IF / THEN / ELSE - indicates a command that is executed if an expression is TRUE!

```
IF (expression) THEN (command)
IF a$="quit" THEN goto 100
IF b-a < 0 THEN print "negative result"
IF err=53 THEN print "out of data" ELSE goto line5:
```

AND / OR - used in conjunction with conditional statements as a connector / modifier

```
if a=0 AND b=4 then goto 100
if dir$="up" AND status$="ok" then goto line4:
if number>0 OR score>100 then print "game over"
```

SELECT CASE / CASE / END SELECT - used to set up a block of options, only ONE of which gets executed, depending on the value of the variable stated in the select statement

```
SELECT CASE NUMBER
CASE 1
  set of instructions for this option
CASE 2
  set of instructions for this option
CASE ELSE
  set of instructions for this option
END SELECT

SELECT CASE ALPHA$
CASE "A"
  set of instructions for this option
CASE "B"
  set of instructions for this option
CASE ELSE
  set of instructions for this option
END SELECT
```

FOR / NEXT / STEP - creates an unconditional loop with an internal index number. Index must be a numeric variable, bottom and top are the two extremes of the index number. STEP may be used to count in increments other than +1.

```
FOR index = bottom TO top STEP 10
do these commands
NEXT index

FOR r = 10 TO 1000 STEP 15
print "my aren't we having fun for the "; r; "th time."
NEXT r
```

DO / LOOP (while / until) - used to execute a section of the program repetitively UNTIL a condition becomes true, or WHILE a certain condition is true

```
DO
a=rnd
b=rnd
c=a+b
LOOP UNTIL c > 1.3
```

LEFT$ - extracts the (, #) left-most characters from an alphanumeric variable and assigns them to another variable or to the same variable.

RIGHT$ - extracts the (, #) right-most characters from an alphanumeric variable and assigns them to another variable or to the same variable.

MID$ - extracts (, ,#) characters from an alphanumeric variable starting with the (,#,)-th character, and assigns them to another variable, or the same variable

```
print LEFT$(variable$, 5)
a$ = LEFT$(a$, 12)
b$ = RIGHT(a$, 12)
print RIGHT$("abcde", 3)
print MID$("abcdefgh", 3, 2)
print MID$(alphastring$, 1, 1)
```

UCASE$ - changes all letters to uppercase

LCASE$ - changes all letters to lowercase

```
print LCASE$(A$)
A$= UCASE$(B$)
COMMAND$= UCASE$("oldcommand")
```

SWAP - switches values in two variables

```
SWAP A$, B$
SWAP CMD$, STATUS$
```

NUMERIC MANIPULATION:

```
* multiplication 16 = 4 * 4
/ division 3 = 27 / 9
+ addition 100 = 63 + 37
- subtraction 54 = 60 - 6
^ exponentiation 100 = 10 ^ 2
square root 8 = 64 ^ 0.5
() brackets used to order of (6 + 4) * 10
operations
< less than sign 4 < 8
> greater than sign 16 > -54
= equal to sign C = A + B ^ 3
```

**INKEY$** - used to find out what the single next character is, from the keyboard buffer. Should be used by assigning the INKEY$ value to an alphanumeric variable, then checking the value of that variable.

```
a$ = INKEY$
cmd$ = INKEY$: if cmd$ = "Q" then goto quit:
do: a$ = INKEY$: loop while a$=""
```

DATA / READ - used to put information into variable from DATA lines stored within the text of the program. Variables used in READ line must match with information written in DATA line

```
READ name$, age, address$
DATA Frank, 14, 123 Nowhere Street
READ a,b,c,d,e,f$
DATA 5,12,65,23,19,ENGLISH
```

TIME$ - used to print the current time, or to assign it to an alphanumeric variable.

```
A$= TIME$
locate 1,1: print TIME$
```

CHR$ - used to print characters other than the standard keyboard letters and numbers. Can be used in a print statement, a condition expression, or assigned to an alphanumeric variable

```
print CHR$(12)
if string$=CHR$(100) then goto quit:
command$=CHR$(65)
```

DEFINT - rounds off (to the nearest whole number) all numeric variables which start with same letter as given in the statement.

```
DEFINT A
DEFINT A-B, Q-Z
```

DIM - creates storage space for multiple variables under a single name, with access provided through index numbers.

```
DIM name$(25)
DIM student$(100), mark(100,5), age(100)
```

RND - accesses the next number from the computer's random number table. (always between ZERO and 1) Can be printed, or assigned to a numeric variable or multiplied to get a random number from within a
larger range.

```
print RND * 100
number = RND * 1000
```

SPACE$ - used to print a long string of spacebar characters or assign a string of variables to an alphanumeric variable.

```
print SPACE$(40)
a$ = SPACE$(65)
```

INPUT - used to get information from the keyboard during running of the program, and storing that information in a variable.

```
INPUT a,b,c,d,e
INPUT number, number2
INPUT "prompt telling you what to type", variable$
```

LET - used to assign a value to a variable during running of a program, without keyboard assistance. Left side must always contain a SINGLE variable, either numeric or alphanumeric.

```
LET g=4
LET status$="quit"
```

LOCATE - changes the current print location to the row and column given

```
LOCATE row, col
LOCATE 15,1: print "this will show up on row 15,
column 1"
```

PRINT - prints variables following it, on the screen at the current print location. Used alone, it prints a blank line.

```
PRINT "message appearing on the screen"
PRINT "combo" + " of " + "messages!"
PRINT "combo of", a$, b$ " variables"
```

; - prints next text right next to this text
, - prints next text tabbed to the next major column

SCREEN - changes the screen from text mode (0), to med-res graphics (1) to hi-res graphics (2).

```
SCREEN 0
SCREEN 1
SCREEN 2
```

WIDTH - changes the screen width from 80 narrow characters to wide characters, or back to 80 width.

```
WIDTH 80
```

PSET - highlights a single pixel on the screen

```
PSET (320,240), 4
```

LINE - draws a line between two points on the screen

```
LINE (31,45) - (135,400), 15
```

COLOR - picks a color for text to be printed in.

```
COLOR 12
```

PALETTE - used to change the ratios of blue, green and red in a given colour

```
palette blue * 65536 + green * 256 + red
```

STEP - used to make the finishing point of a line statement relative to the starting point, rather than a fixed point on the screen

```
LINE (31,45) - step(10,10), 15
```