

Minerva4Q68

Keywords

This Keyword Reference Guide lists all the Minerva4Q68 keywords in alphabetical order: A brief explanation of the keywords function is given followed by loose definition of the syntax and examples of usage.

This guide is a combination of the Sinclair QL manuals Keyword section, the Minerva manual, and the Q68 manual.

The Minerva operating system was originally designed as a replacement ROM operating system for the Sinclair QL computer, currently licenced under GPLv3. This port is aimed at the Q68, an FPGA-based replacement board for the QL. It is not intended as a replacement for the SMSQ/E OS supplied with the Q68, as SMSQ/E is far more extensive and better suited to support the Q68 hardware than the 48K ROM-based Minerva. We just provide this port to demonstrate the Q68's ability to run 'old school' ROM images, give Q68 users the Minerva look and feel, and maybe provide an opportunity to run badly written software that doesn't run on SMSQ/E (but chances are big that this software won't run on Minerva either).

The current Minerva build is based on v1.98, with a few modifications to run successfully on the Q68.

Note - The keywords described in this document are for Minerva4Q68 when it is first started. It does not include any extensions such as Toolkit 2.

© 1984 SINCLAIR RESEARCH LIMITED
© JAN BREDENBEEK
© 1994-2002 TONY TEBBY
© PETER GRAF
© DEREK STEWART
© WOLFGANG LENERZ
© LAURENCE REEVES

ABS maths functions

ABS returns the absolute value of the parameter. It will return the value of the parameter if the parameter is positive and will return zero minus the value of the parameter if the parameter is negative.

The ABS function now takes a list of numeric parameters, and returns the square-root of the sum of the squares of it's parameters. This leaves the original function unchanged, though the one parameter call is not done by squaring and square-rooting the parameter

syntax. **ABS**(*numeric_expression*)
 ABS(*numeric_expression* *[,*numeric_expression*])

example: i. **PRINT ABS(0.5)**
 ii. **PRINT ABS(a-b)**
 iii. **PRINT ABS(a,b,c,d,e)**

ACOS, ASIN

ACOT, ATAN maths functions

ACOS and **ASIN** will compute the arc cosine and the arc sine respectively. **ACOT** will calculate the arc cotangent and **ATAN** will calculate the arc tangent. There is no effective limit to the size of the parameter.

ATAN will provide a 4 quadrant result by taking two parameters. If x is greater than 0, **ATAN** (x,y) give the same results as ATAN (y/x). Otherwise it returns values in the other quadrants (>PI/2 and <-PI/2).

syntax: *angle:= numeric_expression* [in radians]

 ACOS (*angle*) **ACOT** (*angle*)
 ASIN (*angle*) **ATAN** (*angle* [,*angle*])

example: i. **PRINT ATAN(angle)**
 ii. **PRINT ASIN(1)**
 iii. **PRINT ACOT(3.6574)**
 iv. **PRINT ATAN(a-b)**

ADATE clock

ADATE allows the clock to be adjusted.

Note that **ADATE** does not set the battery backed clock in the Q68. To set the battery backed real time clock. First set the date and time with **ADATE**, then execute the clock utility program named 'Q68SETRTC'

syntax: *seconds:= numeric_expression*

 ADATE *seconds*

example: i. **ADATE 3600** {advance the clock 1 hour}
 ii. **ADATE -60** {move the clock back 1 minute}

ALFM memory management

The function **ALFM** will allocate a requested amount of memory from the 'Fast Memory' area and return the base address of the space.

If you try to reserve more memory than is available, then the function returns with an out of memory error and no memory will have been reserved.

Initially there is about 10K bytes of memory available.

syntax: *number_of_bytes := numeric_expression*

ALFM (*number_of_bytes*)

example: **base = ALFM (3000)** {allocate 3000 bytes from the fast memory}

note: Once memory has been allocated with **ALFM**, it cannot be de-allocated

warning: The system does NOT stop you from using more than the amount of memory that you requested, but sooner or later the system is likely to crash.

ARC

ARC_R graphics

ARC will draw an arc of a circle between two specified points in the *window* attached to the default or specified channel. The end points of the arc are specified using the *graphics co-ordinate* system.

Multiple arcs can be drawn with a single **ARC** command.

The end points of the arc can be specified in absolute coordinates (relative to the *graphics origin*) or in relative coordinates (relative to the *graphics cursor*). If the first point is omitted then the arc is drawn from the graphics cursor to the specified point through the specified angle.

