้ายหรือทางขวาตามลำดับ (ทั้งนี้ขึ้นอยู่กับตำแหน่งปัจจุบันด้วย)

นั่นคือ TLn ทำให้เกิดการย้อนกลับ n ตำแหน่ง อย่างไรก็ดี format ถัดไปนั้นอาจจะทำให้ตัวอักขระใน n ตำแหน่งนี้ถูกแทนที่ ไม่ใช่ถูกพิมพ์ทับ

TRn นั้นมีความหมายเช่นเดียว nX

การใช้ TL-format code ทำให้เราสามารถอ่านข้อมูลเดียวกันซ้ำหลายครั้งได้

ตัวอย่าง ข้อมูล |123

คำสั่ง INTEGER NUM

REAL BETA

CHARACTER*3 STR

READ'(I3,TL3,F3.1,TL3,A3)',NUM,BETA,STR

จะทำให้ NUM เก็บค่า 123

BETA เก็บค่า 12.3

STR เก็บค่า '123'

9.6 คำสั่ง ASSIGN และ ASSIGNED GO TO

รูปทั่วไปของคำสั่ง

GO TO ตัวแปรชนิด integer,(n_1,n_2,...,n_k)
ASSIGN เลขประจำคำสั่ง TO ตัวแปรชนิด integer

คำสั่ง assigned GO TO ใช้ตัวแปรชนิด integer ในการเลือกคำสั่งที่จะถูกกระทำต่อไป ในที่นี้ $n_1,n_2,...,n_k$ เป็นเลขประจำคำสั่งของคำสั่งปฏิบัติการ ก่อนคำสั่ง assigned GO TO จะต้องกำหนดเลขประจำคำสั่งให้เก็บไว้ในตัวแปรชนิด integer ก่อนโดยใช้คำสั่ง ASSIGN ดังนั้นตัวแปรชนิด integer จะเท่ากับ n ตัวใดตัวหนึ่งในคำสั่ง assigned GO TO

ตัวอย่าง

```
ASSIGN (12) TO N
   :
GO TO N,(18, (12),23,25,3) --> จะทำให้คำสั่งที่ 12 ถูกทำต่อจากคำสั่งนี้

18 _____
   :
12 _____
   :
```

ค่าของตัวแปร N กำหนดโดยคำสั่ง ASSIGN จะไม่เอาไปใช้คำนวณ