บทที่ 7

GWBASIC COMMANDS, STATEMENTS, FUNCTIONS, AND VARIABLES

คำแนะนำ

รายการต่อไปนี้เป็นการแสดงรายการตามตัวอักษรของคำสั่ง (commands) คำสั่ง (statements) ฟังก์ชัน (functions) และตัวแปร (variables) ใน GWBASIC คำสั่ง (commands) เป็นวิธีต่าง ๆ ที่จะบอก GWBASIC ให้ปฏิบัติการ (action) ในทันที คำสั่งดังกล่าวถูกใช้มากในรูปแบบ direct mode ส่วน คำสั่ง (statements) บ่งบอกพารามิเตอร์ (parameters) ต่าง ๆ ที่ท่านต้องการที่จะกำหนด Executable statements บอก GWBASIC ถึงสิ่งที่จะกระทำถัดไป และ Nonexecutable statements เป็นเหตุให้ไม่มีการกระทำ ตัวอย่างของ nonexecutable statement คือ REM เป็นคำสั่งที่อนุญาตให้ท่านสอดแทรกหมายเหตุ (remark) ในโปรแกรมที่ท่านกำลังเขียน ฟังก์ชันต่าง ๆ ส่งค่าผลลัพธ์ในรูปจำนวนเลข (numeric) หรือในรูปสตริงก์ (string) ตัวแปรต่าง ๆ แสดงมูลค่าต่าง ๆ ที่ถูกใช้ในโปรแกรมของ GWBASIC

ไลน์รูปแบบ (format line) แสดงให้ท่านทราบถึงวิธีที่ป้อนคำสั่งในรูปของ command และ statement ป้อนฟังก์ชัน หรือป้อนตัวแปรเข้าไป ต่อไปนี้เป็นกฎต่าง ๆ ที่ต้องจดจำไว้ :

- คำต่าง ๆ ในรูปตัวอักษรพิมพ์ใหญ่ต้องถูกป้อนเข้าไปตามที่ถูกแสดง GWBASIC จะแปลงคำต่าง ๆ นั้นให้อยู่ในรูป upper case แม้ว่ามันทั้งหลายจะเป็นส่วนหนึ่งของสตริงก์ที่อยู่ในเครื่องหมายคำพูด (quoted string) หมายเหตุ (remark) หรือ DATA STATEMENT ก็ตาม
- ท่านต้องกำหนดเองในรูปของ lower-case bold letters
- เครื่องหมายวงเล็บก้ามปู ([ ]) ที่ห้อมล้อมข่าวสารต่าง ๆ นั้นเป็นสิ่งที่เลือกได้ (optional)

- ... หมายถึง item ที่อยู่ข้างหน้าอาจจะถูกกระทำซ้ำกี่ครั้งก็ได้ตามความจำเป็น
- เครื่องหมายวรรคตอนทั้งหมดยกเว้นเครื่องหมายที่อยู่ใน [ ] ต้องถูกประกอบรวมอยู่ด้วย และต้องอยู่ในตำแหน่งที่ถูกบ่งชี้โดยรูปแบบ (format)

ความถูกต้องแม่นยำ (accuracy) เป็นสิ่งที่จำเป็นยิ่ง ตัวอย่างเช่น

INPUT[;]["prompt";] variable[,variable]...

ในคำสั่งนี้ คำคีย์ INPUT อาจจะถูกติดตามด้วยเครื่องหมายเซมิโคลอน (;) จากนั้น prompt อาจจะถูกประกอบรวมอยู่ด้วยภายในเครื่องหมายคำพูดตามด้วยเครื่องหมายเซมิโคลอน ตามด้วยตัวแปรอย่างน้อยที่สุด 1 ตัวที่ถูกต้องการ ตัวแปรตัวอื่น ๆ อาจจะถูกเพิ่มเติมเข้าไปถ้ามันถูกคั่น (separated) ด้วยเครื่องหมายจุลภาค (,)

จุดประสงค์ของคำสั่งในรูป command และ statement ฟังก์ชัน หรือตัวแปรถูกเสนอแนะถัดไป หลังจากนั้นจะมีรายละเอียดเพิ่มเติมที่บ่งบอกเกี่ยวกับแต่ละจุดประสงค์ของสิ่งต่าง ๆ เหล่านั้น เช่นเดียวกัน ตัวอย่างต่าง ๆ จะถูกกำหนดให้และถูกอธิบาย

ABS Function

รูปแบบ v=ABS(x)

จุดประสงค์ ส่งค่าสัมบูรณ์ (absolute value) ของ x

รายละเอียด x อาจเป็นนิพจน์จำนวนเลข (numeric expression) ใด ๆ

ตัวอย่าง
```
PRINT ABS(8*(-3))
 24
Ok
```
เนื่องจากค่าสัมบูรณ์ของจำนวนเลขโดยปกติมีค่าเป็นบวกหรือเป็นศูนย์ ในตัวอย่างนี้มันมีค่าเป็นจำนวนบวก 24

ASC Function

รูปแบบ v=ASC(A$)

จุดประสงค์ ส่งค่า ASCII code สำหรับอักขระตัวแรกของสตริงก์

รายละเอียด
- A$ อาจเป็นนิพจน์สตริงก์ (string expression) ใด ๆ
- ดูภาคผนวก ค สำหรับ ASCII codes
- ถ้า A$ ว่างเปล่า (null) แล้ว "Illegal function call" error จะถูกส่งไป
- CHR$ function เป็น inverse ของ ASC function ซึ่งแปลงค่า ASCII code ไปเป็นอักขระ

ตัวอย่าง
```
10 A$="TEST"
20 PRINT ASC(A$)
RUN
 84
Ok
```

ตัวอย่างนี้แสดง ASCII code สำหรับ "T" คือ 84

PRINT ASC("TEST") จะแสดงผลลัพธ์เดียวกัน

ATN Function

รูปแบบ ATN(x)

จุดประสงค์ ส่งค่า arctangent ของ x ในรูปเรเดียน

รายละเอียด - นิพจน์ x อาจเป็น numeric type ใด ๆ แต่การประมวลผลของ ATN โดยปกติจะถูกกระทำในรูปของ single precision

- ผลลัพธ์ที่ได้จะเป็นมูลค่าอยู่ในช่วง -pi/2 ถึง pi/2 เมื่อ pi=3.141593

- ถ้าท่านต้องการที่จะแปลงเรเดียนไปเป็นองศา ให้คูณด้วย 180/pi

ตัวอย่าง

```
10 INPUT X
20 PRINT ATN(X)
RUN
? 3
 1.249046
Ok
```

ตัวอย่างนี้แสดงวิธีที่ arctangent ของ 3 ถูกคำนวณค่า

AUTO Command

รูปแบบ AUTO [line number[, increment]]

จุดประสงค์ สร้างไลน์นัมเบอร์ขึ้นโดยอัตโนมัติหลังจากทุก ๆ carriage return

รายละเอียด - line number เป็นหมายเลขของไลน์ที่จะเริ่มต้น

- increment เป็นมูลค่าที่ถูกบวกเพิ่มไปยังแต่ละไลน์นัมเบอร์ที่จะให้กับไลน์นัมเบอร์ถัดไป

- โดยทั่วไป AUTO ถูกใช้สำหรับการป้อนโปรแกรมต่าง ๆ เข้าไป ทำให้ประหยัดเวลาในการที่จะพิมพ์ไลน์นัมเบอร์แต่ละไลน์นัมเบอร์เข้าไป
- AUTO เริ่มต้นที่ line number และเพิ่มแต่ละไลน์นัมเบอร์ที่อยู่ถัด ๆ ไปด้วย increment มูลค่าที่ถูกกำหนดไว้ล่วงหน้า (default) สำหรับมูลค่าทั้งสองนี้คือ 10 ถ้า line number ถูกติดตามด้วยเครื่องหมายจุลภาคแต่ increment ไม่ได้ถูกระบุ increment ล่าสุดที่ถูกระบุในคำสั่ง AUTO จะถูกกำหนดแทน ถ้า line number ถูกละ (omitted) แต่ increment ถูกประกอบรวมอยู่ด้วย แล้วไลน์นัมเบอร์แรกคือ 0
- ถ้า AUTO สร้างไลน์นัมเบอร์ซึ่งเคยถูกใช้ไปแล้ว เครื่องหมายดอกจัน (*) จะถูกพิมพ์ข้างหลังหมายเลขเพื่อบ่งเตือนว่า ข้อมูลเข้า (input) ใด ๆ จะแทนที่ไลน์ที่มีอยู่แล้วนั้น อย่างไรก็ตาม การกด carriage return ทันทีหลังจาก * จะยังคงรักษาไลน์ที่มีอยู่แล้วนั้นไว้และ AUTO จะสร้างไลน์นัมเบอร์ถัดไป
- AUTO สิ้นสุดลง (end) เมื่อท่านกด Ctrl-Break ไลน์ที่ซึ่ง Ctrl-Break ถูกกดจะไม่ถูกเก็บบันทึกไว้ และท่านจะถูกส่งกลับไปยังไลน์คำสั่ง (command line)
- <u>หมายเหตุ</u> : เมื่ออยู่ใน AUTO mode ท่านอาจจะทำการเปลี่ยนแปลงต่าง ๆ ไปยังไลน์ที่เป็นอยู่ในขณะนั้น (current line) เท่านั้น ถ้าท่านต้องการที่จะเปลี่ยนแปลงไลน์อื่น ๆ ท่านต้องออกจาก AUTO

ตัวอย่าง AUTO 100,50

สร้างไลน์นัมเบอร์ 100, 150, 200 ...

AUTO.

สร้างไลน์นัมเบอร์ 10, 20, 30, 40 ...

BEEP Statement

รูปแบบ BEEP

จุดประสงค์ ส่งสัญญาณเสียงพูด

รายละเอียด - BEEP ส่งสัญญาณเสียงที่ 800 เฮิร์ทใน 4 วินาที

- BEEP และ PRINT CHR$(7) มีผลเหมือนกัน

ตัวอย่าง 2430 IF X < 20 THEN BEEP

ถ้า X มีค่าน้อยกว่า 20 speaker จะส่งสัญญาณเสียง

BLOAD Command

รูปแบบ BLOAD filespec [,offset]

จุดประสงค์ โหลด binary data ลงสู่หน่วยความจำ

รายละเอียด - filespec เป็นนิพจน์สตริงก์ที่ทำให้สอดคล้องกับกฎต่าง ๆ สำหรับ file specification ที่ถูกกำหนดให้ในบทที่ 3

- offset เป็นนิพจน์จำนวนเลขมีค่าอยู่ในช่วง 0 ถึง 65535 การโหลด (loading) เริ่มต้นที่แอดเดรส (address) นี้ และถูกระบุในฐานะเป็นกาวะเริ่มแรก (offset) ไปสู่ segment ที่ถูกประกาศโดย DEF SEG statement ล่าสุด

- ถ้า offset ถูกละ แอดเดรสของ offset และของ segment จะถูกกำหนดเป็นแอดเดรสของ offset และของ segment ที่ถูกระบุที่ BSAVE ในกรณีนี้ไฟล์จะถูกโหลดไปในตำแหน่ง (location) เดียวกันกับที่มันถูกเก็บ

- ถ้าอุปกรณ์ (device) ถูกละแล้ว DOS default diskette drive จะถูกกำหนดขึ้น

- คำเตือน : BLOAD ไม่ได้ทำการตรวจเช็คช่วงของแอดเดรส ดังนั้น จึงอาจเป็นไปได้ในการที่จะโหลดไฟล์ ณ ที่ใด ๆ ในหน่วยความจำ ระวังอย่าโหลดไปบนสแต็ค (stack) ของ GWBASIC บน GWBASIC's variable area หรือบนโปรแกรม GWBASIC ของท่าน
- บ่อยครั้ง BLOAD และ BSAVE ถูกใช้สำหรับการโหลดและการเก็บบันทึกโปรแกรมภาษาเครื่อง (machine language programs) ต่าง ๆ (ดู CALL Statement สำหรับวิธีที่จะกระทำโปรแกรมภาษาเครื่องต่าง ๆ จากภายในโปรแกรม GWBASIC) อย่างไรก็ตาม segment ใด ๆ อาจจะถูกระบุในฐานะเป็นแหล่งกำเนิด (source) หรือเป้าหมาย (target) สำหรับคำสั่งต่าง ๆ เหล่านี้ด้วย DEF SEG statement สิ่งนี้ยอมให้ท่านเก็บบันทึกหรือพิมพ์ screen images โดยการเก็บบันทึกจากหรือการโหลดไปยัง screen buffer

ตัวอย่าง

```
10 'Load subroutine at 6000:F000
20 DEF SEG=&H6000 'Set segment to 6000 Hex
30 BLOAD"PROG1",&HF000 'Load PROG1
```

segment address ถูกกำหนดที่ 6000 Hex และโหลด PROG1 ที่ F000

BSAVE Command

รูปแบบ BSAVE *filespec,offset,length*

จุดประสงค์ เก็บบันทึก binary data

รายละเอียด - filespec เป็นนิพจน์สตริงก์สำหรับ file specification ซึ่งสอดคล้องกับกฎต่าง ๆ ที่ถูกกำหนดให้ในบทที่ 3

- offset เป็นจำนวนเต็มมีค่าอยู่ในช่วง 0 ถึง 65535 สิ่งนี้เป็นภาวะเริ่มแรก (offset) ไปสู่ segment ที่ถูกประกาศโดย DEF SEG ล่าสุดเพื่อที่จะทำการการเก็บบันทึก
- ถ้า offset หรือ length ถูกละ จะเกิด Syntax error ขึ้นและการเก็บบันทึกจะถูกยกเลิกไป (cancelled)
- ถ้าชื่อของอุปกรณ์ถูกละแล้ว DOD default diskette drive จะถูกกำหนด
- DEF SEG statement ควรจะถูกประมวลผลก่อน BSAVE statement แอดเดรสที่ถูกกำหนดให้ใน DEF SEG statement จะถูกใช้สำหรับการบันทึก
- บ่อยครั้งที่ BLOAD และ BSAVE ถูกใช้สำหรับการโหลดและการเก็บบันทึกโปรแกรมภาษาเครื่องต่าง ๆ (ดู CALL Statement สำหรับวิธีที่จะประมวลผลโปรแกรมภาษาเครื่องต่าง ๆ จากภายในโปรแกรม GWBASIC) อย่างไรก็ตาม segment ใด ๆ อาจจะถูกระบุในฐานะเป็นแหล่งกำเนิดหรือเป้าหมายสำหรับคำสั่งต่าง ๆ เหล่านี้ด้วย DEF SEG statement สิ่งนี้ยอมให้ท่านเก็บบันทึกและพิมพ์ screen images โดยการเก็บบันทึกจากหรือการโหลดไปสู่ screen buffer

ตัวอย่าง

```
10 'Save the graphic screen buffer
20 SCREEN 105: 'set screen attributes for text and
                graphics
30 SCREEN ,,3,3 'set active and visual pages to
                p a g e  3

                program generates a screen image
```

```
100 DEF SEG = &H1800 'set segment pointer to screen
                     b u f f e r
110 BSAVE "PICTURE",0,27000 'save screen buffer to
                            file  "Picture"
```

DEF SEG ถูกใช้เพื่อที่จะสร้าง (set up) segment address ของ screen buffer ในฐานะเป็น hex 1800 ซึ่งสอดคล้องกันกับคำสั่ง SCREEN ,,3,3 ภาวะเริ่มแรกของ 0 และความยาว (length) 27,000 ระบุอย่างแจ้งชัดว่า entire screen จะถูกเก็บบันทึก

CALL Statement

รูปแบบ CALL numvar [(variable [,variable]...)]

จุดประสงค์ เรียกโปรแกรมภาษาเครื่อง

รายละเอียด
- numvar เป็นชื่อของตัวแปรจำนวนเลข (numeric variable) ที่ซึ่งมูลค่าบ่งบอกจุดที่จะเริ่มต้นในหน่วยความจำของสับรูทีน (subroutine) ที่จะถูกเรียก ภาวะเริ่มแรกของ segment ต้องถูกบ่งบอกใน DEF SEG statement ก่อนหน้า (ดู DEF SEG Statement)
- variable เป็นชื่อของตัวแปรหรือตัวแปรต่าง ๆ ถูกแบ่งแยกด้วยเครื่องหมายจุลภาคซึ่งจะถูกส่งผ่านไปยังสับรูทีน เช่นเดียวกัน ค่าคงที่ต่าง ๆ อาจจะถูกใช้ด้วย
- วิธีอื่นที่จะเรียกสับรูทีนภาษาเครื่องคือด้วยการใช้ USR function

ตัวอย่าง

```
100 DEF SEG=&H8000
110 FOO=0
120 CALL FOO(A,B$,C)
```

ไลน์ 100 กำหนด segment address เป็น 8000 hex F00 ถูกกำหนดค่าเป็นศูนย์ซึ่งเป็นสาเหตุให้การเรียกไปยัง F00 เพื่อที่จะประมวลผลสับรูทีน ณ ตำแหน่ง (location) hex 80000 ตัวแปร A B$ และ C ถูกส่งผ่านไปในฐานะเป็นอาร์กิวเม้นต์ (arguments) ต่าง ๆ ไปยังสับรูทีนภาษาเครื่อง

CALLS Statement

รูปแบบ CALLS numvar [{variable [,variable]...)]

จุดประสงค์ เรียกสับรูทีนภาษาเครื่อง

รายละเอียด
- numvar เป็นชื่อของตัวแปรจำนวนเลขที่ซึ่งมูลค่าบ่งบอกจุดที่จะเริ่มต้นในหน่วยความจำของสับรูทีนที่จะถูกเรียก ภาวะเริ่มแรกของ segment ต้องถูกบ่งบอกใน DEF SEG statement ก่อนหน้า (ดู DEF SEG Statement)
- variable เป็นชื่อของตัวแปรหรือตัวแปรต่าง ๆ ที่ถูกแบ่งแยกโดยเครื่องหมายจุลภาคซึ่งจะถูกส่งผ่านไปยังสับรูทีน เช่นเดียวกัน ค่าคงที่ต่าง ๆ อาจจะถูกใช้ด้วย
- คำสั่งนี้เป็นเหมือนเช่นคำสั่ง CALL ยกเว้นว่า segmented address ของทุก ๆ อาร์กิวเม้นต์ถูกส่งผ่านไป ในขณะที่ CALL ส่งผ่าน unsegmented addresses CALLS ควรจะถูกใช้เมื่อทำการเข้าถึง (accessing) รูทีนต่าง ๆ ของ MS-FORTRAN

ตัวอย่าง อ้างถึงตัวอย่างที่ถูกกำหนดให้ใน CALL Statement

CDBL Function

รูปแบบ CDBL(x)

จุดประสงค์ แปลงค่า x ไปสู่จำนวนเลขต่าง ๆ ในรูป double-precision numbers

รายละเอียด
- x อาจเป็นนิพจน์จำนวนเลขใด ๆ

- ดู "How GWBASIC Converts Numbers from One Precision to Another" ในบทที่ 5 สำหรับกฎต่าง ๆ เกี่ยวกับวิธีที่จะแปลงค่าจาก precision หนึ่งไปยังอีก precision หนึ่ง เช่นเดียวกัน อ้างอิงไปยัง CINT และ CSNG functions สำหรับการแปลงค่าจำนวนเลขต่าง ๆ ไปเป็น integer และ sipgle-precision

ตัวอย่าง

```
10 A=454.67
20 PRINT A;CDBL(A)
RUN
 454.67  454.6700134277344
Ok
```

มูลค่าของ CDBL(A) ถูกต้องแม่นยำถึงตำแหน่งทศนิยมหลักที่สองหลังจากการปัดเศษ (rounding) เนื่องจากมูลค่าแรกเริ่มของ A มีเลขหลังจุดทศนิยม 2 ตำแหน่งเท่านั้น

CHAIN Statement

รูปแบบ

```
CHAIN [MERGE]filespec[,[line] [,ALL][,DELETE range]]
```

จุดประสงค์ เรียกโปรแกรมและส่งผ่านตัวแปรต่าง ๆ ไปยังมัน

รายละเอียด

- filespec เป็นไปตามกฎต่าง ๆ สำหรับ file specification ที่ถูกกำหนดให้ในบทที่ 3 มันเป็นชื่อของโปรแกรมซึ่งถูกเรียก ตัวอย่างเช่น :

```
CHAIN "A:PROG1"
```

- line เป็นไลน์นัมเบอร์หรือไม่ก็เป็นนิพจน์ซึ่งประเมินค่าไปเป็นไลน์นัมเบอร์ในโปรแกรมที่ถูกเรียก ซึ่งจะระบุไลน์ที่ซึ่งโปรแกรมที่ถูกเรียกจะเริ่มต้นการวิ่ง (running) ถ้าไม่ระบุแล้วการประมวลผลจะเริ่มต้นที่ไลน์แรกในโปรแกรมที่ถูกเรียก

- CHAIN "A:PROG1",1000

ณ ที่นี่ ไลน์ 1,000 ไม่ได้ถูกผลกระทบโดยคำสั่ง RENUM ถ้าโปรแกรมที่ถูกเรียกถูกเปลี่ยนลำดับเลขใหม่ (renumbered) คำสั่ง CHAIN ต้องถูกเปลี่ยนเปลี่ยนแปลงเพื่อที่จะชี้ไปยังไลน์นัมเบอร์ไลน์ใหม่

- ALL ระบุว่าทุก ๆ ตัวแปรในโปรแกรมที่เป็นอยู่ในขณะนั้นจะถูกส่งผ่านไปยังโปรแกรมที่ถูกเรียก ถ้า ALL option ถูกละแล้วท่านต้องรวมเอาคำสั่ง COMMON เข้าไว้ด้วยในโปรแกรมที่เป็นอยู่ในขณะนั้นเพื่อที่จะแสดงรายการตัวแปรต่าง ๆ ที่ถูกส่งผ่านไปยังโปรแกรมที่ถูกเรียก ดู "COMMON Statement" ในบทนี้

ตัวอย่างเช่น :

```
CHAIN "A:PROG1",1000,ALL
```

- MERGE นำ section ของรหัส (code) ไปสู่โปรแกรม GWBASIC ในฐานะเป็น overlay นั่นคือ MERGE operation ถูกกระทำกับโปรแกรมที่เป็นอยู่ในขณะนั้นและโปรแกรมที่ถูกเรียก โปรแกรมที่ถูกเรียกต้องอยู่ในรูป ASCII code ถ้ามันถูกรวบรวมเข้าด้วยกัน (merge)

- CHAIN MERGE "A:OVERLAY",1000

หลังจากการใช้ overlay, โดยทั่วไปท่านอาจต้องการที่จะลบ (delete) มันออกไปเพื่อว่า overlay ใหม่อาจจะถูกนำเข้าไป เพื่อที่จะกระทำสิ่งนี้ใช้ DELETE option ซึ่งกระทำเหมือนคำสั่ง DELETE เช่นเดียวกับในคำสั่ง DELETE ไลน์นัมเบอร์ต่าง ๆ ที่ถูกระบุในฐานะเป็นไลน์แรกและไลน์สุดท้ายของช่วงต้องมีอยู่ หรือ "Illegal function call" error จะเกิดขึ้น

- CHAIN MERGE "A:OVERLAY2",1000,DELETE 1000-5000

ตัวอย่างนี้ลบไลน์ 1,000 ถึงไลน์ 5,000 ของโปรแกรมที่เป็นอยู่ในขณะนั้นออกไปก่อนที่จะทำการโหลดใน overlay (โปรแกรมที่ถูกเรียก) ไลน์นัมเบอร์ต่าง ๆ ในช่วงถูกผลกระทบโดยคำสั่ง RENUM

- หมายเหตุ :

1. คำสั่ง CHAIN ปล่อยให้ไฟล์ต่าง ๆ ถูกเปิดอยู่
2. คำสั่ง CHAIN ร่วมกับ MERGE option รักษา OPTION BASE setting ในขณะนั้นให้คงไว้
3. ถ้า MERGE option ถูกละ OPTION BASE setting จะไม่ถูกรักษาให้คงไว้ในโปรแกรมที่ถูกเรียก เช่นเดียวกัน โดยปราศจาก MERGE, CHAIN จะไม่รักษาชนิดต่าง ๆ ของตัวแปร (variable types) หรือฟังก์ชันที่ถูกบ่งบอกโดยผู้ใช้ (user-defined functions) ต่าง ๆ ให้คงไว้เพื่อใช้โดยโปรแกรมที่ถูกเรียก ดังนั้น คำสั่ง DEFINT DEFSNG DEFDBL DEFSTR หรือคำสั่ง DEF FN ที่ประกอบด้วยตัวแปรต่าง ๆ ที่ถูกใช้ร่วมกัน (shared variables) ต้องถูกกำหนดใหม่ (restated) ในโปรแกรมที่ถูกเรียก
4. เมื่อทำการใช้ MERGE option ฟังก์ชันต่าง ๆ ที่ถูกบ่งบอกโดยผู้ใช้ควรจะถูกกำหนด (placeed) ก่อน CHAIN MERGE statements ใด ๆ ในโปรแกรม มิฉะนั้นแล้ว ฟังก์ชันต่าง ๆ เหล่านี้จะ undefined หลังจากที่การ merge เสร็จสิ้นสมบูรณ์

ตัวอย่าง 1

```
10 REM THIS PROGRAM DEMONSTATES CHAINING USING COMMON
   TO PASS VARIABLES.
20 REM SAVE THIS MODULE ON DISK AS "PROG1" USING THE
   A OPTION.
30 DIM A$(2),B$(2)
40 COMMON A$(),B$()
50 A$(1)="VARIABLES IN COMMON MUST BE ASSIGNED"
60 A$(2)="VALUES BEFORE CHAINING."
```

```
70 B$(1)="": B$(2)=""
80 CHAIN "PROG2"
90 PRINT: PRINT B$(1): PRINT: PRINT B$(2): RPINT
100 END
```

```
10 REM THE STATEMENT "DIM A$92),B$(2)" MUST ONLY BE
   EXECUTED ONCE.
20 REM HENCE, IT DOES NOT APPEAR IN THIS MODULE.
30 REM SAVE THIS MODULE ON THE DISK AS "PROG2" USING
   THE 'A' OPTION.
40 COMMON A$(),B$()
50 PRINT: PRINT A$(1);A$(2)
60 B$(1) "NOTE HOW THE OPTION OF SPECIFYING A STARTING
   LINE NUMBER"
70 B$(2)="WHEN CHAINING AVOIDS THE DIMENSION STATEMENT
   IN 'PROG1'."
80 CHAIN "PROG1", 90
90 END
```

ในตัวอย่างนี้ "PROG1" แสดงวิธีที่จะเชื่อมโยง (chain) โปรแกรมอื่นที่ถูกตั้งชื่อว่า "PROG2" "PROG2" แสดงวิธีที่จะเชื่มโยงกลับไปยังไลน์ 90 ใน "PROG1"

ตัวอย่าง 2

```
10 REM THIS PROGRAM DEMONSTATES CHAINING USING THE MERGE
   AND ALL OPTIONS.
```

```
  20 REM SAVE THIS MODULE ON THE DISK AS "MAINPRG".
  30 A$="MAINPRG"
  40 CHAIN MERGE "OVERLAY1",1010,ALL
  50 END
1000 REM SAVE THIS MODULE ON THE DISK AS "OVERLAY1" USING
     THE A OPTION.
1010 PRINT A$; " HAS CHAINED TO OVERLAY1."
1020 A$="OVERLAY1"
1030 B$="OVERLAY2"
1040 CHAIN MERGE "OVERLAY2",1010,ALL,DELETE 1000-1050
1050 END
1000 REM SAVE THIS MODULE ON THE DISK AS "OVERLAY2" USING
     THE A OPTION.
1010 PRINT A$; " HAS CHANGED TO ";B$;"."
1020 END
```

ตัวอย่างนี้แสดงวิธีที่จะใช้ MERGE option ในขณะทำการเชื่อมโยง

CHR$ Function

รูปแบบ CHR$(a)

จุดประสงค์ แปลงค่า ASCII code ไปสู่รูปของอักขระที่ปรากฏ

รายละเอียด
- a ต้องมีค่าอยู่ในช่วง 0 ถึง 255
- ASCII code ถูกแสดงรายการในภาคผนวก ค
- โดยทั่วไป CHR$ ถูกใช้เพื่อที่จะส่งอักขระพิเศษ (special character) ไปยังจอภาพ (screen) หรือไปยังเครื่องพิมพ์ (printer) ตัวอย่างเช่น

อักขระ BEL ซึ่งส่งสัญญาณเสียงพูดอาจจะถูกรวบรวมในฐานะเป็น CHR$(7) ซึ่งเป็นการกล่าวนำไปสู่ error message (แทนที่จะใช้ BEEP) CHR$(12) อาจจะเป็น form feed เพื่อที่จะเคลียร์จอเทอร์มินัลและส่งเคอร์เซอร์ไปยัง home position

- อ้างอิงไปยัง ASC Function เพื่อที่จะดูวิธีที่จะแปลงค่าอักขระกลับไปเป็น ASCII code ของมัน

ตัวอย่าง

```
PRINT CHR$(66)
B
Ok
```

อักขระสำหรับ ASCII CODE 66 คือ B

```
10 IF INKEY$ <> CHR$(13) TIIEN 10
```

โปรแกรมจะวนลูปที่คำสั่ง 10 จนกว่าอักขระ ^C (Control C) จะถูกป้อนเข้าไปจากแป้นคีย์

CINT Function

รูปแบบ CINT(x)

จุดประสงค์ แปลงค่า x ไปสู่จำนวนเต็ม (integer)

รายละเอียด
- x อาจเป็นนิพจน์จำนวนเลขใด ๆ ในช่วง -32768 ถึง 32767
- x ถูกแปลงค่าไปสู่จำนวนเต็มโดยการปัดเศษ (rounding) ส่วนที่เป็นเศษส่วน
- ดู FIX และ INT ซึ่งส่งค่าจำนวนเต็มด้วย เช่นเดียวกัน ดู CDBL และ CSNG สำหรับการแปลงค่าจำนวนเลขต่าง ๆ ไปสู่ single precision หรือ ไปสู่ double precision

ตัวอย่าง PRINT CINT(45.67)

46

Ok

สังเกตวิธีการปัดเศษที่เกิดขึ้น

CIRCLE Statement

รูปแบบ CIRCLE (a,b),r [,color [,start,end [,aspect]]]

จุดประสงค์ วาดรูปอิลลิปซ์ (ellipse) ด้วยจุดศูนย์กลางและรัศมีที่ถูกระบุ

รายละเอียด
- a เป็น x-coordinate สำหรับจุดศูนย์กลางของวงกลม
- b เป็น y-coordinate สำหรับจุดศูนย์กลางของวงกลม
- r เป็นรัศมีของวงกลมในรูป pixels
- color เป็นสัญลักษณ์จำนวนเลขสำหรับสีที่ต้องการ (ดู COLOR Statement) สีที่ถูกกำหนดไว้ล่วงหน้า (default color) คือสีพื้นหน้า (foreground color)
- start และ end เป็นมุมเริ่มต้นและมุมสิ้นสุดมีหน่วยเป็นเรเดียน ช่วงของมูลค่าคือ -2*pi ถึง 2*pi มุมต่าง ๆ เหล่านี้ยอมให้ท่านระบุที่ซึ่งอิลลิปซ์จะเริ่มต้นและสิ้นสุด ถ้ามุมเริ่มต้นและมุมสิ้นสุดมีค่าเป็นลบ (negative) แล้วอิลลิปซ์จะถูกเชื่อมต่อเข้าด้วยกันด้วยไลน์ (line) และมุมต่าง ๆ จะถูกปฏิบัติราวกับว่ามันมีค่าเป็นบวก (positive) สังเกตว่าสิ่งนี้แตกต่างจากการเพิ่ม (adding) 2*pi มุมเริ่มต้นอาจมีค่าน้อยกว่ามุมสิ้นสุด
- aspect เป็น aspect ratio นั่นคือเป็นอัตราส่วนของรัศมี x ต่อรัศมี y Default ratio จะให้ round circle บนจอภาพ
- ถ้า aspect ratio มีค่าน้อยกว่า 1 แล้วรัศมี x จะถูกกำหนดให้ ถ้ามันมีค่ามากกว่า 1 แล้วรัศมี y จะถูกกำหนดให้

- จุดสุดท้ายที่ถูกอ้างอิงหลังจากที่วงกลมถูกวาดคือจุดศูนย์กลางของวงกลม
- โคออร์ดิเนตต่าง ๆ สามารถถูกแสดงในฐานะเป็น absolutes ดังในรูปแบบที่ถูกแสดงข้างบน หรือ STEP option สามารถถูกใช้เพื่อที่จะอ้างอิงจุดที่สัมพันธ์กับจุดที่ซึ่งเพิ่งถูกใช้ไป (most recent point used) รูปแบบของ STEP option คือ :

STEP (a,b)

ตัวอย่างเช่น ถ้า most recent point ที่ถูกอ้างอิงคือจุด (x,y) แล้ว STEP (10,5) จะอ้างอิงจด (x+10,y+5)

ตัวอย่าง สมมติว่าจุดสุดสุดท้ายที่ถูกพล็อตคือ 100,50 แล้ว

```
CIRCLE (200,200),50
```

และ

```
CIRCLE STEP (100,150),50
```

ทั้งคู่จะวาดวงกลมที่ 200,200 ด้วยรัศมี 50 ตัวอย่างแรกใช้ absolute notation ตัวอย่างที่สองใช้ relative notation

```
10 'These examples demonstate use of the CIRCLE statement
20 SCREEN 105: CLS: PI=3.1415926
30 CIRCLE (50,100),40,0,2*PI
40 CIRCLE (150,100),40,,,,1.5
50 CIRCLE (250,100),40,,.25*PI,1.5*PI
60 CIRCLE (350,100),40,,-.25*PI,1.5*PI
70 CIRCLE (450,100),40,,.25*PI,-1.5*PI
80 CIRCLE (550,100),40,,1.25*PI,-.75*PI
```

การตรวจสอบตัวอย่างต่าง ๆเหล่านี้แต่ละตัวอย่างจะช่วยให้ท่านเข้าใจการกระทำของคำสั่ง CIRCLE

หมายเหตุ : โดยปกติคำสั่ง CIRCLE วาดในทิศทางทวนเข็มนาฬิกาจากจุดเริ่มต้นไปยังจุดสุดท้าย

CLEAR Command

รูปแบบ CLEAR [,[n][,m]]

จุดประสงค์ กำหนดค่าตัวแปรจำนวนเลขทุก ๆ ตัวเป็นศูนย์ ตัวแปรสตริงก์ทุก ๆ ตัวเป็น null และปิดไฟล์ทุก ๆ ไฟล์ที่เปิดอยู่ ข้อเลือก (options) ต่าง ๆ กำหนดจุดสิ้นสุดของหน่วยความจำและจำนวนของ stack space

รายละเอียด
- n เป็น byte count ซึ่งถ้าระบุจะกำหนดจำนวนสูงสุดของไบท์สำหรับ GWBASIC work space (ที่ซึ่งโปรแกรมและข้อมูลของท่านถูกเก็บไว้รวมทั้ง interpreter work area) บางทีท่านอาจจะระบุนิพจน์นี้ถ้าท่านต้องการที่จะเก็บรักษา space ในเนื้อที่หน่วยความจำ (storage) สำหรับโปรแกรมภาษาเครื่องต่าง ๆ
- m กำหนด a side stack space สำหรับ GWBASIC Default คือ 512 ไบท์ หรือ 1 ใน 8 ของหน่วยความจำที่สามารถใช้ได้ ขนิดไหนก็ตามที่มีขนาดเล็กกว่า ท่านอาจต้องการที่จะรวบรวมนิพจน์นี้เข้าไปด้วยถ้าท่านใช้คำสั่ง GOSUB ที่ถูกซ้อนกันเป็นโครงข่าย (nested) หรือลูป FOR-NEXT เป็นจำนวนมากในโปรแกรมของท่าน หรือถ้าท่านใช้ PAINT เพื่อที่จะกระทำภาพระบายสีต่าง ๆ
- CLEAR ปลดปล่อยทุก ๆ หน่วยความจำที่ถูกใช้สำหรับข้อมูลให้เป็นอิสระโดยปราศจากการลบโปรแกรมที่เป็นอยู่ในขณะนั้นในหน่วยความจำออกไป หลังจากการใช้ CLEAR อะเรย์ต่าง ๆ จะ undefined ตัวแปรจำนวนเลขต่าง ๆ จะมีมูลค่าเป็นศูนย์ ตัวแปรสตริงก์ต่าง ๆ จะมีมูลค่าเป็น null และข่าวสารใด ๆ ที่กำหนดด้วย DEF statement ใด ๆ จะสูญเสียไป

- CLEAR กำหนด SOUND ใหม่ไปที่ Music Foreground และกำหนด PEN และ STRIG ไปที่ OFF
- เพื่อที่จะปลดปล่อยหน่วยความจำให้เป็นอิสระโดยปราศจากการลบทุก ๆ ข้อมูลของท่านใช้คำสั่ง ERASE

ตัวอย่าง CLEAR

เคลียร์ทุก ๆ ข้อมูลจากหน่วยความจำโดยปราศจากการลบโปรแกรมออกไป

CLEAR ,32768

เคลียร์ข้อมูลและกำหนดขนาดของ work space สูงสุดเป็น 32K

CLEAR ,,2000

เคลียร์ข้อมูล กำหนดขนาดของสแต็คเป็น 2,000 ไบท์

CLEAR ,32768,2000

เคลียร์ข้อมูล กำหนด work space สูงสุดสำหรับ GWBASIC เป็น 32K และกำหนดขนาดของสแต็คเป็น 2,000 ไบท์

CLOSE Statement

รูปแบบ CLOSE [[#]filenum[,[#]filenum...]]