ARC will always draw with absolute coordinates, while **ARC_R** will always draw relative to the graphics cursor.

syntax: *x := numeric_expression*
y := numeric_expression
angle := numeric_expression (in radians)
point := x,y

parameter_2 := | **TO point, angle** (1)
 | **,point TO point, angle** (2)

parameter_1 := | **point TO point, angle** (1)
 | **TO point, angle** (2)

ARC [*channel*,] *parameter_1* * [*parameter_2*]*
ARC_R [*channel*,] *parameter_1* * [*parameter_2*]*

where (1) will draw from the specified point to the next specified point turning through the specified angle

(2) will draw from the last point plotted to the specified point turning through the specified angle

example: i. **ARC 15,10 TO 40,40,PI/2**
 {draw an arc from 15,10 to 40,40 turning through PI/2 radians}
ii. **ARC TO 50,50,PI/2**
 {draw an arc from the last point plotted to 50,50 turning through PI/2 radians}
iii. **ARC_R 10,10 TO 55,45,0.5**
 {draw an arc, starting 10,10 from the last point plotted to 55,45 from the start of the arc, turning through 0.5 radians}

AT windows

AT allows the print position to be modified on an imaginary row/column grid based on the current character size. **AT** uses a modified form of the *pixel coordinate system* where (row 0, column 0) is in the top left hand corner of the window. **AT** affects the print position in the window attached to the specified or default channel.

syntax: *line:= numeric_expression*
column:= numeric_expression

AT [*channel*,] *line*, *column*

example: **AT 10,20 : PRINT "This is at line 10 column 20"**

AUTO SuperBASIC editor

AUTO allows line numbers to be generated automatically when entering programs directly into the computer. **AUTO** will generate the next number in sequence and will then enter the SuperBASIC line editor while the line is typed in. If the line already exists then a copy of the line is presented along with the line number. Pressing **ENTER** at any point in the line will check the syntax of the whole line and will enter it into the program.

AUTO is terminated by pressing **CTRL - SPACE**

AUTO accepts enhanced movement keys as follows:

ALT ←/→	move to start/end of current line
TAB	move along to 8th character from start of buffer
SHIFT-TAB	move back in the same steps as above
CTRL-ALT ←	deletes to start of current visible line
CTRL-ALT →	deletes from current character to total end of line
ESCape	behaves like CTRL/SPACE (Break)
SHIFT-ENTER	behaves pretty much like ENTER
SHIFT-SPACE	behaves pretty much like SPACE

syntax: *first_line:= line_number*
gap:= numeric_expression

AUTO [*first_line*] [,*gap*]

example: i. **AUTO** {start at line 100 with intervals of 10}
ii. **AUTO 10,5** {start at line 10 with intervals of 5}
iii. **AUTO ,7** {start at line 100 with intervals of 7}

BAUD communications

BAUD sets the baud rate for communication via the serial channels. There is only one serial port built into the Q68, so supplying an optional port number is not necessary.

If no port number is supplied, then the command will default to SER1.

syntax: *rate:= numeric_expression*
 port:= numeric_expression

BAUD [*port*,] *rate*

The value of the rate numeric expression must be one of the supported baud rates that are supported by Minerva on the Q68:

1200
2400
4800
9600
19200
38400
57600
115200
230400

If the selected baud rate is not supported, then an error will be generated.

example: i. **BAUD 1,9600** {set SER1 to 9600 baud}
 ii. **BAUD print_speed** {set SER1 to 'print_speed' baud}

BEEP sound

The Q68 tries to emulate the QL's **BEEP** command. This is far from perfect. The sound will often be too "clean".

BEEP can accept a variable number of parameters to give various levels of control over the sound produced. However in the Q68, the "wrap", "random" and "fuzziness" parameters of the **BEEP** command are simply ignored.

The minimum specification requires only a duration and pitch to be specified. **BEEP** used with no parameters will kill any sound being generated.

syntax: *duration*:= *numeric_expression* {range -32768..32767}
 pitch:= *numeric_expression* {range 0..255}
 grad_x:= *numeric_expression* {range -32768..32767}
 grad_y:= *numeric_expression* {range -8..7}
 wrap:= *numeric_expression* {range 0..15} ignored in the Q68
 fuzzy:= *numeric_expression* {range 0..15} ignored in the Q68
 random:= *numeric_expression* {range 0..15} ignored in the Q68

BEEP [*duration*, *pitch*
 [,*pitch_2*, *grad_x*, *grad_y*
 [, *wrap*
 [, *fuzzy*
 [, *random*]]]]]

duration - specifies the duration of the sound in units of 72 microseconds. A duration of zero will run the sound until terminated by another BEEP command.

pitch - specifies the pitch of the sound. A pitch of 1 is high and 255 is low.