จุดประสงค์ รวบรวม (conclude) I/O ไปยัง disk file หรือ device

รายละเอียด
- filenum เป็นหมายเลขที่ถูกใช้บนคำสั่ง OPEN
- CLOSE สิ้นสุดความสัมพันธ์ระหว่างไฟล์หรืออุปกรณ์ (device) ที่เฉพาะเจาะจงและหมายเลขไฟล์ (file number) ของมัน ไฟล์หรืออุปกรณ์อาจจะถูกเปิดอีกครั้งหนึ่งโดยการใช้หมายเลขเดียวกันหรือหมายเลขต่างกัน หรือหมาย

เลขอาจจะถูใช้ใหม่เพื่อที่จะเปิดอุปกรณ์หรือเปิดไฟล์ใด ๆ

- ถ้าไฟล์หรืออุปกรณ์ถูกเปิดสำหรับ sequential output คำสั่ง CLOSE จะเป็นเหตุให้บัฟเฟอร์ (buffer) สุดท้ายจะถูกเขียนไปยังมัน
- ถ้าไม่มีหมายเลขไฟล์ใด ๆ ถูกระบุ แล้ว CLOSE จะเป็นเหตุให้ทุก ๆ อุปกรณ์หรือทุก ๆ ไฟล์ที่เปิดอยู่ถูกปิดไป
- การประมวลผล END NEW RESET หรือ RUN โดยปราศจาก R option นั้นโดยอัตโนมัติจะเป็นเหตุให้ทุก ๆ ไฟล์และอุปกรณ์ต่าง ๆ ที่เปิดอยู่ถูกปิดไป คำสั่ง STOP ไม่ได้ปิดไฟล์หรืออุปกรณ์ใด ๆ

ตัวอย่าง

```
10 CLOSE 2,5
```

หรือ

```
10 CLOSE #2,#5
```

แต่ละคำสั่งจะปิดไฟล์หมายเลข 2 และไฟล์หมายเลข 5

CLS Statement

รูปแบบ

```
CLS
```

จุดประสงค์ ลบเนื้อหา (contents) ต่าง ๆ ของจอภาพที่เป็นอยู่ในขณะนั้นทั้งหมดและกำหนดเคอร์เซอร์ไปยังตำแหน่งที่เรียกว่า home position

รายละเอียด

- ในเท็กซ์โหมด (text mode) CLS เคลียร์ active page ไปยัง background color และกำหนดตำแหน่งเคอร์เซอร์ไปยังขอบบนด้านซ้ายมือสุดของจอภาพ
- ในกราฟฟิคโหมด (graphic mode) ไม่ว่าจะเป็น super เป็น medium หรือเป็น high resolution ก็ตาม CLS จะเคลียร์ screen buffer ไปยัง background color และกำหนดตำแหน่งเคอร์เซอร์ไปยังจุดศูนย์กลางของจอภาพ

- การกด Ctrl-Home จะมีผลเช่นเดียวกับการประมวลผลคำสั่ง CLS

ตัวอย่าง

```
10 CLS
20 PRINT "Hi there!!!"
```

ตัวอย่างนี้จะเคลียร์จอภาพและพิมพ์ข้อความ "Hi there!!!" ที่ขอบบนด้านซ้ายของจอภาพ

COLOR Statement (Graphics)

รูปแบบ

```
COLOR [background] [,palette]
```

จุดประสงค์ กำหนด background color และเลือก color palette สำหรับ foreground

รายละเอียด
- คำสั่งนี้อาจจะถูกใช้ร่วมกับ color/graphics monitor ส่วนมากที่ดัดแปลงให้เหมาะสมใน medium resolution ความพยายามต่าง ๆ ในการที่จะใช้ COLOR ใน high resolution mode จะส่งผลให้เกิด "Illegal Function Call" error ขึ้น คำสั่ง COLOR ไม่ถูกรับรู้ใน super resolution mode (SCREEN 104 หรือ SCREEN 105)
- background เป็นสีในช่วง 0 ถึง 15 ตามที่ถูกระบุในตารางที่ 7.1
- palette สามารถเป็นจำนวนเต็มที่ไม่มีเครื่องหมายหรือนิพจน์จำนวนเลขใด ๆ ถ้ามูลค่าของมันเป็นเลขคี่แล้ว palette 1 จะถูกเลือก มิฉะนั้น palette 2 จะถูกเลือก ตารางที่ 7.2 แสดงสีที่ส่งผลกับแต่ละ palette ถ้ามูลค่าอยู่นอกช่วง 0-254 จะเกิด "Illegal Function Call" error ขึ้น
- Foreground color อาจจะเป็นเช่นเดียวกันกับ background color ดังนั้นจึงเป็นการทำให้อักขระต่าง ๆ ที่ถูกพิมพ์ไม่สามารถเห็นได้
- พารามิเตอร์ใด ๆ อาจจะถูกละ ถ้ามันถูกละแล้วมูลค่าก่อนหน้าจะถูกรักษาไว้

ตารางที่ 7.1

COLOR NUMBERS

No.	Color	No.	Color
0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Brown	14	Yellow
7	White	15	High-Intensity White

ตารางที่ 7.2

PALETTE INFORMATION

Color	Palette 0	Palette 1
1	Green	Cyan
2	Red	Magenta
3	Brown	White

- สีต่างๆ ที่ถูกเลือกโดยคำสั่ง COLOR จะถูกใช้โดยคำสั่งต่าง ๆ ต่อไปนี้คือ คำสั่ง PSET PRESET LINE CIRCLE PAINT และคำสั่ง DRAW
- มูลค่าต่าง ๆ ที่ถูกป้อนเข้าไปภายนอกช่วงจะส่งผลให้เกิด "Illegal function call" error ขึ้น และมูลค่าต่าง ๆ ก่อนหน้าจะถูกรักษาให้คงไว้ (retained)

ตัวอย่าง

```
10 SCREEN 1
20 COLOR 9,0
```

กำหนด background ไปที่ light blue และเลือก palette 0

```
30 COLOR ,3
```

Background ยังคงอยู่ที่ light blue และเลือก palette 1 เนื่องจาก palette เป็นเลขคี่

```
10 CLS
20 SCREEN 1,0          'Select medium resolution - color on
30 COLOR 10,1          'Select light green background -
                        palette 1
40 PRINT "Characters are white (color 3)"
50 LINE (100,100)-(200,150),2,B   'Draw a box with magenta
                                   lines
60 PAINT (150,25),1,2   'Fill the box with cyan
```

nu standard monochrome system โปรแกรมนี้จะพิมพ์ในรูปแบบ reverse video และวาดกล่องและทำให้เต็มกล่อง (fill) ด้วยสีขาว

COLOR Statement (Text)

รูปแบบ COLOR [foreground][,[background][,border]]

จุดประสงค์ กำหนดค่า foreground (นั่นคืออักขระ) color background color และ border color ตามลำดับ

รายละเอียด - ผลของพารามิเตอร์ต่าง ๆ เหล่านี้แต่ละตัวแตกต่างกันไปขึ้นอยู่กับ monitor และ monitor adaptor ที่ถูกใช้ (ดูตารางต่าง ๆ ต่อไปนี้) มูลค่าต่าง ๆ ที่อยู่ภายนอกช่วงต่าง ๆ ที่ถูกแสดงจะเป็นเหตุให้เกิด "Illegal Function Call" error ขึ้น

- เพื่อความสะดวกสบาย, ตารางที่ 7.1 จะถูกแสดงซ้ำ ณ ที่นี้

ตารางที่ 7.1

COLOR NUMBERS

No.	Color	No.	Color
0	Black	a	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Margenta
6	Brown	14	Y e l l o w
7	White	15	High-Intensity White

ตารางที่ 7.3

color ON A STANDARD MONOCHROME MONITOR

Foreground (0-31)	Background (0-15)	
	Even	Odd
0	Black on black	Black, on white
1	White on black underlined	Black on white underlined
2-7	White on black	Black on white
8-15	Same as 0-7 except high intensity.	
16-23	Same as 0-7 except blinking.	
24--31	Same as 0-7 except blinking and high intensity.	

- หมายเหตุ : Border ไม่มีผลกระทบและอาจจะถูกละได้ ถ้ามันถูกกำหนดให้แล้วมันต้องอยู่ในช่วง 0-15 หรือไม่เช่นนั้นจะเกิด "Illegal Function Call" error ขึ้น
- ตัวอย่างต่าง ๆ ต่อไปนี้สาธิตการใช้ของคำสั่ง COLOR ใน text mode บน standard monochrome system

```
10 CLS
20 SCREEN 100          'SCREEN 0,0 could also be used
30 COLOR 7,0: PRINT "Normal - white on black"
40 COLOR 28,1: PRINT "Blinking - light red on blue"
50 COLOR 0,7: PRINT "Reverse Video - black on white"
60 COLOR 7,0           'Return to normal
```

COM(n) Statement

รูปแบบ COM(n) ON

COM(n) OFF

COM(n) STOP

จุดประสงค์ ทำให้ event trapping ของ communications activity บน channel ที่ถูกระบุมีความสามารถ (enable) หรือไร้ความสามารถ (disable)

รายละเอียด
- n เป็นหมายเลขของ communications channel มีค่าเป็น 1 หรือ 2
- คำสั่ง COM(n) ON ทำให้ communications event trapping มีความสามารถโดยคำสั่ง ON COM statement ในระหว่างที trapping ถูกทำให้มีความสามารถและถ้าไลน์นัมเบอร์ที่ไม่ใช่ศูนย์ถูกระบุใน ON COM statement แล้ว GWBASIC จะทำการตรวจเช็คระหว่างทุก ๆ คำสั่งเพื่อที่จะดูว่าการเคลื่อนไหว (activity) ได้ถูกทำให้เกิดขึ้นบน communications channel ถ้ามันเกิดขึ้นแล้วคำสั่ง ON COM statement จะถูกปฏิบัติ
- คำสั่ง COM(n) OFF ทำให้ communications event trapping ไร้ความสามารถ ถ้า event เกิดขึ้นมันจะไม่ถูกจดจำ

- คำสั่ง COM(n) STOP ทำให้ communications event trapping ไร้ความสามารถ แต่ถ้า event เกิดขึ้นมันจะถูกจดจำไว้และคำสั่ง ON COM statement จะถูกปฏิบัติในไม่ช้าที่ trapping ถูกทำให้มีความสามารถ

ตัวอย่าง 10 COM(1) ON

ทำให้ error trapping ของ communications activity บน channel 1 มีความสามารถ

COMMON Statement

รูปแบบ COMMON list of variables

จุดประสงค์ ส่งผ่านตัวแปรต่าง ๆ ไปยังโปรแกรมที่ถูกเชื่อมโยง (chained program)

รายละเอียด - คำสั่ง COMMON ถูกใช้ร่วมกับคำสั่ง CHAIN คำสั่ง COMMON อาจปรากฏ ณ ที่ใด ๆ ในโปรแกรมแม้ว่ามันจะถูกแนะนำว่ามันควรจะปรากฏที่จุดเริ่มต้นของโปรแกรมก็ตาม ตัวแปรตัวเดียวกันไม่สามารถปรากฏในคำสั่ง COMMON มากกว่า 1 คำสั่ง ตัวแปรอะเรย์ (array variables) ต่าง ๆ ถูกระบุโดยการเพิ่มเครื่องหมาย () ไปที่ชื่อของตัวแปร ถ้าตัวแปรทุก ๆ ตัวจะถูกส่งผ่านให้ใช้คำสั่ง CHAIN รวมด้วย ALL option และละคำสั่ง COMMON

- บางผลิตภัณฑ์ของไมโครซอฟท์อนุญาตให้จำนวนมิติต่าง ๆ ในอะเรย์ถูกประกอบรวม (include) อยู่ในคำสั่ง COMMON GWBASIC จะยอมรับไวยกรณ์ (syntax) นั้นแต่จะไม่รับรู้นิพจน์จำนวนเลขของตัวมันเอง ตัวอย่างเช่นคำสั่งต่าง ๆ ต่อไปนี้ทั้งคู่ใช้ได้ (valid) และถูกพิจารณาว่าเท่าเทียมกัน

```
COMMON A(1)
COMMON A(3)
```

ตัวเลขในเครื่องหมายวงเล็บเป็นหมายเลขของมิติไม่ใช่ตัวมิติเอง ตัวอย่างเช่น ในตัวอย่างนี้ตัวแปร A(3) อาจจะสมนัยกันกับคำสั่ง DIM ของ

```
DIM A(5,8,4)
```

ตัวอย่าง

```
100  COMMON  A,B,C,D(),G$
110  CHAIN "PROG3",10
.
.
```

ตัวอย่างนี้เชื่อมโยงโปรแกรม PROG3 ตัวแปร A B C และตัวแปร G$ และอะเรย์ D จะถูกส่งผ่านไป

CONT Command

รูปแบบ CONT

จุดประสงค์ ดำเนินการประมวลผลโปรแกรมต่อหลังจากที่หยุด (break) ไว้

รายละเอียด
- ใช้คำสั่งนี้เพื่อที่จะเริ่มต้นโปรแกรมอีกครั้งหนึ่งหลังจากที่ Ctrl-Break ถูกกด คำสั่ง STOP หรือคำสั่ง END ได้ถูกปฏิบัติหรือได้เกิด error ขึ้น การประมวลผลดำเนินต่อไป ณ จุดที่ซึ่งการหยุดได้ถูกเกิดขึ้น ถ้าการหยุดได้ถูกเกิดขึ้นหลังจากข้อความบ่งเตือน (prompt) จากคำสั่ง INPUT แล้ว การประมวลผลจะดำเนินต่อด้วยการพิมพ์ข้อความบ่งเตือนใหม่อีกครั้งหนึ่ง
- โดยปกติ CONT ถูกใช้ร่วมกับ STOP สำหรับการปรับปรุงแก้ไข (debugging) เมื่อการประมวลผลถูกทำให้หยุดลงท่านสามารถตรวจสอบหรือเปลี่ยนแปลงมูลค่าต่าง ๆ ของตัวแปรต่าง ๆ โดยการใช้คำสั่งต่าง ๆ ในรูปแบบตรง (direct mode statements) ใช้ CONT เพื่อที่จะทำการประมวลผลโปรแกรมต่อไป หรือใช้ GOTO ในรูปแบบตรงซึ่งต้องการไลน์นัมเบอร์ที่แน่นอนเพื่อที่จะดำเนินการประมวลผลต่อไป
- CONT ใช้ไม่ได้ (invalid) เมื่อโปรแกรมถูกปรับปรุงในระหว่างการหยุด

COS Function

รูปแบบ COS(x)

จุดประสงค์ ส่งค่าฟังก์ชันโคไซน์ (cosine function) ของมุม

รายละเอียด - x เป็นมุมที่ซึ่งโคไซน์จะถูกคำนวณค่า มูลค่าของ x ต้องอยู่ในรูปเรเดียน เพื่อที่จะแปลงค่าจากองศาไปเป็นเรเดียน ให้คูณองศาด้วย pi/180 เมื่อ pi=3.141593

- การคำนวณค่าของ COS(x) ถูกกระทำในรูป single precision

ตัวอย่าง
```
10 X=2*COS(.4)
20 PRINT X
RUN
 1.842122
Ok
```
ตัวอย่างนี้แสดงผลลัพธ์ของ 2 คูณด้วยโคไซน์ของ .4

CSNG Function

รูปแบบ CSNG(x)

จุดประสงค์ แปลงคา x ไปสู่จำนวนเลขในรูป single precision

รายละเอียด - x อาจเป็นนิพจน์จำนวนเลขใด ๆ

ตัวอย่าง
```
10 A# = 975.3421#
20 PRINT A#; CSNG(A#)
RUN
 975.3421  975.3421
Ok
```
มูลค่าของจำนวนเลขในรูป double precision ของ A# ถูกส่งค่าในฐานะ

เป็น CSNG(A#)

CSRLIN Function

รูปแบบ x = CSRLIN

จุดประสงค์ ส่งค่าตำแหน่งของไลน์ปัจจุบันที่เคอร์เซอร์อยู่

รายละเอียด - มูลค่าที่ถูกส่งจะอยู่ในช่วง 1 ถึง 24

- สำหรับตำแหน่งของสคมภ์ที่เคอร์เซอร์อยู่ ให้ดูฟังก์ชัน POS

- LOCATE กำหนดไลน์ของเคอร์เซอร์

ตัวอย่าง 10 Y = CSRLIN

20 ▫ POS(0)

30 LOCATE 20,1 :FRINT "HELLO"

40 LOACTE X,Y

ไลน์ 10 บันทึกไลน์ปัจจุบัน ไลน์ 20 บันทึกสคมภ์ปัจจุบัน ไลน์ 30 เป็นเหตุให้ HELLO ถูกพิมพ์บนไลน์ 20 ไลน์ 40 เคอร์เซอร์จะถูกส่งกลับไปยังตำแหน่งเริ่มต้นของมัน

CVI CVS CVD Functions

รูปแบบ CVI(Z-byte string)

CVS(4-byte string)

CVD(8-byte string)

จุดประสงค์ แปลงค่าตัวแปรสตริงก์ (string variables) ไปสู่ตัวแปรจำนวนเลข (numeric variables)

รายละเอียด - มูลค่าจำนวนเลขต่าง ๆ ที่ถูกอ่านจาก rendom file ต้องถูกแปลงค่าจากสตริงก์ไปสู่จำนวนเลขต่าง ๆ CVI แปลงค่าสตริงก์ขนาด 2 ไบท์ไปสู่

จำนวนเต็ม (integer) CVS แปลงค่าสตริงก์ขนาด 4 ไบท์ไปสู่จำนวนเลขในรูป single presion CVD แปลงค่าสตริงก์ขนาด 8 ไบท์ไปสู่จำนวนเลขในรูป double precision

- เพื่อที่จะแปลงค่ามูลค่าจำนวนเลขต่าง ๆ ไปสู่มูลค่าสตริงก์ต่าง ๆ ให้ดูฟังก์ชัน MKI$ MKS$ และฟังก์ชัน MKD$

ตัวอย่าง

```
70 FIELD #1,4 AS N$, 12 AS B$, . . .
80 GET #1
90 Y=CVS(N$)
```

ไลน์ 70 มี random file (#1) ประกอบด้วยฟิลด์ต่าง ๆ ที่แน่นอน ไลน์ 80 อ่านเร็คคอร์ดจากไฟล์ และไลน์ 90 ใช้ฟังก์ชัน CVS เพื่อที่จะแปลงค่า (N$) เป็นตัวเลขในรูป single precision

DATA Statement

รูปแบบ DATA list of constants

รายละเอียด เก็บค่าคงที่จำนวนเลข (numeric constants) และค่าคงที่สตริงก์ (string constants) ซึ่งถูกเข้าถึง (accessed) โดยคำสั่ง READ ของโปรแกรมตั้งแต่ 1 คำสั่งขึ้นไป

รายละเอียด

- คำสั่ง DATA เป็น nonexecutable และอาจถูกวางในที่ใด ๆ ในโปรแกรม คำสั่ง DATA อาจจะบรรจุค่าคงที่ต่าง ๆ มากมายเท่าที่จะเหมาะสมบนหนึ่งไลน์ จำนวนเลขใด ๆ ของคำสั่ง DATA อาจจะถูกใช้ในโปรแกรม คำสั่ง READ เข้าถึงคำสั่ง DATA ตามลำดับของไลน์นัมเบอร์ ข้อมูลที่ถูกกำหนดให้ในคำสั่งต่าง ๆ เหล่านี้อาจจะถูกคิดถึงในฐานะเป็นรายการที่ต่อเนื่องของรายการข้อมูล (items) ต่าง ๆ 1 รายการโดยไม่คำนึงถึงจำนวนมากน้อยเท่าไรของรายการข้อมูลอยู่บนไลน์หรือที่ซึ่งไลน์ต่าง ๆ ถูกวางในโปรแกรม

- list of constants อาจบรรจุค่าคงที่จำนวนเลขต่าง ๆ ในรูปแบบใด ๆ ไม่ว่าจะเป็น fixed-point floating-point หรือ integer นิพจน์จำนวนเลขต่าง ๆ ไม่สามารถใช้ใน list) ค่าคงที่สตริงก์ต่าง ๆ ในคำสั่ง DATA ต้องอยู่ภายในเครื่องหมายคำพูดคู่ก็ต่อเมื่อมันบรรจุเครื่องหมายจุลภาค (,) โคลอน (:) หรือที่ว่างต่าง ๆ ที่อยู่ข้างหน้าหรือข้างหลังที่มีความสำคัญ (leading or trailing spaces) มิฉะนั้นแล้วเครื่องหมายคำพูดนี้จะไม่จำเป็น
- ชนิดของตัวแปร (จำนวนเลขหรือสตริงก์) ที่ถูกกำหนดในคำสั่ง READ ต้องสอดคล้องกับค่าคงที่ที่สัมพันธ์กันในคำสั่ง DATA
- ใช้คำสั่ง RESTORE เพื่อที่จะอ่านคำสั่ง DATA ใหม่

ตัวอย่าง อ้างอิงไปยังคำสั่ง READ

DATE$ Statement

รูปแบบ DATE$ = string expression

จุดประสงค์ กำหนดวันเดือนปี (date)

รายละเอียด - string expression จะถูกป้อนเข้าไปดังต่อไปนี้

mm-dd-yy

mm/dd/yy

mm-dd-yyyy

mm/dd/yyyy

ตัวอย่าง 10 DATE$="02-11-84"

วันเดือนปีปัจจุบันถูกกำหนดเป็นวันที่ 11 กุมภาพันธ์ ปีค.ศ.1984

DATE$ Varible

รูปแบบ v$=DATE$

จุดประสงค์ ส่งค่าวันเดือนปีของระบบ (system date)

รายละเอียด - ตัวแปรนี้ส่งสตริงก์ของ 10 อักขระในรูปแบบ mm-dd-yyyy เมื่อ mm คือเดือน (00 ถึง 12) dd คือวันที่ (01 ถึง 31) และ yyyy คือปีค.ศ. (1980 ถึง 2099)

ตัวอย่าง 10 PRINT DATE$

วันเดือนปีจะถูกพิมพ์ตามที่ถูกกำหนดด้วยคำสั่ง DATE$

DEF FN Statement

รูปแบบ DEF FNname[(parameter list)]=function definition

จุดประสงค์ - บ่งบอกและตั้งชื่อฟังก์ชันที่ถูกเขียนโดยผู้ใช้ (user)

รายละเอียด - name ต้องเป็นชื่อตัวแปรที่ใช้ได้ (valid) ชื่อที่ถูกนำหน้าโดย FN นี้จะเป็นชื่อของฟังก์ชัน

- parameter list ประกอบด้วยชื่อตัวแปรต่าง ๆ ใน function definition ซึ่งจะถูกแทนที่เมื่อฟังก์ชันถูกเรียก Items ใน list ถูกแบ่งแยกด้วยเครื่องหมายจุลภาค

- function definition เป็นนิพจน์ซึ่งกระทำการกระทำของ (operation) ของฟังก์ชัน มันถูกจำกัดขอบเขตใน 1 logical line ชื่อตัวแปรต่าง ๆ ซึ่งปรากฏในนิพจน์นี้สงวนไว้เพื่อที่จะบ่งบอกฟังก์ชันเท่านั้น มันไม่ส่งผลกระทบต่อตัวแปรต่าง ๆ ของโปรแกรมซึ่งมีชื่อเดียวกัน ชื่อตัวแปรที่ถูกใช้ใน function definition อาจจะปรากฏหรือไม่ปรากฏใน parameter list ถ้ามันปรากฏอยู่ มูลค่าของพารามิเตอร์จะถูกจัดจ่ายเมื่อฟังก์ชันถูกเรียก มิฉะนั้นแล้วมูลค่าปัจจุบันของตัวแปรจะถูกใช้

- ตัวแปรต่าง ๆ ใน parameter list แทนแบบหนึ่งต่อหนึ่งของตัวแปรอาร์กิวเมนต์ (argument variables) ต่าง ๆ หรือมูลค่าต่าง ๆ ซึ่งจะถูกกำหนดให้ในฟังก์ชันเรียก (function call)
- คำสั่งนี้อาจจะบ่งบอกเป็นฟังก์ชันจำนวนเลขหรือไม่ก็เป็นฟังก์ชันสตริงก์ ถ้าชนิด (type) ถูกระบุในชื่อของฟังก์ชันแล้วมูลค่าของนิพจน์จะถูกกระทำให้เป็นชนิดนั้นก่อนที่มันจะถูกส่งค่าไปยังคำสั่งที่เรียก (calling statement) ถ้าชนิดถูกระบุในชื่อฟังก์ชันและชนิดของอาร์กิวเมนต์ไม่สอดคล้องกัน จะเกิด "Type mismatch" error ขึ้น
- คำสั่ง DEF FN ต้องถูกป้อนเข้าไปก่อนที่ฟังก์ชันฟังก์ชันที่มันบ่งบอกอาจจะถูกเรียก ถ้าฟังก์ชันถูกเรียกก่อนที่มันจะถูกบ่งบอก จะเกิด "Undefined user function" error ขึ้น ในรูปแบบตรง (direct mode) คำสั่ง DEF FN ไม่สามารถใช้ได้

ตัวอย่าง

```
410 DEF FNAB(X,Y)=X*3/Y^2
420 T = FNAB(I,J)
```

ไลน์ 410 บ่งบอกฟังก์ชัน FNAB ฟังก์ชันถูกเรียกในไลน์ 420

DEF SEG Statement

รูปแบบ DEF SEG [=address]

จุดประสงค์ กำหนดค่า segment address ปัจจุบันที่จะถูกอ้างอิงโดยคำสั่ง BLOAD BASVE CALL CALLS หรือคำสั่ง POKE ที่ตามมา

รายละเอียด - address เป็นเลขจำนวนเต็มในช่วง 0 ถึง 65535

- แอดเดรส (address) ที่ถูกระบุถูกเก็บรักษาไว้เพื่อใช้ในฐานะเป็นเซกเมนต์ (segment) ที่ถูกต้องการโดย BLOAD BSAVE CALL CALLS POKE USR และ PEEK
- การป้อนมูลค่าต่าง ๆ ออกนอกช่วงของแอดเดรสเข้าไปจะส่งผลให้เกิด "Illegal function call" error ขึ้นและมูลค่าก่อนหน้าจะถูกเก็บรักษาไว้
- ถ้า address ถูกละแล้วเซกเมนต์ที่จะถูกใช้จะถูกกำหนดไปที่ GWBASIC data segment (DS) สิ่งนี้เป็น initial default value
- ถ้า address ถูกกำหนดให้แล้วมันควรจะถูกกำหนดฐานบนขอบเขตของ 16 ไบท์ (16-byte boundary) GWBASIC ไม่ได้ตรวจเช็คความถูกต้องของแอดเดรสที่ถูกระบุ
- หมายเหตุ : DEF และ SEG ต้องถูกแบ่งแยกด้วยช่องว่าง (space) ไม่เช่นนั้นแล้ว GWBASIC จะแปลความหมายคำสั่ง DEFSEG=100 หมายถึง "กำหนดมูลค่า 100 ไปที่ตัวแปร DEFSEG"

ตัวอย่าง

```
10 DEF SEG=&HB800 'Set segment address to B800 Hex
20 DEF SEG 'Restore segment to GWBASIC data segment
```

ไลน์ต่าง ๆ ข้างบนนี้กำหนด (set) และ restore data segment

DEFtype Statements

รูปแบบ DEFtype range(s) of letters

จุดประสงค์ ประกาศชนิดต่าง ๆ ของตัวแปรเป็น integer single precision double precision หรือ string

รายละเอียด - type คือ INT SNG DBL หรือ STR

- ชื่อตัวแปรใด ๆ ที่เริ่มต้นด้วยอักษรที่ถูกระบุใน range of letters จะถูกพิจารณาเป็นชนิดของตัวแปรที่ถูกระบุในส่วนของ type ของคำสั่ง อย่างไรก็ตาม อักขระการประกาศชนิด (type declaration character) มีลำดับการมาก่อนเหนือคำสั่ง DEFtype
- ถ้าไม่มีคำสั่งการประกาศชนิดถูกป้อนเข้าไปแล้ว GWBASIC จะกำหนดว่าตัวแปรทุก ๆ ตัวที่ปราศจากอักขระการประกาศค่าต่าง ๆ เป็นตัวแปรในรูป single precision

ตัวอย่าง 10 DEFDBL L-P

ตัวแปรทุก ๆ ตัวที่เริ่มต้นด้วยอักษร L M N O และอักษร P จะเป็นตัวแปรในรูป double precision

10 DEFSTR A

ตัวแปรทุก ๆ ตัวที่เริ่มต้นด้วยอักษร A เป็นตัวแปรสตริงก์ (string variable)

10 DEFINT I-N,W-Z

ตัวแปรทุก ๆ ตัวที่เริ่มต้นด้วยอักษร I J K L M N W X Y และอักษร Z จะเป็นตัวแปรชนิด integer

DEF USR Statement

รูปแบบ DEF USR[digit]=integer expression

จุดประสงค์ ระบุแอดเดรสเริ่มต้นของสับรูทีนภาษาแอสเซมบลี

รายละเอียด
- digit อาจเป็นตัวเลขใด ๆ จาก 0 ถึง 9 ตัวเลขสอดคล้องกันกับตัวเลขของ USR routine ที่ซึ่งแอดเดรสจะถูกระบุ ถ้า digit ถูกละแล้ว DEF USR0 จะถูกกำหนดขึ้น

- integer expression เป็นแอดเดรสเริ่มต้นของ USR routine
- จำนวนใด ๆ ของคำสั่ง DEF USR อาจปรากฏในโปรแกรมเพื่อที่จะบ่งบอกแอดเดรสเริ่มต้นต่าง ๆ ของสับรูทีนใหม่ ด้วยเหตุนี้จะอนุญาตให้เข้าถึงสับรูทีนมากมายเท่าที่จำเป็น

ตัวอย่าง

```
200 DEF USR0=24000
210 X=USR0(Y-2/2.89)
```

เรียกสับรูทีนที่ absolute location 24000

DELETE Command

รูปแบบ DELETE [line number] [-line number]

จุดประสงค์ ลบไลน์ต่าง ๆ ของโปรแกรม

รายละเอียด
- โดยปกติ GWBASIC จะหันกลับไปยังระดับคำสั่ง (command level) หลังจากที่ DELETE ถูกประมวลผลแล้ว
- ถ้า line number ไม่มีอยู่ จะเกิด "Illegal function call" error ขึ้น

ตัวอย่าง DELETE 40

ลบไลน์ 40

DELETE 40-100

ลบไลน์ 40 ถึงไลน์ 100

DELETE -40

ลบไลน์ทุก ๆ ไลน์จนถึงและรวมทั้งไลน์ 40

DIM Statement

รูปแบบ DIM list of subscripted variables

จุดประสงค์ ระบุมูลค่าสูงสุดต่าง ๆ สำหรับสับสคริพท์ต่าง ๆ ของตัวแปรอะเรย์และจัดสรรเนื้อที่หน่วยความจำ (storage) ไปตามนั้น

รายละเอียด
- ถ้าชื่อของตัวแปรอะเรย์ถูกใช้โดยปราศจากคำสั่ง DIM แล้วมูลค่าสูงสุดของสับสคริพท์ของอะเรย์จะถูกกำหนดเป็น 10 ถ้าสับสคริพท์ที่ถูใช้มีค่ามากกว่าค่าสูงสุดที่ถูกระบุแล้ว จะเกิด "Subscript out of range" error ขึ้น มูลค่าต่ำสุดสำหรับสับสคริพท์โดยปกติเป็นศูนย์ แม้ว่ามันจะถูกระบุด้วยคำสั่ง OPTION BASE ก็ตาม
- คำสั่ง DIM กำหนดค่าของทุก ๆ สมาชิกของอะเรย์ที่ถูกระบุให้มีค่าเริ่มต้นเป็นศูนย์
- ตามทฤษฎีแล้ว จำนวนสูงสุดของมิติ (dimension) ต่าง ๆ ที่ถูกอนุญาตในคำสั่ง DIM คือ 255 อย่างไรก็ตาม โดยแท้จริงแล้วจำนวนนั้นอาจจะเป็นไปไม่ได้เนื่องจากชื่อและเครื่องหมายวรรคตอนถูกนับในฐานะเป็นช่องว่าง (spaces) ต่าง ๆ บนไลน์ด้วยและตัวไลน์เองมีขอบเขตจำกัดที่ 255 อักขระ จำนวนของมิติต่าง ๆ ถูกจำกัดขอบเขตมากยิ่งขึ้นโดยจำนวนของหน่วยความจำ (memory) ที่สามารถใช้ได้

ตัวอย่าง

```
10 DIM A(20)
20 FOR I=0 TO 20
30 READ A(1)
40 NEXT I
.
.
```

ตัวอย่างนี้อ่าน ตัวเลข 20 ตัวจากคำสั่ง DATA เข้าสู่อะเรย์ A

DRAW Statement

รูปแบบ DRAW string expression

จุดประสงค์ วาดไลน์ตามที่ถูกบ่งบอกโดยนิพจน์สตริงก์ (string expression)

รายละเอียด

- string expression เป็นคำสั่งย่อย (subcommand) ซึ่งเรียกสำหรับการเคลื่อนไหว (บน ล่าง ซ้าย ขวา) สี (color) มุม และ scale factor
- คำสั่งย่อยแต่ละคำสั่งต่อไปนี้เริ่มต้นการเคลื่อนย้ายจากตำแหน่งกราฟฟิคปัจจุบัน (current graphics position) โดยปกติสิ่งนี้คือโคออร์ดิเนตของจุดของกราฟฟิคต่างๆ ล่าสุดที่ถูกพล็อตด้วย LINE หรือ PSET Default คือจุดศูนย์กลางของจอภาพ

  U n Move up (scale factor*n) points

  D n Move down

  L n Move left

  R n Move right

  E n Move diagonal up and right

F 'n Move diagonal up and left

G n Move diagonal down and left

H n Move digonal down and right

M x,y Move absolute หรือ realtive ถ้า x ถูกนำหน้าโดยเครื่องหมายบวก (+) หรือเครื่องหมายลบ (-) แล้ว x และ y จะถูกบวกเพิ่มไปยังตำแหน่งกราฟฟิคปัจจุบันและถูกเชื่อมโยง (connected) กับตำแหน่งปัจจุบันโดยไลน์ มิฉะนั้นแล้วไลน์จะถูกวาดไปยังจุด x,y จากตำแหน่งปัจจุบันของเคอร์เซอร์

Prefix commands ต่อไปนี้อาจจะนำหน้าคำสั่งการเคลื่อนย้ายข้างบนใด ๆ :

B เคลื่อนย้ายแต่ไม่พล็อตจุดใด ๆ

N เคลื่อนย้ายแต่หันกับไปยังตำแหน่งเริ่มต้นเมื่อถูกกระทำ

An กำหนดค่ามุม n n อาจอาจอยู่ในช่วงจาก 0 ถึง 3 เมื่อ 0 คือ 0 องศา 1 คือ 90 องศา และ 3 คือ 270 องศา รูปต่าง ๆ ที่ถูกหมุนรอบไป 90 องศาหรือ 270 องศาถูกกำหนกสเกล (scaled) เพื่อว่ามันจะปรากฏขนาดเดียวกันกับ 0 องศาหรือ 180 องศาบน monitor screen ด้วย standard aspect ration ของ 4/3

Cn กำหนดสี n n อาจอยู่ในช่วง 0 ถึง 3

Sn กำหนด scale factor n อาจอยู่ในช่วงจาก 1 ถึง 255 scale factor ถูกคูณด้วยระยะทางต่าง ๆ ที่ถูกกำหนดให้ด้วยคำสั่ง U D L R หรือคำสั่ง relative M ให้ระยะทางที่แท้จริงที่ถูกเดินทาง

x string expression

ประมวลผลสตริงก์ย่อย (substring) คำสั่งนี้อนุญาตให้ท่านประมวล-ผลสตริงก์ย่อยจากสตริงก์ ท่านสามารถมีสตริงก์หนึ่งประมวลผลสตริงก์อื่น ซึ่งประมวลผลสตริงก์ที่ 3 และเป็นเช่นนี้ต่อไป

อาร์กิวเมนต์จำนวนเลข (numeric arguments) ต่าง ๆ สามารถเป็นค่าคงที่ต่าง ๆ อย่างเช่น "123" หรือ "=variable" เมื่อ variable เป็นชื่อของตัวแปร

ตัวอย่าง

```
10 SCREEN 105
20 CLS
30 PSET (160,100),1
40 DRAW "V20R30D20L30w
```

ตัวอย่างนี้จะวาดรูปกล่อง

EDIT Command

รูปแบบ EDIT line number

จุดประสงค์ ป้อน edit mode เข้าไป ณ ไลน์ที่ถูกระบุ

รายละเอียด - หลังจากที่ทำการป้อนหมายเลขของไลน์ที่จะถูกบรรณาธิกรณ์ (editing) เข้าไป GWBASIC จะพิมพ์ไลน์ตลอดไลน์เพื่อที่ท่านจะทำการบรรณาธิกรณ์

END Statement

รูปแบบ END

จุดประสงค์ สิ้นสุดการประมวลโปรแกรม ปิดไฟล์ทุก ๆ ไฟล์และหันกลับไปยังระดับคำสั่ง

รายละเอียด - คำสั่ง END อาจจะวาง ณ ที่ใด ๆ ในโปรแกรมเพื่อที่จะสิ้นสุดการประมวลผล ไม่เหมือน STOP คำสั่ง END ไม่ได้ทำให้ข้อความ "Break" ถูกพิมพ์และ

END ปิดไฟล์ทุก ๆ ไฟล์

- คำสั่ง END ที่ตอนท้ายของโปรแกรมสามารถเลือกได้ (optional) โดยปกติ GWBASIC จะหันกลับไปยังระดับคำสั่งหลังจากการประมวลผล END

ตัวอย่าง
```
520 IF K>1000 THEN END ELSE GOTO 20
```
จากตัวอย่างนี้ ถ้า K มีค่ามากกว่า 1000 จะสิ้นสุดโปรแกรม มิฉะนั้นแล้วโปรแกรมจะย้าย (branch) ไปยังไลน์หมายเลข 20

EOF Function

รูปแบบ EOF(file number)

จุดประสงค์ ตรวจสอบต่าง ๆ สำหรับเงื่อนไข end-of-file

รายละเอียด
- ฟังก์ชันนี้ส่งค่า -1 (จริง) ถ้าพบว่าข้อมูลใน sequential file หมดแล้ว เราใช้ฟังก์ชันนี้เพื่อที่จะหลีกเลี่ยงกรณีของ "Input past end" errors
- เมื่อ EOF ถูกใช้ร่วมกับอุปกรณ์การติดต่อสื่อสาร (communication device) นิยามของเงื่อนไข end-of-file จะขึ้นอยู่กับ mode (ASCII หรือ binary) กับอุปกรณ์ถูกเปิด ใน binary mode EOF จะเป็นจริงเมื่อคิวข้อมูลเข้า (input queue) ว่างเปล่า นั่นคือ LOC(n)=0 และจะเป็นเท็จเมื่อคิวข้อมูลเข้าไม่ว่างเปล่า ใน ASCII mode EOF จะเป็นเท็จจนกว่า Control-Z จะถูกได้รับและจากนั้นมันจะยังคงเป็นจริงอยู่จนกว่าอุปกรณ์จะถูกปิด

ตัวอย่าง
```
10 OPEN "I", 1, "DATA"
20 c=o
30 IF EOF(1) THEN 100
40 IN-PUT #1,M(C)
50 C=C+1:GOTO 30
```

.

.

.

อ่านข่าวสารจากไฟล์ที่ชื่อ DATA ไปในอะเรย์ M และเมื่อจุดจบของไฟล์มาถึง โปรแกรมจะย้ายไปยังไลน์ 100

ERASE Statement

รูปแบบ ERASE arrayname[,arrayname]...

จุดประสงค์ ลบอะเรย์ต่าง ๆ ออกจากโปรแกรม

รายละเอียด
- arrayname เป็นชื่อของอะเรย์ที่ท่านต้องการจะลบออกไป
- คำสั่งนี้สามารถถูกใช้เพื่อที่จะทำให้เนื้อที่ทำงาน (workspace) สามารถเรียกหาได้ เช่นเดียวกันมันสามารถถูกใช้เพื่อที่จะกำหนดมิติของอะเรย์ใหม่อีกครั้งหนึ่ง ถ้าท่านพยายามที่จะกำหนดอะเรย์ใหม่โดยปราศจากการลบมันออกไปก่อนจะทำให้เกิด "Duplicate Definition" error ขึ้น

ตัวอย่าง

.

.

.

```
450  ERASE  A,B
460  DIM  B(99)
```

.

.

.

ตัวอย่างนี้แสดงวิธีที่จะกำหนดมิติของอะเรย์ "B" ใหม่

ERR and ERL Variables

รูปแบบ **ERR**

ERL

จุดประสงค์ ส่งรหัสความผิดพลาด (error code) และหมายเลขของไลน์ที่ซึ่ง error ได้เกิดขึ้น

รายละเอียด

- เมื่อรูทีนการจัดการความผิดพลาด (error-handling routine) ถูกป้อนเข้าไป ตัวแปร ERR บรรจุรหัสความผิดพลาดสำหรับ error และตัวแปร ERL บรรจุไลน์นัมเบอร์ของไลน์ในที่ซึ่ง error ถูกตรวจจับ (detected) โดยปกติตัวแปร ERR และตัวแปร ERL ถูกใช้ในคำสั่ง IF...THEN เพื่อที่จะชี้ทางเดินของโปรแกรมในรูทีนการจัดการความผิดพลาด อ้างถึงคำสั่ง ON ERROR ในบทนี้
- ถ้าคำสั่งซึ่งเป็นเหตุให้เกิด error เป็นคำสั่งในรูปแบบตรง (direct mode) แล้ว ERL จะบรรจุ 65535 เพื่อที่จะตรวจสอบว่า error ถูกเกิดขึ้นใน direct statement ใช้รูปฟอร์ม :

  ```
  IF 65535 = ERR TREN . . .
  ```

  มิฉะนั้นแล้วใช้

  ```
  IF ERR = error code THEN ...
  IF ERL = line number THEN ...
  ```

- ทำให้แน่ใจว่าใส่ไลน์นัมเบอร์ทางด้านขวามือของตัวกระทำแสดงความสัมพันธ์ (relational operator) เพื่อว่ามันจะสามารถถูกเรียงลำดับใหม่โดยคำสั่ง RENUM
- เนื่องจาก ERL และ ERR เป็นตัวแปรต่าง ๆ ที่ถูกสงวน (reserve) ไว้ ทั้งคู่อาจไม่ปรากฏทางด้านซ้ายของเครื่องหมายเท่ากับในคำสั่ง LET
- รหัสความผิดพลาดต่าง ๆ ของ GWBASIC ถูกแสดงไว้ในภาคผนวก ฉ

ดูคำสั่ง ERROR ด้วย

ตัวอย่าง

```
10 ON ERROR GOT0 60
20 OPEN "I",#1,"JUNK.DAT"
30 CLOSE #1
40 PRINT "The file exists"
50 END
60 IF ERR=53 AND ERL=20 THEN PRINT "File dose not exist"
70 PRINT "Error";ERR;"at line";ERL
80 STOP
```

ตัวอย่างนี้แสดงวิธีที่จะตรวจสอบสำหรับการมีอยู่ของไฟล์

ERROR Statement

รูปแบบ ERROR n

จุดประสงค์ สร้างจำลอง (simulate) การเกิดขึ้นของ error ของ GWBASIC โดยผู้ใช้ ทั้งนี้เพื่อให้ท่านบ่งบอกรหัสความผิดพลาดต่าง ๆ เอง

รายละเอียด
- n ต้องเป็นนิพจน์จำนวนเต็มมีค่าระหว่าง 0 ถึง 255
- ถ้ามูลค่าของ n เป็นเช่นเดียวกันกับรหัสความผิดพลาดที่มีใช้อยู่แล้ว (ดูภาคผนวก ฉ) คำสั่ง ERROR จะสร้างการเกิดขึ้น (occurance) ของ error นั้น นั่นคือ ข้อความผิดพลาด (error message) ที่สอดคล้องกับรหัสจะถูกพิมพ์และการประมวลผลจะหยุดลง (ดูตัวอย่างแรกข้างล่าง)
- เพื่อที่จะบ่งบอกรหัสความผิดพลาดที่เป็นของท่านเอง ใช้มูลค่าที่แตกต่างจากรหัสความผิดพลาดที่ถูกใช้โดย GWBASIC (เราเสนอแนะให้ท่านใช้มูลค่าต่างๆ ที่มีค่าสูงสุดเท่าที่จะเป็นไปได้ที่จะอนุญาตสำหรับรหัสความผิดพลาดต่าง ๆ เพิ่มเติมที่จะถูกเพิ่มเข้าไปยัง GWBASIC) จากนั้นรหัสความผิดพลาดใหม่ของท่าน

อาจจะถูกตรวจสอบในรูทีนการจัดการความผิดพลาด(ดูตัวอย่างที่สองข้างล่าง)

- ถ้าท่านได้ทำการบ่งบอกรูทีนการจัดการความผิดพลาดด้วย ON ERROR โปรแกรมจะป้อนรูทีนนี้เข้าไปเมื่อมันตรวจพบความผิดพลาด (error)
- ถ้าท่านบ่งบอกรหัสที่เป็นของท่านเองและไม่ได้ยึดถือความผิดพลาดในรูทีนการจัดการความผิดพลาด GWBASIC จะพิมพ์ข้อความ "Unprintable error" และการประมวลผลจะหยุดลง

ตัวอย่างที่ 1

```
LIST
10 S=10
20 T=5
30 ERROR S+T
40 END
Ok
RUN
String too long in line 30
```

หรือในรูปแบบตรง :

```
Ok
ERROR15                 (ท่านพิมพ์ไลน์นี้)
String too long         (GWBASIC พิมพ์ไลน์นี้)
Ok
```

ตัวอย่างข้างบนนี้แสดง "String too long" error

ตัวอย่างที่ 2

```
110 ON ERROR GOT0 400
120 INPUT "WHAT IS YOUR BET";B
130 IF B>5000 THEN ERROR 210



400 IF ERR=210 THEN PRINT "HOUSELIMIT IS $5000"
410 IF ERL=130 THEN RESUME 120
 .
 .
 .
```

ตัวอย่างนี้ดัก (trap) ความผิดพลาดด้วยรหัส 210

EXP Function

รูปแบบ v=EXP(x)

จุดประสงค์ ให้ค่า e ยกกำลัง x

รายละเอียด
- x อาจเป็นนิพจน์จำนวนเลขใด ๆ
- ฟังก์ชันนี้ส่งค่า e ฐานของลอการิทึมธรรมชาติยกกำลัง x ถ้า x มีค่ามากกว่า 88.02969 จะเกิด overflow error ขึ้น หลังจากที่ข้อความผิดพลาด "Overflow" ถูกพิมพ์ ค่าที่มากมายของเครื่อง (machine infinity) พร้อมด้วยเครื่องหมายจะถูกจัดจ่ายในฐานะเป็นผลลัพธ์และการประมวลผลจะดำเนินการต่อไป

ตัวอย่าง
```
10 X=5
20 PRINT EXP(X-1)
```

```
RUN
 54.59815
Ok
```

ตัวอย่างนี้คำนวณค่าของ e ยกกำลัง 4

FIELD Statement

รูปแบบ FIELD [#]file number,field width AS string variable...