Pitch_2 - specifies a second pitch level between which the sound will 'bounce'

grad_x - defines the time interval between pitch steps.

grad_y - defines the size of each step, *grad_x* and *grad_y* control the rate at which the pitch bounces between levels.

wrap - will force the sound to wrap around the specified number of times. If wrap is equal to 15 the sound will wrap around forever:

fuzzy - defines the amount of fuzziness to be added to the sound.

random - defines the amount of randomness to be added to the sound.

note: Currently sound is not implemented in Minerva4Q68

BEEPING sound

BEEPING is a function which will return zero (false) if the Q68 is currently not beeping and a value of one (true) if it is beeping.

syntax: **BEEPING**

example: **100 DEFine PROCedure be_ quiet**
 110 BEEP
 120 END DEFine
 130 IF BEEPING THEN be_ quiet

BLOCK windows

BLOCK will fill a block of the specified size and shape, at the specified position relative to the origin of the *window* attached to the specified, or default *channel*.

BLOCK has been enhanced to accept any 16-bit signed integer values for width, height, and x and y positions. Normal usage is unaffected, but one can use it very similarly to the graphics routines, in that it will now just fill in any part of the area that lies within the screen. e.g.

BLOCK 200,20,-195,-10,7 will behave as **BLOCK 10,5,0,0,7**

The reason for this enhancement is that we wanted something that would accept **BLOCK 2,2,-2,0,7**, and various others, that the JS ROM fails to error trap correctly.

This was causing trouble in some software that used such invalid **BLOCK** commands, and got away with it. Be careful if you try out the invalid **BLOCK** commands on non-Minerva ROM's; the example above actually draws itself on the right hand edge of a full size screen, but we wouldn't guarantee that other parameters won't cause a crash!

BLOCK uses *the pixel coordinate system*.

syntax: *width:= numeric_expression*
 height:= numeric_expression
 x:= numeric_expression
 y:= numeric_expression
 colour:= numeric_expression {range 0 ... 255}

BLOCK [*channel*,] *width, height, x, y, colour*

example: i. **BLOCK 10,10,5,5,7** {10x10 pixel white block at 5,5}
 ii. **BLOCK 200,20,-195,-10,7**

BORDER windows

BORDER will add a border to the window attached to the specified *channel*, or default channel.

For all subsequent operations except **BORDER** the window size is reduced to allow space for the **BORDER**. If another **BORDER** command is used then the full size of the original window is restored prior to the border being added; thus multiple **BORDER** commands have the effect of changing the size and colour of a single border. Multiple borders are not created unless specific action is taken.

If **BORDER** is used without specifying a colour then a transparent border of the specified width is created.

syntax: *width:= numeric_expression*
 colour:= numeric_expression {range 0 ... 255}

BORDER [*channel*,] *width* [, *colour*]

example: i. **BORDER 10,0,7** {black and white stipple border}
 ii. **100 REMark Lurid Borders**
 110 FOR thickness = 50 to 2 STEP -2
 120 BORDER thickness, RND(0 TO 255)
 130 END FOR thickness
 140 BORDER 50

CALL machine code

Machine code can be accessed directly from SuperBASIC by using the **CALL** command. **CALL** can accept up to 13 long word parameters which will be placed into the 68000 data and address registers (D1 to D7, A0 to A5) in sequence.

No data is returned from **CALL**.

syntax: *address:= numeric_expression*
 data:= numeric_expression

CALL *address*, *[*data*]* {13 data parameters maximum}

example: i. **CALL 262144,0,0,0**
 ii. **CALL 262500,12,3,4,1212,6**

warning: Address register A6 should not be used in routines called using this command. To return to SuperBASIC use the instructions:

MOVEQ #0,D0
RTS

CARD_INIT SD cards

CARD_INIT initialises the SD card reader in the Q68. Card1 is automatically initialised at boot time.

By design, card2 is not initialised at boot time, though this will depend on your configuration options. If it is not initialised, you have to initialise it yourself. You can do this with the **CARD_INIT** command. The card itself is not touched by this command (it is not formatted, written to or anything).

syntax: *card:= numeric_expression* {1 or 2}

CARD_INIT *card*

example: **CARD_INIT 2**

CHR\$ SuperBASIC

CHR\$ is a function which will return the character whose value is specified as a parameter: **CHR\$** is the inverse of **CODE**.

syntax: **CHR\$(numeric_expression)**

example: i. **PRINT CHR\$(27)** {print ASCII escape character}
 ii. **PRINT CHR\$(65)** {print A}

ELLIPSE, ELLIPSE_R graphics

CIRCLE will draw a circle (or an ellipse at a specified angle) on the screen at a specified position and size. The circle will be drawn in the *window* attached to the specified or default channel.