จุดประสงค์ จัดสรรเนื้อที่ (space) สำหรับตัวแปรต่าง ๆ ในบัฟเฟอร์ (buffer) ของ random file

รายละเอียด
- ก่อนที่คำสั่ง GET หรือคำสั่ง PUT สามารถถูกปฏิบัติ ท่านต้องใช้คำสั่ง FIELD เพื่อที่จะกำหนดรูปแบบ (format) บัฟเฟอร์ของ random file
- file number เป็นตัวเลขภายใต้ไฟล์ที่ถูกเปิด
- field width เป็นจำนวนของอักขระต่าง ๆ ที่จะถูกจัดสรรไปยังตัวแปรสตริงก์ (string variable)
- จำนวนรวมของไบท์ต่าง ๆ ที่ถูกจัดสรรในคำสั่ง FIELD ต้องไม่มากเกินกว่าความยาวของเร็คคอร์ดซึ่งถูกระบุเมื่อไฟล์ถูกเปิด มิฉะนั้นแล้วจะเกิด"FIELD overflow" error ขึ้น (defaule record length คือ 128 ไบท์)
- จำนวนใด ๆ ของคำสั่ง FIELD ต่าง ๆ อาจจะถูกประมวลผลสำหรับไฟล์เดียวกัน ในขณะเดียวกันคำสั่ง FIELD ทุก ๆ คำสั่งซึ่งถูกประมวลผลจะยังคงมีผลอยู่
- หมายเหตุ : อย่าใช้ชื่อตัวแปรที่ถูกกำหนดขอบเขต (fielded variable name) ในคำสั่ง INPUT หรือในคำสั่ง LET เมื่อชื่อตัวแปรถูกกำหนดขอบเขต (fielded) มันชี้ไปยังสถานที่ที่ถูกต้องในบัฟเฟอร์ของ random file

ถ้าคำสั่ง INPUT หรือคำสั่ง LET ต่อ ๆ มาพร้อมด้วยชื่อตัวแปรนั้นถูกประมวลผล,ตัวชี้ (pointer) ของตัวแปรจะถูกย้ายไปยังเนื้อที่สตริงก์ (string space)

ตัวอย่าง

```
FIELD#1,20 AS N$,10 AS ID$,40 AS ADD$
```

จัดสรรเนื้อที่ 20 ตำแหน่ง(ไบท์)แรกในบัฟเฟอร์ของ random file ไปยังตัวแปรสตริงก์ N$ 10 ตำแหน่งถัดไปไปยัง ID$ และ 40 ตำแหน่งถัดไปไปยัง ADD$ FIELD ไม่ได้แทนข้อมูลใด ๆ ลงในบัฟเฟอร์ของ random file

```
10 OPEN "R",#1,"A:PHONELST",35
15 FIELD #1,2 AS RECNBR$,33 AS DUMMY$
20 FIELD #1,25 AS NAME$,10 AS PHONENBR$
25 GET #1
30 TOTAL=CVI(RECNBR$)
35 FOR I=2 TO TOTAL
40 GET #1, I
45 PRINT NAME$, NAME$
50 NEXT I
```

แสดงคำสั่ง multiple defined FIELD ในคำสั่งที่ 15 ฟิลด์ขนาด 35 ไบท์ถูกบ่งบอกสำหรับเร็คคอร์ดแรกเพื่อที่จะเก็บรักษาแทรค (track) ของจำนวนของเร็คคอร์ดต่าง ๆ ในไฟล์ ในลูป (loop) ของคำสั่งต่าง ๆ ถัดไป (35-50) คำสั่งที่ 20 บ่งบอกฟิลด์สำหรับแต่ละชื่อและแต่ละหมายเลขโทรศัพท์

```
10 FOR LOOP%=0 TO 7
20 FIELD #1, (LOOP%*16) AS OFFSET$, 16 AS A$(LOOP%)
30 NEXT LOOP%
```

แสดงการสร้างของคำสั่ง FIELD โดยการใช้อะเรย์ของสมาชิกต่าง ๆ ที่มีขนาดเท่ากัน ผลลัพธ์จะสมนัยกันกับการประกาศเดี่ยว ๆ ดังนี้

```
FIELD #1,16 AS A$(0),16 AS A$(1),...,16 AS A$(6),16 AS
      A$(7)
```

```
10 DIM SIZE% (4%): REM ARRAY OF FIELD SIZES
20 FOR LOOP%=0 TO 4
30     READ SIZE% (LOOP%)
40 NEXT LOOP%
50 DATA 9, 10,12,21,41
```

```
120 DIM A$(4%): REM ARRAY OF FIELDED VARIABLES
130  OFFSET%=0
140 FOR LOOP%=0 TO 4%
150 FIELD #1,OFFSET% AS OFFSET$,SIZE%(LOOP%) AS A$(LOOP%)
160 OFFSET%=OFFSET%+SIZE%(LOOP%)
170 NEXT LOOP%
```

สร้างฟิลด์ในลักษณะเดียวกันกับตัวอย่างก่อนหน้า อย่างไรก็ตาม ขนาดของสมาชิกแปรเปลี่ยนไปตามแต่ละสมาชิก การประกาศที่เท่าเทียมกันคือ

FIELD #1,SIZE%(0) AS A$(0),SIZE%(1) AS A$(1),...

SIZE%(4) AS A$(4%)

FILES Statement

รูปแบบ FILES [filespec]

จุดประสงค์ พิมพ์ชื่อของไฟล์ต่าง ๆ ที่อยู่บนดิสค์ที่ถูกระบุ

รายละเอียด

- filespec ประกอบด้วยชื่อไฟล์และ optional device designation
- ถ้า filespec ถูกละแล้วไฟล์ทุก ๆ ไฟล์ของไดร์ฟที่ถูกเลือกในขณะนั้นจะถูกแสดงรายการ filespec อาจจะประกอบด้วยเครื่องหมายคำถาม (?) หรือเครื่องหมายดอกจัน (*) ที่ถูกใช้ในฐานะเป็น wild cards เครื่องหมาย ? จะจับคู่อักขระตัวเดียวใด ๆ ในชื่อไฟล์หรือส่วนขยาย (extension) เครื่องหมาย * จะจับคู่อักขระตั้งแต่หนึ่งตัวขึ้นไปเริ่มต้นที่ตำแหน่งนั้น เครื่องหมายดอกจันเป็นสัญญลักษณ์ย่อสำหรับชุดลำดับของเครื่องหมายคำถาม

ตัวอย่าง FILES

แสดงไฟล์ทุก ๆ ไฟล์บนไดร์ฟที่ถูกกำหนด (logged drive) ในขณะนั้น

FILES "*.BAS"

แสดงไฟล์ทุก ๆ ไฟล์ที่มีส่วนขยายเป็น .BAS

FILES "B:*.*"

แสดงไฟล์ทุก ๆ ไฟล์ที่อยู่ในดิสค์ในไดร์ฟ B

FILES "B:"

คำสั่งนี้เท่าเทียมกันกับคำสั่ง FILES "B:*.*"

FILES "TEST?.BAS"

แสดงไฟล์ทุก ๆ ไฟล์ที่มี 5 ตัวอักขระที่ชื่อต่าง ๆ เริ่มต้นด้วย "TEST"
และจบลงด้วยส่วนขยาย .BAS

FIX Function

รูปแบบ FIX(x)

จุดประสงค์ ปัดเศษ (truncate) ของ x ทิ้งไปให้อยู่ในรูปจำนวนเต็ม

รายละเอียด
- x อาจเป็นนิพจน์จำนวนเลขใด ๆ
- FIX ส่งมูลค่าของตัวเลขต่าง ๆ ที่อยู่ทางด้านซ้ายของจุดทศนิยมโดยปราศจากการปัดเศษ (rounding) มูลค่าของตัวเลขต่าง ๆ ที่อยู่ทางด้ายขวาของจุดทศนิยม
- ข้อแตกต่างระหว่าง FIX และ INT ก็คือว่า FIX ไม่ได้ส่งค่าตัวเลขที่มีค่าต่ำกว่าถัดไปเมื่อ x มีมูลค่าเป็นลบ (ดู INT และ CINT)
- สังเกตได้ว่า FIX ไม่ได้เปลี่ยนแปลงมูลค่าของตัวเลขต่าง ๆ ที่อยู่ทางด้านซ้ายของจุดทศนิยม
- INT จะเปลี่ยนแปลงมูลค่าเมื่อ x มีมูลค่าเป็นลบ

ตัวอย่าง
```
PRINT FIX(58.75)
 58
Ok

PRINT  FIX(-28.87)
-28
Ok
```

FOR and **NEXT** Statements

รูปแบบ FOR variable=x TO y [STEP z]

.

.

NEXT [variable][,variable]...

จุดประสงค์ บ่งบอกพารามิเตอร์ต่าง ๆ สำหรับลูป

รายละเอียด

- variable เป็นตัวแปรจำนวนเต็มหรือตัวแปรในรูป single precision ที่จะถูกใช้ในฐานะเป็นตัวนับ (counter) การประมวลผลโปรแกรมจะเร็วยิ่งขึ้นสำหรับตัวแปรจำนวนเต็มต่าง ๆ
- x เป็นนิพจน์จำนวนเลขซึ่งเป็นมูลค่าเริ่มต้น (initial value) ของตัวนับ
- y เป็นนิพจน์จำนวนเลขซึ่งเป็นมูลค่าสุดท้าย (final value) ของตัวนับ
- z เป็นค่าคงที่จำนวนเต็มหรือค่าคงที่ในรูป single precision ที่จะถูกใช้ในฐานะเป็นตัวเพิ่มขึ้น (increment)
- ไลน์ต่าง ๆ ของโปรแกรมที่อยู่ถัดจากคำสั่ง FOR จะถูกประมวลผลจนกว่าคำสั่ง NEXT จะถูกพบ จากนั้นตัวนับจะถูกทำให้เพิ่มขึ้นโดยมูลค่าของ step z ถ้าท่านไม่ได้ระบุมูลค่า z การเพิ่มขึ้นจะถูกกำหนดเป็น 1 การตรวจสอบถูกกระทำซ้ำเพื่อที่จะดูว่ามูลค่าของตัวนับขณะนี้มากกว่ามูลค่าสุดท้าย y แล้วหรือยัง ถ้ามันยังไม่มากกว่า GWBASIC จะย้ายกลับไปยังคำสั่งข้างหลังคำสั่ง FOR และกระบวนการ (process) จะถูกกระทำซ้ำอีก ถ้ามันมีค่ามากกว่า การประมวลผลจะดำเนินการต่อไปกับคำสั่งต่าง ๆ ที่อยู่ถัดจากคำสั่ง NEXT สิ่งนี้เราเรียกว่าลูปของ FOR...NEXT (FOR...NEXT loop)
- ถ้า z มีค่าเป็นลบ การตรวจสอบจะเป็นไปในทางกลับกัน ตัวนับจะถูกลดลงในแต่ละครั้งที่ผ่านลูป และลูปจะถูกประมวลผลจนกว่าตัวนับจะมีค่าน้อยกว่ามูลค่า

สุดท้าย

- โครงร่าง (body) ของลูปจะถูกข้ามไปถ้า x มีค่ามากกว่า y เมื่อ z เป็นจำนวนบวก หรือถ้า x มีค่าน้อยกว่า y เมื่อ z เป็นจำนวนลบ ถ้า z มีค่าเป็นศูนย์ ลูปไม่สิ้นสุด (infinite loop) จะถูกสร้างขึ้นแม้ว่าท่านจะให้วิธีบางวิธีที่กำหนดตัวนับให้มีมูลค่ามากกว่ามูลค่าสุดท้ายก็ตาม
- ลูปที่ซ้อนกัน (Nested Loops)

  ลูปต่าง ๆ ของ FOR...NEXT อาจจะถูกทำให้ซ้อนกัน (nested) นั่นคือ ลูป FOR...NEXT อาจจะถูกกำหนดอยู่ภายในลูป FOR...NEXT อื่น เมื่อลูปซ้อนกัน แต่ละลูปต้องมีชื่อตัวแปรเดี่ยว ๆ ในฐานะเป็นตัวนับของมัน คำสั่ง NEXT ของลูปที่อยู่ด้านในต้องปรากฏก่อนคำสั่ง NEXT ของลูปที่อยู่ด้านนอก ถ้าลูปต่าง ๆ ที่ซ้อนกันมีจุดจบเดียวกัน คำสั่ง NEXT เดี่ยว ๆ อาจจะถูกใช้สำหรับทุก ๆ ลูป FOR ของมัน
- ตัวแปรหรือตัวแปรต่าง ๆ ในคำสั่ง NEXT อาจจะถูกละในกรณีที่ซึ่งคำสั่ง NEXT จะจับคู่กันกับคำสั่ง FOR ที่เพิ่งผ่านมา (most recent FOR) การใช้ชื่อตัวแปรต่าง ๆ ในคำสั่ง NEXT เป็นเหตุให้โปรแกรมต่าง ๆ นั้นประมวลผลช้าลงอย่างมาก
- ถ้าคำสั่ง NEXT ถูกพบก่อนคำสั่ง FOR ที่สอดคล้องกับมัน ข้อความผิดพลาด "NEXT without FOR" จะถูกพิมพ์ออกมาและการประมวลผลจะถูกทำให้สิ้นสุดลง

ตัวอย่างที่ 1

```
10 K=10
20 FOR I=1 TO K STEP 2
30 PRINT I;
40 K=K+10
50 PRINT K
```

```
60 NEXT
RUN
 1 20
 3 30
 5 40
 7 50
 9 60
Ok
```

ตัวอย่างนี้แสดงลูป FOR...NEXT ด้วยมูลค่าของ STEP มีค่าเป็น 2

ตัวอย่างที่ 2

```
10 J=0
20 FOR I=1 TO J
30 PRINT I
40 NEXT. I
```

ในตัวอย่างนี้ ลูปไม่ได้ประมวลผลเนื่องจากมูลค่าเริ่มต้นของลูปมีค่าเกินกว่ามูลค่าสุดท้าย

ตัวอย่างที่ 3

```
10 I=5
20 FOR I=1 TO I+5
30 PRINT I;
40 NEXT
RUN
 1 2 3 4 5 6 7 8 9 1 0
ok
```

ลูปประมวลผล 10 ครั้ง มูลค่าสุดท้ายของตัวแปรของลูปโดยปกติถูกกำหนดก่อนที่มูลค่าเริ่มต้นจะถูกกำหนด

FHE Function

รูปแบบ v = FRE(x)

v = FRE(x$)

จุดประสงค์ ส่งค่าจำนวนของไบท์ต่าง ๆ ในหน่วยความจำที่ยังไม่ถูกใช้โดย GWBASIC

รายละเอียด - x และ x$ เป็นดัมมีอาร์กิวเมนต์ (dummy arguments)

- FRE เป็นเหตุให้เกิดการทำ housecleaning ก่อนที่จะส่งค่าจำนวนของไบท์ต่าง ๆ ที่ว่าง (free bytes) ถึงแม้ว่าโดยปกติ GWBASIC กระทำ housecleaning เมื่อมันทำการวิ่ง (running) ออกนอกพื้นที่ทำงาน (work area) ที่สามารถใช้ได้ก็ตาม ท่านอาจต้องการที่จะใช้ FRE(x$) เป็นครั้งคราวเพื่อที่จะประหยัดเวลาที่มันจะต้องปฏิบัติเพื่อที่จะทำ "clean house" นั่นก็คือ เพื่อที่จะปลดปล่อยพื้นที่ที่ไม่ได้ถูกใช้ (unused area) ให้เป็นอิสระอีกครั้งหนึ่งและเพื่อที่จะรวบรวมและ compress ทุก ๆ พื้นที่ที่เป็นประโยชน์

- เนื่องจาก CLEAR ,n กำหนดจำนวนสูงสุดของไบท์ต่าง ๆ สำหรับพื้นที่ทำงานของ GWBASIC มูลค่าที่ถูกส่งโดย FRE จะเป็น 2.5K ถึง 4K (ขนาดของ interpreter work area ที่ถูกสงวนไว้) ซึ่งมีค่าน้อยกว่าจำนวนของไบต์ต่าง ๆ ที่ถูกกำหนดโดย CLEAR

ตัวอย่าง
```
PRINT FRE(0)

 14542

Ok
```

มูลค่าที่ถูกส่งไปบนคอมพิวเตอร์ที่ท่านใช้จะแปรเปลี่ยนไปตาม install option

GET Statement (Files)

รูปแบบ GET [#]filenum [,number]

จุดประสงค์ อ่านเร็คคอร์ดจาก random file ไปยัง random buffer

รายละเอียด

- filenum เป็นตัวเลขภายใต้ไฟล์ที่ถูกเปิด
- number เป็นหมายเลขเร็คคอร์ดที่จะถูกอ่าน มีค่าในช่วง 1 ถึง 32767 ถ้า number ถูกละ แล้วเร็คคอร์ดถัดไป (หลังจาก GET ล่าสุด) จะถูกอ่านไปสู่บัฟเฟอร์
- หลังจากที่คำสั่ง GET ถูกประมวลผล INPUT # และ LINE INPUT # อาจจะถูกใช้เพื่อที่จะอ่านอักขระต่าง ๆ จากบัฟเฟอร์ของ random file สำหรับข่าวสารเพิ่มเติม ให้ดูที่ภาคผนวก ก, Sequential and Random Files
- คำสั่ง GET และคำสั่ง PUT ยอมรับ (allow) ข้อมูลเข้าและข้อมูลออกที่มีความยาวที่ถูกกำหนดคงที่แน่นอน (fixed-length input and output) สำหรับ COM files

ตัวอย่าง

```
50 CLS
60 INPUT "Name of the file to copy: ",IN.FILE$
70 INPUT "Name of the output file: ",OUT.FILE$
80 OPEN "R",#1,IN.FILE$,128
90 FIELD #1, 128 AS IN.DATA$
100 ' Check to see if the file exists -- if it does
      then stop all action
110 OPEN "I",#2,OUT.FILE$
120 CLOSE #2
130 PRINT "File exists.  cannot  copy"
```

```
140 GOT0  330
150 ' Print  a  message  for  the  user
160 PRINT  "Copying  file"
170 OPEN "R",#2,OUT.FILE$,128
180 FIELD  #2, 128 AS  OUT.DATA$
190 ' Read  from  file  #1  and  copy  the  information  to  file  #2
200 GET  #1
210 ' Check  to  see  if  we  are  at  the  end  of  the  file
220 '  The  error  handler
230 IF  EOF(1) = -1  THEN  GOT0  280
240 LSET  OUT.DATA$  =  IN.DATA$
250 PUT  #2
260 ' and  do  it  again!
270 GOT0  200
280  CLOSE
290  PRINT  "Copy  complete"
300  END
310 IF  ERR = 53  AND  ERL = 110  THEN  CLOSE  #2 : RESUME  160
320 PRINT : PRINT "Error";ERR;"at line";ERL
330 END
```

ตัวอย่างนี้คัดลอกไฟล์ผ่านการใช้ของคำสั่ง GET และคำสั่ง PUT

GET Statement (Graphics)

รูปแบบ GET (x1,y1)-(x2,y2),arrayname

จุดประสงค์ - (x1,y1)-(x2,y2) เป็นรูปสี่เหลี่ยม (rectangle) บนจอภาพการพิมพ์ (display screen) Rectangle ถูกบ่งบอกเช่นเดียวกันกับที่รูปสี่เหลี่ยมถูกวาดโดยคำสั่ง LINE โดยการใช้ ,b option

- arrayname เป็นชื่อที่ถูกกำหนดไปยังสถานที่ที่ซึ่งจะยึดครอง (hold) ภาพ (image) อะเรย์สามารถมีชนิดเป็นชนิดใด ๆ ยกเว้นชนิด string มันต้องถูกกำหนดมิติที่ใหญ่เพียงพอที่จะยึดครองภาพทั้งหมด (entire image) แม้ว่าอะเรย์จะมีชนิดเป็น integer ก็ตาม รายละเอียด (contents) ต่าง ๆ ของอะเรย์หลังจาก GET จะไม่มีความหมายเมื่อถูกแปลโดยตรง
- คำสั่ง GET เคลื่อนย้ายภาพของจอภาพ (screen image) ที่ถูกจำกัดขอบเขตโดยรูปสี่เหลี่ยม (rectangle) และถูกอธิบายโดยจุดต่าง ๆ ที่ถูกระบุไปสู่อะเรย์
- คำสั่ง PUT เคลื่อนย้ายภาพที่ถูกเก็บในอะเรย์ไปบนจอภาพ
- จำนวนไบท์ต่อสมาชิกของอะเรย์จะเป็นดังนี้ :

  2 สำหรับ integer

  4 สำหรับ single precision

  8 สำหรับ double precision

- ตัวอย่างเช่น : สมมติว่าท่านต้องการ GET ภาพขนาด 10x12 ไปในอะเรย์ชนิด integer จำนวนของไบท์ต่าง ๆ ที่ถูกต้องการจะเป็น 4+INT((10*2+7)/8)*12 หรือ 40 ไบท์ ดังนั้นท่านจะต้องการอะเรย์ชนิด integer ที่ประกอบด้วยสมาชิกอย่างน้อยที่สุด 20 สมาชิก
- เราอาจจะตรวจสอบมิติของ x และมิติของ y และแม้แต่ตัวข้อมูลเองถ้าอะเรย์ชนิด integer ถูกใช้ มิติของ x อยู่ในสมาชิกตัวที่ศูนย์ของอะเรย์และมิติของ y อยู่ในสมาชิกตัวที่หนึ่ง อย่าลืมว่า integers ถูกเก็บโดยไบท์ต่ำ (low byte) เป็นครั้งแรก จากนั้นเป็นไบท์สูง (high byte) แต่ข้อมูลถูก

เคลื่อนย้ายจากไบท์สูงก่อน (left most)n และจากนั้นก็เป็นไบท์ต่ำ

GOSUB and RETURN Statements

รูปแบบ GOSUB line

RETURN line

รายละเอียด
- line เป็นหมายเลขของไลน์แรกของสับรูทีน
- สับรูทีนอาจจะถูกเรียกกี่ครั้งก็ได้ในโปรแกรมและสับรูทีนอาจจะถูกเรียกจากภายในสับรูทีนอื่น แต่ละสับรูทีนต่าง ๆ ที่ซ้อนกันถูกจำกัดขอบเขตเท่าที่หน่วยความจำจะสามารถเป็นไปได้
- คำสั่ง RETURN เป็นเหตุให้ GWBASIC ย้ายกลับไปยังคำสั่งต่อจากคำสั่ง GOSUB ที่เพิ่งผ่านมา (most recent GOSUB) สับรูทีนอาจจะประกอบด้วยคำสั่ง RETURN มากกว่า 1 คำสั่ง ควรจะบ่งบอกเงื่อนไขทางตรรกของการส่งค่าที่จุดต่าง ๆ ที่แตกต่างกันในสับรูทีน
- line อาจจะถูกประกอบรวมอยู่ด้วยในคำสั่ง RETURN เพื่อที่จะส่งค่ากลับไปยังไลน์นัมเบอร์ที่ระบุเฉพาะจากสับรูทีน อย่างไรก็ตามให้ใช้รูปแบบ (type) นี้ของ RETURN ด้วยความระมัดระวัง เนื่องจาก GOSUBs WHILES หรือ FORs อื่นใดซึ่งถูกปฏิบัติการในขณะเดียวกันกับที่ GOSUB จะยังปฏิบัติการอยู่ และความผิดพลาดต่าง ๆ อย่างเช่น "FOR without NEXT" อาจจะเป็นผลลัพธ์ที่ได้
- สับรูทีนต่าง ๆ อาจปรากฏ ณ ที่ใด ๆ ในโปรแกรม แต่ขอแนะนำว่าก่อนใช้สับรูทีนควรชี้ให้เห็นความแตกต่างอย่างง่าย ๆ จากโปรแกรมหลัก (main program)

เพื่อที่จะป้องกันรายการเข้า(entry) ที่ไม่ได้เจตนาแต่เกิดเข้าไปสู่สับรูทีน โดยการนำหน้าด้วยคำสั่ง STOP, END หรือคำสั่ง GOTO ซึ่งบ่งชี้การควบคุมไปรอบ ๆ สับรูทีน

ตัวอย่าง

```
10 GOSUB 40
20 PRINT "BACK FROM SUBROUTINE"
30 END
40  PRINT  "SUBROUTINE";
50  PRINT " IN";
60 PRINT " PROGRESS"
70 RETURN
RUN
SUBROUTINE IN PROGRESS
BACK FROM SUBROUTINE
Ok
```

ตัวอย่างนี้แสดงวิธีที่สับรูทีนทำงาน GOSUB โดยที่ไลน์ 10 เรียกสับรูทีนในไลน์ 40 ดังนั้นโปรแกรมจะย้ายไปที่ไลน์ 40 และเริ่มต้นทำการประมวลผลคำสั่งต่าง ๆ ที่นั่นจนกระทั่งมันพบคำสั่ง RETURN ในไลน์ 70 ณ จุดนั้นโปรแกรมจะวกกลับไปยังคำสั่งข้างหลังการเรียกสับรูทีน (subroutine call) ผลก็คือคือจะส่งไปยังไลน์ 20 คำสั่ง END ในไลน์ 30 นั้นใช้เพื่อป้องกันสับรูทีนจากการที่จะถูกปฏิบัติการอีกเป็นครั้งที่สอง

GOTO Statement

รูปแบบ GOT0 line

จุดประสงค์ ย้ายอย่างไม่มีเงื่อนไขออกนอกลำดับปกติของโปรแกรมไปยังไลน์นัมเบอร์ที่ถูกระบุ

รายละเอียด
- line เป็นหมายเลขของไลน์ในโปรแกรม
- ถ้า line เป็นหมายเลขของคำสั่งปฏิบัติการ (executable statement) คำสั่งนั้นและคำสั่งอื่นที่อยู่ต่อจากนั้นจะถูกประมวลผล ถ้า line อ้างอิงไปยังคำสั่งไม่ปฏิบัติการ (nonexecutable statement) อย่างเช่นคำสั่ง REM หรือคำสั่ง DATA แล้วโปรแกรมจะดำเนินการต่อไปที่คำสั่งปฏิบัติการคำสั่งแรกที่ถูกพบหลังจาก line
- คำสั่ง GOTO สามารถถูกใช้ในรูปแบบตรงเพื่อที่จะป้อนโปรแกรมเข้าไปใหม่ ณ จุดที่ถูกต้องการ สิ่งนี้สามารถใช้เป็นประโยชน์ในการปรับปรุงแก้ไข (debugging)
- ใช้ ON...GOTO เพื่อที่จะย้ายเคอร์เซอร์ไปยังไลน์ต่าง ๆ ที่แตกต่างกันซึ่งขึ้นอยู่กับผลลัพธ์ของนิพจน์

ตัวอย่าง

```
5 DATA 5,7,10
10 READ R
20 PRINT "R =";R,
30 A=3.14*R^2
40 PRINT "AREA =";A
50 GOT0 5
RUN
R = 5          AREA = 78.5
R = 7          AREA = 153.86
R = 12         AREA = 452.16
Out of data in 10
```

ok

คำสั่ง GOTO ในไลน์ 50 นำโปรแกรมไปสู่ลูปที่ไม่สิ้นสุด (infinite loop) ซึ่งถูกทำให้หยุดลงเมื่อโปรแกรมวิ่งออกนอกข้อมูลในคำสั่ง DATA (ไม่มีข้อมูลให้อ่าน)

GWBASIC Command

รูปแบบ GWBASIC [filespec] [/F:files] [/S: bsizel
[/C: combuffer] [/M: max workspace]

จุดประสงค์ ทำให้ GWBASIC วิ่ง (run) ภายใต้เงื่อนไขต่าง ๆ ที่ถูกระบุจาก ระดับคำสั่งของ DOS

รายละเอียด
- filespec เป็นชื่อของโปรแกรมที่จะถูกเรียก (load) และถูกประมวลผล ถ้าไม่มีส่วนขยาย (extension) ถูกจัดจ่ายให้และความยาวของชื่อไฟล์มีความยาวน้อยกว่า 8 อักขระแล้ว .BAS จะถูกใช้ในฐานะเป็นส่วนขยายที่กำหนดไว้ก่อนหน้า (default extension) ข้อเลือก (option) นี้เป็นเหตุให้กระทำการกระทำเดียวกันกับ RUN filespec
- /F:files กำหนดจำนวนไฟล์ต่าง ๆ ซึ่งอาจจะถูกเปิด ณ เวลาใดเวลาหนึ่งในระหว่างการประมวลผลของโปรแกรม GWBASIC ไฟล์แต่ละไฟล์ต้องการหน่วยความจำ 188 ไบท์สำหรับบล็อคควบคุม (control block) บวกกับจำนวนของไบท์ต่าง ๆ ที่ถูกกำหนดด้วย /S: option ถ้า /F: option ถูกละ หมายเลขที่กำหนดไว้ล่วงหน้า (default) คือ 3 จะถูกใช้ มูลค่าสูงสุดที่อาจจะถูกใช้จะเป็น 15
- /S:bsize กำหนดขนาดของบัฟเฟอร์สำหรับใช้กับ random files พารา-มิเตอร์สำหรับความยาวของเร็คคอร์ดอาจจะมีค่าไม่เกินกว่ามูลค่านี้ ขนาดของบัฟเฟอร์ที่ถูกกำหนดไว้ล่วงหน้าคือ 128 ไบท์ มูลค่าสูงสุดซึ่งอาจจะถูกป้อนเข้า

ไปคือ 32767 การใช้ /S:512 ถูกเสนอแนะสำหรับการกระทำที่ถูกปรับปรุงกับ random files เนื่องจากคำสั่งนี้สอดคล้องกับขนาดของเซกเตอร์ทางกายภาพ (physical sector size) บนฟล็อปปีดิสค์

- /C:combuff ใช้ร่วมกับ asynchronous communication adaptor และกำหนดขนาดของบัฟเฟอร์สำหรับข้อมูลที่จะได้รับ (บัฟเฟอร์สำหรับการส่งถ่ายข้อมูลโดยปกติกำหนดเป็น 128 ไบท์) มูลค่าสูงสุดซึ่งอาจจะถูกป้อนเข้าไปสำหรับข้อเลือกนี้คือ 32767 ถ้าข้อเลือกนี้ถูกละ receive buffer จะกำหนดเป็น 256 ไบท์ สำหรับ high-speed line มูลค่าที่ถูกเสนอแนะคือ /C:1024 ถ้าท่านมี 2 asynchronous communications adaptors ข้อเลือกนี้จะกำหนดทั้งสอง recieve buffers การป้อน /C:0 ทำให้ RS232 support ไร้ความสามารถ (disable)

- /M:max workspace กำหนดจำนวนสูงสุดของไบท์ซึ่งอาจจะถูกใช้ในฐานะเป็นเนื้อที่ทำงานของ GWBASIC เนื่องจาก GWBASIC ใช้หน่วยความจำสูงสุด 64K ไบท์เท่านั้น นั่นคือมูลค่าสูงสุดซึ่งอาจจะกำหนด (hex FFFF) ข้อเลือกนี้สามารถถูกใช้เพื่อที่จะสงวน (reserve) เนื้อที่ว่าง (space) สำหรับสับรูทีนภาษาเครื่องต่าง ๆ หรือสำหรับเนื้อที่หน่วยความจำของข้อมูลที่พิเศษเฉพาะ (special data storage) ถ้าข้อเลือกนี้ถูกละ หน่วยความจำทุก ๆ หน่วยสามารถใช้ได้ (สูงสุดถึง 64K)

- files, bsize, combuffer และ max workspace อาจจะเป็นเลขฐานสิบ เลขฐานแปด (ถูกนำหน้าโดยอักษร O) หรือเป็นเลขฐานสิบหก (ถูกนำหน้าโดย &H)

- ตัวอย่างต่าง ๆ บางตัวอย่างข้างล่างนี้เป็น GWBASIC commands :

```
GWBASIC PAYROLL.ABC
```

GWBASIC จะใช้หน่วยความจำทุก ๆ หน่วย ใช้ไฟล์ได้สูงสุดถึง 3 ไฟล์ใน

ขณะใดขณะหนึ่งและโปรแกรม PAYROLL.ABC จะถูกเรียกและถูกประมวลผล

GWBASIC INVENT/F:6

GWBASIC จะใช้หน่วยความจำทุก ๆ หน่วย ใช้ไฟล์ได้สูงสุดถึง 6 ไฟล์และโปรแกรม INVENT.BAS จะถูกเรียกและถูกประมวลผล

GWBASIC/M:32768

ขนาดสูงสุดของเนื้อที่ทำงานกำหนดเป็น 32768 หรือ 32K ของหน่วยความจำและไฟล์ไม่มากกว่า 3 ไฟล์จะถูกใช้ในแต่ละครั้ง ไม่มีโปรแกรมถูกวิ่งและข้อความบ่งเตือน (prompt) "Ok" จะปรากฏ

GWBASIC TTY/C:512

GWBASIC ใช้หน่วยความจำทุก ๆ หน่วยและ 3 ไฟล์ RS232 receive buffer ถูกจัดสรร 512 ไบท์และ transmit buffer ถูกจัดสรร 128 ไบท์ โปรแกรม TTY.BAS จะถูกเรียกและถูกประมวลผล

HEX$ Function

รูปแบบ	HEX$(x)
จุดประสงค์	ส่งค่าสตริงก์ซึ่งแทนมูลค่าเลขฐานสิบหกของอาร์กิวเมนต์ฐานสิบ
รายละเอียด	- x เป็นจำนวนเต็มอยู่ในช่วง -32768 ถึง 65535
	- ดูฟังก์ชัน OCT$ สำหรับการแปลงค่าของเลขฐานแปด
ตัวอย่าง	10 PRINT HEX$(27)
	RUN
	1B

```
Ok
```

มูลค่าของ 27 ในฐานสิบคือ 1B ในฐานสิบหก

IF Statement

รูปแบบ IF expression [,] THEN clause [ELSE clause]

IF expression [,] GOTO line [[,] ELSE clause]

จุดประสงค์ ทำการตัดสินใจเกี่ยวกับทางเดินของโปรแกรม (program flow) ซึ่งขึ้นอยู่กับผลลัพธ์ของนิพจน์

รายละเอียด
- expression อาจเป็นนิพจน์จำนวนเลขใด ๆ
- clause อาจเป็นคำสั่งของ GWBASIC ชุดลำดับ (sequence) ของคำสั่งต่าง ๆ (ถูกแบ่งแยกโดยเครื่องหมายจุลภาค) หรืออาจเป็นหมายเลขของไลน์ที่จะเคลื่อนย้ายไป
- line เป็นหมายเลขของไลน์ที่มีอยู่ในโปรแกรม
- ถ้าผลลัพธ์ของนิพจน์มีค่าไม่เป็นศูนย์แล้วอนุประโยค THEN หรือ GOTO จะถูกประมวลผล THEN อาจจะถูกติดตามด้วยไลน์นัมเบอร์สำหรับการเคลื่อนย้าย หรือไม่ก็ถูกติดตามด้วยคำสั่งที่จะถูกประมวลผลตั้งแต่ 1 คำสั่งขึ้นไป GOTO โดยปกติถูกติดตามด้วยไลน์นัมเบอร์
- ถ้าผลลัพธ์ของนิพจน์เป็นศูนย์แล้วอนุประโยค THEN, หรือ GOTO จะไม่ถูกรับรู้ และอนุประโยค ELSE ถ้ามีอยู่จะถูกประมวลผล การประมวลผลดำเนินต่อไปด้วยคำสั่งปฏิบัติการถัดไป
- การซ้อนกันของคำสั่ง IF หลาย ๆ คำสั่ง : คำสั่ง IF...THEN...ELSE หลาย ๆ คำสั่งอาจจะถูกซ้อนกันเป็นโครงข่าย การซ้อนกันถูกจำกัดขอบเขตโดยความยาวของไลน์เท่านั้น ตัวอย่างเช่น

```
IF X>Y THEN PRINT "GREATER" ELSE IF Y>X THEN
PRINT "LESS THAN" ELSE PRINT "EQUAL"
```

เป็นคำสั่งที่ใช้ได้ ถ้าคำสั่งไม่ได้ประกอบด้วยจำนวนเดียวกันของอนุประโยค ELSE และ THEN แล้วแต่ละ ELSE จะจับคู่กันกับ THEN ที่ไม่เข้าคู่กันที่อยู่ใกล้กันที่สุด ตัวอย่างเช่น

```
IF A=B THEN IF B=C THEN PRINT "A=C"
ELSE PRINT "A<>C"
```

คำสั่งนี้จะไม่พิมพ์ "A<>C" เมื่อ A<>B

- ถ้าท่านป้อนคำสั่ง IF...THEN ในรูปแบบตรงและมันชี้การควบคุมไปยังไลน์-นัมเบอร์แล้วจะเกิด "Undefined line number" error เป็นผลลัพธ์ขึ้น แม้ว่าท่านจะได้ป้อนไลน์ที่ประกอบด้วยหมายเลขที่ถูกระบุในรูปแบบตรงเข้าไปแล้วก็ตาม
- หมายเหตุ : เมื่อทำการใช้ IF เพื่อที่จะตรวจสอบความเท่ากันของมูลค่าซึ่งเป็นผลลัพธ์ของการคำนวณในรูป single precision หรือในรูป double precision อย่าลืมว่าการแทนภายในของมูลค่าอาจจะไม่ถูกต้องแน่นอน นี่เป็นเพราะว่ามูลค่าในรูปของ single precision หรือ double precision ถูกเก็บภายในในรูปแบบของ floating point binary format ดังนั้นการตรวจสอบควรจะต่อกับช่วงที่อยู่เหนือช่วงของความถูกต้องของมูลค่าที่อาจแปรเปลี่ยนไป ตัวอย่างเช่น เพื่อที่จะตรวจสอบ computed variable A กับมูลค่า 1.0 ใช้

```
IF ABS(A-1.0) < 1.0E-6 THEN ...
```

การตรวจสอบนี้ส่งผลลัพธ์ค่าจริง (true result) ถ้ามูลค่าของ A เป็น 1.0 ด้วย relative error น้อยกว่า 1.0E-6

ตัวอย่างที่ 1 200 IF I THEN GET #1,I

คำสั่งนี้อ่านเร็คคอร์ดที่ I ถ้า I ไม่มีค่าเป็นศูนย์

ตัวอย่างที่ 2 100 IF (I>10) and (I<20) THEN DB=1979-1: GOT0 300

ELSE PRINT "OUT OF RANGE"

ถ้า I มีค่าอยู่ระหว่าง 10 และ 20 DB จะถูกคำนวณค่าและการประมวลผลจะย้ายไปยังไลน์ 300 ถ้า I ไม่มีค่าอยู่ในช่วงนี้ ข้อความ "OUT OF RANGE" จะถูกพิมพ์ สังเกตการใช้คำสั่ง 2 คำสั่งในอนุประโยค THEN

ตัวอย่างที่ 3 210 IF IOFLAG THEN PRINT A$ ELSE LPRINT A$

คำสั่งนี้เป็นเหตุให้ผลลัพธ์จะถูกพิมพ์ไปยังจอภาพหรือไปยังเครื่องพิมพ์ขึ้นอยู่กับมูลค่าของตัวแปร IOFLAG ถ้า IOFLAG เป็นศูนย์แล้วผลลัพธ์จะไปที่เครื่องพิมพ์มิฉะนั้นแล้วผลลัพธ์จะไปที่จอภาพ

INKEY$ Varibles

รูปแบบ INKEY$

จุดประสงค์ อ่านอักขระจากคีย์บอร์ด

รายละเอียด - INKEY$ ส่ง actual character ที่ถูกได้รับจากเทอร์มินอลหรือส่งสตริงก์ว่างเปล่า (null string) ถ้าไม่มีสตริงก์อักขระหรือสตริงก์ของ 2 อักขระซึ่งบ่งชี้ extended character code ข่าวสารเกี่ยวกับ extended character codes ถูกกำหนดในภาคผนวก ค

- มีรหัสจากคีย์บอร์ดอยู่ 4 รหัสซึ่งมีหน้าที่พิเศษเฉพาะและไม่สามารถถูกดัก (trapped) โดยฟังก์ชัน INKEY$ รหัสทั้งสี่คือ :

Ctrl-Break : หยุดการประมวลผลโปรแกรม

Ctrl-Num Lock : หยุดการประมวลผลโปรแกรมชั่วขณะ

Ctrl-Alt-Del : กำหนดระบบใหม่

Ptrsc : พิมพ์ภาพของจอภาพที่เป็นอยู่ในขณะนั้นไปที่เครื่องพิมพ์

ตัวอย่าง

```
50 CLS
60 LOCATE 10,10
70 PRINT "Press RETURN to continue ";
80 LOCATE 10,30
90 RET$ = INKEY$
100 IF RET$ = "" THEN GOTO 80
110 IF ASC(RET$) <> 13 THEN GOTO 80
120 END
```

ตัวอย่างนี้ใช้ฟังก์ชัน INKEY$ เพื่อที่จะสอบถามคีย์บอร์ดจนกว่าคีย์ RETURN ซึ่งมีมูลค่าแอสกี (ASCII value) เป็น 13 จะถูกกด

INP Function

รูปแบบ `V = INP (i)`

จุดประสงค์ ส่งค่าไบท์ที่ถูกอ่านจาก port address i

รายละเอียด
- i ต้องอยู่ในช่วง 0 ถึง 65535
- ตารางที่ 7.5 คือ port address map
- จากตาราง เครื่องหมาย * หมายถึง not in decode และเครื่องหมาย ** หมายถึงขณะที่ power on, nonmaskable, NMI, ไปยัง 8088 ถูกทำให้ไร้ความสามารถ (disabled) โดยฮาร์ดแวร์ภายนอก (external hardware) สิ่งนี้สามารถกำหนดและกำหนดใหม่ได้ดังต่อไปนี้

เพื่อที่จะกำหนด mask : เขียนบันทึก 80H ไปที่ I/O address OAOH

เพื่อที่จะลบ mask : เขียนบันทึก OOH ไปที่ I/O address OAOH

- INP เป็นฟังก์ชันเติมเต็ม (complementary function) ของคำสั่ง OUT (ดูคำสั่ง OUT)

ตัวอย่าง 100 A=INP(4255)

ไบท์ถูกอ่านจาก port address 4255 และถูกกำหนดค่าไปที่ตัวแปร A

ตารางที่ 7.5

PORT ADDRESS MAP

Hex Range	Address Bit 9	8	7	6	5	4	3	2	1	0	Device
00-OF	0	0	0	0	0	*	A3	A2	Al	A0	DMA Chip 8237-2
20-21	0	0	0	0	1	*	*	*	*	A0	Interrupt 8259A
40-43	0	0	0	1	0	*	*	*	Al	A0	Timer 8253-5
60-63	0	0	0	1	1	*	*	*	Al	A0	PPI 8255-5
80-83	0	0	1	0	0	*	*	*	Al	A0	DMA Page Registers
AX**	0	0	1	0	1						NMI Mask Registers
cx	0	0	1	1	0						Reserved
EX	0	0	1	1	1						Reserved
200-20F	1	0	0	0	0	0	A3	A2	Al	A0	Game I/O Adapter
278-27F	1	0	0	1	1	1	1	*	Al	A0	Reserved

ตารางที่ 7.5 (ต่อ)

Hex Range	Address Bit 8 8 7 8 5 4 3	2	1 0	Device
2F8-2FF	1 0 1 1 1 1 1	A2	Al A0	Reserved
300-307	1 1 0 0 0 0 0	A2	Al A0	Hard Disk I/O Ports
378-37F	1 1 0 1 1 1 1	*	Al A0	Auxiliary Parallel Printer Port
3B0-3BF	1 1 1 0 1 1 A3	A2	Al A0	Built-in Parallel Printer Port and Monochrome Display
3D0-3DF	1 1 1 1 0 1 A3	A2	Al A0	Color Graphics Adapter
3F0-3F7	1 1 1 1 1 1 0	A2	Al A0	5 1/4" Drive
3F8-3FF	1 1 1 1 1 1 1	'A2	Al A0	TP RS-232C CD (Serail Port)

*Not in decode

INPUT Statement

รูปแบบ INPUT[;] ["prompt";] variable[,variable]...

จุดประสงค์ รับข้อมูลเข้า (input) จากคีย์บอร์ดในระหว่างการประมวลผลโปรแกรม

รายละเอียด - "prompt" เป็นค่าคงที่สตริงก์ (string constant) ซึ่งจะถูกใช้เพื่อที่จะบ่งเตือน (prompt) สำหรับข้อมูลเข้าที่ถูกต้องการ

- variable เป็นชื่อของตัวแปรจำนวนเลขหรือตัวแปรสตริงก์หรือชื่อของสมาชิกของอะเรย์ซึ่งจะรับข้อมูลเข้า
- เมื่อโปรแกรมเห็นคำสั่ง INPUT มันจะหยุดชั่วขณะและพิมพ์เครื่องหมายคำถามเพื่อที่จะบ่งชี้ว่ามันกำลังคอยข้อมูลอยู่ ถ้า "prompt" ถูกประกอบรวมอยู่ด้วยมันจะถูกพิมพ์ก่อนเครื่องหมายคำถาม จากนั้นท่านอาจจะป้อนข้อมูลที่ถูกต้องการจากคีย์บอร์ด
- ท่านอาจใช้เครื่องหมายจุลภาคแทนเครื่องหมายเซมิโคลอนหลังจาก prompt string เพื่อที่จะกำจัดเครื่องหมายคำถามออกไป ตัวอย่างเช่น คำสั่ง INPUT "ENTER BIRTHDATE",B$ จะพิมพ์ข้อความบ่งเตือน (prompt) โดยไม่มีเครื่องหมายคำถาม
- ถ้า INPUT ถูกติดตามทันทีด้วยเครื่องหมายเซมิโคลอนและ Enter เครื่องหมายเคอร์เซอร์จะยังคงอยู่บนไลน์เดียวกันในฐานะเป็นผลตอบสนองของท่าน
- ข้อมูลที่ท่านป้อนเข้าไปถูกกำหนดค่าไปยังตัวแปรหรือตัวแปรต่าง ๆ ที่ถูกกำหนดให้ในรายการของตัวแปร data items ที่ถูกจัดจ่ายต้องถูกแบ่งแยกโดยเครื่องหมายจุลภาคและจำนวนของ data items ต้องเท่ากันกับจำนวนของตัวแปรต่าง ๆ ในรายการ
- ชนิดของ data item แต่ละตัวที่ถูกป้อนเข้าไปต้องสอดคล้องกับชนิดที่ถูกระบุโดยชื่อของตัวแปร (สตริงก์ต่าง ๆ ที่ถูกป้อนเข้าไปเพื่อตอบสนองต่อคำสั่ง INPUT ไม่จำเป็นต้องถูกคลุมอยู่ภายในเครื่องหมายคำพูดแม้ว่ามันทั้งหลายจะบรรจุเครื่องหมายจุลภาคหรือที่ว่างข้างหน้าหรือที่ว่างข้างท้ายที่มีนัยสำคัญ (significant leading or trailing blanks) ก็ตาม)
- ถ้าท่านตอบสนองต่อ INPUT ด้วย items ที่มากเกินไปหรือน้อยเกินไปหรือด้วยชนิดของมูลค่าที่ไม่ถูกต้องตามประเภท (อักษรต่าง ๆ แทนตัวเลขต่าง ๆ เป็นต้น) แล้ว GWBASIC จะพิมพ์ข้อความ "?Redo from start" และไม่

กำหนดมูลค่าข้อมูลใด ๆ ไปยังตัวแปรต่าง ๆ จนกว่าท่านจะให้ผลตอบสนองที่สามารถยอมรับได้

ตัวอย่าง

```
10 INPUT X
20 PRINT  X "SQUARED IS" X^2
30 END
RUN
?
```

เครื่องหมายคำถามที่ถูกพิมพ์เป็นข้อความบ่งเตือน (prompt) ที่บ่งบอกให้ท่านป้อนบางสิ่งบางอย่างเข้าไป ถ้าท่านป้อน 5 เข้าไปแล้วตามด้วย Return โปรแกรมจะดำเนินการต่อไป :.

```
.
.
.
? 5
 5 SQUARED IS 25
Ok
```

```
10 PI=3.14
20 INPUT "WHAT IS THE RADIUS";R
30 A=PI*R^2
40 PRINT "THE AREA OF THE CIRCLE IS";A
50 END
RUN
 WHAT IS THE RADIUS?
```

ข้อความบ่งเตือนถูกประกอบรวมอยู่ในไลน์ 20 ดังนั้นในขณะนี้คอมพิวเตอร์จะบ่งเตือนด้วย "WHAT IS THE RADIUS?" สมมติว่าท่านตอบสนองด้วย 7.4 โปรแกรมจะดำเนินการต่อไป :

.

.

```
WHAT IS THE RADIUS? 7.4
 THE AREA QF THE CIRCLE IS 171.9464
Ok
```

INPUT$ Function

รูปแบบ INPUT$(n[,[#]filenum])

จุดประสงค์ อ่านข้อมูลจากแป้นพิมพ์หรือจากไฟล์หมายเลข f i lenum

รายละเอียด

- n เป็นจำนวนของอักขระต่าง ๆ ที่จะถูกอ่านจากไฟล์
- filenum เป็นหมายเลขของไฟล์ที่ถูกใช้บนคำสั่ง OPEN ถ้า filenum ถูกละแล้วเครื่องจะรับข้อมูลที่แป้นพิมพ์
- ถ้าแป้นพิมพ์ถูกใช้สำหรับป้อนข้อมูลเข้าแล้วจะไม่มีอักขระใด ๆ ถูกพิมพ์ปรากฏบนจอภาพ อักขระทุก ๆ ตัว (รวมทั้งอักขระควบคุมต่าง ๆ) ถูกส่งค่าผ่านไปตลอด ยกเว้น Ctrl-Break ซึ่งถูกใช้เพื่อที่จะหยุดการประมวลผลของฟังก์ชัน INPUT$
- เมื่อกำลังทำงานกับ communication files เรามักจะเลือกใช้คำสั่ง INPUT$ มากกว่าคำสั่งข้อมูลเข้า (input statements) ประเภทอื่น ๆ ทั้งนี้เนื่องจากเราจะสามารถส่งผ่านอักขระทุก ๆ ตัวได้ยกเว้น Ctrl-Break

ตัวอย่างที่ 1

```
40 CLS
50 INPUT "File to dump: ";F$
60 HX$ = APACE$(2)
70 OPEN "1",#1,F$
80 IF EOF(l) THEN GOTO 130
90 CHAR$ = INPUT$(1,#1)
100 LSET HX$ = RIGHT$("0"+HEX$(ASC(CHAR$)),2)
110 PRINT HX$;" ";
120 GOT0 80
130 CLOSE
140 END
```

โปรแกรมนี้ใช้ INPUT$ เพื่อที่จะพิมพ์ HEX dump ของไฟล์

INPUT% Statement

รูปแบบ INPUT#filenum, variable [,variable]...

จุดประสงค์ อ่าน data items จาก sequential device หรือจาก sequential file และกำหนดค่าไปยังตัวแปรต่าง ๆ ของโปรแกรม

รายละเอียด
- filenum เป็นหมายเลขที่ถูกใช้เมื่อไฟล์ถูกเปิด
- variable เป็นชื่อของตัวแปรซึ่งจะมี item ในไฟล์ถูกกำหนดค่าไปยังมัน มันอาจเป็นตัวแปรสตริงก์หรือตัวแปรจำนวนเลขหรือเป็นสมาชิกของอะเรย์
- ชนิดของข้อมูลในไฟล์ต้องสอดคล้องกับชนิดที่ถูกระบุโดยชื่อตัวแปร ไม่เหมือน INPUT คือไม่มีเครื่องหมายคำถามถูกพิมพ์ด้วย INPUT#
- data items ในไฟล์ควรจะปรากฏเท่าที่กำหนดไว้ ถ้าหากข้อมูลจะถูกพิมพ์เพื่อตอบสนองต่อคำสั่ง INPUT แล้วมูลค่าจำนวนเลขต่าง ๆ ช่องว่าง

ต่าง ๆ ที่อยู่ข้างหน้า (leading spaces) carriage returns และ line feeds จะไม่ถูกรับรู้ อักขระตัวแรกที่ถูกป้อนเข้าไปที่ไม่ใช่ space, carriage return หรือ line feed ถูกกำหนดที่จะเป็นจุดเริ่มต้นของตัวเลข ตัวเลขสิ้นสุดลงด้วย space, carriage return, line feed หรือด้วยเครื่องหมายจุลภาค

- ถ้า GWBASIC จะทำการตรวจสอบข้อมูลสำหรับ string items แล้ว leading spaces, carriage returns และ line feeds จะไม่ถูกรับรู้เช่นกัน อักขระตัวแรกที่พบซึ่งไม่ใช่ space, carriage return หรือ line feed จะถูกกำหนดเป็นจุดเริ่มต้นของ string item ถ้าอักขระตัวแรกเป็นเครื่องหมายคำพูดแล้ว string item จะประกอบด้วยอักขระต่าง ๆ ที่ถูกอ่านระหว่างเครื่องหมายคำพูดแรกและเครื่องหมายคำพูดที่สอง ดังนั้น สตริงก์ที่ถูกคลุม (quoted string) อาจจะไม่บรรจุเครื่องหมายคำพูดในฐานะเป็นอักขระตัวหนึ่ง ถ้าอักขระตัวแรกของสตริงก์ไม่เป็นเครื่องหมายคำพูด สตริงก์นั้นจะเรียกว่าเป็น quoted string ซึ่งจะสิ้นสุดลงด้วยเครื่องหมายจุลภาค carriage return หรือ line feed หรือหลังจากที่อักขระ 255 ตัวถูกอ่าน ถ้า end-of-file มาถึงในขณะที่ numeric item หรือ string item จะเป็นข้อมูลเข้า item นั้นจะถูกยกเลิกไป (cancelled)

- INPUT# สามารถถูกใช้กับ random file ด้วย

ตัวอย่าง ดูภาคผนวก ก, Sequential and Random Files

INSTR Function

รูปแบบ INSTR([n,]a$,b$)

จุดประสงค์ ส่งค่าของตำแหน่งของ b$ ใน a$ เริ่มต้นที่ n

รายละเอียด - n เป็นนิพจน์จำนวนเลขมีค่าอยู่ในช่วง 1 ถึง 255

- a$, b$ อาจเป็นตัวแปรสตริงก์ นิพจน์สตริงก์ หรือค่าคงที่สตริงก์

- ถ้า n>LEN(a$) หรือถ้า a$ เป็น null หรือถ้า b$ ไม่สามารถถูกพบ INSTR จะส่งค่าศูนย์ ถ้า b$ เป็น null แล้ว INSTR จะส่งค่า n (หรือ 1 ถ้า n ไม่ได้ถูกระบุ)

- ถ้า n มีค่าออกนอกช่วงแล้ว "Illegal function call" error จะถูกส่งค่าไป

ตัวอย่าง

```
10 A$=ABCDEB"
20 B$="B"
30 PRINT INSTR(A$,B$);INSTR(4,A$,B$)
RUN
 2 6
Ok
```

ตัวอย่างนี้ค้นหาสตริงก์ "B" ภายในสตริงก์ "ABCDEB" เมื่อสตริงก์ถูกค้นหาจากจุดเริ่มต้น "B" ถูกพบที่ตำแหน่งที่ 2 เมื่อการค้นหาเริ่มต้นที่ตำแหน่งที่ 4 "B" ถูกพบที่ตำแหน่งที่ 6

INT Function

รูปแบบ INT(x)

จุดประสงค์ ส่งค่าจำนวนเต็มที่ใหญ่ที่สุดที่มีค่าน้อยกว่าหรือเท่ากับ x

รายละเอียด - x เป็นนิพจน์จำนวนเลขใด ๆ

- ดูฟังก์ชัน FIX และฟังก์ชัน CINT ซึ่งส่งค่าจำนวนเต็มเช่นกัน

ตัวอย่าง

```
PRINT INT(99.89)
 99
```

```
PRINT INT(-12.11)
 -13
Ok
```

ตัวอย่างข้างบนแสดงวิธีที่ INT โดยปกติจะส่งตัวเลขทั้งหมดซึ่งมีค่าน้อยกว่าเลขฐานสิบ

KEY Statement

รูปแบบ KEY n, x$

KEY LIST

KEY ON

KEY OFF

จุดประสงค์ กำหนดมูลค่าต่าง ๆ ของฟังก์ชันคีย์ พิมพ์หรือเปิดหรือปิดการพิมพ์ของ soft keys

รายละเอียด

- n เป็นหมายเลขของฟังก์ชันคีย์ มีค่าอยู่ในช่วง 1 ถึง 10
- x$ เป็นนิพจน์สตริงก์มีความยาว 1 ถึง 15 อักขระซึ่งจะถูกกำหนดค่าไปยังคีย์ อย่าลืมว่าค่าคงที่สตริงก์ต่าง ๆ ต้องถูกคลุมอยู่ภายในเครื่องหมายคำพูด
- คำสั่งนี้อนุญาตให้ท่านกำหนดฟังก์ชันคีย์ต่าง ๆ เพื่อที่จะพิมพ์ชุดลำดับใด ๆ ของอักขระต่าง ๆ อักขระสูงสุดถึง 15 อักขระอาจจะถูกกำหนดค่าไปยังแต่ละ soft key เมื่อคีย์ถูกกด สตริงก์จะเป็นข้อมูลเข้าไปยัง GWBASIC
- มูลค่าเริ่มต้นของฟังก์ชันคีย์ต่าง ๆ ถูกกำหนดให้ในตารางที่ 7.6
- KEY LIST แสดงรายการมูลค่าของคีย์แต่ละคีย์บนจอภาพ อักขระทุก ๆ 15 อักขระของแต่ละมูลค่าจะถูกกำหนดให้
- KEY ON เป็นเหตุให้มูลค่าต่าง ๆ ของคีย์ถูกพิมพ์บนไลน์ที่ 25 เมื่อความกว้างเป็น 40 คีย์ 5 คีย์ใน 10 คีย์จะถูกพิมพ์และเมื่อความกว้างเป็น 80 คีย์ทั้ง 10 คีย์จะถูกพิมพ์ อย่างไรก็ตาม อักขระ 6 อักขระแรกของแต่ละมูลค่าเท่านั้น

ตารางที่ 7.6

FUNCTION KEY VALUES

Key	Value	Key	Value
F1	LIST	F2	RUN + CHR$(13)
F3	LOAD"	F4	SAVE"
F5	CONT + CHR$(13)	F6	,"LPT1:"
F7	TRON + CHR$(13)	F8	TROFF + CHR$(13)
F9	KEY	F10	SCREEN 0,0,0

จะถูกพิมพ์ ON เป็นสถานะที่กำหนดไว้แล้วล่วงหน้า (default state) สำหรับการพิมพ์ soft key

- KEY OFF ลบการพิมพ์คีย์ที่ไลน์ที่ 25 ออกไป ฟังก์ชันคีย์ต่าง ๆ ไม่สามารถถูกเห็นได้ (disabled)
- การกำหนดค่าสตริงก์ความยาวศูนย์ (null string) ไปยัง soft key ไม่สามารถกระทำได้
- เมื่อ soft key ถูกกำหนดค่า INKEY$ จะส่งอักขระ 1 ตัวของ soft key ในแต่ละครั้งที่มันถูกเรียก ถ้า soft key ถูกทำให้ไร้ความสามารถแล้ว INKEY$ จะส่งสตริงก์ขนาด 2 อักขระ อักขระแรกเป็น binary zero และอักขระที่สองจะเป็น key scan code ภาคผนวก ค, ASCII Character Codes แสดงรายการของ code นี้

- ข่าวสารบนไลน์ที่ 25 ของจอภาพจะไม่ถูกเลื่อน (scrolled) ถ้าท่านได้ใช้ KEY OFF แล้วท่านอาจพิมพ์ข่าวสารบนไลน์ที่ 25 โดยการใช้ LOCATE 25,1 ถูกติดตามด้วย PRINT

ตัวอย่าง 10 KEY 1,"MENU"+CHR$(13)

สตริงก์ "MENU"+Enter ถูกกำหนดค่าไปยังคีย์ 1 สิ่งนี้ให้การอ้างอิงอย่างง่ายไปยังการพิมพ์เมนู (menu disply) ในโปรแกรมที่ถูกกำหนดให้

40 KEY 1,""

Soft key ถูกทำให้ไร้ความสามารถ

KEY(n) Statement

รูปแบบ KEY(n) ON

KEY(n) OFF

KEY(n) STOP

จุดประสงค์ Enable และ disable trapping คีย์ที่ถูกระบุ

รายละเอียด - n เป็นตัวเลขมีค่าอยู่ในช่วง 1 ถึง 14 ซึ่งบ่งชี้ที่ซึ่งคีย์ต่าง ๆ ต่อไปนี้จะถูก trapped :

1-10 Function keys F1-F10

11 Cursor Up

12 Cursor Left

13 Cursor Right

14 Cursor Down

- ON KEY(n) ต้องถูกกำหนดด้วยหมายเลขของไลน์เป็นครั้งแรกก่อนที่ KEY(n) จะตรวจสอบว่าคีย์ได้ถูกกดไปแล้ว ถ้าเป็นเช่นนั้น โปรแกรมจะกระทำราวกับ

ว่าคำสั่ง GOSUB ได้ถูกประมวลผลและเคลื่อนย้ายไปยังรูทีนของ trap ณ ไลน์นัมเบอร์ที่ถูกระบุ

- KEY(n) STOP หยุด trapping แต่ถ้าคีย์ถูกกดต่อมา คำสั่งนี้จะเป็นเหตุให้เกิด trap ทันทีถ้า trapping on turned back on
- KEY(n) OFF ไม่ได้หยุด trapping เพียงเท่านั้น แต่ว่าคีย์ที่ถูกกดต่อมาจะไม่สามารถจำมันได้
- คำสั่งนี้ไม่มีผลต่อคำสั่ง INPUT หรือคำสั่ง INKEY$ รูทีนของ trap ที่แตกต่างกันสำหรับคีย์แต่ละคีย์ต้องถูกสร้างขึ้นสำหรับใช้กับคำสั่งต่าง ๆ เหล่านี้
- หมายเหตุ : KEY(n) ON และ KEY ON เป็นคำสั่งที่แตกต่างกันด้วยความหมายต่าง ๆ ที่แตกต่างกันมากมาย ดู KEY ON สำหรับข่าวสารเพิ่มเติม

ตัวอย่าง

```
40 CLS
50 ' Enable function keys 1 to 4
60 FOR 1% =1 TO 4
70     KEY(1%) ON
80 NEXT 1%
90 ' Disable function keys 5 to 10 and the arrow keys (up,
     left, down, right)
100 FOR 1% = 5 TO 14
110     KEY(1%) OFF
120 NEXT 1%
130 ' Determine which subroutine to use if a function key
      is pressed
140 ON KEY(l) GOSUB 240
150 ON KEY(2) GOSUB 250
```

```
160 ON KEY(3) GOSUB 260
170 ON KEY(4)  GOSUB 270
180 ' Print  a  message
190 LOCATE 10,10
200 PRINT "Press F1, F2, F3, or F4";
210 LOCATE 20,20 : PRINT " ";
220  GOT0  210
230 ' Subroutines
240 PRINT "You pressed F1"; : RETURN
250 PRINT "You pressed F2"; : RETURN
260 PRINT "You pressed F3"; : RETURN
270 PRINT "You pressed F4"; : RETURN
```

โปรแกรมนี้สาธิตการใช้ของคำสั่ง KEY ON KEY OFF และคำสั่ง ON KEY(n)

KILL Command

รูปแบบ KILL filespec

จุดประสงค์ ลบไฟล์ออกจากดิสเกตต์

รายละเอียด - filespec เป็นชื่อของไฟล์หรือชื่อของอุปกรณ์ที่ถูกต้อง ชื่ออุปกรณ์ต้องเป็นดิสเกตต์ไดร์ฟ ถ้าชื่ออุปกรณ์ถูกละ DOS default drive จะถูกใช้

- KILL สามารถถูกใช้กับทุก ๆ ไฟล์ของดิสเกตต์ (diskette files) มันคล้ายคลึงกับคำสั่ง ERASE ใน DOS คำสั่งนี้ต้องการการใช้ของส่วนขยายถ้ามันปรากฏอยู่

- &lv&n เปิดเมื่อคำสั่ง KILL ถูกปฏิบัติแล้วจะเกิด "File already open" error ขึ้นเป็นผลลัพธ์

ตัวอย่าง 10 KILL "A: TEST.BAS"

ตัวอย่างนี้จะลบไฟล์ TEST.BAS ออกจากไดร้ฟ์ A

LEFT$ Function

รูปแบบ LEFT$(X$,I)

จุดประสงค์ ส่งค่าสตริงก์ที่ประกอบด้วยอักขระทางด้านซ้ายสุด (leftmost) ของ X$

รายละเอียด - I ต้องมีค่าอยู่ในช่วง 0 ถึง 255 ถ้า I มีค่ามากกว่าจำนวนอักขระต่าง ๆ ใน X$ (LEN(X$)) แล้วทั้งสตริงก์ (X$) จะถูกส่งค่าไป ถ้า I=0 แล้ว สตริงก์ว่างเปล่า (null string มีความยาวเป็นศูนย์) จะถูกส่งค่าไป

- ดูฟังก์ชัน MID$ และฟังก์ชัน RIGHT$ ประกอบด้วย

ตัวอย่าง
```
10 A$="BASIC"
20 B$=LEFT$(A$,3)
30 PRINT B$
RUN
 BAS
Ok
```
อักขระ 3 ตัวแรกของ "BASIC" จะถูกพิมพ์

LEN Function

รูปแบบ LEN(x$)

จุดประสงค์ ส่งค่าจำนวนของอักขระต่าง ๆ ใน x$

รายละเอียด - อักขระต่าง ๆ ที่ไม่สามารถถูกพิมพ์ (nonprinting characters) และช่องว่าง (blanks) ต่าง ๆ จะถูกนับรวมเข้าไปด้วย

ตัวอย่าง
```
10 X$ = "WESTLAKE VILLAGE, CA"
```

```
20 PRINT LEN(X$)
RUN
 20
Ok
```

ในสตริงก์ "WESTLAKE VILLAGE, CA" มีอักขระทั้งหมด 20 ตัวเนื่องจาก เครื่องหมายจุลภาคและช่องว่างต่าง ๆ ถูกนับด้วย

LET Statement

รูปแบบ [LET] variable=expression

จุดประสงค์ กำหนดมูลค่าของนิพจน์ไปยังตัวแปร

รายละเอียด - คำ LET เป็น optional เครื่องหมายเท่ากับเป็นสิ่งเพียงพอเมื่อทำการกำหนดค่านิพจน์ไปยังชื่อตัวแปร

ตัวอย่าง

```
110 LET D=12
120 LET E=12^2
130 LET F=12^4
140 LET SUM=D+E+F
```

หรือ

```
110 D=12
120 E=12^2
130 E=12^4
140 SUM=D+E+F
```

โปรแกรมทั้งสองนี้จะเท่าเทียมกัน

LINE Statement

รูปแบบ LINE [(x1,y1)]-(x2,y2) [,[color][,b[f]]]

จุดประสงค์ วาดเส้นตรง (line) หรือกล่อง (box) บนจอภาพ

รายละเอียด
- คำสั่งนี้สามารถถูกใช้ได้เฉพาะในกราฟฟิคโหมดเท่านั้น
- (x1,y1) เป็นโคออร์ดิเนตสำหรับจุดเริ่มต้นของไลน์
- (x2,y2) เป็นจุดสิ้นสุดของไลน์
- color เป็นหมายเลขของสีในที่ซึ่งไลน์จะถูกวาด ถ้า ,b หรือ ,bf option ถูกใช้แล้ว กล่องจะถูกวาดด้วยสีนี้
- เมื่อโคออร์ดิเนตอยู่นอกช่วงที่ถูกกำหนดให้ โคออร์ดิเนตที่อยู่นอกช่วงนั้นจะถูกกำหนดให้เป็นมูลค่าที่ใช้ได้ที่ใกล้เคียงกันมากที่สุด (closest legal value)
- ,bf วาด filled box ใน foreground
- โคออร์ดิเนตจาก STEP (xoffset,yoffset) สามารถถูกใช้แทนที่ absolute coordinate ตัวอย่างเช่น กำหนดว่า most recent point ที่ถูกอ้างอิงคือ (0,0) คำสั่ง LINE STEP (10,5) จะระบุจุดที่ภาวะเริ่มต้น (offset) 10 จาก x และภาวะเริ่มต้น 5 จาก y
- ถ้า STEP option ถูกใช้สำหรับโคออร์ดิเนตที่สองบนคำสั่ง LINE แล้วมันจะสัมพันธ์กับโคออร์ดิเนตแรกในคำสั่ง วิธีอื่นเท่านั้นที่จะสร้าง "most recent point" ขึ้นใหม่ก็คือทำการกำหนดค่าเริ่มต้นใหม่ด้วยคำสั่ง CLS และคำสั่ง SCREEN

ตัวอย่าง

```
SCREEN  105
10 LINE -(x2,y2)
```

วาดไลน์จากจุดล่าสุดไปยังจุด x2,y2 ใน foreground color นี่เป็นรูปฟอร์มที่ง่ายที่สุดของคำสั่ง LINE

```
20 LINE (0,0)-(639,324)
```

วาดเส้นทะแยงมุมขวางจอภาพ (downward)

```
30 LINE (0,100)-(639,100)
```

วาดไลน์ขวางจอภาพ

```
40 LINE (10,10)-(200,200),2
```

วาดไลน์ในสีหมายเลข 2

```
10 ON ERROR GOTO 50
20 CLS
30 LINE -(rnd*639,rnd*324),rnd*4
40 GO TO 20
50 IF ERR=5 THEN RESUME 30
```

วาดไลน์ต่าง ๆ เรื่อย ๆ ไปโดยการใช้คุณสมบัติต่าง ๆ ของการสุ่ม (random attributes)

```
10 FOR x-0 to 639
20 LINE (x,0)-(x,324), x AND 1
```

```
30 NEXT
```

วาด alternating line on-line off pattern

```
10 LINE (0,0)-(100,100),,b
```

วาดกล่องใน foreground (สังเกตว่าสีไม่ได้ถูกประกอบรวมอยู่ด้วย)

```
20 LINE STEP (0,0)-STEP (200,200),2,bf
```

วาด filled box ใน foreground ด้วยสีหมายเลข 2 โคออร์ดิเนต ต่าง ๆ ถูกกำหนดให้ตามภาวะเริ่มแรกต่าง ๆ

LINE INPUT Statement

รูปแบบ LINE INPUT[;]["prompt";] stringvar

จุดประสงค์ ป้อนข้อมูลเข้าไปตลอดไลน์ (สูงสุดถึง 254 อักขระ) โดยไม่รับรู้ตัวกำหนดขอบเขต (dilimiters) ต่าง ๆ ไปยังตัวแปรสตริงก์

รายละเอียด
- "prompt" เป็นค่าคงที่สตริงก์ที่ถูกพิมพ์บนจอภาพก่อนที่ข้อมูลเข้าจะถูกรับเข้าไป เครื่องหมายคำถามจะไม่ถูกพิมพ์แม้ว่ามันจะเป็นส่วนหนึ่งของ prompt string ก็ตาม
- stringvar เป็นชื่อของตัวแปรสตริงก์หรือชื่อของสมาชิกอะเรย์ที่ซึ่งไลน์จะถูกกำหนดค่า ข้อมูลเข้าทุกข้อมูลจากตอนท้ายของ "prompt" ไปจนถึง carriage return จะถูกกำหนดค่าไปยัง stringvar อย่างไรก็ตาม ถ้าชุดลำดับของ linefeed/carriage return (ลำดับนี้เท่านั้น) ถูกพบ อักขระทั้งสองจะถูกส่งค่าไป (echoed) แต่ carriage return จะไม่ถูกรับรู้ linefeed จะถูกใส่เข้าไปใน stringvar และข้อมูลเข้าจะดำเนินการต่อไป

- ถ้า LINE INPUT ถูกติดตามด้วยเครื่องหมายเซมิโคลอนทันที แล้ว carriage return ที่ถูกพิมพ์โดยผู้ใช้ที่ตอนท้ายของไลน์ข้อมูลเข้าจะไม่ส่ง carriage return/linefeed sequence ไปที่เทอร์มินัล
- ออกจาก LINE INPUT โดยการกด Ctrl-Break GWBASIC จะหันกลับไปยังระดับคำสั่ง การกด CONT จะดำเนินการประมวลผลต่อไปที่ LINE INPUT

ตัวอย่าง ดูคำสั่ง LINE INPUT #

LINE INPUT # Statement

รูปแบบ LINE INPUT #filenum, stringvar

จุดประสงค์ อ่านตลอดไลน์ (สูงสุดถึง 255 อักขระ) จากซีเควนเชียลไฟล์โดยไม่รับรู้ตัวกำหนดขอบเขตต่าง ๆ ไปยังตัวแปรสตริงก์

รายละเอียด
- filenum เป็นหมายเลขของไฟล์
- stringvar เป็นชื่อของตัวแปรสตริงก์ที่ซึ่งไลน์จะถูกกำหนดค่า
- LINE IPUT # อ่านอักขระทุก ๆ อักขระในซีเควนเชียลไฟล์จนกว่ามันจะมาถึง carriage return/line feed sequence ซึ่งมันไม่ปฏิบัติ (omit) สังเกตว่า line feed/carriage return sequence ถูกส่งค่าในฐานะเป็นส่วนหนึ่งของสตริงก์
- LINE INPUT # ใช้ประโยชน์พิเศษเฉพาะถ้าไลน์แต่ละไลน์ของไฟล์ถูกแตกออกเป็นฟีลด์ (fields) ต่าง ๆ หรือถ้าโปรแกรมของ GWBASIC ถูกเก็บบันทึกใน ASCII MODE ที่จะถูกอ่านในฐานะเป็นข้อมูลโดยโปรแกรมอื่น (ดูคำสั่ง SAVE)
- LINE INPUT # สามารถถูกใช้สำหรับ random file ได้ด้วย ดูภาคผนวก ก, Sequential and Random Files

ตัวอย่าง

```
10 OPEN "O", 1, "LIST"
20 LINE INPUT "CUSTOMER INFOBMATION? ";C$
30 PRINT #1, C$
40 CLOSE 1
50 OPEN "I", 1, "LIST"
60 LINE INPUT #1, C$
70 PRINT C$
80 CLOSE 1
RUN
CUSTOMER INFORMATION? LINDA JONES 234,4 MEMPHIS
LINDA JONES 234,4 MEMPHIS
```

โปรแกรมนี้บันทึกข่าวสารแรกเริ่มไปยังซีเควนเชียลไฟล์และอ่านมันกลับมาด้วยคำสั่ง LINE INPUT #

LIST Command

รูปแบบที่ 1 LIST [line1]

รูปแบบที่ 2 LIST [line1] [-[line2]] [,device]

จุดประสงค์ แสดงรายการทั้งหมดหรือแสดงส่วนของโปรแกรมที่เป็นอยู่ในหน่วยความจำในขณะนั้น

รายละเอียด
- line มีค่าอยู่ในช่วง 0 ถึง 65529
- device เป็นนิพจน์สตริงก์อย่างเช่น SCRN หรือ LPT
- รูปแบบที่ 1

  ถ้า line ถูกละ แล้วโปรแกรมจะถูกแสดงรายการเริ่มต้นที่ไลน์นัมเบอร์ต่ำสุด

  ถ้า line ถูกประกอบรวมอยู่ด้วย แล้วไลน์ที่ถูกระบุเท่านั้นจะถูกแสดงรายการ

- รูปแบบที่ 2

รูปแบบนี้อนุญาตให้มีข้อเลือกต่าง ๆ ต่อไปนี้ :

1. ถ้าไลน์แรกเท่านั้นถูกระบุและถูกติดตามด้วยเครื่องหมายไฮเฟน (-) ไลน์นั้นและไลน์ทุก ๆ ไลน์ที่มีหมายเลขสูงกว่าจะถูกแสดงรายการ
2. ถ้าไลน์ที่สองเท่านั้นถูกระบุร่วมกับเครื่องหมายไฮเฟนที่อยู่นำหน้าแล้ว ไลน์ทุก ๆ ไลน์จากจุดเริ่มต้นของโปรแกรมจนกระทั่งถึงไลน์นั้นจะถูกแสดงรายการ
3. ถ้าไลน์ทั้งสองถูกระบุ แล้วไลน์ตลอดช่วงที่ถูกระบุจะถูกแสดงรายการ
4. ถ้า device ถูกละ แล้วการแสดงรายการจะถูกแสดงบนจอภาพ

- ใช้ Control-Break เพื่อที่จะหยุด LIST

- โดยปกติ GWBASIC จะหันกลับไปยังระดับคำสั่งหลังจาก LIST

ตัวอย่าง รูปแบบที่ 1

```
LIST
```

แสดงโปรแกรมที่เป็นอยู่ในหน่วยความจำในขณะนั้น

```
LIST 500
```

แสดงไลน์หมายเลข 500

รูปแบบที่ 2

```
LIST 150-
```

แสดงรายการทุก ๆ ไลน์จากไลน์หมายเลข 150 ไปจนสิ้นสุดโปรแกรม

```
LIST -1000
```

แสดงทุก ๆ ไลน์จากไลน์หมายเลขต่ำสุดจนถึงไลน์หมายเลข 1000

LIST 150-1000

แสดงไลน์ต่าง ๆ ตั้งแต่ไลน์หมายเลข 150 จนถึงไลน์หมายเลข 1000

LIST 150-1000,LPT1:

แสดงไลน์ต่าง ๆ ตั้งแต่ไลน์หมายเลข 500 จนถึงไลน์หมายเลข 1000 บนเครื่องพิมพ์ผลลัพธ์ (line printer)

LLIST Command

รูปแบบ LLIST [line1][-[line2]]

จุดประสงค์ แสดงรายการทั้งหมดหรือแสดงส่วนของโปรแกรมในหน่วยความจำที่เป็นอยู่ในขณะนั้นบนเครื่องพิมพ์ (LPT1:)

รายละเอียด - LLIST กำหนดเครื่องพิมพ์ขนาดความกว้าง 132 อักขระ

- LLIST ปฏิบัติตามข้อเลือกต่าง ๆ ของ LIST รูปแบบที่ 2

- GWBASIC จะหันกลับไปยังระดับคำสั่งหลังจาก LLIST

ตัวอย่าง ดูคำสั่ง LIST

LOAD Command

รูปแบบ LOAD filespec[,R]

จุดประสงค์ เรียก (load) โปรแกรมจากอุปกรณ์ที่ถูกระบุลงในหน่วยความจำและโดยเลือกได้วิ่งมันหรือไม่วิ่ง

รายละเอียด - filespec เป็นชื่อของไฟล์

- โดยปกติ R option วิ่งโปรแกรมหลังจากที่มันได้ถูกเรียกเรียบร้อยแล้ว

- ถ้าอุปกรณ์ถูกละ DOS default diskette drive จะถูกใช้ ถ้าไม่มีส่วนขยายถูกจัดจ่ายให้และชื่อไฟล์มีความยาวน้อยกว่าหรือเท่ากับ 8 ตัว

อักขระแล้ว ส่วนขยาย .BAS จะถูกใส่เข้าไปที่ชื่อไฟล์

- LOAD ปิดไฟล์ทุก ๆ ไฟล์ที่เปิดอยู่และลบตัวแปรทุก ๆ ตัวและไลน์ทุก ๆ ไลน์ของโปรแกรมที่ปรากฏอยู่ในหน่วยความจำในขณะนั้นออกไปก่อนที่มันจะเรียกโปรแกรมที่ถูกกำหนดไว้
- ถ้า R option ถูกใช้ โปรแกรมจะปฏิบัติการหลังจากที่มันถูกเรียกและไฟล์ข้อมูลที่เปิดอยู่ทุก ๆ ไฟล์จะยังคงเปิดอยู่ ดังนั้น LOAD ร่วมด้วย R option อาจจะถูกใช้เพื่อที่จะเชื่อมโยงโปรแกรมหลาย ๆ โปรแกรม (หรือส่วนต่าง ๆ ของโปรแกรมเดียวกัน) และข่าวสารอาจจะถูกส่งผ่านระหว่างโปรแกรมต่าง ๆ โดยการใช้ไฟล์ข้อมูล (data files) ต่าง ๆ

ตัวอย่าง LOAD "B:MYPROG"

โปรแกรมชื่อ MYPROG ในไดร์ฟ์ B จะถูกเรียกลงสู่หน่วยความจำ

LOAD "STRTRK" R

โปรแกรมชื่อ STRTRK ถูกเรียกและถูกปฏิบัติการ

LOC Function

รูปแบบ LOC(filenum)