CIRCLE uses the *graphics coordinate system* and can use absolute coordinates (i.e. relative to the *graphics origin*), and relative coordinates (i.e. relative to the *graphics cursor*). For relative coordinates use **CIRCLE_R**.

Multiple circles or ellipses can be plotted with a single call to **CIRCLE**. Each set of parameters must be separated from each other with a semi colon (;)

The word **ELLIPSE** can be substituted for **CIRCLE** if required.

```

syntax:  x:= numeric_expression
         y:= numeric_expression
         radius:= numeric_expression
         eccentricity:= numeric_expression
         angle:= numeric_expression           {range 0..2PI}

```

$$\begin{aligned} parameters := & \mid x, y, & (1) \\ & \mid radius, eccentricity, angle & (2) \end{aligned}$$

where

- (1) will draw a circle
- (2) will draw an ellipse of specified eccentricity and angle

CIRCLE [*channel*,] *parameters**[; *parameters*]*

<i>x</i> -	horizontal offset from the graphics origin or graphics cursor
<i>y</i> -	vertical offset from the graphics origin or graphics cursor
<i>radius</i> -	radius of the circle eccentricity the ratio between the major and minor axes of an ellipse.
<i>Angle</i> -	the orientation of the major axis of the ellipse relative to the screen vertical. The angle must be specified in radians.

example: i. **CIRCLE 50,50,20** {a circle at 50,50 radius 20}
 ii. **CIRCLE 50,50,20,0.5,0** {an ellipse at 50,50 major axis 20 eccentricity 0.5 and aligned with the vertical axis}

CLEAR SuperBASIC

CLEAR will clear out the SuperBASIC variable area for the current program and will release the space for Minerva.

syntax: **CLEAR**

example: **CLEAR**

comment: **CLEAR** can be used to restore to a known state for the SuperBASIC system. For example, if a program is broken into (or stops due to an error) while it is in a procedure then SuperBASIC is still in the procedure even after the program has stopped. **CLEAR** will reset SuperBASIC. {See **CONTINUE**, **RETRY**.}

CLOSE devices

CLOSE will close the specified *channel*. Any *window* associated with the channel will be deactivated.

CLOSE without a parameter will close all channels from #3 and above.

It will not report an error if a channel is not open.

syntax: *channel* := #*numeric_expression*

CLOSE [*channel*]

example: i. **CLOSE #4**
ii. **CLOSE #input_channel**
iii. **CLOSE** {closes all channels from #3 and above}

CLS windows

Will clear the *window* attached to the specified or default *channel* to the current **PAPER** colour, excluding the border if one has been specified. **CLS** will accept an optional parameter which specifies if only a part of the window must be cleared.

syntax: *part* := *numeric_expression*

CLS [*channel*,] [*part*]

where: *part* = 0 - whole screen (default if no parameter)
part = 1 - top excluding the cursor line
part = 2 - bottom excluding the cursor line
part = 3 - whole of the cursor line
part = 4 - right end of cursor line including the cursor position

example: i. **CLS** {the whole window}
ii. **CLS 3** {clear the cursor line}
iii. **CLS #2,2** {clear the bottom of the window on channel 2}

CODE SuperBASIC

CODE is a function which returns the internal code used to represent the specified character. If a string is specified then **CODE** will return the internal representation of the first character of the string.

CODE is the inverse of **CHR\$**.

syntax: **CODE** (*string_expression*)

example: i. **PRINT CODE("A")** {prints 65}
ii. **PRINT CODE("SuperBASIC")** {prints 83}

CONTINUE

RETRY error handling

CONTINUE allows a program which has been halted to be continued. **RETRY** allows a program statement which has reported an error to be re-executed.

As the **RETRY** and **CONTINUE** exit from an error clause without resetting the **WHEN ERROR**.

syntax: **CONTINUE**
RETRY

example: **CONTINUE**
RETRY

warning: A program can only continue if:

1. No new lines have been added to the program
2. No new variables have been added to the program
3. No lines have been changed

The value of variables may be set or changed.

COPY

COPY_N devices

COPY will copy a file from an input device to an output device until an end of file marker is detected. **COPY_N** will not copy the header (if it exists) associated with a file and will allow Disk files to be correctly copied to another type of device.