จุดประสงค์ พิมพ์ตำแหน่งปัจจุบันในไฟล์

รายละเอียด
- filenum เป็นหมายเลขที่ถูกใช้เมื่อไฟล์ถูกเปิด
- กับ random files LOC ส่งค่าหมายเลขเร็คคอร์ดที่เพิ่งถูกอ่านหรือถูกบันทึก ถ้าไฟล์ถูกเปิดไว้แต่ไม่ได้ถูกบันทึก LOC จะส่งค่าศูนย์
- กับ sequentials files LOC ส่งค่าจำนวนของเร็คคอร์ดต่าง ๆ (128-byte blocks) ที่ถูกอ่านจากหรือถูกเขียนไปยังไฟล์ตั้งแต่มันถูกเปิด ตัวเลขนี้จะมีค่าเป็น 1 แม้แต่ก่อนที่ข้อมูลเข้าใด ๆ ไปยังไฟล์เนื่องจากว่า

GWBASIC อ่านเซกเตอร์แรกของไฟล์เมื่อมันถูกเปิด

- กับ communications file LOC พิมพ์จำนวนของอักขระต่าง ๆ ในบัฟเฟอร์ข้อมูลเข้า (input buffer) ที่กำลังคอยที่จะถูกอ่าน ถ้ามีอักขระมากกว่า 255 อักขระ LOC จะส่งค่าเป็น 255 (ขนาดที่กำหนดเป็น default คือ 256 อักขระ แต่มันอาจถูกเปลี่ยนแปลงโดย /C: option บนคำสั่ง GWBASIC สิ่งนี้ตัดความจำเป็นสำหรับการตรวจสอบสำหรับขนาดของสตริงก่อนที่จะทำการอ่านข้อมูลลงสู่บัฟเฟอร์เนื่องจากสตริงถูกจำกัดขอบเขตที่ 255 อักขระ

ตัวอย่าง 200 IF LOC(1)>50 THEN STOP

โปรแกรมจะถูกทำให้หยุดลงหลังจากเร็คคอร์ดที่ 50 ในไฟล์

LOCATE Statement

รูปแบบ LOCATE [y][,[x][,[cursor][,[start][,stop]]]

จุดประสงค์ ย้ายเคอร์เซอร์ไปยังตำแหน่งที่ถูกระบุบน active screen พารามิเตอร์ต่าง ๆ ที่เป็นข้อเลือกจะเปิดหรือปิดเคอร์เซอร์ที่กระพริบ (blinking cursor) และบ่งบอกขนาดของมัน

รายละเอียด
- y เป็นตัวเลขในช่วง 1 ถึง 25 บ่งชี้หมายเลขของไลน์ของจอภาพที่ซึ่งเคอร์เซอร์จะถูกวาง (placed)
- x เป็นตัวเลขในช่วง 1 ถึง 40 หรือ 1 ถึง 80 ขึ้นอยู่กับความกว้างของจอภาพที่ใช้อยู่ที่จะบ่งบอกตัวเลขของสดมภ์ของจอภาพที่ซึ่งเคอร์เซอร์จะถูกวาง
- cursor มีค่าเป็น 0 บ่งชี้ว่าเคอร์เซอร์ปิด (ไม่สามารถมองเห็นได้) หรือมันมีค่าเป็น 1 เพื่อที่จะบ่งชี้ว่ามันเปิด (สามารถมองเห็นได้)
- start เป็นตัวเลขในช่วง 0 ถึง 13 ที่บ่งชี้เคอร์เซอร์ที่จะเริ่มต้น scan ไลน์

- stop เป็นตัวเลขในช่วง 0 ถึง 13 ที่บ่งชี้ว่าเคอร์เซอร์หยุด scan ไลน์
- ถ้าท่านมีทั้ง text และ graphic pages อย่าลืมว่า LOCATE ย้ายเคอร์เซอร์ใน active page ของ mode ที่จะถูกใช้ในขณะนั้น
- cursor, start และ stop อาจจะถูกใช้ใน text mode เท่านั้น
- ถ้ามูลค่าต่าง ๆ ที่ถูกป้อนเข้าไปอยู่นอกช่วงที่ถูกกำหนดให้ จะส่งผลให้เกิด "Illegal function call" error ขึ้น และมูลค่าต่าง ๆ ก่อนหน้าจะถูกรักษาไว้ (retained)
- start และ stop ทำให้ท่านสามารถที่จะสร้างเคอร์เซอร์ขนาดใด ๆ โดยการบ่งชี้จุดเริ่มต้นและจุดสิ้นสุด scan lines ไลน์ต่าง ๆ เหล่านี้จะเป็นจำนวนเลขจาก 0 ที่ตอนบนสุดของตำแหน่งอักขระถึง 12 ที่ตอนล่าง (7 บน most compatible color/graphics monitor adaptors)
- ถ้า start ถูกกำหนดให้โดยไม่มี stop แล้ว stop จะกำหนดเป็นมูลค่าของ start ถ้า start มีค่ามากกว่า stop มันจะเป็น two-part cursor ในที่ซึ่งเคอร์เซอร์ wrap จากไลน์ตอนล่างสุดกลับไปยังไลน์ตอนบนสุด
- LOCATE,,1 เปิดเคอร์เซอร์ (ดูตัวอย่างข้างล่าง)
- พารามิเตอร์ใด ๆ อาจจะถูกละไว้ พารามิเตอร์ต่าง ๆ ที่ถูกละไว้จะกำหนดค่าเป็นมูลค่าปัจจุบัน
- การกระพริบของเคอร์เซอร์ (cursor blink) ไม่สามารถเลือกได้และโดยปกติจะกระพริบ 4 ครั้งต่อวินาที

ตัวอย่าง

```
10 LOCATE 1,1
20 LOACTE 5,5:PRINT "HELLO"
30 LOACTE ,,1
40 LOACTE ,,,12
```

ไลน์ 10 ย้ายเคอร์เซอร์ไปยัง home position ในขอบบนด้านซ้ายของจอภาพ ไลน์ 20 พิมพ์ "HELLO" แถวที่ 5 สดมภ์ที่ 5 ไลน์ 30 ทำให้เคอร์เซอร์สามารถเห็นได้โดยไม่มีการเปลี่ยนแปลงตำแหน่งของมัน ในไลน์ 40 ตำแหน่งของเคอร์เซอร์และการมองเห็นได้ไม่ถูกเปลี่ยนแปลง เคอร์เซอร์จะพิมพ์ที่ตอนล่างสุดของอักขระที่เริ่มต้นและสิ้นสุดบน scan line 12

LOF Function

รูปแบบ LOF (file number)

จุดประสงค์ ส่งค่าความยาวของไฟล์เป็นไบท์

ตัวอย่าง
```
110 IF REC*RECSIZ>LOF(1) THEN PRINT "INVALID ENTRY"
```
ในตัวอย่างนี้ ตัวแปร REC และตัวแปร RECSIZ บรรจุหมายเลขเร็คคอร์ดและความยาวของเร็คคอร์ดตามลำดับ การคำนวณกำหนดว่าเร็คคอร์ดที่ถูกระบุถัดไปเป็น end-of-file หรือไม่

LOG Function

รูปแบบ LOG(x)

จุดประสงค์ ส่งค่า natural log ของ x

รายละเอียด - x เป็นนิพจน์จำนวนเลขที่มีค่ามากกว่าศูนย์

- natural log คือลอการิทึมฐาน e

ตัวอย่าง
```
PRINT LOG(45/7)

 1.860752

o    k
```
ตัวอย่างนี้คำนวณค่าลอการิทึมของนิพจน์ 45/7

LPOS Function

รูปแบบ LPOS(n)

จุดประสงค์ ส่งค่าตำแหน่งปัจจุบันของ print head ในบัฟเฟอร์ของเครื่องพิมพ์

รายละเอียด - n เป็นตัวเลขที่ถูกกำหนดค่าไปยัง line printer (0, 1 หรือ 2)

- ฟังก์ชันนี้ไม่จำเป็นต้องให้ตำแหน่งทางกายภาพ (physical position) ของ print head

ตัวอย่าง 10 IF LPOS(0)>50 THEN LPRINT CHR$(13)

ตัวอย่างนี้ส่งอักขระ carriage return ไปที่เครื่องพิมพ์ถ้าความยาวของ LPOS(0) มากกว่า 50 อักขระ

LPRINT and LPRINT USING Statements

รูปแบบ LPRINT [list of expresssions] [;]

'LPRINT USING string exp; list of expressions [;]

จุดประสงค์ พิมพ์ข้อมูลบนเครื่องพิมพ์ (LPT1:)

รายละเอียด - list of expressions เป็นรายการของนิพจน์จำนวนเลขหรือนิพจน์สตริงก์ที่จะถูกพิมพ์ นิพจน์ต่าง ๆ เหล่านี้ต้องถูกแบ่งแยกโดยเครื่องหมายจุลภาคหรือเครื่องหมายเซมิโคลอน

- string exp เป็นค่าคงที่สตริงก์หรือตัวแปรสตริงก์ซึ่งบ่งชี้รูปแบบที่จะถูกใช้สำหรับการพิมพ์

- คำสั่งต่าง ๆ เหล่านี้คล้ายคลึงกับคำสั่ง PRINT และคำสั่ง PRINT USING ยกเว้นว่าผลลัพธ์จะส่งไปยังเครื่องพิมพ์ ดูคำสั่ง PRINT และคำสั่ง PRINT USING

ตัวอย่าง 10 A=123

20 LPRINT "THIS IS OUTPUT ON THE PRINTER"

```
30 LPRINT USING "####";A
RUN
ตัวอย่างนี้พิมพ์สิ่งต่อไปนี้บนเครื่องพิมพ์
THIS IS OUTPUT ON THE PRINTER
 123
```

LSET and RSET Statements

รูปแบบ LSET stringvar = x$

RSET stringvar = x$

จุดประสงค์ ย้ายข้อมูลจากหน่วยความจำไปยังบัฟเฟอร์ของ random file (ในการจัดเตรียมสำหรับคำสั่ง PUT)

รายละเอียด
- stringvar เป็นตัวแปรที่ถูกบ่งบอกในคำสั่ง FIELD
- x$ เป็นนิพจน์สตริงก์
- ถ้า x$ ต้องการไบท์ต่าง ๆ น้อยกว่าจำนวนไบท์ที่ถูกระบุในคำสั่ง FIELD สำหรับ stringvar แล้ว LSET จะ left-justifies สตริงก์ในฟิลด์และ RSET จะถูกจัดการแบบ right-justifies สตริงก์ (ช่องว่าง (spaces) ต่าง ๆ ถูกใช้เพื่อที่จะใส่ให้เต็มตำแหน่งต่าง ๆ ที่พิเศษเฉพาะ) ถ้า x$ มีความยาวเกินไปสำหรับฟิลด์แล้ว อักขระต่าง ๆ จะถูกตัดทิ้งจากทางขวามือ
- มูลค่าจำนวนเลขต่าง ๆ ต้องถูกแปลงค่าไปสู่สตริงก์ต่าง ๆ ก่อนที่มันจะเป็น LSET หรือ RSET ดูฟังก์ชัน MKI$ MKS$ และฟังก์ชัน MKD$
- ดูภาคผนวก ก, Sequential and Random Files
- หมายเหตุ : เช่นเดียวกันฟังก์ชันเหล่านี้อาจจะถูกใช้กับตัวแปรสตริงก์ที่ไม่ได้ถูกบ่งบอกในคำสั่ง FIELD เพื่อที่จะ justify สตริงก์ในฟิลด์ที่ถูกกำหนดให้ สิ่งนี้เป็นประโยชน์เมื่อทำการกำหนดรูปแบบผลลัพธ์ที่จะถูกพิมพ์

ตัวอย่าง 110 A$=SPACE$(20)

120 RSET A$=N$

ตัวอย่างนี้ right-justifies สตริงก์ N$ ในฟิลด์ขนาด 20 ตัวอักขระ

150 LSET A$-MKS$(AMT)

ตัวอย่างนี้แปลงค่า AMT ซึ่งเป็นมูลค่าจำนวนเลขไปสู่สตริงก์และ left-justifies มันในฟิลด์ A$ เพื่อการเตรียมพร้อมสำหรับ คำสั่ง PUT (file)

MERGE Command

รูปแบบ MERGE filespec

จุดประสงค์ รวมแอสกีไฟล์ (ASCII file) ที่ถูกระบุเข้ากับโปรแกรมที่ปรากฏอยู่ในหน่วย-ความจำในขณะนั้น

รายละเอียด
- filespec เป็นชื่อของไฟล์
- โปรแกรมที่ถูกรวม (merge) ต้องเป็น ASCII format มิฉะนั้นแล้วจะเกิด "Bad file mode" error ขึ้น
- ถ้าไลน์ใด ๆ ในดิสค์ไฟล์มีหมายเลขต่าง ๆ เดียวกันกับหมายเลขต่าง ๆ ในโปรแกรมในหน่วยความจำ ไลน์ต่าง ๆ จากไฟล์บนดิสค์จะแทนที่ไลน์ต่าง ๆ ในหน่วยความจำ (การ merge อาจจะคิดว่าเป็นการสอดใส่ (inserting) ไลน์ต่าง ๆ ของโปรแกรมบนดิสค์ไปสู่โปรแกรมในหน่วยความจำ)
- ถ้าชื่ออุปกรณ์ถูกละ DOS default drive จะถูกกำหนด
- GWBASIC จะหันกลับไปสู่ระดับคำสั่งหลังจากคำสั่ง MERGE

ตัวอย่าง MERGE "A:NUMBRS"

ไฟล์ NUMBRS บนไดร้ฟ์ A ถูกรวมเข้าด้วยกันกับโปรแกรมในหน่วยความจำ

MID$ Function

รูปแบบ v$ = MID$(X$,n[,m])

จุดประสงค์ ส่งค่าสตริงก์ความยาว m อักขระจาก X$ เริ่มต้นด้วยอักขระตัวที่ n

รายละเอียด - n และ m ต้องมีค่าอยู่ในช่วง 1 ถึง 255 ถ้า m ถูกละหรือถ้ามีจำนวนอักขระน้อยกว่า m อักขระที่อยู่ทางด้านขวามือของอักขระตัวที่ n แล้วอักขระทุก ๆ อักขระทางขวามือสุด (right most) เริ่มต้นด้วยอักขระตัวที่ n จะถูกส่งค่าไป ถ้า n มีค่ามากกว่าจำนวนของอักขระต่าง ๆ ใน X$ (LEN(X$)) แล้ว MID$ จะส่งค่าสตริงก์ว่างเปล่า (null string)

- ดูฟังก์ชัน LEFT$ และฟังก์ชัน RIGHT$ ด้วย

ตัวอย่าง

```
LIST
10 A$="GOOD "
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$;MID$(B$,9,7)
ok
RUN
 GOOD  EVENING
Ok
```

MID$ ถูกใช้เพื่อที่จะเลือกส่วนหนึ่งของสตริงก์ B$

MID$ Statement

รูปแบบ MID$(string expl,n[,m]=string exp2

จุดประสงค์ แทนที่ส่วนหนึ่งของสตริงก์หนึ่งด้วยสตริงก์อีกสตริงก์หนึ่ง

รายละเอียด - n และ m เป็นนิพจน์จำนวนเต็ม และ string expl. และstring exp2 เป็นนิพจน์สตริงก์

- อักขระต่าง ๆ ใน string expl เริ่มต้นที่ตำแหน่งที่ n ถูกแทนที่โดยอักขระต่าง ๆ ใน string exp2 Optional m อ้างอิงถึงจำนวนอักขระต่าง ๆ จาก string exp2 ซึ่งจะถูกใช้ในการแทนที่ ถ้า m ถูกละแล้ว string exp2 ทั้งหมดจะถูกใช้ อย่างไรก็ตาม ไม่คำนึงถึงว่า m จะถูกละหรือถูกประกอบรวมอยู่ด้วย การแทนที่ของอักขระต่าง ๆ จะไม่ยาวเกินไปกว่าความยาวแรกเริ่มของ string expl
- ดูฟังก์ชัน MID$ ด้วย

ตัวอย่าง

```
10 A$="KANSAS CITY, MO"
20 MID$(A$,14)="KS"
30 PRINT A$
RUN
 KANSAS CITY, KS
```

ตัวอย่างนี้แสดงอักขระ 2 ตัวในสตริงก์ A$ ที่จะถูกแทนที่โดย "KS"

MKI$, MKS$, MKD$ Functions

รูปแบบ

```
v$ = MKI$ (integer expression)
v$ = MKS$ (single-precision expression)
v$ = MKD$ (double-precision expression)
```

จุดประสงค์ แปลงค่ามูลค่าจำนวนเลขต่าง ๆ ไปสู่มูลค่าสตริงก์ต่าง ๆ

รายละเอียด

- มูลค่าจำนวนเลขใด ๆ ที่ถูกแทนที่ในบัฟเฟอร์ของ random file ด้วยคำสั่ง LSET หรือคำสั่ง RSET ต้องถูกแปลงค่าไปสู่สตริงก์ MKI$ จะแปลงค่าจำนวนเต็มไปสู่สตริงก์ขนาด 2 ไบท์ MKS$ แปลงค่าจำนวนเลขในรูป single precision ไปสู่สตริงก์ขนาด 4 ไบท์ ส่วน MKD$ แปลงค่าจำนวนเลขในรูป double precision ไปสู่สตริงก์ขนาด 8 ไบท์

- เหมือน STR$ ฟังก์ชันต่าง ๆ เหล่านี้ไม่ได้เปลี่ยนแปลงไบท์ต่าง ๆ ของข้อมูล
- อ้างอิงไปที่ฟังก์ชัน CVI, CVS, CVD และอ้างอิงไปยังภาคผนวก ก, Sequential and Random Files

ตัวอย่าง

```
90 AMT=(K+T)
100 FIELD #1,8 AS D$,20 AS N$
110 LSET D$=MKS$(AMT)
120 LSET N$=A$
130 PUT #1
```

ไฟล์หมายเลข 1 (#1) มีฟิลด์ต่าง ๆ ที่ถูกบ่งบอกในไลน์ 100 ไลน์ 110 ใช้ MKS$ เพื่อที่จะแปลงค่า AMT ไปสู่มูลค่าสตริงก์และใช้ LSET เพื่อที่จะใส่มูลค่าของ AMT ไปสู่บัฟเฟอร์ของ random file ไลน์ 120 แทนที่สตริงก์ลงสู่บัฟเฟอร์ และไลน์ 130 บันทึกข้อมูลจากบัฟเฟอร์ไปยังไฟล์

NAME Command

รูปแบบ NAME old filename AS new filename

จุดประสงค์ เปลี่ยนชื่อไฟล์ของดิสเกตต์ (diskette file) ใหม่

รายละเอียด
- old filename เป็นชื่อของไฟล์ที่มีอยู่แล้ว
- new filename เป็นชื่อไฟล์ที่ท่านต้องการเรียกใช้
- หลังจากใช้คำสั่ง NAME แล้วไฟล์จะปรากฏอยู่บนดิสค์เดียวกัน ในเนื้อที่ของ disk space เดียวกันด้วยชื่อใหม่

- ถ้าชื่ออุปกรณ์ไม่ได้ถูกกำหนดให้ DOS default drive จะถูกกำหนดให้ในคำสั่งนี้ ส่วนขยายของไฟล์ไม่ได้กำหนดไว้ล่วงหน้า (default) ไปที่ .BAS

ตัวอย่าง

```
Ok
NAME "A:ACCTS" AS "LEDGER"
Ok
```

ไฟล์ซึ่งถูกกำหนดชื่อเป็น ACCTS แต่แรกเริ่มบนดิสเกตต์ในไดร์ฟ A ในขณะนี้จะถูกกำหนดชื่อเป็น LEDGER

NEW Command

รูปแบบ NEW

จุดประสงค์ ลบโปรแกรมที่ปรากฏในหน่วยความจำขณะนั้นออกไปและเคลียร์ (clear) ตัวแปรทุก ๆ ตัว

รายละเอียด
- คำสั่งนี้เคลียร์หน่วยความจำก่อนที่จะทำการป้อนโปรแกรมใหม่เข้าไป GWBASIC จะกลับไปยังระดับคำสั่งหลังจาก NEW ถูกประมวลผล
- ดู TRON และ TROFF ด้วย

ตัวอย่าง

```
NEW
```

โปรแกรมในหน่วยความจำจะถูกลบออกไป

OCT$ Function

รูปแบบ v$ = OCT$(n)

จุดประสงค์ ส่งค่ามูลค่าเลขฐานแปดของเลขฐานสิบ

รายละเอียด
- n มีค่าเป็นจำนวนเต็มในช่วง -32768 ถึง 65535
- ดูฟังก์ชัน HEX$ สำหรับการแปลงค่าเลขฐานสิบหกด้วย

ตัวอย่าง

```
Ok
PRINT OCT$(24)
30
Ok
```

ตัวอย่างนี้แสดงว่าจำนวนเลข 24 ในฐานสิบคือจำนวนเลข 30 ในฐานแปด

ON COM(n) Statement

รูปแบบ ON COM(n) GOSUB line number

จุดประสงค์ ระบุไลน์นัมเบอร์แรกของสับรูทีนที่จะถูกกระทำเมื่อการกระทำ (activity) เกิดขึ้นบน communications channel

รายละเอียด
- line number เป็นหมายเลขของไลน์แรกของสับรูทีน
- n เป็นหมายเลขของ communiactions port
- ON COM จะถูกประมวลผลก็ต่อเมื่อคำสั่ง COM(n) ON ถูกประมวลผล ถ้า trapping ของเหตุการณ์ (event trapping) ถูกทำให้มีความสามารถ และถ้า line number ในคำสั่ง ON COM ไม่เป็นศูนย์แล้ว GWBASIC จะตรวจสอบระหว่างคำสั่งต่าง ๆ เพื่อที่จะดูว่า communication activity ได้เลือก channel ใดไปใช้ ถ้า communications activity กำหนดขึ้นแล้ว GOSUB จะส่งไปปฏิบัติการยังไลน์ที่ระบุ
- ถ้าคำสั่ง COM OFF ได้ถูกประมวลผลสำหรับ communications แล้ว GOSUB จะไม่ถูกกระทำและจะไม่ถูกจดจำ
- ถ้าคำสั่ง COM STOP ถูกประมวลผลสำหรับ communications channel แล้ว GOSUB จะไม่ถูกกระทำแต่จะถูกกระทำในทันทีที่คำสั่ง COM ON ถูกประมวลผล

- ไลน์นัมเบอร์ 0 ทำให้ communications trap ไร้ความสามารถ
- เมื่อ event trap เกิดขึ้น (นั่นคือ GOSUB ถูกกระทำ) automatic COM STOP จะถูกประมวลผล ทั้งนี้จะทำให้ recursive trap จะไม่เกิดขึ้น (take place) RETURN จาก trapping subroutine โดยอัตโนมัติจะกระทำคำสั่ง COM ON แม้ว่า explicit COM OFF จะถูกกระทำภายในสับรูทีนก็ตาม
- RETURN line อาจจะถูกใช้เพื่อที่จะส่งค่าไปยังไลน์นัมเบอร์ที่เฉพาะเจาะจงของ trapping subroutine อย่างไรก็ตาม การใช้รูปแบบของ return นี้ควรระมัดระวังผลอันเนื่องจาก GOSUBs WHILEs หรือ FOR ใด ๆ ที่เป็น active ในขณะที่ trap ยังคง active อยู่ เพราะอาจจะทำให้เกิดข้อผิดพลาด (errors) ต่าง ๆ อย่างเช่น "FOR without NEXT" ขึ้นก็ได้
- Event trapping ไม่ได้เกิดขึ้นแทนที่เมื่อ GWBASIC ไม่ได้ทำการประมวลผลโปรแกรม และโดยอัตโนมัติ event trapping จะถูกทำให้ไร้ความสามารถเมื่อ error trap เกิดขึ้น

ON ERROR GOTO Statement

รูปแบบ ON ERROR GOTO line

จุดประสงค์ ช่วยให้ข้อผิดพลาดต่าง ๆ ที่เกิดขึ้นจะถูกดัก (trapped) ออกไปโดยการระบุไลน์แรกของ error-handling subroutine

รายละเอียด
- line เป็นหมายเลขของไลน์แรกของ error-handling subroutine ถ้า n ไม่มีอยู่ จะเกิด "Undefined line number" error ขึ้น
- เมื่อ error-handling subroutine ถูกกำหนดขึ้นให้ปฏิบัติการแล้ว errors ที่ถูกตราจจับ (detected) หรือรวมทั้ง errors ในรูปแบบตรง

(ตัวอย่างเช่น syntax error) จะเป็นเหตุให้เกิดการกระโดดไปยัง error-handling routine ที่ถูกระบุในคำสั่งดังกล่าว

- เพื่อที่จะทำให้ error handling ไม่ปฏิบัติ เรามีสิทธิจะใช้คำสั่ง ON ERROR GOTO 0 เพื่อให้ errors ต่าง ๆ ที่ปรากฏมาจะถูกพิมพ์ข้อความผิดพลาดและหยุดการประมวลผล
- คำสั่ง ON ERROR GOTO 0 ซึ่งปรากฏใน error-trapping routines เป็นเหตุให้ GWBASIC หยุดและพิมพ์ข้อความผิดพลาดสำหรับ error ซึ่งเป็นสาเหตุให้เกิด trap ขึ้น มีข้อเสนอแนะว่า ควรจะมี error-handling routines ปรากฏในการประมวลผลทุกครั้งเผื่อไว้สำหรับ error ซึ่งอาจตรวจไม่พบด้วยสายตาของเรา
- หมายเหตุ : ถ้า error เกิดขึ้นในระหว่างการประมวลผลของ error-handling routine ข้อความผิดพลาดนั้นจะถูกพิมพ์และการประมวลผลจะสิ้นสุดลง error trapping ไม่ได้เกิดขึ้นภายใน error-handling routine

ตัวอย่าง

```
10 ON ERROR GOTO 1000
   .
   .
   .
1000 PRINT "Error";ERR;"at line";ERL
   .
   .
   .
```

คำสั่งต่าง ๆ เหล่านี้เป็นคำสั่งตัวอย่าง (sample statements) ต่าง ๆ ซึ่งจะถูกใช้ใน error handling

ON... **GOSUB** and ON... **GOTO** Statements

รูปแบบ ON n GOT0 line[,line]...

ON n GOSUB line[,line]...

จุดประสงค์ ย้ายไปยังไลน์นัมเบอร์ใดไลน์นัมเบอร์หนึ่งในหลายไลน์นัมเบอร์ที่ถูกระบุขึ้นอยู่กับมูลค่าที่ถูกระบุ

รายละเอียด
- n เป็นตัวเลขมีค่าอยู่ในช่วง 0 ถึง 255 ถ้า n มีค่าเป็นลบหรือมีค่ามากกว่า 255 จะเกิด "Illegal function call" error ขึ้น ถ้า n มีค่าเป็น 0 หรือมีค่ามากกว่าจำนวนของ items ในรายการ (แต่น้อยกว่าหรือเท่ากับ 255) แล้ว GWBASIC จะดำเนินการต่อไปด้วยคำสั่งปฏิบัติการถัดไป
- มูลค่าของ n บ่งบอกไลน์นัมเบอร์ในรายการที่ซึ่งจะถูกใช้สำหรับการย้าย ตัวอย่างเช่น ถ้า n มีค่าเป็น 3 ไลน์นัมเบอร์ที่สามในรายการจะเป็นจุดหมาย (destination) ของการย้าย
- ในคำสั่ง ON...GOSUB ไลน์นัมเบอร์แต่ละไลน์ในในรายการต้องเป็นไลน์นัมเบอร์แรกของสับรูทีน

ตัวอย่าง 100 ON L-l GOT0 150,300,320,390

โปรแกรมจะย้ายไปยังไลน์ 150 ถ้า L-1 มีค่าเท่ากับ 1 ไปยังไลน์ 300 ถ้า L-l มีค่าเท่ากับ 2 ไปยังไลน์ 320 ถ้า L-l มีค่าเท่ากับ 3 และไปยังไลน์ 390 ถ้า L-1 มีค่าเท่ากับ 4 โปรแกรมจะดำเนินการไปยังคำสั่งถัดไปถ้า L-1 มีค่าเท่ากับ 0 หรือมีค่ามากกว่า 4

ON KEY(n) Statement

รูปแบบ ON KEY(n) **GOSUB** line number

จุดประสงค์ ระบุไลน์นัมเบอร์แรกของสับรูทีนที่จะถูกกระทำเมื่อฟังก์ชันคีย์ที่ถูกระบุหรือเมื่อคีย์กำหนดทิศทางของเคอร์เซอร์ถูกกด

รายละเอียด
- n เป็นหมายเลขของฟังก์ชันคีย์
- line number เป็นหมายเลขของไลน์แรกของสับรูทีนซึ่งจะถูกกระทำเมื่อฟังก์ชันคีย์ที่ถูกระบุหรือเมื่อคีย์กำหนดทิศทางของเคอร์เซอร์ถูกกด
- line number ที่เป็น 0 ทำให้ event trap ไร้ความสามารถ
- คำสั่ง ON KEY จะถูกประมวลผลก็ต่อเมื่อคำสั่ง KEY(n) ON ได้ถูกประมวลผลเพื่อที่จะทำให้ event trapping มีความสามารถ ถ้า event trapping ถูกทำให้มีความสามารถและถ้า line number ในคำสั่ง ON KEY มีค่าไม่เป็น 0 แล้ว GWBASIC จะตรวจเช็คคำสั่งต่าง ๆ เพื่อที่จะดูว่าฟังก์ชันคีย์ที่ถูกระบุหรือคีย์กำหนดทิศทางของเคอร์เซอร์ได้ถูกกด ถ้าเป็นเช่นนั้น GOSUB จะถูกกระทำไปยังไลน์ที่ถูกระบุ
- ถ้าคำสั่ง KEY(n) OFF ได้ถูกประมวลผลสำหรับสำหรับคีย์ที่ถูกระบุแล้ว GOSUB จะไม่ถูกกระทำและและจะไม่ถูกจดจำ
- ถ้าคำสั่ง KEY STOP ได้ถูกประมวลผลสำหรับคีย์ที่ถูกระบุแล้ว GOSUB จะถูกกระทำหลังจากที่คำสั่ง KEY(n) ON ถูกประมวลผล
- เมื่อ event trap เกิดขึ้น (นั่นคือ GOSUB ถูกกระทำ) automatic KEY(n) STOP จะถูกประมวลผลเพื่อว่า recursive trap จะไม่สามารถเกิดขึ้น RETURN จาก trapping subroutine จะกระทำคำสั่ง KEY(n) โดยอัตโนมัติแม้ว่า KEY(n) OFF จะถูกกระทำภายในสับรูทีนอย่างชัดแจ้งก็ตาม
- RETURN line number จากคำสั่ง RETURN อาจจะถูกใช้เพื่อที่จะส่งค่าไปยังไลน์นัมเบอร์ที่ถูกระบุจาก trapping subroutine อย่างไรก็ตาม ใช้รูปแบบของการส่งนี้ด้วยความระมัดระวังเนื่องจาก GOSUBs WHILEs หรือ FORs อื่นใดซึ่ง active อยู่ในขณะที่ trap จะยังคง active อยู่ และ

ข้อผิดพลาด (errors) ต่าง ๆ อย่างเช่น "FOR without NEXT" อาจจะเกิดขึ้นได้

- event trapping ไม่ได้เกิดขึ้นแทนที่เมื่อ GWBASIC ไม่ได้ทำการประมวลผลโปรแกรมและ event trapping จะถูกทำให้ไร้ความสามารถโดยอัตโนมัติเมื่อ error trap เกิดขึ้น
- หมายเหตุ : เมื่อคีย์ถูก trap การเกิดขึ้นของคีย์จะถูกทำลาย ดังนั้นท่านไม่สามารถใช้คำสั่ง INPUT หรือคำสั่ง INPUT$ ต่อ ๆ มาเพื่อที่จะค้นหาคีย์ซึ่งเป็นสาเหตุให้เกิด trap ดังนั้นถ้าท่านปรารถนาที่จะกำหนดค่าฟังก์ชันต่าง ๆ ที่แตกต่างกันไปยังคีย์ต่าง ๆ ที่เฉพาะเจาะจง ท่านต้องกำหนดสับรูทีนที่แตกต่างกันสำหรับคีย์แต่ละคีย์ขึ้นจะเป็นวิธีที่ค่อนข้างจะดีกว่าการที่จะกำหนดฟังก์ชันที่แตกต่างมากมายภายในสับรูทีนเดียวกัน

ตัวอย่าง

```
10 KEY 4,TRONN                    'assigns soft key 4
20 KEY(4) ON                      'enables event trapping
.
.
.
70 ON KEY(4) GOSUB 200
.
.
.
(Function key 4 pressed)
.
.
.
```

```
200                                 'Subroutine for TRON
```

ตัวอย่างนี้เป็นการแสดงสับรูทีนสำหรับฟังก์ชันคีย์ F4

ON PEN Statement

รูปแบบ ON PEN GOSUB line number

จุดประสงค์ ระบุไลน์นัมเบอร์แรกของสับรูทีนที่จะถูกกระทำเมื่อ light pen ถูก activate

รายละเอียด
- line number เป็นไลน์แรกของสับรูทีน
- ถ้า line number มีค่าเป็น 0 แล้ว trap จะถูกทำให้ไร้ความสามารถ
- คำสั่ง ON PEN จะถูกประมวลผลก็ต่อเมื่อคำสั่ง PEN ON ได้ถูกประมวลผลแล้ว ถ้า event trapping ถูกทำให้มีความสามารถและถ้า line number ไม่เป็น 0 แล้ว GWBSIC จะตรวจเช็คระหว่างคำสั่งต่าง ๆ เพื่อที่จะดูว่า pen ได้ถูก activate ถ้ามันถูก activate แล้ว GOSUB จะถูกกระทำไปยังไลน์ที่ถูกระบุ
- ถ้าคำสั่ง PEN OFF ถูกประมวลผล GOSUB จะไม่ถูกกระทำและไม่ถูกจดจำ
- ถ้าคำสั่ง PEN STOP ถูกประมวลผล GOSUB จะไม่ถูกกระทำแต่จะถูกกระทำหลังจากที่คำสั่ง PEN ON ถูกประมวลผล
- เมื่อ event trapping เกิดขึ้น (นั่นคือ GOSUB ถูกกระทำ) automatic PEN STOP จะถูกประมวลผลเพื่อว่าปรากฏการณ์ของ recursive traps จะไม่สามารถเกิดขึ้นแทนที่ RETURN จาก trapping subroutine จะกระทำคำสั่ง PEN ON โดยอัตโนมัติแม้ว่า PEN OFF จะถูกกระทำภายในสับรูทีนอย่างชัดแจ้งก็ตาม
- RETURN line number อาจจะถูกใช้เพื่อที่จะส่งค่าไปยังไลน์นัมเบอร์ที่ระบุเฉพาะจาก trapping subroutine อย่างไรก็ตามใช้มันด้วยความระมัดระวังเนื่องจาก GOSUBs WHILEs หรือ FORs อื่นใดซึ่ง active ใน

ขณะที่ trap จะยังคง active อยู่ และข้อผิดพลาด (errors) ต่าง ๆ อย่างเช่น "FOR without NEXT" อาจจะเกิดขึ้นก็ได้

- event trapping ไม่ได้เกิดขึ้นแทนที่เมื่อ GWABSIC ไม่ได้ทำการประมวลผลโปรแกรมและ trapping จะถูกทำให้ไร้ความสามารถโดยอัตโนมัติเมื่อ error trap เกิดขึ้น

ON STRIG(n) Statement

รูปแบบ ON STRIG(n) GOSUB line number

จุดประสงค์ ระบุไลน์นัมเบอร์แรกของสับรูทีนที่จะถูกกระทำเมื่อจอยสตริคทริกเกอร์ (joystrick trigger) ถูกกด

รายละเอียด
- n เป็นหมายเลขของจอยสตริคทริกเกอร์ line number เป็นไลน์แรกของสับรูทีน ถ้า line number มีค่าเป็นศูนย์แล้ว trap จะถูกทำให้ไร้ความสามารถ
- คำสั่ง ON STRIG จะถูกประมวลผลก็ต่อเมื่อคำสั่ง STRIG ON ได้ถูกประมวลผลแล้ว ถ้า event trapping ถูกทำให้มีความสามารถและถ้า line number ไม่เป็น 0 แล้ว GWBASIC จะตรวจเช็คระหว่างคำสั่งต่าง ๆ เพื่อที่จะดูว่าจอยสตริคทริกเกอร์ได้ถูกกด ถ้ามันถูกกด GOSUB จะถูกกระทำไปยังไลน์ที่ถูกระบุ
- ถ้าคำสั่ง STRIG OFF ถูกประมวลผล GOSUB จะไม่ถูกกระทำและไม่ถูกจดจำ
- ถ้าคำสั่ง STRIG STOP ถูกประมวลผล GOSUB จะไม่ถูกกระทำแต่จะถูกกระทำในไม่ช้าไม่นานที่คำสั่ง STRIG ON ถูกประมวลผล
- เมื่อ event trap เกิดขึ้น (นั่นคือ GOSUB ถูกกระทำ) automatic STRIG STOP จะถูกประมวลผลเพื่อว่า recursive traps ไม่สามารถเกิดขึ้น RETURN จาก trapping subroutine จะกระทำคำสั่ง STRIG ON โดย

อัตโนมัติแม้ว่า STRIG OFF ถูกกระทำภายในสับรูทีนอย่างชัดแจ้งก็ตาม

- RETURN line number อาจจะถูกใช้เพื่อที่จะส่งค่าไปยังไลน์นัมเบอร์ที่ระบุเฉพาะจาก trapping subroutine อย่างไรก็ตามใช้มันด้วยความระมัดระวังเนื่องจาก GOSUBs WHILEs หรือ FORs อื่นใดที่ active อยู่ขณะที่ trap จะยังคง active อยู่ และข้อผิดพลาด (errors) ต่าง ๆ อย่างเช่น "FOR without NEXT" อาจจะเป็นผลลัพธ์ที่ได้
- event trapping ไม่ได้เกิดขึ้นแทนที่เมื่อ GWBASIC ไม่ได้ทำการประมวลผลโปรแกรมและ trapping จะถูกทำให้ไร้ความสามารถโดยอัตโนมัติเมื่อ error trap เกิดขึ้น

OPEN Statement

รูปแบบ OPEN mode,[#]file number,filespec[,record length]

OPEN filespec[FOR mode] AS [#]file number

[LEN=record length]

จุดประสงค์ อนุญาตให้ข้อมูลเข้าหรือข้อมูลออก (I/O) ไปยังไฟล์หรือไปยังอุปกรณ์

รายละเอียด

- mode เป็นนิพจน์สตริงก์ที่ซึ่งอักขระตัวแรกเป็นสิ่งใดสิ่งหนึ่งต่อไปนี้ :
  - O ระบุว่าเป็น sequential output mode
  - I ระบุว่าเป็น sequential input mode
  - R ระบุว่าเป็น random input/output mode
  - A ระบุว่าเป็น sequential output mode และกำหนดตัวชี้ไฟล์ (file pointer) ที่จุดจบของไลน์และบันทึกหมายเลขในฐานะเป็นเร็คคอร์ดสุดท้ายของไฟล์ คำสั่ง PRINT# หรือคำสั่ง WRITE# จะขยาย (append) ไฟล์
- ถ้า mode ถูกละไว้แล้ว default R จะถูกกำหนดขึ้น

- file number มีค่าเป็นจำนวนเต็มระหว่าง 1 และ 15 ตัวเลขจะเกี่ยวข้องกับไฟล์นานเท่านานตราบเท่าที่ไฟล์ถูกเปิดอยู่ (OPEN) ตัวเลขตัวเลขเป็นสัญลักษณ์ที่เชื่อมโยงกับไฟล์ และจะถูกใช้เพื่อที่จะอ้างอิงคำสั่งเกี่ยวกับ disk I/O ไปยังไฟล์
- filespec เป็นนิพจน์สตริงก์สำหรับชื่อของไฟล์ (ดูบทที่ 3)
- record length เป็นเลขจำนวนเต็มซึ่งถ้าถูกประกอบรวมอยู่ด้วยแล้วจะกำหนดความยาวของเร็คคอร์ด (record length) สำหรับ random files อย่าใช้ option นี้กับ sequential files
- record length ไม่สามารถมีค่าเกินค่าสูงสุดที่ถูกกำหนดด้วย /S: option ถ้า record length option ไม่ได้ถูกกำหนด ความยาวที่ถูกกำหนดล่วงหน้าจะเป็น 128 ไบท์
- ดิสค์ไฟล์ต้องถูกเปิดก่อนที่การกระทำ disk I/O ใด ๆ จะสามารถถูกกระทำบนไฟล์นั้น OPEN จัดสรรบัฟเฟอร์สำหรับ I/O ไปยังไฟล์หรือไปยังอุปกรณ์และกำหนดรูปแบบ (mode) ของการเข้าถึงซึ่งจะถูกใช้ร่วมกับบัฟเฟอร์
- หมายเหตุ : ไฟล์สามารถถูกเปิดสำหรับ sequential input หรือถูกเปิดสำหรับ random access บนหมายเลขไฟล์ตั้งแต่ 1 หมายเลขขึ้นไปในแต่ละครั้ง อย่างไรก็ตาม ไฟล์อาจจะถูกเปิดสำหรับ output บนหมายเลขไฟล์เพียง 1 หมายเลขเท่านั้นในแต่ละครั้ง

ตัวอย่าง

```
10 OPEN "I",2,"INVEN"
```

เปิดไฟล์ "INVEN" สำหรับ sequential input

```
10 OPEN "MAILING.DAT" FOR APPEND AS 1
```

เปิดไฟล์ "MAILING.DAT" สำหรับการขยาย

OPEN "COM... Statement

รูปแบบ OPEN "COMn: [speed][,[parity][,[data][,[stop][,RS]
[,CS[n]][,DS[n]][,CD[n]][,BIN][,ASC][,LF]]]]"
AS [#]device number

จุดประสงค์ เปิดและกำหนดค่าเริ่มต้นของ communications channel สำหรับอินพุท/เอ้าพุท

รายละเอียด
- n เป็นหมายเลขของ port ที่จะถูกเปิด
- speed เป็น baud rate ของอุปกรณ์ที่จะถูกเปิดมีหน่วยเป็นบิทต่อวินาที
- parity ระบุ parity ของอุปกรณ์ที่จะถูกเปิด รายการเข้า (entries) ต่าง ๆ ที่ใช้ได้คือ : N(none), E(even) หรือ O(odd)
- stop ระบุ stop bit รายการเข้าต่าง ๆ ที่ใช้ได้คือ : 1, 1.5 หรือ 2
- RS ยับยั้ง (suppress) RTS (Request To Send)
- CS[n] ควบคุม CTS (Clear To Send)
- DS[n] ควบคุม DSR (Data Set Ready)
- CD[n] ควบคุม CD (Carrier Detect)
- BIN เปิดอุปกรณ์ใน binary mode BIN ถูกเลือกโดย default แม้ว่า ASC จะถูกระบุก็ตาม
- ASC เปิดอุปกรณ์ใน ASCII mode
- LF ระบุว่า line feed จะถูกส่งไปหลังจาก carriage return
- device number เป็นหมายเลขของอุปกรณ์ที่จะถูกเปิด
- คำสั่งนี้ต้องถูกประมวลผลก่อนที่อุปกรณ์จะสามารถถูกใช้สำหรับ RS232 communication
- format errors ใด ๆ ในคำสั่งนี้จะส่งผลให้เกิด "Bad File name" error พารามิเตอร์ที่ไม่ถูกต้องจะไม่ถูกแสดง

- "Device timeout" error จะเกิดขึ้นถ้า Data Set Ready (DSR) ไม่ได้ถูกตรวจจับ
- speed, parity, data และ stop option ต้องถูกแสดงรายการตามลำดับที่มันถูกแสดง options ที่ยังคงเหลืออยู่อาจจะถูกแสดงรายการในลำดับใด ๆ ก็ได้หลังจาก option 4 option แรก
- ใน binary mode tabs จะไม่ถูกขยายไปยัง spaces carriage return จะไม่ถูกส่ง (forced) ไปที่ end-of-line และ Ctrl-Z ไม่ได้ถูกปฏิบัติในฐานะเป็น end-of-file เมื่อ channel ถูกปิด Ctrl-Z จะไม่ถูกส่งไปบน RS232 line
- ใน ASC mode นั้น tabs จะถูกขยายไปยัง spaces cat-rage return จะถูกสงไปที่ end-of-line Ctrl-Z จะถูกปฏิบัติในฐานะเป็น end-of-file และ XON/XOFF ถูกทำให้มีความสามารถ เมื่อ channel ถูกปิด Ctrl-Z จะถูกส่งไปที RS232 line
- LF อนุญาตให้ communication files ที่จะถูกพิมพ์บน serial line printer เมื่อ LF ถูกระบุ อักขระ line feed (OAR) จะถูกส่งไปโดยอัตโนมัติหลังจากแต่ละอักขระ carriage return (OCK) สิ่งนี้รวมทั้ง carriage return ที่ถูกส่งไปในฐานะเป็นผลลัพธ์ของการกำหนดความกว้าง สังเกตว่า INPUT# และ LINE INPUT# เมื่อถูกใช้เพื่อที่จะอานจาก COM file ซึ่งถูกเปิดด้วย LF option จะหยุดลงเมื่อมันเห็น carriage return จะไม่รับรู้ (ignoring) line feed
- LF option ถูกระงับ (superseded) โดย BIN option

ตัวอย่าง

```
10 OPEN "COM1:9600,N,8,1,BIN" AS #2
```

เปิด communications channel 1 ด้วยความเร็ว 9,600 baud ด้วย no parity bit 6 data bits และ 1 stop bit อินพุท/เอ้าพุทจะ

อยู่ในรูป binary mode ไลน์อื่น ๆ ในโปรแกรมในขณะนี้อาจจะเข้าถึง channel 1 ตามอุปกรณ์หมายเลข 2 (device number 2)

OPTION BASE Statement

รูปแบบ OPTION BASE n

จุดประสงค์ ประกาศมูลค่าต่ำสุดสำหรับสับสคริพท์ (subscripts) ต่าง ๆ ของอะเรย์

รายละเอียด - n มีค่าเป็น 0 หรือ 1

- default base มีค่าเป็น 0 ถ้าคำสั่ง

  OPTION BASE 1

  ถูกปฏิบัติแล้ว มูลค่าต่ำสุดของสับสคริพท์ที่อาจมีคือ 1

ตัวอย่าง 10 OPTION BASE 1

OUT Statement

รูปแบบ OUT I,J

จุดประสงค์ ส่งไบท์ไปยัง machine output port

รายละเอียด - I เป็นหมายเลข port มีค่าเป็นจำนวนเต็มในช่วง 0 ถึง 65535

- J เป็นข้อมูลที่จะถูกส่งถ่าย (transmitted) มีค่าเป็นจำนวนเต็มในช่วง 0 ถึง 255

- port address map ถูกเสนอแนะในตารางที่ 7.5

ตัวอย่าง 100 OUT 12345,255

มูลค่า 255 ถูกส่งไปที่ output port 12345

PAINT Statement

รูปแบบ PAINT (xstart,ystart)[,paint color[,border color]]

จุดประสงค์ ระบายภาพให้เต็ม (fill image) ด้วยสีที่ถูกระบุ

รายละเอียด
- xstart และ ystart เป็นโคออร์ดิเนตต่าง ๆ ที่ซึ่งการระบายสี (painting) จะเริ่มต้น โดยปกติการระบายสีจะเริ่มต้นบนจุดที่ไม่ใช่ขอบ (non-border point)
- paint color เป็นหมายเลขของสีที่จะถูกวางในพื้นที่ของรูปภาพ ถ้าหาก paint color ไม่ได้ถูกระบุแล้ว foregroung color จะถูกใช้
- border color บ่งชี้สีของขอบของรูปภาพที่จะถูกระบายสีให้เต็ม เมื่อ boder color ถูกพบ การระบายสีของไลน์และทิศทางที่เป็นอยู่ในขณะนั้นจะหยุดลง ถ้า border color ไม่ถูกระบุ paint color จะถูกใช้
- การระบายสีจะเสร็จสมบูรณ์เมื่อ entire line เท่าเทียมกันกับ paint color
- คำสั่งนี้สามารถถูกใช้เพื่อที่จะระบายสีให้เต็มรูปภาพใด ๆ แต่การระบายสีของขอบที่เว้า ๆ แหว่ง ๆ หรือการระบายสีรูปภาพต่าง ๆ ที่ยุ่งยากซับซ้อนอาจจะส่งผลให้เกิด " Out of Memory" error ขึ้น ถ้าสิ่งนี้เกิดขึ้นให้ใช้ CLEAR เพื่อที่จะเพิ่มจำนวนของ stack space ที่สามารถใช้ได้ให้มากขึ้น

ตัวอย่าง

```
10 SCREEN 105
20 CLS
30 CIRCLE (320,160),100,1
40 PAINT (320,160),1
```

วาดวงกลมที่ตำแหน่ง (location) 320,160 ด้วยรัศมี 100 และจากนั้นระบายสีให้เต็ม (fill-in) วงกลมด้วยคำสั่ง PAINT

PEEK Function

รูปแบบ PEEK(I)

จุดประสงค์ ส่งไบท์ที่อ่านจากที่ตั้งหน่วยความจำ (memory loaction) ที่ I

รายละเอียด - มูลค่าที่ถูกส่งมีค่าเป็นจำนวนเต็มอยู่ในช่วง 0 ถึง 255 I ต้องมีค่าอยู่ในช่วง -32768 ถึง 65535 I เป็นภาวะเริ่มแรก (offset) จาก segement ปัจจุบันซึ่งถูกบ่งบอกโดยคำสั่ง DEF SEG ล่าสุด

- ดูคำสั่ง POKE ซึ่งเป็นฟังก์ชันเติมเต็ม (complentary) ของคำสั่งนี้

ตัวอย่าง x = PEEK (1327)

ตัวอย่างนี้เรียกหาไบท์ที่ตำแหน่งหน่วยความจำที่ 1327

PEN Statement and Function

รูปแบบ PEN ON

PEN OFF

PEN STOP

x = PEN(n)

จุดประสงค์ อ่าน light pen และทำให้ trapping the pen มีความสามารถ ไร้ความสามารถ หรือหยุดลง

รายละเอียด - x เป็นตัวแปรจำนวนเลขที่จะรับมูลค่าของ PEN

- n เป็นตัวเลขมีค่าจาก 0 ถึง 9 ฟังก์ชันนี้ดัก (trap) downstrokes ของ light pen โดยการอ่านพารามิเตอร์ต่าง ๆ ของ n ดังต่อไปนี้ :

0 บ่งชี้ว่า pen was down since last poll ส่งค่า -1 ถ้า down 0 ถ้าไม่ใช่ down

1 ส่งค่าโคออร์ดิเนตของ x ของจุดที่ซึ่ง pen ถูกกดครั้งล่าสุด

2 ส่งค่าโคออร์ดิเนตของ y ของจุดที่ซึ่ง pen ถูกกดครั้งล่าสุด

3 ส่งผลค่าของสวิทช์ของ pen ในขณะนั้น -1 ถ้า down 0 ถ้า up

4 ส่งค่าโคออร์ดิเนตของ x ที่ใช้ได้ที่ถูกรับรู้ล่าสุด

5 ส่งค่าโคออร์ดิเนตของ y ที่ใช้ได้ที่ถูกรับรู้ล่าสุด

6 ส่งค่าตำแหน่งของแถว (โรว์) ของอักขระเมื่อ pen ถูกกดครั้งล่าสุด

7 ส่งค่าตำแหน่งของสดมภ์ (คอลัมน์) ของอักขระเมื่อ pen ถูกกดครั้งล่าสุด

8 ส่งค่าแถวของอักขระที่ถูกรับรู้ครั้งล่าสุดที่ซึ่ง pen ถูกกำหนดตำแหน่ง

9 ส่งค่าสดมภ์ของอักขระที่ถูกรับรู้ครั้งล่าสุดที่ซึ่ง pen ถูกกำหนดตำแหน่ง

- PEN ON ทำให้ทั้งฟังก์ชันที่ถูกอ่าน (read function) และการทำกับดัก ของเหตุการณ์ (event trapping) มีความสามารถ
- PEN STOP ทำให้ทั้งฟังก์ชันที่ถูกอ่านของ light pen และการทำกับดักของ เหตุการณ์ไร้ความสามารถแต่จะจดจำเหตุการของ PEN เพื่อว่ามันสามารถถูก ดักในไม่ช้าไม่นานที่มี PEN ON
- PEN OFF ไม่ได้ทำให้ทั้งฟังก์ชันที่ถูกอ่านและการทำกับดักของเหตุการณ์ไร้ ความสามารถเท่านั้น มันไม่ได้จดจำกิจการ (activity) ที่ตามมาด้วย
- ในตอนเริ่มต้นนั้น pen ถูกปิด (off) คำสั่ง PEN ON ต้องถูกประมวลผล ก่อนการเรียกต่าง ๆ ของฟังก์ชันที่ถูกอ่านของ pen ใด ๆ จะสามารถถูกทำ ถ้าฟังก์ชันถูกเรียกเมื่อ pen ถูกปิดอยู่ จะส่งผลให้เกิด "Illegal function call" error ขึ้น
- เช่นเดียวกัน PEN ON ทำให้การทำกับดักของเหตุการณ์มีความสามารถโดย คำสั่ง ON PEN (ดูคำสั่ง ON PEN ในบทนี้) ในขณะที่การทำกับดักถูกทำให้ มีความสามารถและถ้าไลน์นัมเบอร์ที่ไม่ใช่ศูนย์ถูกระบุในคำสั่ง ON PEN แล้ว GWBASIC จะตรวจเช็คระหว่างคำสั่งทุก ๆ คำสั่งเพื่อที่จะดูว่า light pen ได้ถูก activate ถ้ามันถูก activate โปรแกรมจะย้ายไปที่คำสั่ง ON PEN

- pen จะไม่ถูกใช้ในพื้นที่ที่เป็นขอบของจอภาพ มูลค่าต่าง ๆ ที่ถูกส่งจากพื้นที่นั้นจะไม่ถูกต้องละเอียดลออ
- เพื่อเพิ่มความเร็วของการประมวลผลโปรแกรม ใช้ PEN OFF สำหรับโปรแกรมต่าง ๆ ที่ไม่ได้ทำการใช้ light pen

ตัวอย่าง

```
5 CLS
10 PEN ON
20 P=PEN(3)
30 LOCATE 1,1 : PRINT "PEN IS";
40 IF P THEN PRINT "DOWN" ELSE PRINT "UP"
50 GOTO 20
```

ตัวอย่างนี้สร้างลูปที่ไม่สิ้นสุด (infinite loop) เพื่อที่จะพิมพ์สถานะของสวิทช์ของ pen ในปัจจุบัน (UP/DOWN)

PLAY Statement

รูปแบบ PLAY string

จุดประสงค์ เล่นดนตรี

รายละเอียด - string เป็นนิพจน์ที่ประกอบด้วยคำสั่งอักขระเดี่ยว ๆ ต่าง ๆ (single character commands) ดังต่อไปนี้ :

- n กำหนดระยะคั่นระหว่างเสียงดนตรีที่เรียกว่า octave ในขณะนั้น มีทั้งหมด 7 octaves เป็นจำนวนเลขจาก 0 ถึง 6 แต่ละ octave เริ่มต้นด้วย C และจบลงด้วย B C ตอนกลางเริ่มต้น octave ที่ 3 ส่วน default octave คือ octave ที่ 4

A ถึง G กับข้อเลือก #, + หรือ -

เล่นโน้ตที่ถูกบ่งชี้ใน octave ปัจจุบัน โน้ตที่ถูกติดตามด้วยเครื่องหมาย # หรือเครื่องหมาย + บ่งชี้เสียงแหลมคม (sharp) ส่วนโน้ตที่ถูกติดตามด้วยเครื่องหมาย - บ่งชี้เสียงต่ำ (flat) เครื่องหมาย #, + หรือเครื่องหมาย - ต้องสอดคล้องกันกับคีย์สีดำบนเปียนโน

N n โน้ตถูกกำหนดตัวเลขจาก 0 ถึง 84 0 บ่งชี้หยุด (rest) สิ่งนี้เป็นรูปฟอร์มสำรองสำหรับ O n และ A-G

L n กำหนดความยาวของตัวโน้ตต่าง ๆ ที่จะปฏิบัติในช่วง 1 ถึง 64 ความยาวของโน้ตที่แท้จริงคือ 1/n ตารางที่ 7.7 อธิบายมูลค่าต่าง ๆ บางมูลค่าซึ่งสามารถถูกใช้

ตารางที่ 7.7

NOTE LENGTH EQUIVALENTS

Length	Equivalent
L1	whole note
L2	half note
L3	one note of a three-half-note triplet
L4	quarter note
L5	one of a quarter note triplet
.	
.	
64	sixty-fourth note

ตัวอย่างเช่น ความยาวอาจจะติดตามโน้ตด้วยเมื่อท่านต้องการที่จะเปลี่ยนแปลงมันสำหรับโน้ตนั้นเพียงเท่านั้น ดังนั้น A16 จะเท่าเทียมกันกับ L16A

P n หยุด (rest) ช่วงอาจจะเป็นจาก 1 ถึง 64 ดูตารางที่ 7.7 สำหรับความเท่าเทียมกัน

ด็อท (dot) หรือ จุด (period) ข้างหลังโน้ตเป็นเหตุให้มันถูกเล่นในฐานะเป็น dotted note (ความยาวของมันถูกคูณด้วย 3/2) ด็อทมากกว่า 1 ด็อทถูกยอมรับซึ่งจะเป็นเหตุให้ความยาวของโน้ตจะถูกทำให้เพิ่มขึ้นโดย ทวีคูณของ 3/2 ที่เหมาะสม ดังนั้น "C.." เล่น 9/4 นานเท่านานเท่าที่ความยาวของมันบ่งชี้ และ "C..." จะเล่น 27/8 เช่นเดียวกัน ดอทต่าง ๆ อาจจะปรากฏข้างหลัง rest (P) เพื่อที่มันจะถูกกำหนดความยาวในวิธีเดียวกัน

T n จังหวะ (tempo) กำหนดหมายเลขของ quarter notes ใน 1 นาที ในช่วง 32 ถึง 255 คำสั่ง SOUND แสดงรายการจังหวะทั่วไป (common tempos) ต่าง ๆ และการเคาะจังหวะ (beats) ต่าง ๆ ที่สมนัยกันต่อนาที default จะเป็น 120

MF Music foreground เสียงแต่ละเสียงเริ่มต้นหลังจากเสียงก่อนหน้าได้เสร็จสิ้นแล้วเท่านั้น สิ่งนี้เป็น default จะใช้ได้สำหรับ SOUND ด้วย

MB Music background เสียงแต่ละเสียงถูกบันทึกไปยังบัฟเฟอร์ซึ่งอนุญาตให้โปรแกรมดำเนินการประมวลผลต่อไปในระหว่างที่ดนตรีเล่นใน background จำนวนเลขสูงสุดของโน้ตต่าง ๆ (หรือ rests) ซึ่งสามารถถูกเล่นใน background ในแต่ละครั้งจะเป็น 32 ข้อเลือกนี้ถูกต้องใช้ได้ (valid) สำหรับ SOUND ด้วย

- MN Music normal (nonlegato) สั่งนี้เป็นการกำหนดค่าที่เป็น default เมื่อ ML หรือ MS ทั้งสองไม่ถูกระบุ แต่ละโน้ตถูกเล่นที่ 7/8L ที่จะเป็นการสร้าง nonlegato sound ขึ้น

- ML Music legato โน้ตแต่ละโน้ตเล่นความยาวเต็ม (full length) ที่กำหนดด้วย L ซึ่งสร้างโน้ตต่าง ๆ ที่เชื่อมต่อกัน (connected notes)

- MS Music staccato โน้ตแต่ละโน้ตเล่นที่ 3/4L เพื่อที่จะสร้าง strong disconnected sound

- X variable;

ประมวลผลสตริงก์ที่ถูกระบุ

- สังเกตว่าเนื่องจาก slow clock หยุดชะงัก rate โน้ตบางโน้ตจะไม่เล่นที่ high tempos ตัวอย่างเช่น L64 ที่ T255 ท่านอาจจะค้นพบการประกอบรวมกันต่าง ๆ อื่นใดตามที่ท่านทดลองด้วยความเป็นไปได้ต่าง ๆ ของคำสั่งนี้

ตัวอย่าง

```
10 REM CHIMES OF BIG BEN
20 PLAY "MBMLO3L4ECDO2MSL1GMLL4GO3DEMSL1C"
```

ตัวอย่างนี้เล่นเพลง "Sounds heard in London"

POINT Function

รูปแบบ POINT (xcoordinate,ycoordinate)

จุดประสงค์ อ่านมูลค่าของสีของจุดบนจอภาพ

รายละเอียด
- xcoordinate และ ycoordinate เป็นโคออร์ดิเนตต่าง ๆ ของจุดของจอภาพซึ่งจะถูกอ้างอิงถึง
- ถ้าจุดที่ถูกระบุอยู่นอกช่วง มูลค่า -1 จะถูกส่งค่าไป

ตัวอย่าง 5 SCREEN 2

```
10 IF POINT(i,i)<>0 THEN PRESET (i,i) ELSE PSET(i,i)
20 PSET (i,i),1-POINT(i,i)
```

ตัวอย่างนี้แสดงวิธี 2 วิธีที่จะสลับ (invert) จุด (i,i)

POKE Statement

รูปแบบ POKE i,j

จุดประสงค์ บันทึกข้อมูลสู่ตำแหน่งของหน่วยความจำ

รายละเอียด
- i และ j เป็นนิพจน์จำนวนเต็ม
- i เป็นแอดเดรสของหน่วยความจำซึ่งต้องมีค่าอยู่ในช่วง -32768 ถึง 65535 i เป็นภาวะเริ่มแรกจาก segment ปัจจุบันซึ่งถูกกำหนดโดยคำสั่ง DEF SEG ล่าสุด (ดูคำสั่ง DEF SEG)
- j เป็น data byte
- ใช้คำสั่งนี้ด้วยความระมัดระวัง มันจะเป็นเหตุให้เกิดปัญหาต่าง ๆ ที่เข้มงวดถ้ามันถูกใช้ไม่ถูกต้อง
- ฟังก์ชันเติมเต็มของ POKE คือ PEEK (ดูฟังก์ชัน PEEK)

ตัวอย่าง
```
10 POKE &H5A00,&HFF
```

ตัวอย่างนี้ใส่ FF hex ลงสู่ 5A00 hex ใน segment ปัจจุบัน

POS Function

รูปแบบ v = POS(n)

จุดประสงค์ ส่งค่าตำแหน่งของสคแมท์ของเคอร์เซอร์ที่เป็นอยู่ในขณะนั้น

รายละเอียด
- n เป็นดัมมีอาร์กิวเมนต์ (dummy argument)

- มูลค่าที่ถูกส่งไปจะอยู่ในช่วง 1 ถึง 40 หรือในช่วง 1 ถึง 80 ขึ้นอยู่กับการกำหนดค่าของ WIDTH ในขณะนั้น CSRLIN หาตำแหน่งของแถวของเคอร์เซอร์
- ดูฟังก์ชัน LPOS ด้วย

ตัวอย่าง IF POS(0)>50 THEN BEEP

สัญญาณเสียงจะถูกเปล่งขึ้นถ้าเคอร์เซอร์อยู่ถัดจากตำแหน่งที่ 50 บนจอภาพ

PRESET Statement .

รูปแบบ PRESET (xcoordinate,ycoordinate)[,color]

จุดประสงค์ วาดจุดที่ถูกระบุบนจอภาพ

รายละเอียด
- xcoordinate และ ycoordinate ระบุจุดที่จะถูกกำหนด
- color เป็นหมายเลขของสีที่จะถูกใช้สำหรับจุดที่ถูกระบุ
- PRESET ทำงานเหมือนกับ PSET ยกเว้นว่า ถ้า color ถูกระบุแล้ว background color จะถูกเลือก
- ถ้าโคออร์ดิเนตออกนอกช่วงที่ถูกกำหนดให้ จะไม่มีการกระทำใด ๆ เกิดขึ้น ไม่มีข้อความผิดพลาดถูกกำหนดให้

ตัวอย่าง
```
5 REM DRAW A LINE FROM (0,0) TO (100,100)
10 FOR I=0 TO 100
20 PRESET (I,I),1
30 NEXT
35 REM NOW ERASE THAT LINE
40 FOR I=0 TO 100
50 PRESET STEP (-1,-1)
60 NEXT
```

ตัวอย่างนี้วาดไลน์จากจุด (0,0) ถึงจุด (100,100) และจากนั้นทำการลบมันออกไปโดยทำการเขียนทับ (overwriting) มันด้วย background color

PRINT Statement

รูปแบบ PRINT [list of expressions]

จุดประสงค์ แสดงผลข้อมูลบนจอภาพ

รายละเอียด - ถ้า list of expressions ถูกละแล้วไลน์ว่างเปล่า (blank line) 1 ไลน์จะถูกพิมพ์ ถ้า list of expressions ถูกประกอบรวมอยู่ด้วยมูลค่าต่าง ๆ ของนิพจน์ต่าง ๆ จะถูกพิมพ์ที่เทอร์มินัล นิพจน์ต่าง ๆ ในรายการอาจจะเป็นนิพจน์จำนวนเลขและ/หรือนิพจน์สตริงก์ (สตริงก์ต่าง ๆ ต้องถูกปิดล้อมอยู่ภายในเครื่องหมายคำพูด)

- <u>ตำแหน่งการพิมพ์</u>

ตำแหน่งของ item แต่ละ item ที่จะถูกพิมพ์ (printed item) ถูกกำหนดโดยเครื่องหมายวรรคตอนซึ่งถูกใช้เพื่อที่จะแบ่งแยก items ในรายการ GWBASIC แบ่งไลน์ออกเป็นโซนของการพิมพ์ (print zones) ต่าง ๆ ของ 14 ช่องว่าง (spaces) ในแต่ละโซน ใน list of expressions เครื่องหมายจุลภาคเป็นเหตุให้มูลค่าถัดไปจะถูกพิมพ์ที่จุดเริ่มต้นของโซนถัดไป เครื่องหมายเซมิโคลอนเป็นเหตุให้มูลค่าถัดไปจะถูกพิมพ์ทันทีหลังจากมูลค่าล่าสุด การพิมพ์ช่องว่างตั้งแต่ 1 ช่องขึ้นไประหว่างนิพจน์ต่าง ๆ จะมีผลเหมือนกันกับการพิมพ์การพิมพ์เซมิโคลอน

- ถ้าเครื่องหมายจุลภาคหรือเครื่องหมายเซมิโคลอนสิ้นสุด list of expressions แล้วคำสั่ง PRINT ถัดไปจะเริ่มต้นการพิมพ์บนไลน์เดียวกันโดยการเว้นที่ว่าง (spacing) ถ้า list of expressions สิ้นสุดลงโดยปราศจากเครื่องหมายจุลภาคหรือปราศจากเครื่องหมายเซมิโคลอน

แล้ว carriage return จะถูกพิมพ์ที่ตอนท้ายของไลน์ ถ้าไลน์ที่ถูกพิมพ์มีความยาวมากกว่าความกว้างของเทอร์มินัลแล้ว GWBASIC จะไปยัง physical line ถัดไปและดำเนินการพิมพ์ต่อไป

- จำนวนเลขต่าง ๆ ที่ถูกพิมพ์โดยปกติถูกติดตามด้วยช่องว่าง 1 ที่ จำนวนบวกต่าง ๆ ถูกนำหน้าด้วยช่องว่าง 1 ที่ จำนวนลบต่าง ๆ ถูกนำหน้าด้วยเครื่องหมายลบ ตัวเลขต่าง ๆ ในรูป single precision ซึ่งสามารถถูกแสดงด้วยตัวเลขน้อยกว่าหรือเท่ากับ 6 หลักในรูป unscaled format มีความถูกต้องไม่น้อยกว่าที่มันสามารถถูกแสดงในรูป scaled format เป็นผลลัพธ์โดยการใช้ unscaled format ตัวอย่างเช่น 1E-7 เป็นผลลัพธ์ดังนี้คือ .0000001 และ 1E-8 เป็นผลลัพธ์ดังนี้คือ 1E-8 ตัวเลขต่าง ๆ ในรูป double precision ซึ่งสามารถถูกแสดงด้วยตัวเลขน้อยกว่าหรือเท่ากับ 16 หลักในรูป unscaled format มีความถูกต้องไม่น้อยกว่าที่มันสามารถถูกแสดงในรูป scaled format จะเป็นผลลัพธ์โดยการใช้ unscaled format ตัวอย่างเช่น ID-15 เป็นผลลัพธ์ดังนี้คือ .00000000 00000001 และ 1D-16 เป็นผลลัพธ์ดังนี้คือ 1D-16
- ดูคำสั่ง LPRINT และคำสั่ง LPRINT USING ด้วย

ตัวอย่าง

```
10 x=5
20 PRINT X+5, X-5, X*(-5), X^5
30 END
RUN
 10                          -25.
Ok
```

เครื่องหมายจุลภาคในคำสั่ง PRINT เป็นเหตุให้มูลค่าที่จะถูกพิมพ์แต่ละมูลค่าอยู่ที่จุดเริ่มต้นของโซนของการพิมพ์ถัดไป

```
LIST
10 INPUT X
20 PRINT X "SQUARED IS" X^2 "AND";
30 PRINT X "CUBED IS" X^3
40 PRINT
50 GOTO 10
Ok
RUN
? 9
 9 SQUARED IS 81 AND 9 CUBED IS 729

? 21
 21 SQUARED IS 441 AND 21 CUBED IS 9261

?
```

เครื่องหมายเซมิโคลอนที่ตอนท้ายของไลน์ 20 เป็นเหตุให้คำสั่ง PRINT ทั้งสอง ถูกพิมพ์บนไลน์เดียวกัน ไลน์ 40 เป็นเหตุให้ไลน์ว่างเปล่าถูกพิมพ์ก่อนข้อความบ่งเตือน (prompt) ถัดไป

```
10 FOR X = 1 TO 5
20 J=J+5
30 K=K+10
40 ?J;K;
50 NEXT X
```

```
Ok
RUN
 5  10  10  20  15  30  20  40  25  50
Ok
```

เครื่องหมายเซมิโคลอนในคำสั่ง PRINT เป็นเหตุให้มูลค่าที่จะถูกพิมพ์แต่ละมูลค่าถูกพิมพ์ทันทีหลังจากมูลค่าที่อยู่ก่อนหน้า (อย่าลืมว่าโดยปกติตัวเลขต้องถูกติดตามด้วยช่องว่าง จำนวนบวกต่าง ๆ ถูกนำหน้าด้วยช่องว่าง) ในไลน์ 40 เครื่องหมาย ? ถูกใช้แทนคำว่า PRINT

PRINT USING Statement

รูปแบบ PRINT USING v$; list of expressions [;]

จุดประสงค์ พิมพ์สตริงก์หรือตัวเลขต่าง ๆ โดยการใช้รูปแบบ (format) ที่ถูกระบุ

รายละเอียด
- v$ เป็นค่าคงที่สตริงก์หรือเป็นตัวแปรสตริงก์ที่ประกอบด้วยอักขระที่จะกำหนดรูปแบบพิเศษต่าง ๆ (special formatting characters) ซึ่งบ่งบอก (determine) ฟิลด์และรูปแบบของสตริงก์ต่าง ๆ หรือตัวเลขต่าง ๆ ที่จะถูกพิมพ์ ดู String Fields และ Numeric Fields ข้างล่าง
- list of expressions ประกอบด้วยนิพจน์สตริงก์หรือนิพจน์จำนวนเลขต่าง ๆ ที่จะถูกพิมพ์ถูกแบ่งแยกโดยเครื่องหมายเซมิโคลอน
- String Fields

  เมื่อทำการพิมพ์สตริงก์ต่าง ๆ ด้วย PRINT USING อักขระ 1 ใน 3 ตัวอาจจะถูกใช้เพื่อที่จะกำหนดรูปแบบฟิลด์ของสตริงก์ :

  ! ระบุว่าเฉพาะอักขระแรกในสตริงก์ที่ถูกกำหนดให้เท่านั้นจะถูกพิมพ์

"n\spaces\" ระบุว่า 2+n อักขระจากสตริงก์จะถูกพิมพ์ ถ้าเครื่องหมาย \\ ถูกพิมพ์โดยปราศจาก spaces แล้วอักขระ 2 ตัวจะถูกพิมพ์ กับ 1 space อักขระ 3 ตัวจะถูกพิมพ์ และเป็นเช่นนี้เรื่อยไป

- ถ้าสตริงก์มีความยาวมากกว่าฟิลด์ อักขระที่เกินไปจะไม่ถูกรับรู้ ถ้าฟิลด์มีความยาวมากกว่าสตริงก์ สตริงก์จะถูกเก็บชิดซ้าย (left-justified) ในฟิลด์และที่ว่างต่าง ๆ จะถูกใส่ให้เต็มฟิลด์ทางขวามือ

ตัวอย่าง :

```
10 A$="LOOK": B$="OUT"
20  PRINT-USING  "!";A$;B$
30 PRINT USING "\ \";A$;B$
50  PRINT  USING "\   \";A$;B$;"!!"
RUN
 LO
 LOOKOUT
 LOOK OUT  !!
```

& ระบุ variable-length string filed เมื่อฟิลด์ถูกระบุด้วย & สตริงก์จะถูกพิมพ์โดยปราศจากการปรับปรุงแก้ไข (modification)

ตัวอย่าง :

```
10 A$="LOOK": B$="OUT"
20  PRINT  USING  "!";A$;
30  PRINT  USING "&";B$
RUN
LOUT
```

- Numeric Fields

เมื่อทำการพิมพ์ตัวเลขต่าง ๆ ด้วย PRINT USING อักขระพิเศษต่าง ๆ ต่อไปนี้อาจจะถูกใช้เพื่อที่จะกำหนดรูปแบบฟิลด์จำนวนเลข (numeric field) :

แทนแต่ละตำแหน่งตัวเลข (digit position) ตำแหน่งตัวเลขต่าง ๆ โดยปกติจะถูกใส่ให้เต็ม (filled) ถ้าตัวเลขที่จะถูกพิมพ์มีตัวเลขต่าง ๆ น้อยกว่าตำแหน่งที่ถูกระบุแล้ว ตัวเลขจะถูกพิมพ์ชิดขวา (right-justified) ในฟิลด์ (ถูกนำหน้าด้วยช่องว่างต่าง ๆ)

. จุดทศนิยม (decimal point)

อาจจะถูกใส่เข้าไปในตำแหน่งใด ๆ ในฟิลด์ ถ้าสตริงก์กำหนดรูปแบบ (format string) ระบุว่าตัวเลขจะนำหน้าจุดทศนิยม โดยปกติตัวเลขจะถูกพิมพ์ (เป็น 0 ถ้าจำเป็น) ตัวเลขต่าง ๆ จะถูกปัดเศษแบบ (rounded) ตามความจำเป็น

ตัวอย่าง :

```
PRINT  USING "##.##";.78
 0.78

PRINT USING "###.##";978.654
 978.65

PRINT USING "##.##   ";10.2,5.3,66.789,.234
 10.20    5.30   66.79    0.23
```

ในตัวอย่างสุดท้าย ช่องว่าง 3 ที่จะถูกใส่ไปที่ตอนท้ายของสตริงก์กำหนดรูปแบบเพื่อที่จะแบ่งแยกมูลค่าต่าง ๆ ที่ถูกพิมพ์บนไลน์

+ ที่จุดเริ่มต้นหรือที่ตอนท้ายของสตริงก์กำหนดรูปแบบจะเป็นเหตุให้เครื่อง-หมายของตัวเลข (+ หรือ -) ถูกพิมพ์ก่อนหรือหลังตัวเลข

- ที่ตอนท้ายของสตริงก์กำหนดรูปแบบจะเป็นเหตุให้จำนวนลบต่าง ๆ ถูกพิมพ์ด้วยเครื่องหมายลบที่ที่อยู่ข้างท้าย

ตัวอย่าง :

```
PRINT USING "+##.##   ";-68.95,2.4,55.6,-.9
 -68.95    +2.40   +55.60    -0.90

PRINT USING "##.##-   ";-68.95,22.449,-7.01
 68.95-   22.45    7.01-
```

** ที่จุดเริ่มต้นของสตริงก์กำหนดรูปแบบเป็นเหตุให้ช่องว่างต่าง ๆ ที่อยู่ข้างหน้าในฟิลด์จำนวนเลขถูกใส่ให้เต็มด้วยเครื่องหมายดอกจัน (*) เช่นเดียวกัน ** ระบุตำแหน่งต่าง ๆ ของตัวเลขมากกว่า 2 ตัวขึ้นไป

ตัวอย่าง :

```
PRINT USING "**#.# ";12.39,-0.9,765.1
 *12.4   *.0.9   765.1
```

$$ เป็นเหตุให้เครื่องหมายดอลลาร์ (dollar sign) ถูกพิมพ์ทันทีติดกับตัวเลขที่ถูกกำหนดรูปแบบตัวที่อยู่ทางซ้ายมือสุด $$ ระบุตำแหน่งของตัวเลขมากกว่า 2 หลักขึ้นไป 1 ตัวของ $$ จะเป็นเครื่องหมาย $ รูปแบบเอกโปเนนเชียลไม่สามารถถูกใช้กับ $$ จำนวนลบต่าง ๆ ไม่สามารถถูกใช้แม้ว่าเครื่องหมายลบจะอยู่ข้างท้ายทางขวามือก็ตาม

ตัวอย่าง :

```
PRINT USING "$$###.##";456.78
```

$456.78

**$

ที่จุดเริ่มต้นของสตริงก์กำหนดรูปแบบรวมผลของสัญลักษณ์ทั้ง 2 ตัว ข้างบนเข้าด้วยกัน ช่องว่างต่าง ๆ ที่อยู่ข้างหน้าจะถูกใส่ให้เต็มด้วยเครื่องหมายดอกจัน และเครื่องหมายดอลลาร์จะถูกพิมพ์ก่อนตัวเลข **$ ระบุตำแหน่งตัวเลขมากว่า 3 หลัก 1 หลักจะเป็นเครื่องหมายดอลลาร์

ตัวอย่าง :

```
PRINT USING "**$##.##";2.34
 ***$2.34
```

เครื่องหมายจุลภาค (comma) ที่อยู่ทางด้านซ้ายของจุดนิยมในสตริงก์กำหนดรูปแบบเป็นเหตุให้เครื่องหมายจุลภาคถูกพิมพ์ไปทางด้านซ้ายของทุก ๆ ตัวเลขที่สามทางด้านซ้ายของจุดทศนิยม เครื่องหมายจุลภาคที่ตอนท้ายของสตริงก์กำหนดรูปแบบถูกพิมพ์ในฐานะเป็นส่วนหนึ่งของสตริงก์ เครื่องหมายจุลภาคระบุตำแหน่งของตัวเลขอื่น ๆ มันไม่มีผลถ้าถูกใช้กับรูปแบบเอ็กโปเนนเชียล (^^^^)

ตัวอย่าง :

```
PRINT USING "####,.##";1234.5
 1.234.50


PRINT USING "####.## ";1234.5
 1234.50,
```

^^^^ เครื่องหมายตก (carets)

อาจจะถูกวางไว้ข้างหลังอักขระกำหนดตำแหน่งตัวเลขต่าง ๆ เพื่อที่จะระบุรูปแบบเอกโพเนนเชียล เครื่องหมายตก 4 ตัว (^^^^) ยอมให้มีช่องว่างสำหรับ E+xx หรือ D+xx ถูกพิมพ์ ตัวเลขนันสำคัญต่าง ๆ จะถูก left-justified และเลขยกกำลัง (exponent) จะถูกปรับ-ปรุงแก้ไขให้เหมาะสม แม้ว่าเครื่องหมาย + ที่อยู่ข้างหน้า หรือเครื่อง-หมาย + หรือ - ที่อยู่ข้างท้ายจะถูกระบุก็ตาม ตำแหน่งตัวเลข 1 ตำแหน่งจะถูกใช้ทางด้านซ้ายของจุดทศนิยมเพื่อที่จะพิมพ์ช่องว่างหรือพิมพ์เครื่องหมายลบ

ตัวอย่าง :

```
PRINT USING "##.##^^^^";234.56
 2.35E+02

PRINT USING ".####^^^^-";-888888
 .8889E+06-

PRINT USING "+.##^^^^";123
 +.12E+03
```

_ (underscore) ในสตริงก์กำหนดรูปแบบเป็นเหตุให้อักขระถัดไปจะเป็นผลลัพธ์ในฐานะเป็น literal character

ตัวอย่าง :

```
PRINT USING "_!##.##_!";12.34
 !12.34!
```

ตัวของ literal character เองอาจจะเป็น underscore โดย

การแทน "_" ในสตริงก์กำหนดรูปแบบ

% ถูกพิมพ์ข้างหน้าตัวเลขถ้าตัวเลขที่จะถูกพิมพ์มีค่าใหญ่กว่าฟิลด์จำนวนเลขที่ถูกระบุ ถ้าการปัดเศษเป็นเหตุให้ตัวเลขมีค่ามากเกินกว่าฟิลด์แล้ว เครื่องหมาย % จะถูกพิมพ์ข้างหน้าตัวเลขที่ถูกปัดเศษนั้น

ตัวอย่าง :

```
PRINT USING "##.##";111.22
 %111.22

PRINT USING ".##";.999
 %1.00
```

ถ้าจำนวนของตัวเลขต่าง ๆ ที่ถูกระบุมีค่าเกินกว่า 24 ตัวแล้วจะส่งผลให้เกิด Illegal function call" error ขึ้น

PRINT # and PRINT # USING Statements

รูปแบบ PRINT #filenum, [USING v$;] list of expressions

จุดประสงค์ บันทึกข้อมูลเรียงตามลำดับไปยังไฟล์

รายละเอียด
- filenum เป็นตัวเลขที่ถูกใช้เมื่อไฟล์ถูกเปิด
- v$ ประกอบด้วยอักขระต่าง ๆ ที่จะกำหนดรูปแบบดังถูกอธิบายในคำสั่ง PRINT USING
- list of expressions ประกอบด้วยนิพจน์จำนวนเลขและ/หรือนิพจน์สตริงก์ที่จะถูกเขียนไปยังไฟล์
- PRINT # ไม่ได้อัด (compress) ข้อมูลบนไฟล์ ภาพ (image) ของข้อมูลจะถูกเขียนไปยังไฟล์เพียงเพื่อมันจะถูกพิมพ์บนจอภาพด้วยคำสั่ง PRINT ด้วยเหตุผลนี้ ข้อควรระวังควรจะเกิดขึ้นเพื่อที่จะจำกัดขอบเขตข้อมูลบนไฟล์

ดังถูกแสดงรายละเอียดข้างล่าง เพื่อว่ามันจะเป็นข้อมูลเข้าอย่างถูกต้อง จากไฟล์

- ใน list of expressions นิพจน์จำนวนเลขต่าง ๆ จะถูกจำกัดขอบเขตโดยเซมิโคลอน ตัวอย่างเช่น :

```
PRINT #1,A;B;C;X;Y;Z
```

(ถ้าเครื่องหมายจุลภาคถูกใช้ในฐานะเป็นตัวจำกัดขอบเขตแล้ว ที่ว่างพิเศษ (extra blanks) ต่าง ๆ ที่ถูกใส่เข้าไประหว่างฟิลด์การพิมพ์ต่าง ๆ จะถูกบันทึกไปยังไฟล์ด้วย)

- นิพจน์สตริงก์ต่าง ๆ ต้องถูกแบ่งแยกโดยเครื่องหมายเซมิโคลอนในรายการ เพื่อที่จะกำหนดรูปแบบนิพจน์สตริงก์ต่าง ๆ อย่างถูกต้องบนดิสค์ ใช้ตัวจำกัดขอบเขตที่แจ้งชัดใน list of expressions

ตัวอย่างเช่น ให้ A$="CAMERA" และ B$="93604-1" คำสั่ง

```
PRINT#1,A$;B$
```

จะบันทึก

```
CAMERA93604-1
```

ไปยังไฟล์ เนื่องจากว่าไม่มีตัวจำกัดขอบเขตต่าง ๆ คำสั่งนี้จะไม่เป็นข้อมูลเข้าในฐานะเป็นสตริงก์ที่แตกต่างกัน 2 สตริงก์ เพื่อที่จะแก้ไขปัญหาให้ถูกต้อง ให้ใส่ตัวจำกัดขอบเขตที่แจ้งชัดไปในคำสั่ง PRINT# ดังต่อไปนี้ :

```
PRINT#1,A$;",";B$
```

จะบันทึก

```
CAMERA,93604-1
```

ไปยังไฟล์ ซึ่งสามารถถูกอ่านกลับมาไปสู่ตัวแปรสตริงก์ 2 ตัว

- ถ้าตัวสตริงก์ต่าง ๆ เองบรรจุเครื่องหมายจุลภาค เครื่องหมายเซมิโคลอน ที่ว่างต่าง ๆ ที่อยู่ข้างหน้าที่มีนัยสำคัญ carriage returns หรือ line

feed ใช้ CHR$(34) เพื่อที่บันทึกข้อมูลทั้งหลายไปยังไฟล์ด้วยเครื่องหมาย

คำพูด

ตัวอย่างเช่น ให้ A$="CAMERA, AUTOMATIC" และ B$="93604-1"

คำสั่ง

```
PRINT #1,A$;B$
```

จะบันทึกสิ่งต่อไปนี้ไปยังไฟล์ :

```
CAMERA,  AUTOMATIC  93804-l
```

และคำสั่ง

```
INPUT #1,A$,B$
```

จะใส่ข้อมูล "CAMER" ไปยัง A$ และ "AUTOMATIC 93604-l" ไปยัง B$

เพื่อที่จะแบ่งแยกสตริงก์ต่าง ๆ เหล่านี้ในไฟล์โดยเฉพาะ ใช้ CHR$(34)

เพื่อที่จะบันทึกเครื่องหมายคำพูดคู่ไปยังไฟล์ คำสั่ง

```
PRINT #1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
```

บันทึกภาพดังต่อไปนี้ :

```
"CAMERA,  AUTOMATIC""  93604-1"
```

และคำสั่ง

```
INPUT #1,A$,B$
```

จะใส่ข้อมูล "CAMERA, AUTOMATIC" ไปยัง A$ และ "93604-l" ไปยัง

B$

- คำสั่ง PRINT # อาจจะถูกใช้ร่วมกับ USING option เพื่อที่จะควบคุม

รูปแบบของไฟล์ ตัวอย่างเช่น :

```
PRINT #1,USING"$$###.##,";J:K;L
```

- ดูคำสั่ง WRITE

- ดูภาคผนวก ก, Sequential and Random Files

PSET Statement

รูปแบบ PSET (x coordinate,y coordinate)[,color]

จุดประสงค์ กำหนดจุดบนจอภาพ

รายละเอียด - x coordinate และ y coordinate ระบุจุดบนจอภาพ

- color เป็นหมายเลขของสีที่จะถูกใช้

- เมื่อ GWBASIC scan มูลค่าโคออร์ดิเนตต่าง ๆ โดยจะยอมให้มูลค่าเหล่านั้นอยู่นอกขอบของจอภาพ (ขนาดของจอภาพสามารถถูกปรับปรุงแก้ไขให้เหมาะสมด้วยคำสั่ง WIDTH) อย่างไรก็ตาม มูลค่าต่าง ๆ ที่อยู่นอกช่วง -32768 ถึง 32767 จะเป็นเหตุให้เกิด "Overflow" error ขึ้น

- PSET อนุญาตให้สีถูกละทิ้ง(left off) จากไลน์คำสั่ง ถ้าสีนั้นถูกละทิ้งแล้ว default จะเป็น foreground color

ดูคำสั่ง PRESET

ตัวอย่าง

```
5 REM DRAW A LINE FROM (0,0) TO (100,100)
10 FOR i=0 TO 100
20 FSET (i,i)
30 NEXT

35 REM NOW ERASE THAT LINE
40 FOR i=0 TO 100
50 PSET STEP (-1,-1),0
60 NEXT
```

ตัวแย่างนี้วาดไลน์จากจุด (0,0) ไปยังจุด (100,100) และจากนั้นลบไลน์นั้นโดยการบันทึกทับ (overwriting) ส่วนนั้นด้วย background color

PUT Statement (Files)

รูปแบบ PUT [#] filenum [,number]

จุดประสงค์ เพื่อที่จะบันทึกเร็คคอร์ดจาก random buffer ไปยัง random file

รายละเอียด - filenum เป็นตัวเลขภายใต้ไฟล์ที่ถูกเปิด

- number เป็นตัวเลขสำหรับเร็คคอร์ดที่จะถูกบันทึก มีค่าอยู่ในช่วง 1 ถึง 32767 ถ้าละทิ้งแล้ว เร็คคอร์ดจะถูกบันทึกในหมายเลขเร็คคอร์ดที่สามารถเรียกได้โดยลำดับถัดไป (หลังจาก PUT ล่าสุด)

- PRINT #, PRINT # USING, WRITE #, LSET และ RSET อาจจะถูกใช้เพื่อที่จะใส่อักขระต่าง ๆ ในบัฟเฟอร์ของ random file ก่อนที่คำสั่ง PUT จะถูกประมวลผล กับคำสั่ง WRITE # แล้ว GWBASIC จะทำการเพิ่มที่ว่างต่าง ๆ ในบัฟเฟอร์จนกระทั่งถึง carriage return

- การอ่านหรือการบันทึกผ่านจุดจบของบัฟเฟอร์ จะเกิด "Field Overflow" error ขึ้น

- GWBASIC อาจจะกั้น (buffer) ข้อมูลที่ถูกบันทึกไปยังไฟล์และหน่วงเหนี่ยว (defer) disk access จนกว่ามันจะถูกกั้นไว้ 512 อักขระ

- สำหรับ communication files แล้ว ขนาดจำนวนของไบท์ต่าง ๆ ที่จะถูกบันทึกไปยังไฟล์ได้โดยมีข้อมูลไม่เกินมูลค่าที่กำหนดโดยเงื่อนไขของ /S: switch บนคำสั่ง GWBASIC

PUT Statement (Graphics)

รูปแบบ PUT (x,y) ,array [,action]

จุดประสงค์ สร้างภาพของกราฟฟิค (graphic image) ไปบนพื้นที่ของจอภาพที่ถูกระบุ

รายละเอียด - (x,y) ระบุจุดที่ซึ่งภาพที่ถูกเก็บจะถูกเพิ่มพ่นจอภาพ จุดที่ถูกระบุเป็นโคออร์ดิเนตของบนด้านซ้ายสุด (top left corner) ของภาพ ถ้าภาพ

ที่จะถูกเคลื่อนย้ายใหญ่มากเกินกว่าที่จะเหมาะสมกับขนาดบนจอภาพแล้ว จะเกิด "Illegal function call" error ขึ้น

- array เป็นชื่อของอะเรย์จำนวนเลข (numeric array) ซึ่งบรรจุข้อมูลที่จะถูกเคลื่อนย้าย ข้อมูลที่เฉพาะเกี่ยวกับอะเรย์จะถูกกำหนดให้ในคำสั่ง GET (Graphics)
- action เป็น PSET, PRESET, AND, OR หรือไม่ก็เป็น XOR อย่างใดอย่างหนึ่ง default คือ XOR
- PSET เคลื่อนย้ายข้อมูลจากอะเรย์ไปบนจอภาพ
- PRESET ให้ผลเช่นเดียวกันกับ PSET เว้นแต่ว่า negative image (black on white) จะถูกสร้างขึ้นแทน
- AND ถูกใช้เมื่อภาพจะถูกเคลื่อนย้ายและก็ต่อเมื่อภาพปรากฏอยู่เรียบร้อยแล้วบนจอภาพเท่านั้น
- OR วางทับอยู่บนยอด (superimpose) ภาพไปบนภาพที่ปรากฏอยู่
- XOR เป็นโหมดพิเศษที่ถูกใช้บ่อยครั้ง ซึ่งเป็นเหตุให้จุดต่าง ๆ บนจอภาพถูกทำให้กลับตรงข้ามกัน (inverse) กับจุดที่ซึ่งปรากฏอยู่ในภาพของอะเรย์ การกระทำนี้เหมือนกันอย่างมากกับการกระทำของเคอร์เซอร์ เมื่อภาพถูก PUT ตาม background ที่ยุ่งยากซับซ้อนสองครั้ง background จะถูกเก็บเก็บไว้โดยไม่ถูกเปลี่ยนแปลง วิธีการนี้จะทำให้ท่านสามารถย้ายเป้าหมาย (object) ไปรอบ ๆ จอภาพโดยไม่ทิ้งร่องรอยของ background เดิมไว้เลย
- action ที่เป็น default คือ XOR
- ตารางที่ 7.8 บรรจุผลต่าง ๆ ของคำสั่ง PUT เมื่อถูกใช้ใน medium resolution mode

ตารางที่ 7.8

EFFECTS OF AND, OR, AND XOR ON COLOR*

Action	Array Names	Color			
		0	1	2	3
AND	0	0	0	0	0
	I	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
OR	0	0	1	2	3
	1	1	1	3	3
	2	2	3	2	3
	3	3	3	3	.3
XOH	0	0	1	2	3
	1	1	0	3	2
	2	2	3	0	1
	3	3	2	1	0

*in medium resolution

ขั้นตอนต่าง ๆ ต่อไปนี้จะใช้เพื่อที่จะแสดงการเคลื่อนไหว (animate)

เป้าหมาย (object):

1. PUT object(s) บนจอภาพ
2. คำนวณตำแหน่งใหม่ของ object(s) อีกครั้งหนึ่ง
3. PUT object(s) บนจอภาพครั้งที่สองที่ตำแหน่งที่เก็บ (location(s)) เดิมเพื่อที่จะเคลื่อนย้าย image(s) เดิม
4. ไปยังขั้นที่ 1 แต่คราวนี้ PUT object(s) ที่ตำแหน่งที่เก็บใหม่

การเคลื่อนไหว (movement) ที่ถูกกระทำโดยวิธีนี้จะทิ้ง background ไว้โดยไม่ถูกเปลี่ยนแปลง การเคลื่อนไหว (flicker) สามารถตัดทอนลงโดยการทำเวลาให้น้อยลงระหว่างขั้นที่ 4 และขั้นที่ 1 และโดยการทำให้แน่ใจว่ามีเวลาที่หน่วงเหนี่ยวให้ช้าลง (time delay) ระหว่างขั้นที่ 1 และขั้นที่ 3 ถ้า object มากกว่า 1 object จะถูกเคลื่อนไหวแล้วทุก ๆ object ควรจะถูกปฏิบัติในทันทีทันใด 1 ขั้นในแต่ละครั้ง

- ถ้าเราไม่ต้องการจะเก็บ background ไว้ การเคลื่อนไหวก็สามารถกระทำโดยการใช้ PSET แทน แนวความคิดนี้ก็คือเพื่อที่จะทิ้งขอบเขต (border) รอบ ๆ ภาพซึ่งใหญ่หรือใหญ่กว่าระยะทางสูงสุดที่ object จะเคลื่อนไหวได้ ดังนั้นเมื่อ object ถูกเคลื่อนย้าย ขอบเขตนี้จะส่งผลให้เกิดการลบจุดใด ๆ ออกไป วิธีนี้อาจจะเป็นวิธีที่ค่อนข้างจะเร็วกว่าวิธีที่จะใช้ XOR ที่อธิบายมาแล้ว ทั้งนี้เนื่องจากว่า 1 PUT เท่านั้นที่ถูกต้องการเพื่อที่จะเคลื่อนย้าย object (แม้ว่าท่านต้อง PUT ภาพที่ใหญ่กว่าก็ตาม)

RANDOMIZE Statement

รูปแบบ RANDOMIZE [n]

จุดประสงค์ คัดเลือก (reseed) ตัวสร้างเลขสุ่ม (random number generator)

รายละเอียด n เป็นนิพจน์จำนวนเต็มที่ถูกใช้ในฐานะเป็น random number seed ถ้าไม่กำหนด n แล้ว GWBASIC จะหยุดการประมวลผลโปรแกรมชั่วคราวและแสดงข้อความดังนี้

Random Number Seed (-32768 to 32767)?

ก่อนที่จะประมวลผล RANDOMIZE

ถ้าตัวสร้างเลขสุ่มไม่ถูกคัดเลือกแล้ว ฟังก์ชัน RND จะส่งค่าชุดลำดับของเลขสุ่มต่าง ๆ ชุดเดียวกันในแต่ละครั้งที่โปรแกรมถูกประมวลผล (run) เพื่อที่จะเปลี่ยนแปลชุดลำดับของเลขสุ่มต่าง ๆ วางคำสั่ง RANDOMIZE ที่จุดเริ่มต้นของโปรแกรมและเปลี่ยน seed กับการประมวลผลแต่ละครั้ง

ตัวอย่าง

```
10 RANDOMIZE
20 INPUT "NUMBER OF RANDOM NUMBERS ";N
30 FOR I = 1 TO N
40 PRINT RND;
50 NEXT I
Ok
RUN
Randon number seed (-32768 to 32767)? 6
NUMBER OF RANDOM NUMBERS ? 3
 .4417627   .1085309   .182628
Oh
```

READ Statement

รูปแบบ READ variable [,variable]

จุดประสงค์ อ่านมูลค่าต่าง ๆ จากคำสั่ง DATA และกำหนดค่าของมันทั้งหลายไปยังตัวแปรต่าง ๆ (ดูคำสั่ง DATA)

รายละเอียด
- variable เป็นตัวแปรจำนวนเลขหรือตัวแปรสตริงก์หรือเป็นสมาชิกของอะเรย์ซึ่งจะรับมูลค่าที่ถูกอ่าน
- คำสั่ง READ โดยปกติแล้วต้องถูกใช้ร่วมกับคำสั่ง DATA คำสั่ง READ ต่าง ๆ กำหนดค่าตัวแปรต่าง ๆ ไปยังมูลค่าต่าง ๆ ของคำสั่ง DATA แบบ 1 ต่อ 1 (one-to-one basis) ตัวแปรต่าง ๆ ของคำสั่ง READ อาจจะเป็นตัวแปรจำนวนเลขหรือตัวแปรสตริงก์ และมูลค่าต่าง ๆ ที่ถูกอ่านต้องสอดคล้องกับชนิดต่าง ๆ ของตัวแปรที่ถูกระบุ ถ้ามันไม่สอดคล้องกันจะเกิด "Syntax error" ขึ้น
- คำสั่ง READ เดี่ยว ๆ อาจจะเข้าถึง (access) คำสั่ง DATA ตั้งแต่ 1 คำสั่งขึ้นไป (โดยที่ DATA ทั้งหลายจะถูกเรียกมาใช้อย่างเรียงตามลำดับ) หรือคำสั่ง READ หลาย ๆ คำสั่งอาจดึงข้อมูลจากคำสั่ง DATA เดียวกันมาใช้ก็ได้ ถ้าจำนวนของตัวแปรต่าง ๆ ในรายการมีจำนวนมากกว่าจำนวนของสมาชิกต่าง ๆ ในคำสั่ง DATA แล้วคำสั่ง READ ต่าง ๆ ถัด ๆ มาจะเริ่มต้นทำการอ่านข้อมูลที่สมาชิกตัวแรกที่ยังไม่ถูกอ่าน ถ้าไม่มีคำสั่ง READ ต่าง ๆ ถัด ๆ มาแล้ว ข้อมูลที่ยังเหลืออยู่ (extra data) จะไม่ถูกดำเนินการ
- เพื่อที่จะอ่านข้อมูลจากคำสั่ง DATA ใหม่อีกครั้งหนึ่ง ให้ใช้ RESTORE (ดูคำสั่ง RESTORE)

ตัวอย่างที่ 1 .

```
80 FOR I=1 TO 10
90 READ A(1)
```

```
100 NEXT I
1.10 DATA3.08,5.19,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.37
```

ส่วนหนึ่งของโปรแกรมนี้อ่านมูลค่าต่าง ๆ ของจากคำสั่ง DATA ต่าง ๆ ไปสู่อะเรย์ A หลังจากการประมวลผล มูลค่าของ A(1) จะเป็น 3.08 และอื่น ๆ ต่อไป

ตัวอย่างที่ 2

```
LIST
10 PRINT "CITY", "STATE", " ZIP"
20 READ C$,S$,Z
30 DATA "DENVER,", COLORADO, 60211
40 PRINT C$,S$,Z
Ok'
RUN
 CITY          STATE          ZIP
 DENVER,       COROLADO       80211
Ok
```

โปรแกรมนี้อ่านข้อมูลสตริงก์และข้อมูลจำนวนเลขจากคำสั่ง DATA ในไลน์ 30 สังเกตว่า DENVER ต้องการเครื่องหมายคำพูด (quotes) เนื่องจากเครื่องหมายจุลภาคที่อยู่ถัดจากมัน ในทางตรงกันข้ามไม่มีเครื่องหมายคำพูดคลุม CORORADO ที่จำเป็น

REM Statement

รูปแบบ REM remark

จุดประสงค์ อนุญาตให้รายละเอียดต่าง ๆ ที่เป็นการอธิบายถูกใส่เข้าไปในโปรแกรม

รายละเอียด
- remark อาจเป็นชุดลำดับใด ๆ ของอักขระต่าง ๆ
- คำสั่ง REM ต่าง ๆ จะไม่ถูกประมวลผลแต่ปรากฏให้เห็นเสมือนกับคำสั่งหนึ่ง ๆ ในโปรแกรม
- คำสั่ง REM ต่าง ๆ อาจจะถูกย้าย (branched into) จากคำสั่ง GOTO หรือคำสั่ง GOSUB การประมวลผลจะดำเนินการต่อไปด้วยคำสั่งปฏิบัติการคำสั่งแรกข้างหลังคำสั่ง REM
- รายละเอียดต่าง ๆ อาจจะถูกเพิ่มเติมไปที่ตอนท้ายของไลน์โดยทำการนำหน้ารายละเอียดด้วยเครื่องหมายคำพูดเดี่ยว (single quotation mark) แทนที่ :REM
- หมายเหตุ : เมื่อทำการใช้ remarks ใน GWBASIC คำสั่ง remark ทั้งหลายจะยังคงอยู่ในหน่วยความจำ

ตัวอย่าง

```
120 REM CALCULATE AVERATE VELOCITY
130 FOR I=1 TO 20
140 SUM=SUM + V(I)
150  NEXT I
```

หรือ

```
120 FOR I=1 TO 20      'CALCULATE AVERAGE VELOCITY
130 SUM=SUM + V(I)
140 NEXT I
```

RENUM Command

รูปแบบ RENUM [newnum] [,[oldnum] [,[increment]]

จุดประสงค์ เรียงลำดับไลน์ต่าง ๆ ของโปรแกรมใหม่

รายละเอียด
- newnum เป็นไลน์นัมเบอร์แรกที่จะถูกใช้ในลำดับใหม่ default คือ 10
- oldnum เป็นไลน์ในโปรแกรมที่ใช้อยู่ในขณะนั้นที่ซึ่งการเรียงลำดับใหม่จะถูกเริ่มต้น
- increment เป็นการเพิ่มขึ้นที่จะถูกใช้ในลำดับใหม่ default คือ 10
- RENUM จะมีผลให้เปลี่ยนแปลงทุก ๆ การอ้างอิงไลน์นัมเบอร์ต่าง ๆ ที่อยู่ต่อจากคำสั่ง GOTO, GOSUB, THEN, ELSE, ON...GOTO, ON...GOSUB, RESTORE, RESUME และคำสั่ง ERL เพื่อที่จะส่งผลให้เกิดเป็นไลน์นัมเบอร์ใหม่ ๆ ถ้าไลน์นัมเบอร์ที่ไม่มีอยู่ปรากฏหลังคำสั่งใดคำสั่งหนึ่งในคำสั่งต่าง ๆ เหล่านี้ ข้อความผิดพลาด "Undefined line number yyyyy in xxxxx" จะถูกพิมพ์ ไลน์นัมเบอร์ที่ไม่ถูกต้องซึ่งอ้างอิง yyyyy จะไม่ถูกเปลี่ยนแปลงโดย RENUM แต่ไลน์นัมเบอร์ xxxxx อาจจะถูกเปลี่ยนแปลง
- หมายเหตุ : RENUM ไม่สามารถถูกใช้เพื่อที่จะเปลี่ยนแปลงลำดับของไลน์ต่าง ๆ ของโปรแกรม (ตัวอย่างเช่น RENUM 15,30 เมื่อโปรแกรมมี 3 ไลน์นัมเบอร์คือไลน์นัมเบอร์ 10, 20 และไลน์นัมเบอร์ 30) หรือไม่สามารถถูกใช้เพื่อที่จะสร้างไลน์นัมเบอร์ต่าง ๆ ที่มีค่ามากกว่า 65529 ขึ้น "Illegal function call" error จะเป็นผลลัพธ์ที่ได้

ตัวอย่าง RENUM

คำสั่งนี้จะเรียงหมายเลขไลน์ของโปรแกรมใหม่ทั้งหมด ไลน์นัมเอร์แรกจะเป็น 10 และไลน์ต่าง ๆ ที่สืบเนื่องกันมาจะถูกทำให้เพิ่มขึ้นครั้งละ 10

RENUM 300,,50

ตลอดโปรแกรมจะถูกเรียงลำดับหมายเลขใหม่ ไลน์นัมเบอร์แรกจะเป็น 300 และไลน์ต่าง ๆ ที่เรียงลำดับถัดไปมีไลน์นัมเบอร์ทวีค่าครั้งละ 50

RENUM 1000,900,20

โปรแกรมจะถูกเรียงลำดับหมายเลขใหม่จากไลน์ 900 ไลน์นัมเบอร์ใหม่ไลน์แรกจะเป็น 1000 และแต่ละไลน์ที่สืบเนื่องกันมาจะถูกทำให้เพิ่มขึ้นครั้งละ 20

RESET Command

รูปแบบ RESET

จุดประสงค์ ปิดไฟล์ทุก ๆ ไฟล์บนไดร์ฟ์ทุก ๆ ไดร์ฟ์

รายละเอียด คำสั่งนี้คล้ายคลึงกันกับคำสั่ง CLOSE (ดูคำสั่ง CLOSE)

RESTORE Statement

รูปแบบ RESTORE [line]

จุดประสงค์ อนุญาตให้คำสั่ง DATA ต่าง ๆ ถูกอ่านใหม่จากไลน์ที่ถูกระบุ

รายละเอียด - line เป็นไลน์นัมเบอร์ของคำสั่ง DATA ในโปรแกรม

- หลังจากที่คำสั่ง RESTORE ถูกประมวลผล คำสั่ง READ ถัดไปจะเข้าถึง item แรกในคำสั่ง DATA คำสั่งแรกในโปรแกรม ถ้า line ถูกระบุ คำสั่ง READ ถัดไปจะเข้าถึง ITEM แรกในคำสั่ง DATA ที่ถูกระบุ

ตัวอย่าง
```
10 DATA 1,2,3
20 DATA 4,5,6
30 READ A,B,C
40 PRINT A,B,C
```

```
50 READ A,B,C
60 PRINT A,B,C
70 RESTORE 20
80 READ A,B,C
90 PRINT A,B,C
Ok
RUN
 1            2            3
 4            5            6
 4            5            6
Ok
```

RESUME Statement

รูปแบบ RESUME

RESUME 0

RESUME NEXT

RESUME line

จุดประสงค์ คำเนินการประมวลผลโปรแกรมต่อไปหลังจากที่โพรซีเดอร์การค้นหาความผิดพลาด (error recovery procedure) ได้ถูกกระทำ

รายละเอียด
- ใช้รูปแบบใดรูปแบบหนึ่งใน 4 รูปแบบที่ถูกแสดงข้างบนขึ้นอยู่กับที่ซึ่งการประมวลผลจะกระทำต่อไป (resume)
- RESUME หรือ RESUME เป็นเหตุให้การประมวลผลจะกระทำต่อไปที่คำสั่งซึ่งเป็นเหตุให้เกิดความผิดพลาด
- RESUME NEXT เป็นเหตุให้การประมวลผลจะกระทำต่อไปที่คำสั่งที่อยู่ถัดจาก

คำสั่งซึ่งเป็นเหตุให้เกิดความผิดพลาด

- RESUME line เป็นเหตุให้การประมวลผลจะกระทำต่อไปที่ไลน์ line
- ถ้าคำสั่ง RESUME ไม่อยู่ในสับรูทีนการตรวจจับความผิดพลาด (error handling subroutine) แล้วข้อความ "RESUME without error" จะถูกพิมพ์

ตัวอย่าง

```
10 ON ERROR GOTO 900
.
.
.
900 IF (ERR=230)AND(ERL 90) THEN PRINT "TRY AGAIN":
    RESUME 80
```

ในรูทีนการตรวจจับความผิดพลาดที่เริ่มต้นที่ไลน์ 900 resume เป็นเหตุให้โปรแกรมส่งไปยังไลน์ 80 เมื่อ error 230 เกิดขึ้นในไลน์ 90

RETURN Statement

รูปแบบ RETUEN [line number]

จุดประสงค์ เป็นเหตุให้ GWBASIC หันกลับ (return) ไปยังคำสั่งคำสั่ง GOSUB ที่เพิ่งผ่านมา (most recent GOSUB Statement)

รายละเอียด - ถ้า line number ถูกระบุ การประมวลผลจะหันกลับไปยังไลน์นัมเบอร์ที่ถูกระบุในโปรแกรม ข้อเลือกนี้ควรจะถูกใช้ด้วยความระมัดระวังอย่างมาก

RIGHT$ Function

รูปแบบ v$=RIGHT$(a$,n)

จุดประสงค์ ส่งค่าอักขระ n ตัวทางด้านขวามือสุดของสตริงก์ a$

รายละเอียด – a$ เป็นนิพจน์สตริงก์ใด ๆ

– n เป็นจำนวนเต็มซึ่งจำนวนอักขระต่าง ๆ มากน้อยเท่าไรที่จะเป็นผลลัพธ์

– ถ้า n มีค่ามากกว่าหรือเท่ากับจำนวนของอักขระต่าง ๆ ใน a$ แล้ว RIGHT$ จะส่งค่า a$ ถ้า n=0 แล้วสตริงก์ว่างเปล่า(ความยาวเป็นศูนย์จะถูกส่งค่า)

– ดูฟังก์ชัน MID$ และฟังก์ชัน LEFT$ ด้วย

ตัวอย่าง

```
10 TEST$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 FOR I=1 TO 5
30 PRINT RIGHT$(TEST$,I)
40 NEXT I
RUN
 Z
 YZ
 XYZ
 WXYZ
 VWXYZ
Ok
```

คำสั่ง RUN เป็นเหตุให้ไลน์ 5 ไลน์ถูกพิมพ์ ไลน์แต่ละไลน์จะส่งค่าอักขระที่เพิ่มขึ้นจากทางขวาสุดของสตริงก์

RND Function

รูปแบบ V = RND[(x)]

จุดประสงค์ ส่งค่าเลขสุ่มระหว่าง 0 และ 1

รายละเอียด – x เป็นนิพจน์จำนวนเลขที่จะก่อให้เกิดผลต่อมูลค่าที่ถูกส่ง

- ชุดลำดับเดียวกันของเลขสุ่มต่าง ๆ จะถูกสร้างขึ้นในแต่ละครั้งที่โปรแกรมถูกประมวลผลแม้ว่าตัวสร้างเลขสุ่มจะถูกคัดเลือกก็ตาม (ดูคำสั่ง RANDOMIZE)
- เมื่อ x<0 คัดเลือกตัวสร้างเลขสุ่มโดยการใช้ x สิ่งนี้เริ่มต้นชุดลำดับเดียวกันอีกครั้งหนึ่ง เพื่อที่จะสร้างชุดลำดับที่แตกต่างกัน ท่านต้องใช้มูลค่าที่แตกต่างไปจากมูลค่าสำหรับ x
- ถ้า x>0 หรือถ้า x ถูกละไว้แล้ว RND จะสร้างเลขสุ่มถัดไปในชุดลำดับ
- ถ้า x=0 เลขสุ่มล่าสุดที่ถูกสร้างขึ้นจะถูกกระทำซ้ำ

```
10  RANDOMIZE
20 PRINT RND*100
 79.51105
```

ตัวอย่างนี้แสดงวิธีที่ RND call สามารถสร้างมูลค่าระหว่าง 0 และ 100

RUN Command

รูปแบบ

```
RUN  [line]
RUN filespec[,R]
```

จุดประสงค์ ประมวลผลโปรแกรมที่เป็นอยู่ในหน่วยความจำในขณะนั้น

รายละเอียด
- line เป็นหมายเลขของไลน์ในโปรแกรมที่ซึ่งการประมวลผลจะเริ่มต้น
- filespec เป็นชื่อของไฟล์หรือโปรแกรม สำหรับดิสค์ไฟล์ต่าง ๆ default extension จะเป็น .BAS
- ถ้า line ถูกละ การประมวลผลโปรแกรมจะเริ่มต้นที่ไลน์นัมเบอร์ต่ำสุด
- RUN filespec เรียก (load) ไฟล์จากดิสเกตต์ลงสู่หน่วยความจำและประมวลผลมัน ก่อนที่จะทำการเรียกโปรแกรมที่ถูกต้องการ RUN จะปิดไฟล์ทุก ๆ ไฟล์ที่เปิดอยู่และลบรายละเอียด (contents) ต่าง ๆ ในหน่วย-

ความจำในขณะนั้นออกไป อย่างไรก็ตาม ด้วย "R" option ไฟล์ข้อมูลทุก ๆ ไฟล์ยังคงเปิดอยู่

- โดยปกติ GWBASIC จะหันกลับไปยังระดับคำสั่งหลังจากคำสั่ง RUN โดยเพิ่มเติม คำสั่งนี้จะปิดเสียงใด ๆ ที่กำลังประมวลผลอยู่และกำหนดใหม่ไปที่ Music Foreground (ดูคำสั่ง PLAY) PEN และ STRIG จะถูกกำหนดใหม่ไปที่ OFF

ตัวอย่าง RUN "B:NEWFIL",R

โปรแกรม NEWFIL จะถูกเรียกจากไดร์ฟ B และถูกประมวลผลด้วยไฟล์ต่าง ๆ ที่ยังคงทิ้งให้เปิดอยู่

SAVE Command

รูปแบบ SAVE filespec[,A]

SAVE filespec[,P]

จุดประสงค์ บันทึกโปรแกรมไฟล์ (program file) บนดิสเกตต์

รายละเอียด - filespec เป็นชื่อของไฟล์ซึ่งจะถูกบันทึก

- ถ้าชื่อไฟล์มีความยาวน้อยกว่าหรือเท่ากับ 8 อักขระและไม่มีส่วนขยายถูกจัดจ่ายให้ แล้วส่วนขยาย .BAS จะถูกเพิ่มเติมไปที่ชื่อ ถ้า filespec มีอยู่แล้ว ไฟล์จะถูกบันทึกซ้ำ

- ใช้ A option เพื่อที่จะบันทึกไฟล์ในรูป ASCII format มิฉะนั้นแล้ว GWABSIC จะบันทึกไฟล์ในรูป compressed binary format ASCII format ใช้เนื้อที่ (space) มากกว่าบนดิสค์แต่การขับจากดิสค์มาใช้งาน (disk access) บางครั้งต้องการว่าไฟล์ต่าง ๆ อยู่ในรูป ASCII format ตัวอย่างเช่น คำสั่ง MERGE ต้องการไฟล์ในรูป ASCII format และบางคำสั่งควบคุมระบบ (operating system commands) ต่าง ๆ อย่างเช่น

LIST อาจต้องการไฟล์ในรูป ASCII format

- ใช้ P option เพื่อที่จะป้องกันไฟล์โดยทำการบันทึกมันในรูป encoded binary format เมื่อไฟล์ที่ถูกป้องกันจะถูกประมวลผล (หรือถูกเรียก) ในเวลาต่อมา ความพยายามใด ๆ ที่จะแสดงรายการหรือบรรณาธิกรณ์ (edit) มันจะล้มเหลว ไม่มีวิธีที่จะ "unprotect" โปรแกรมแต่ละโปรแกรม
- ดูภาคผนวก ก, Sequential and Random Files

ตัวอย่าง SAVE "A:COM2",A

ตัวอย่างนี้บันทึก COM2.BAS บนไดร์ฟ์ A ในรูป ASCII format

SAVE "B:PROG",P

ตัวอย่างนี้บันทึก PROG บนไดร์ฟ์ B เพื่อว่ามันจะไม่ถูกเปลี่ยนแปลง

SCREEN Function

รูปแบบ v = SCREEN(row,col[,z])

จุดประสงค์ ส่งอักขระหรือสี ณ แถว(ไลน์)และสดมภ์ของ active screen ที่ถูกระบุ

รายละเอียด
- row มีค่าเป็นตัวเลขในช่วง 1 ถึง 25
- col มีค่าเป็นตัวเลขในช่วง 1 ถึง 40 หรือในช่วง 1 ถึง 80 ขึ้นอยู่กับ WIDTH
- z เป็นนิพจน์จำนวนเลขซึ่งประเมินค่าไปสู่มูลค่าจริง (true) หรือมูลค่าเท็จ (false) มันใช้ได้เฉพาะใน text mode เท่านั้น
- ภาคผนวก A, ASCII Character Codes แสดงรายการรหัส (codes') ต่าง ๆ ที่เป็นไปได้

- ใน text mode ถ้า z ถูกประกอบรวมอยู่ด้วยและมีค่าไม่เป็นศูนย์แล้ว color attribute จะถูกส่งค่าไปแทนรหัส ASCII (ดูคำสั่ง COLOR สำหรับรายการของสีและตัวเลขต่าง ๆ ที่ถูกเกี่ยวข้องสัมพันธ์กัน)
- ดูคำสั่ง SCREEN ด้วย

ตัวอย่าง 10 A = SCREEN (20,21)

ถ้าอักขระที่จุด 20,21 เป็น X แล้ว A จะมีค่าเป็น 88

110 A = SCREEN (10,10,1)

A จะเป็น color attribute ของอักขระที่ตำแหน่ง (location) (10,10)

SCREEN Statement

รูปแบบ SCREEN [model [,[burst][,[apage][,vpage] 11

จุดประสงค์ กำหนด screen attributes สำหรับใช้ในคำสั่งต่าง ๆ ที่ตามกันมา (subsequent statements)

รายละเอียด - mode อาจเป็นสิ่งใด ๆ ดังต่อไปนี้ :

0 text mode ณ ความกว้างที่เป็นอยู่ในขณะนั้น (40 หรือ 80) การแสดงผล (display) บน "current" monitor ตามที่ถูกเลือกก่อนหน้า สิ่งนี้เป็นมูลค่าที่เป็น default

1 medium resolution graphics mode (320x200) แสดงผลบนด้านซ้ายมือสุดของ monochrome screen แม้ว่ามันจะเป็น color/graphics monitor adaptor ก็ตาม

2 high resolution graphics mode (640x200) แสดงผลบนสองในสามส่วนตอนบนสุดของ monochrome screen แม้ว่ามันจะเป็น

color/graphics monitor adaptor ก็ตาม

100 ส่งบังคับ (force) text mode ไปบน monochrome screen

101 สับกลไก (switch) จาก monochrome display ไปสู่ color/ graphics adaptor (ถ้าสามารถเป็นไปได้)

104 กำหนดจอภาพสำหรับ super-resolution graphics

105 กำหนดจอภาพสำหรับ mixed mode (super-resulotion graphics and text) ใน mode นี้ text อาจจะปกคลุม graphics image ใด ๆ

- burst เป็นนิพจน์จำนวนเลขที่จะส่งผลเป็นมูลค่าจริงหรือมูลค่าเท็จซึ่งส่งผล (enable) ต่อสี กับ color/graphics mode และ burst มีผลต่อสีดังต่อไปนี้ :

mode	burst	Effect on Color
0	0	Disabled
	1	Knabled
1	0	Enabled
	1	Disabled
2		
100		
101		None
104		
105		

- apage (active page) เป็น 3, 4, 5, 6 หรือ 7 สำหรับ active page ในขณะนั้น ค่าใดค่าหนึ่งจะถูกบันทึกไปยังหน่วยความจำ กับ color /graphics adaptor แล้ว apage มีค่าเป็นตัวเลขระหว่าง 0 ถึง 7 สำหรับความกว้าง 40 หรือมีค่าระหว่าง 0 ถึง 3 สำหรับความกว้าง 80 มันถูกใช้กับ mode 0 เท่านั้น
- vpage เป็น 3, 4, 5, 6 หรือ 7 สำหรับ virtual page ในขณะนั้น ค่าใดค่าหนึ่งจะถูกเขียนบนจอภาพ กับ color/graphics adaptor vpage ใช้ได้เฉพาะใน text mode เท่านั้น และถ้ามันถูกละไว้แล้ว default จะเป็น apage
- สังเกตว่า บ่อยครั้งในระหว่างที่ vpage และ apage จะเป็นเช่นเดียวกัน ท่านอาจจะแสดงผลหนึ่งหน้าบนจอภาพในขณะที่ทำการเขียนไปยังหน้าอื่น และดังนั้น vpage อาจจะแตกต่างกว่า apage
- ถ้า mode เท่ากับ 104, 105 หรือเท่ากับ 0 และเฉพาะ apge และ vpage เท่านั้นที่ถูกระบุแล้ว ผลที่ได้จะเป็นการเปลี่ยนแปลงหน้าการพิมพ์ต่าง ๆ สำหรับการปรากฏผล โดยการจัดการกับพารามิเตอร์ต่าง ๆ เหล่านี้ ท่านสามารถพิมพ์หนึ่งหน้าในขณะที่ทำการสร้างหน้าอื่นและจากนั้นสับกลไก visual pages ในทันทีทันใด
- กับระบบต่าง ๆ ซึ่งมีหน่วยความจำ 128K ถ้า mode เท่ากับ 0 แล้ว default ของ apage และ vpage จะเป็น 0 แต่ถ้า mode เท่ากับ 100-105 แล้ว default ของ apage และ vpage จะเป็น 7
- ถ้า mode ของจอภาพใหม่เป็นเช่นเดียวกันกับ mode ก่อนหน้าแล้ว เฉพาะพารามิเตอร์ใหม่ต่าง ๆ เท่านั้นจะถูกปรับปรุงให้เหมาะสม (updated)
- แอดเดรสของหน่วยความจำ (memory address) สำหรับ super-resolution graphics page ที่ถูกกำหนดให้จะเป็น

[page*800H]:0000 ท่านจำเป็นที่จะต้องใส่แอดเดรสนี้ในคำสั่ง DEF SEG ก่อนที่จะทำการใช้คำสั่ง BLOAD

- สังเกตว่าเคอร์เซอร์ถูกใช้เนื้อที่ร่วมกัน (shared) ระหว่างหน้าต่าง ๆ ทุก ๆ หน้า ดังนั้นเมื่อทำการสับกลไกกลับไปมาระหว่าง active pages ที่จะบันทึกตำแหน่งเคอร์เซอร์บน active page ในขณะนั้นก่อนที่จะทำการเปลี่ยนแปลงไปยังหน้าอื่นโดยใช้ POS(0) และ CSRLIN เพื่อที่จะดำเนินการดังกล่าว จากนั้นเคอร์เซอร์สามารถถูกสร้างขึ้นใหม่ (restored) บนหน้าแรกเริ่ม (original page) ด้วย LOCATE
- พารามิเตอร์ใด ๆ อาจจะถูกละไว้ ยกเว้นสำหรับ vpage พารามิเตอร์ต่าง ๆ ที่ถูกละไว้จะกำหนดเอามูลค่าเดิมขึ้นแทน
- ในกรณีของ standard monochrome display และ color/graphics monitor adaptor นั้น มีข้อเสนอแนะว่าควรใช้คำสั่ง SCREEN 0,0,0 และคำสั่ง WIDTH 80 ในตอนเริ่มต้นโปรแกรมเสมอ
- มูลค่าต่าง ๆ ที่ออกนอกช่วงจะเป็นเหตุให้เกิด "Illegal function call" error ขึ้นและมูลค่าต่าง ๆ ก่อนหน้าจะถูกเก็บรักษาไว้ (retained)

ตัวอย่าง

```
10 CLEAR ,19202
20 SCREEN 105,,3,3
30 CLS
```

ไลน์ 10 สงวน (reserve) หน่วยความจำสำหรับกราฟฟิคต่าง ๆ เมื่อท่านมี 128K system (ถ้าท่านมี system ที่มีหน่วยความจำมากกว่านี้แล้ว คำสั่งนี้ก็ไม่จำเป็นต้องใช้) ไลน์ 20 มีผลทำให้ text และ super-resolution graphics มีความสามารถที่จะถูกพิมพ์และถูกเขียนไปยัง screen 3 ไลน์ 30 เคลียร์ (clear) ทั้ง alphanumeric screen และ graphics screen ในหน่วยความจำ

10 SCREEN 0,1,0,0

ตัวอย่างนี้เป็นตัวอย่างสำหรับ color/graphics monitor adaptor ซึ่งจะทำการคัดเลือก text mode ด้วยสีและกำหนด active page และ virtual page เป็น 0

SGN Function

รูปแบบ v = SGN(x)

จุดประสงค์ ส่ง เครื่องหมายของ X

รายละเอียด - X เป็นนิพจน์จำนวนเลขใด ๆ

- ถ้า x>0 แล้ว SGN(x) ส่งค่า 1

ถ้า x=0 แล้ว SGN(x) ส่งค่า 0

ถ้า x<0 แล้ว SGN(x) ส่งค่า - 1

ตัวอย่าง ON SGN(X)+2 GOTO100,200,300

โปรแกรมจะย้ายไปยังไลน์ 100 ถ้า X มีค่าเป็นลบ ไปยังไลน์ 200 ถ้า X มีค่าเป็นศูนย์ และไปยังไลน์ 300 ถ้า X มีค่าเป็นบวก

SIN Function

รูปแบบ v = SIN(x)

จุดประสงค์ ส่งค่าไซน์ของ x

รายละเอียด - x เป็นมุมในหน่วยเรเดียน

- SIN(x) ถูกประเมินค่าอยู่ในรูป single precision

- ดูฟังก์ชัน COD ด้วย

ตัวอย่าง PRINT SIN(1.5)

.9974951

Ok

SOUND Function

รูปแบบ SOUND freq, duration

จุดประสงค์ สร้างเสียงผ่านเครื่องพูด (speaker)

รายละเอียด
- freq เป็นความถี่มีหน่วยเป็นเฮิร์ท (จำนวนรอบต่อวินาที) และมีค่าอยู่ในช่วง 37 ถึง 32767
- duration เป็นนิพจน์จำนวนเลขมีค่าอยู่ในช่วง 0 ถึง 65535 ซึ่งบ่งชี้ความยาวที่ถูกต้องการใน clock ticks ซึ่งเกิดขึ้น 18.2 ครั้งต่อวินาที
- ดูตารางที่ 7.9 สำหรับข่าวสารเกี่ยวกับความถี่ต่าง ๆ ที่ถูกสร้างขึ้นโดยคำสั่งนี้ อ้างอิงไปยังคำสั่ง PLAY สำหรับข่าวสารเกี่ยวกับการสร้างโน้ตต่าง ๆ ที่ต่อเนื่องโดยไม่มีการหยุดระหว่างคำสั่งต่าง ๆ

ตารางที่ 7.9

NOTE FREQUEMCIES FOR FOUR OCTAVES

Note	Frequeny	Note	Frequency
C	130.8	'A	220.0
D	146.8	B	246.9
E	164.8	c*	261.6
F	174.6	D	293.7
G	196.0	E	329.6

*Middle C

ตารางที่ 7.9 (ต่อ)

Note	Frequency	Note	Frequency
F	349.2	A	880.0
G	302.0	B	987.8
A	440.0	C	1046.5
B	493.9	D	1174.7
C	523.3	E	1318.5
D	587.3	F	1396.9
E	659.3	G	1568.0
F	698.5	A	1760.0
G	784.0	B	1975.5

- หมายเหตุ : สำหรับโน้ตต่าง ๆ ใน octave อื่น ๆ ใช้โน้ตที่สัมพันธ์กันใน octave x (โน้ตใน octave X-l) x 2
- ถ้า rest เป็น duration ที่ถูกต้องการ ใช้ 32767
- ตารางที่ 7.10 แสดงรายการของบาง typical tempos

ตารางที่ 7.10

TEMPO CALCULATIONS

Beats/ Minute	Tempo	Ticks/ Beat
	Larghissimo (very slow)	
40-60	Largo	27-18
60-66	Larghetto	18-17
	Grave	
	Lent0	
66-76	Adagio	17-14
	Adagietto (slow)	
76-108	Andante	14-10
	Andantino (medium)	
108-120	Moderato	10-9
	Allegretto (fast)	
120-168	Allegro	9-7
	Vivace	
	Veloce	
168-208	Presto	7-5
	Prestissimo (very fast)	

SPA&$ Function

รูปแบบ v$ = SPACE$(x)

จุดประสงค์ ส่งค่าสตริงก์ของ X ช่องว่าง

รายละเอียด - x ต้องเป็นเลขจำนวนเต็มในช่วง 0 ถึง 255

- ดูฟังก์ชัน SPC ด้วย

ตัวอย่าง

```
10 FOR I = 1 TO 5
20 X$ = SPACE$(I)
30 PRINT X$;I
40 NEXT I
RUN
 1
  2
   3
    4
     5
Ok
```

ตัวเลขแต่ละตัวจะตามด้วยที่ว่าง 1 ช่องว่างบนไลน์ GWBASIC ใส่ช่องว่างข้างหน้าจำนวนบวกต่าง ๆ ซึ่งส่งผลให้เกิดที่ว่าง I+1 ช่องว่างที่อยู่ข้างหน้า

SPC Function

รูปแบบ PRINT SPC(x)

จุดประสงค์ ข้ามไป x ช่องว่างในคำสั่ง PRINT

รายละเอียด - x ต้องมีค่าเป็นจำนวนเต็มอยู่ในช่วง 0 ถึง 255

- SPC อาจจะถูกใช้กับคำสั่ง PRINT, LPRINT และคำสั่ง PRINT # เท่านั้น

- GWBASIC จะทำโดยถือเสมือนว่า SPC มีเครื่องหมายเซมิโคลอนที่ถูกบ่งบอกอยู่ข้างหลังมัน ดังนั้น ถ้า SPC เกิดขึ้นที่ตอนท้ายของรายการของ data items แล้ว GWBASIC จะไม่ใส่ carriage return
- ดูฟังก์ชัน SPACE$ ด้วย

ตัวอย่าง

```
PRINT "OVER" SPC(15) "THERE"
 OVER               THERE
'Ok
```

มีช่องว่าง 15 ที่ระหว่าง OVER และ THERE

SQR Function

รูปแบบ `v = SQR(x)`

จุดประสงค์ ส่งค่ารากที่สองของ x

รายละเอียด - x ต้องมีค่ามากกว่าหรือ เท่ากับศูนย์

ตัวอย่าง

```
10 FOR X = 10 TO 25 STEP 5
20 PRINT, X, SQR(X)
30  NEXT
RUN
 10             3.162278
 15             3.872984
 20             4.472136
 2       5      5
Ok
```

รากที่สองของ 10, 15, 20 และ 25 จะถูกคำนวณค่าตามลำดับ

STICK Function

รูปแบบ v = STICK(n)

จุดประสงค์ รับข้อมูลเข้าจากคันบังคับ (joystick) ในรูปของโคออร์ดิเนตของ x และโคออร์ดิเนตของ y

รายละเอียด - n เป็นจำนวนเต็มมีค่าจาก 0 ถึง 3 ดังต่อไปนี้ :

0 = x coordinate ของ joystick A

1 = y coordinate ของ joystick A

2 = x coordinate ของ joystick B

3 = y coordinate ของ joystick B

- ในครั้งแรกให้ใช้ STICK(0) ก่อน ฟังก์ชันนี้จะเป็นเหตุให้พารามิเตอร์ทั้งหมด 4 ตัวจะถูกได้รับ (obtained) จาก joystick จากนั้นการเรียกไปยังฟังก์ชันต่าง ๆ STICK(1) ถึง STICK(3) จะส่งค่าของมูลค่าต่าง ๆ ที่ถูกรวบรวมไว้โดยฟังก์ชัน STICK(0) สิ่งนี้ถูกต้องการเพื่อที่จะยอมให้พารามิเตอร์ข้อมูลเข้า (input parameters) ต่าง ๆ ทั้งหมดส่งผลเป็นตำแหน่งของ STICK ณ ขณะเวลาที่เฉพาะ (particular instant)

ตัวอย่าง

```
10 FOR I = 1 TO 3
20 FOR J = 0 TO 3
30  PRINT  STICK(J);
40 NEXT  J
50 PRINT
60 NEXT  I
Ok
```

STOP Statement

รูปแบบ STOP

จุดประสงค์ หยุดการประมวลผลชั่วคราวและหันกลับไปยังระดับคำสั่ง

รายละเอียด

- คำสั่ง STOP ต่าง ๆ อาจจะถูกใช้ ณ ที่ใด ๆ ในโปรแกรม
- STOP เป็นเหตุให้ข้อความต่อไปนี้ถูกพิมพ์ :

```
break in line nnnnn
```

- ไม่เหมือนคำสั่ง END คำสั่ง STOP ไม่ได้ปิดไฟล์ต่าง ๆ
- GWBASIC จะกลับไปยังระดับคำสั่งหลังจากที่ STOP ถูกประมวลผลแล้ว การประมวลผลโปรแกรมดำเนินการต่อไปด้วย CONT (ดูคำสั่ง CONT)

ตัวอย่าง

```
10 INPUT A,B,C
20 K=A^2*5.3:L=B^3/.26
30 STOP
40 M=C*K+100:PRINT M
RUN
? 1,2,3
BREAK IN 30
Ok
PRINT L
 30.76923
Ok
CONT
 115.9
Ok
```

ตัวอย่างนี้แสดงวิธีที่ STOP สามารถถูกใช้เพื่อที่จะตรวจสอบค่าของตัวแปรต่าง ๆ

STR$ Function

รูปแบบ v$ = STR$(x)

จุดประสงค์ ส่งค่าสตริงก์ที่เป็นการแทนของมูลค่าของ x

รายละเอียด
- x เป็นนิพจน์จำนวนเลขใด ๆ
- ถ้า x มีค่าเป็นบวกแล้ว สตริงก์ที่ถูกส่งค่าจะบรรจุที่ว่างที่อยู่ข้างหน้า (leading blank) ซึ่งเป็นช่องว่างที่ถูกสงวนไว้สำหรับเครื่องหมายลบ เมื่อตัวเลขมีค่าเป็นจำนวนลบ
- ดูฟังก์ชัน VAL ด้วย

ตัวอย่าง

```
5 REM arithmetic for kids
10 INPUT "TYPE A  NUMBER";N
20 ON LEN(STR$(N)) GOSUB 30,100,200,300,400,500
```

โปรแกรมจะย้ายไปยังสับรูทีนต่าง ๆ ขึ้นอยู่กับตัวเลขที่ถูกป้อนเข้าไป STR$ แปลงค่าตัวเลขไปสู่สตริงก์และการโยกย้ายจะขึ้นอยู่กับความยาวของสตริงก์

STRIG Statement and Function

รูปแบบ

```
STRIG ON
STRIG OFF
STRIG STOP
x = STRIG (n)
```

จุดประสงค์ ส่งสถานะต่าง ๆ ของ joystick trigger ที่ถูกระบุและทำให้การทำ trapping ของ joystick มีความสามารถ, ไร้ความสามารถ หรือหยุดกระทำ

รายละเอียด
- x เป็นตัวแปรจำนวนเลขสำหรับเป็นที่เก็บผลลัพธ์ของฟังก์ชัน
- n เป็นตัวเลขจาก 0 ถึง 3 ที่จะทำการระบุ trigger ซึ่งจะถูกตรวจเช็คดังต่อไปนี้ :
  - 0 ส่งค่า -1 ถ้า trigger A ถูกกดนับตั้งแต่คำสั่ง STRIG(0) ล่าสุด ส่งค่า 0 ถ้า trigger A ไม่ถูกกด
  - 1 ส่งค่า -1 ถ้า trigger A เป็น down ในขณะนั้น ส่งค่า 0 ถ้า trigger A ไม่เป็น down
  - 2 ส่งค่า -1 ถ้า trigger B ถูกกดนับตั้งแต่คำสั่ง STRIG(2) ล่าสุด ส่งค่า 0 ถ้า trigger B ไม่ถูกกด
  - 3 ส่งค่า -1 ถ้า trigger B เป็น down ในขณะนั้น ส่งค่า 0 ถ้า trigger B ไม่เป็น down
- STRIG ON ทำให้ trapping มีความสามารถ
- STRIG STOP ทำให้ trapping ไร้ความสามารถแต่ถ้า joystick ถูกกดเหตุการณ์นั้นจะถูกจดจำได้และถูก trapped ในไม่ช้าไม่นานที่มี STRIG ON
- STRIG OFF ไม่เพียงแต่จะทำให้ trapping ไร้ความสามารถเพียงเท่านั้นมันไม่ได้จดจำ เหตุการณ์ที่ตามมาด้วย
- STRIG ON ต้องการคำสั่ง ON STRIG (ดูคำสั่ง ON STRIG) ในระหว่างที่ trapping ถูกทำให้มีความสามารถและถ้าไลน์นัมเบอร์ที่ไม่ใช่ศูนย์ถูกระบุในคำสั่ง ON STRIG แล้ว GWBASIC จะตรวจเช็คระหว่างคำสั่งทุก ๆ คำสั่งเพื่อที่จะดูว่า joystick trigger ได้ถูกกดแล้ว

- เมื่อ trap เกิดขึ้น การเกิดขึ้น (occurance) ของเหตุการณ์นั้นจะถูกทำลายไป ดังนั้นโดยปกติแล้วฟังก์ชัน STRIG(n) จะส่งค่าเท็จภายในสับรูทีนแม้ว่าเหตุการณ์ได้ถูกกระทำเนื่องซ้ำเนื่องจาก trap ก็ตาม ดังนั้นถ้าท่านปรารถนาที่จะกระทำโพรซิเดอร์ (procedures) ต่าง ๆ ที่แตกต่างกันสำหรับ joystick แล้วท่านต้องสร้างสับรูทีนที่แตกต่างกันสำหรับ joystick แต่ละ joystick ค่อนข้างดีกว่าที่จะรวบรวมโพรซิเดอร์ทุกโพรซิเดอร์ในสับรูทีนเดียว

```
10 IF STRIG(0) THEN BEEP
20 GOTO 10
```

ลูปที่ไม่สิ้นสุด (endless loop) ถูกสร้างขึ้นเพื่อที่จะส่งเสียง (beep) เมื่อใดก็ตามที่ trigger botton บน joystick ถูกกด

STRING$ Function

รูปแบบ `v$ = STRING$(x,m)`

`v$ = STRING$(x,a$)`

จุดประสงค์ ส่งสตริงก์ความยาว x ที่ซึ่งอักขระต่าง ๆ ทั้งหมดมี ASCII code m หรือเป็นอักขระตัวแรกของ a$

รายละเอียด
- x,m เป็นเลขจำนวนเต็มมีค่าอยู่ในช่วง 0 ถึง 255
- a$ เป็นนิพจน์สตริงก์ใด ๆ

ตัวอย่าง

```
10 X$ = STRING$(10,45)
20 PRINT X$ "MONTHLY REPORT" X$
RUN
----------MONTHLY REPORT----------
Ok
```

ตัวอย่างนี้พิมพ์สตริงก์ของเครื่องหมายไฮเฟน (-) โดยการกระทำซ้ำมูลค่า ASCII ของ 45

SWAP Statement

รูปแบบ SWAP variable1, variable2

จุดประสงค์ สับเปลี่ยนมูลค่าต่าง ๆ ของตัวแปร 2 ตัว

รายละเอียด
- variable1, variable2 เป็นชื่อของตัวแปรสองตัวหรือเป็นชื่อของสมาชิกต่าง ๆ ของอะเรย์
- ชนิดใด ๆ ของตัวแปรอาจจะถูกสับเปลี่ยน (integer, single precision, double precision, string) แต่ตัวแปรทั้งสองตัวต้องมีชนิดเดียวกันหรือไม่เช่นนั้นจะส่งผลให้เกิด "Type mismatch" error ขึ้น

ตัวอย่าง

```
LIST
10 A$="ONE " : B$=" ALL " : C$="FOR"
20 PRINT A$ C$ B$
30 SWAP A$, B$
40 PRINT A$ C$ B$
RUN
 ONE ALL FOR
 FOR ALL ONE
Ok
```

ไลน์ 30 เป็นเหตุให้ A$ มีมูลค่าเป็น ALL และ B$ มีมูลค่าเป็น ONE

SYSTEM Command

รูปแบบ SYSTEM

จุดประสงค์ สิ้นสุด GWBASIC และหันกลับไปยัง DOS

รายละเอียด - คำสั่งนี้กระทำ "warm boot" นั่นคือ ไฟล์ทุก ๆ ไฟล์ที่ถูกเปิดอยู่จะถูกปิด และการควบคุมจะถูกส่งกลับไปยัง DOS

TAB Function

รูปแบบ TAB(I)

จุดประสงค์ เลื่อนไปยังตำแหน่งที่ I

รายละเอียด - I เป็นจำนวนเต็มมีค่าอยู่ในช่วง 1 ถึง 255

- ถ้าตำแหน่งการพิมพ์ในขณะนั้น (current print position) อยู่พ้นจาก space I แล้ว TAB จะไปยังตำแหน่งนั้นบนไลน์ถัดไป Space 1 เป็นตำแหน่งทางซ้ายมือสุดและตำแหน่งทางขวามือสุดจะเป็นความกว้างที่ถูกบ่งบอก

- TAB อาจจะถูกใช้เฉพาะในคำสั่ง PRINT, LPRINT และคำสั่ง PRINT # เท่านั้น

ตัวอย่าง

```
10 PRINT "NAME" TAB(25) "AMOUNT" : PRINT
20 READ A$,B$
30 PRINT A$ TAB(25) B$
40 DATA "G. T. JONES","$25.00"
RUN
 NAME                      AMOUNT

 G. T. JONES               $25.00
Ok
```

TAB เป็นเหตุให้สดมภ์ของ NAME และสดมภ์ของ AMOUNT วางอยู่ในแนว

TAN Function

รูปแบบ v = TAN(x)

จุดประสงค์ ส่งค่า tangent ของ x

รายละเอียด - x เป็นมุมมีหน่วยเป็นเรเดียน เพื่อที่จะแปลงค่าขององศาให้เป็นเรเดียน ให้คูณด้วย PI/180 เมื่อ PI=3.141593

- TAN(x) ถูกคำนวณอยู่ในรูป single precision

- ถ้า TAN มีค่ามากเกินไป (overflow) ข้อความผิดพลาด "Overflow" จะถูกพิมพ์ ค่ามากที่สุดของเครื่อง (machine infinity) พร้อมด้วยเครื่องหมายที่เหมาะสมจะถูกกำหนดในฐานะเป็นผลลัพธ์และการประมวลผลจะดำเนินการต่อไป

ตัวอย่าง 10 Y=Q*TAN(X)/2

Y มีค่าเท่ากับ Q คูณด้วย tangent ของ X หารด้วย 2

TIME$ Statement

รูปแบบ TIME$=a$

จุดประสงค์ กำหนดเวลา

รายละเอียด - a$ ส่งสตริงก์ 1 สตริงก์จากสตริงก์ต่าง ๆ ต่อไปนี้ :

hh กำหนดชั่วโมง นาทีและวินาทีจะมี default เป็น 00

hh:mm กำหนดชั่วโมงและนาที วินาทีจะมี default เป็น 00

hh:mm:ss กำหนดชั่วโมง นาทีและวินาที

- คำสั่งนี้เป็นคอมพลีเมนต์ของตัวแปร TIME$ ซึ่งทำให้ได้เวลากลับคืนมา (retrieve the time)

ตัวอย่าง 10 TIME$="17:00:00"

เวลาปัจจุบัน (current time) กำหนดเป็น 5.00 p.m.

TIME$ Variable

รูปแบบ a$ = TIME$

จุดประสงค์ ทำให้ได้เวลาปัจจุบัน (current time) กลับคืนมา

รายละเอียด - ฟังก์ชันนี้ส่งสตริงก์ขนาด 8 อักขระในรูปแบบ hh:mm:ss เมื่อ hh เป็นชั่วโมง (00 ถึง 23) mm เป็นนาที (00 ถึง 59) และ ss เป็นวินาที (00 ถึง 59 ดังนั้น 8:00 p.m. จะถูกแสดงเป็น 20:00:00)

- เพื่อที่จะกำหนดเวลา ดูคำสั่ง TIME$

ตัวอย่าง 10 PRINT TIME$

พิมพ์เวลาที่กำหนดด้วยคำสั่ง TIME$

TRON and TROFF Command

รูปแบบ TRON

TROFF

จุดประสงค์ กำหนดรอยทาง (trace) การประมวลผลของคำสั่งต่าง ๆ ของโปรแกรม

รายละเอียด - เพื่อช่วยเหลือในการปรับปรุงแก้ไข (debugging) คำสั่ง TRON ทำให้ธงสัญญาณกำหนดรอยทาง (trace flag) มีความสามารถ โดยจะพิมพ์ไลน์นัมเบอร์ของโปรแกรมแต่ละไลน์ตามที่มันถูกประมวลผล ตัวเลขต่าง ๆ ที่ปรากฏจะถูกห้อมล้อมอยู่ภายในเครื่องหมายวงเล็บเหลี่ยม ([ ])

ธงสัญญาณกำหนดรอยทางถูกทำให้ไร้ความสามารถโดยคำสั่ง TROFF

ตัวอย่าง

```
TRON
Ok
LIST
10 K=10
20 FOR J=1 TO 2
30 L=K+10
40  PRINT  J;K;L
50 K=K+10
60  NEXT
70 END
Ok
RUN
[10][20][30][40] 1 10 20
[50][60][30][40] 2 20 30
[50][60][70]
Ok
TROFF
Ok
```

TRON และ TROFF ถูกใช้เพื่อที่จะกำหนดรอยทางการประมวลผลของลูป ไลน์นัมเบอร์ต่าง ๆ จะอยู่ภายในเครื่องหมายวงเล็บเหลี่ยม ([ ]) ตัวเลขอื่น ๆ เป็นมูลค่าต่าง ๆ ของ J, K และ L ตามที่ถูกพิมพ์โดยโปรแกรมในไลน์ 40

USR Function

รูปแบบ v = USR[n](arg)

จุดประสงค์ เรียกสับรูทีนภาษาเครื่องด้วย arg

รายละเอียด - n มีค่าอยู่ในช่วง 0 ถึง 9 และสอดคล้องกันกับตัวเลขที่ถูกกำหนดด้วยคำสั่ง DEF USR สำหรับรูทีนนั้น (ดูคำสั่ง DEF USR) ถ้า n ถูกละแล้ว USR0 จะถูกกำหนด

- arg เป็นนิพจน์จำนวนเลขใด ๆ หรือเป็นตัวแปรสตริงก์ซึ่งจะเป็นอาร์กิว-เม้นต์ (argument) ไปยังสับรูทีนภาษาเครื่อง

- สำหรับฟังก์ชัน USR แต่ละฟังก์ชัน คำสั่ง DEF USR ที่สอดคล้องกันต้องถูกประมวลผลเพื่อที่จะบ่งบอกภาวะเริ่มแรก (offset) ของ USR call ภาวะเริ่มแรกและเซกเม้นต์แอดเดรส(segment address) ของ active DEF SEG ที่เป็นอยู่ในขณะนั้นนี้จะกำหนดแอดเดรสเริ่มต้นของสับรูทีน

- ถ้าเซกเม้นต์อื่นที่นอกเหนือจากเซกเม้นต์ที่เป็น default จะถูกใช้แล้ว คำสั่ง DEF SEG ต้องถูกประมวลผลก่อนไปยัง USR function call แอดเดรสที่ถูกกำหนดให้ในคำสั่ง DEF SEG เป็นสิ่งกำหนดเซกเม้นต์แอด-เดรสของสับรูทีน

ตัวอย่าง

```
100 DEF SEG = &H8000
110 DEF USR0 = 0
120 X = 5
130 Y = USR0(X)
140 PRINT Y
```

รูทีนภาษาเครื่องที่เซกเม้นต์ 8000 hex ภาวะเริ่มต้นเป็น 0 ถูกเรียกด้วย arg ที่กำหนดค่าเป็น 5

VAL Function

รูปแบบ v = VAL(x$)

จุดประสงค์ ส่งมูลค่าที่เป็นจำนวนเลขของสตริงก์ x$

รายละเอียด - x$ เป็นนิพจน์สตริงก์

- ฟังก์ชัน VAL จะนำเอาที่ว่างต่าง ๆ ที่อยู่ข้างหน้า (leading blanks) tabs และ line feeds ออกจากอาร์กิวเม้นต์สตริงก์ ตัวอย่างเช่น

VAL(" -3")

ส่งค่า -3

- VAL(x$) จะส่งค่า 0 ถ้าอักขระตัวแรกของ x$ ไม่เป็นจำนวนเลข

- สำหรับการแปลงค่าจำนวนเลขและการแปลงค่าสตริงก์ ดูฟังก์ชัน STR$

ตัวอย่าง

```
10  READ  NAME$,CITY$,STATE$,ZIP$
20 IF VAL(ZIP$)<90000 OR VAL(ZIP$)>96699
   THEN PRINT NAME$ TAB(25) "OUT OF STATE"
30 IF VAL(ZIP$)>=90801 AND VAL(ZIP$)<=90815
   THEN PRINT NAME$ TAB(25) "LONG BEACH"
```

ในตัวอย่างนี้ VAL ถูกใช้เพื่อที่จะบ่งชี้ชื่อต่าง ๆ ของแต่ละบุคคลที่อาศัยอยู่นอกรัฐคาลิฟฟอร์เนีย (ไลน์ 20) ไลน์ 30 VAL ถูกใช้เพื่อที่จะบ่งชี้ชื่อต่าง ๆ ของแต่ละบุคคลที่อาศัยอยู่ในเมืองลองบีช รัฐคาลิฟฟอร์เนีย

VARPTR Function

รูปแบบ v = VARPTR(variable)

v = VARPTR(#filenum)

จุดประสงค์ ส่งแอดเดรสในหน่วยความจำของตัวแปรหรือของบล็อกควบคุมไฟล์ (file control block)

รายละเอียด - variable เป็นตัวแปรจำนวนเลขหรือตัวแปรสตริงก์หรือเป็นสมาชิกของอะเรย์ในโปรแกรมของท่าน variable ต้องมีมูลค่าที่ถูกกำหนดไว้ก่อนหน้าแล้ว มิฉะนั้นจะเกิด "Illegal function call" error ขึ้น

- filenum เป็นตัวเลขภายใต้ไฟล์ที่ถูกเปิด
- โดยปกติ VARPTR ถูกใช้เพื่อที่จะรับ (obtain) แอดเดรสของตัวแปรหรืออะเรย์เพื่อว่ามันอาจจะถูกส่งผ่านไปยังสับรูทีนภาษาเครื่อง การเรียกฟังก์ชันในรูป VARPTR(A(0)) โดยปกติถูกระบุเมื่อทำการส่งผ่านอะเรย์เพื่อว่าสมาชิกของอะเรย์ที่ถูกกำหนดแอดเดรสต่ำที่สุดจะถูกส่งค่าไป
- หมายเหตุ : ตัวแปร simple variable ทุก ๆ ตัวควรจะถูกกำหนดค่าก่อนที่จะทำการเรียก VARPTR สำหรับอะเรย์เนื่องจากแอดเดรสต่าง ๆ ของอะเรย์ต่าง ๆ จะเปลี่ยนแปลงไปเมื่อใดก็ตามที่ตัวแปร simple variable ตัวใหม่ถูกกำหนดค่า
- สำหรับรูปแบบทั้งสองนั้น แอดเดรสจะมีค่าเป็นจำนวนเต็มในช่วง 0 ถึง 65535
- สำหรับตัวแปรสตริงก์ต่าง ๆ แอดเดรสของไบท์แรกของ string descriptor จะถูกส่งค่าไป สำหรับ sequential files แล้ว แอดเดรสเริ่มต้นของบัฟเฟอร์ของดิสค์ไอ/โอ (disk I/O buffer) ที่ถูกกำหนดค่าไปยัง filenum จะถูกส่งค่าไป สำหรับ random files แอดเดรสของบัฟเฟอร์ของฟิลด์ (field buffer) ที่ถูกกำหนดค่าไปยัง filenum จะถูกส่งค่าไป

ตัวอย่าง

```
100 X=USR(VARPTR(Y))
```

ตัวอย่างนี้ส่งค่าแอดเดรสเริ่มแรก (offset address) ของตัวแปร Y

VARPTR Function

รูปแบบ v$ = VARPTR(variable)

จุดประสงค์ ส่งค่าสตริงก์ซึ่งบ่งบอกชนิดของตัวแปรและแอดเดรสของตัวแปรนั้นในหน่วยความจำ

รายละเอียด - variable เป็นชื่อของตัวแปร (numeric, string, or array) ที่ปรากฏ(มี)อยู่ในโปรแกรม

- ฟังก์ชันนี้ถูกใช้กำหนดเริ่มแรกร่วมกับ PLAY และ DRAW ในโปรแกรมต่าง ๆ ซึ่งจะถูกแปล (complied) ต่อมา

- มูลค่าต้องถูกกำหนดไปยัง variable ก่อนที่ฟังก์ชันนี้จะปฏิบัติการ มิฉะนั้นจะเกิด "Illegal function call" error ขึ้น

- VARPTR ส่งค่าสตริงก์ขนาด 3 ไบท์ในรูป :

ไบท์ 0	ชนิด (type)
ไบท์ 1	ไบท์ต่ำ (low byte) ของแอดเดรสของตัวแปร
ไบท์ 2	ไบท์สูง (high byte) ของแอดเดรสของตัวแปร

ชนิด (type) บ่งชี้ชนิดของตัวแปรดังต่อไปนี้ :

2	integer
3	string
4	single-precision
8	double-precision

- มูลค่าที่ถูกส่งไปจะเป็นเช่นเดียวกันกับ CHR$ (type+MKI$(VARPTR(variable))

- เนื่องจากแอดเดรสต่าง ๆ ของอะเรย์ต่าง ๆ เปลี่ยนแปลงไปเมื่อใดก็ตามที่ตัวแปร simple variable ตัวใหม่ถูกกำหนดค่า ตัวแปร simple variable ทุกตัวควรจะถูกกำหนดค่าก่อนการเรียก VARPTR สำหรับสมาชิกของอะเรย์

ตัวอย่าง

```
100 A$ = "ABCDEFG"
200 PLAY "X" + VARPTR(A$)
```

VARPTR ได้ถูกใช้เพื่อที่จะบ่งชี้สตริงก์ "ABCDEFG"

WAIT Statement

รูปแบบ WAIT port, I[,J]

จุดประสงค์ หยุดการประมวลผลชั่วคราวในระหว่างการ monitoring ตามสถานะของ machine input port

รายละอียด
- port เป็นตัวเลขในช่วง 0 ถึง 65535 ซึ่งเป็นแอดเดรสของ port
- I,J เป็นนิพจน์จำนวนเต็มต่าง ๆ มีค่าอยู่ในช่วง 0 ถึง 255
- คำสั่ง WAIT เป็นเหตุให้การประมวลผลถูกทำให้หยุดลงชั่วคราวจนกว่า machine input port ที่ถูกระบุจะปรับปรุง (develop) รูปแบบของบิทที่ถูกระบุ (specified bit pattern)
- ข้อมูลที่ถูกอ่าน ณ port จะ XOR กับ J และจะ AND กับ I ถ้าผลลัพธ์ที่ได้เป็นศูนย์แล้ว GWBASIC จะวนกลับไปและอ่านข้อมูลที่ port อีกครั้งหนึ่ง ถ้าผลลัพธ์ที่ได้ไม่เป็นศูนย์แล้ว การประมวลผลจะดำเนินการต่อไปกับคำสั่งที่อยู่ถัดไป ถ้า J ถูกละไว้แล้ว มันจะถูกกำหนดค่าเป็น 0
- หมายเหตุ : อาจเป็นไปได้ที่จะป้อนลูปที่ไม่สิ้นสุด (infinite loop) เข้าไปด้วย WAIT ในกรณีที่ซึ่งมันจะเป็นสิ่งจำเป็นที่จะ reboot ด้วย Ctrl-Alt-Del หรือ System Reset เพื่อที่จะหลีกเลี่ยงสถานะการณ์เช่นนี้ WAIT ต้องมีมูลค่าที่ถูกระบุ ณ port ในระหว่างจุดบางจุด (some point) ในการประมวลผลโปรแกรม

ตัวอย่าง

```
10 WAIT 40,8
```

WAIT หยุดการประมวลผลชั่วขณะจนกว่าจะมี 1 ในตำแหน่งบิทที่สี่ใน port 40

WHILE and WEND Statements

รูปแบบ WHILE expression

.

.

(loop statements)

.

WEND

จุดประสงค์ ประมวลผลชุดลำดับ (series) ของคำสั่งต่าง ๆ ในลูปนานเท่านานตราบเท่าที่เงื่อนไขที่ถูกกำหนดให้เป็นจริง

รายละเอียด

- expression เป็นนิพจน์จำนวนเลขใด ๆ
- ถ้า expression มีค่าไม่เป็นศูนย์ (นั่นคือเป็นจริง) แล้วคำสั่งต่าง ๆ ในลูป (loop statements) จะถูกประมวลผลจนกว่าคำสั่ง WEND จะถูกพบ จากนั้น GWBASIC จะส่งการปฏิบัติการกลับไปยังคำสั่ง WHILE และตรวจสอบ expression ถ้าหากว่ายังคงเป็นจริงอยู่ กระบวนการจะถูกกระทำซ้ำ ถ้าไม่เป็นจริง การประมวลผลจะกระทำต่อไปกับคำสั่งที่อยู่ถัดจากคำสั่ง WEND
- ลูปต่าง ๆ ของ WHILE...WEND อาจจะถูกซ้อนกันในหลาย ๆ ระดับ WEND แต่ละ WEND จะจับคู่ (match) กับ WHILE ที่ใกล้เคียงกันมากที่สุด (most recent WHILE) คำสั่ง WHILE ที่ไม่ถูกจับคู่จะเป็นเหตุให้เกิด "WHILE without WEND" error ขึ้นและคำสั่ง WEND ที่ไม่ถูกจับคู่จะเป็นเหตุให้เกิด "WEND without WHILE" error ขึ้น

ตัวอย่าง

```
90 'BUBBLE SORT ARRAY A$
100 FLIPS=1 'FORCE ONE PASS THRU LOOP
110 WHILE FLIPS
115         FLIPS=0
120         FOR I=1 TO J-1
130                 IF A$(I)>A$(I+1) THEN
                            SWAP A$(I),A$(I+1):FLIPS=1
140         NEXT I
150 WEND
```

สมาชิกต่าง ๆ ของ A$ ถูกเรียงลำดับตามตัวอักษร (A$ ถูกบ่งบอกด้วยสมาชิก J ตัว)

WIDTH Statement

รูปแบบ

```
WIDTH size
WIDTH filenum,size
WIDTH device,size
```

จุดประสงค์ กำหนดความกว้างของไลน์สำหรับจอภาพหรือเครื่องพิมพ์เป็นจำนวนของอักขระต่าง ๆ

รายละเอียด
- size เป็นตัวเลขจาก 0 ถึง 255 ความกว้างที่ default จะเป็น 80 อักขระ หลังจากที่อักขระต่าง ๆ ของ size ถูกแสดงผลหรือถูกพิมพ์แล้ว GWBASIC จะใส่ carriage return line feed sequence เข้าไปก่อนที่อักขระถัดไปจะถูกปฏิบัติ
- filenum เป็นตัวเลขจาก 1 ถึง 15 ตัวเลขของไฟล์ที่ถูกเปิดไปที่อุปกรณ์ (device) ถูกแสดงรายการข้างล่าง

- device เป็นสิ่งใดสิ่งหนึ่งต่อไปนี้ : SCRN:, LPT1:, LFT':, COM1:, หรือ COM2:
- รูปแบบแรกกำหนดความกว้างของจอภาพเป็น 40 สดมภ์หรือไม่ก็เป็น 80 สดมภ์ตาม WIDTH "SCRN:", size
- การระบุ WIDTH 255 ทำให้ความกว้างของไลน์ไม่สิ้นสุด (line width infinite)
- สังเกตว่า WIDTH 80 ที่อยู่ต่อจาก SCREEN 1 ส่ง (force) จอภาพไปสู่ high resolution และ WIDTH 40 ที่อยู่ต่อจาก SCREEN 2 จะส่งจอภาพไปสู่ medium resolution
- การเปลี่ยนแปลงความกว้างของจอภาพจะเป็นเหตุให้จอภาพจอภาพถูกเคลียร์และสีของจอภาพที่เป็นขอบ (border screen color) จะถูกกำหนดเป็นสีดำ ดูคำสั่ง SCREEN
- WIDTH filenum,size อนุญาตให้เปลี่ยนแปลงความกว้างของอุปกรณ์ที่ถูกเกี่ยวข้องกับ filenum ในขณะใด ๆ ที่ไฟล์ถูกเปิด เราสามารถใช้กับ LPT1:, LPT2:, COM1: และ COM2:
- WIDTH device,size เก็บ (store) การกล่าวอ้างของความกว้าง (width assignment) โดยปราศจากการเปลี่ยนแปลงการกำหนดค่าปัจจุบัน คำสั่ง OPEN ลำดับถัดมาจะใช้มูลค่าใหม่ ถ้าอุปกรณ์ถูกเปิดไว้ความกว้างจะไม่เปลี่ยนแปลงทันที
- LPRINT, LLIST และ LIST,"LPTn:" ทั้งหมดได้รับผลกระทบโดยคำสั่งนี้
- มูลค่าต่าง ๆ ที่อยู่นอกช่วงส่งผลให้เกิด "Illegal function call " error ขึ้นและมูลค่าก่อนหน้าจะถูกรักษาไว้
- หมายเหตุ : สำหรับ communication files การเปลี่ยนแปลงความกว้างไม่ได้เปลี่ยนแปลงขนาดหรือ contents ของบัฟเฟอร์ที่รับหรือบัฟเฟอร์ที่ส่ง

ตัวอย่าง

```
10 WIDTH "LPT1:",72
20 OPEN "LPT1:" FOR OUTPUT AS #1
.
.
.
100 WIDTH #1,60
```

ไลน์ 10 เก็บความกว้างของเครื่องพิมพ์เป็นจำนวนอักขระ 72 ตัวต่อไลน์ ไลน์ 20 เปิดไฟล์ #1 ไปยังเครื่องพิมพ์และกำหนดความกว้างเป็น 75 สำหรับคำสั่ง PRINT #1,... ต่าง ๆ ต่อมาและไลน์ 100 เปลี่ยนแปลงความกว้างของเครื่องพิมพ์ในปัจจุบันเป็น 60 อักขระต่อไลน์

WRITE Statement

รูปแบบ WRITE [list of expressions]

จุดประสงค์ แสดงผลลัพธ์ข้อมูลบนจอภาพ

รายละเอียด
- list of expressions ประกอบด้วยนิพจน์จำนวนเลขและ/หรือนิพจน์สตริงก์ถูกแบ่งแยกด้วยเครื่องหมายจุลภาคหรือเครื่องหมายเซมิโคลอน
- ถ้า list of expressions ถูกละแล้ว ไลน์ว่างเปล่า (blank line) จะเป็นผลลัพธ์
- เมื่อ items เป็นผลลัพธ์ item แต่ละตัวจะถูกแบ่งแยกจากตัวล่าสุดด้วยเครื่องหมายจุลภาค สตริงก์ต่าง ๆ ถูกกำหนดขอบเขตด้วยเครื่องหมายคำพูด หลังจากที่ item ตัวล่าสุดในรายการถูกพิมพ์ GWBASIC จะใส่ carriage return/line feed เข้าไป
- ดูคำสั่ง PRINT ด้วย

ตัวอย่าง

```
10 A=80: B=90: C$="THAT'S ALL"
20 WRITE A,B,C$
R U N
 80,90,"THAT'S ALL"
Ok
```

WRITE ถูกใช้กับทั้งมูลค่าจำนวนเลขและมูลค่าสตริงก์

WRITE # Statement

รูปแบบ WRITE #filenum, list of expressions

จุดประสงค์ บันทึกข้อมูลไปยัง sequential file

รายละเอียด
- filenum เป็นตัวเลขภายใต้ไฟล์ที่ถูกเปิด
- list of expressions ประกอบด้วยนิพจน์สตริงก์และ/หรือนิพจน์จำนวนเลข ถูกแบ่งแยกโดยเครื่องหมายจุลภาคหรือเครื่องหมายเซมิโคลอน
- WRITE # และ PRINT # คล้ายคลึงกันมากยกเว้นว่า WRITE # ใส่เครื่องหมายจุลภาคระหว่าง item ต่าง ๆ ที่ถูกบันทึกและทำการคลุมสตริงก์ต่าง ๆ ด้วยเครื่องหมายคำพูดและไม่ใส่ที่ว่างข้างหน้าจำนวนบวก
- carriage return/line feed ถูกใส่เข้าไปหลังจากที่ item ล่าสุดในรายการถูกบันทึก

ตัวอย่าง ให้ A$="HELLO" และ B$="FOLKS" คำสั่ง :

```
WRITE #1,A$,B$
```

บันทึกภาพ (image) ต่อไปนี้ไปยังไฟล์ :

```
"HELLO","FOLKS"
```

เช่นเดียวกับคำสั่ง INPUT # ตัวอย่างเช่น INPUT #1,A$,B$

จะบันทึก (input) "HELLO" ไปที่ A$ และ "FOLKS" ไปที่ B$