Headers are associated with directory-type devices and should be removed using **COPY_N** when copying to non-directory devices, e.g. **flp1** is a directory device; **ser1** is a non-directory device.

syntax: **COPY** *device* **TO** *device*
COPY_N *device* **TO** *device*

It must be possible to input from the source device and it must be possible to output to the destination device.

example: i. **COPY flp1_data_file TO con_** {copy to default window}
ii. **COPY neti_3 TO flp1_data** {copy data from network station to flp1_data.}
iii. **COPY_N flp1_test_data TO ser1_** {copy flp1_test_data to serial port 1 removing header information}

COS maths functions

COS will compute the cosine of the specified argument.

syntax: *angle:= numeric_expression* {range -10000..10000 in radians}

COS (*angle*)

example: i. **PRINT COS(theta)**
ii. **PRINT COS(3.141592654/2)**

COT maths functions

COT will compute the cotangent of the specified argument.

syntax: *angle:= numeric_expression* {range -30000..30000 in radians}

COT (*angle*)

example: i. **PRINT COT(3)**
 ii. **PRINT COT(3.141592654/2)**

CSIZE window

Sets a new character size for the *window* attached to the specified or default *channel*.

The standard size is 0,0 in 512/1024 mode and 2,0 in 256 mode.

In other screen resolutions the standard size 0,0.

Width defines the horizontal size of the character space. Height defines the vertical size of the character space. The character size is adjusted to fill the space available.

width	size	height	size
0	6 pixels	0	10 pixels
1	8 pixels	1	20 pixels
2	12 pixels		
3	16 pixels		

syntax: *width:= numeric_expression* {range 0..3}
 height:= numeric_expression {range 0..1}

CSIZE [*channel*,] *width*, *height*

example: i. **CSIZE 3,0**
 ii. **CSIZE 3,1**

CURSOR windows

CURSOR allows the screen cursor to be positioned anywhere in the window attached to the specified or default *channel*.

CURSOR uses the *pixel coordinate system* relative to the window origin and defines the position for the top left hand corner of the cursor. The size of the cursor is dependent on the character size in use.

If **CURSOR** is used with four parameters then the first pair is interpreted as graphics coordinates (using the graphics coordinate system) and the second pair as the position of the cursor (in the pixel coordinate system) relative to the first point.

This allows diagrams to be annotated relatively easily.

syntax: *x:= numeric_expression*
 y:= numeric_expression

CURSOR [*channel*,] *x*, *y* [,*x*, *y*]

example: i. **CURSOR 0,0**
 ii. **CURSOR 20,30**
 iii. **CURSOR 50,50,10,10**

DATA

READ

RESTORE SuperBASIC

READ, **DATA** and **RESTORE** allow embedded data, contained in a SuperBASIC program, to be assigned to variables at run time.

DATA is used to mark and define the data, **READ** accesses the data and assigns it to variables and **RESTORE** allows specific data to be selected.

DATA allows data to be defined within a program. The data can be read by a **READ** statement and the data assigned to variables. A **DATA** statement is ignored by SuperBASIC when it is encountered during normal processing.

syntax: **DATA** *[*expression*,]*

READ reads data contained in **DATA** statements and assigns it to a list of variables. Initially the data pointer is set to the first **DATA** statement in the program and is incremented after each **READ**. Re-running the program will not reset the data pointer and so in general a program should contain an explicit **RESTORE**.

An error is reported if a **READ** is attempted for which there is no **DATA**.

syntax: **READ** *[*identifier*,]*

RESTORE restores the data pointer, i.e. the position from which subsequent **READs** will read their data. If **RESTORE** is followed by a line number then the data pointer is set to that line. If no parameter is specified then the data pointer is reset to the start of the program.

syntax: **RESTORE** [*line_number*]

example: i. **100 REMark Data statement example**
 110 DIM weekdays\$(7,4)
 120 RESTORE
 130 FOR count= 1 TO 7 : READ weekdays\$(count)
 140 PRINT weekday\$
 150 DATA "MON","TUE","WED","THUR","FRI"
 160 DATA "SAT","SUN"

```
ii. 100 DIM month$(12,9)
    110 RESTORE
    120 REMark Data statement example
    130 FOR count=1 TO 12 : READ month$(count)
    140 PRINT month$
    150 DATA "January", "February", "March"
    160 DATA "April","May","June"
    170 DATA "July","August","September"
    180 DATA "October","November","December"
```

warning: An implicit **RESTORE** is not performed before running a program. This allows a single program to run with different sets of data. Either include a **RESTORE** in the program or perform an explicit **RESTORE** or **CLEAR** before running the program.

DATE\$

DATE clock

DATE\$ is a function which will return the date and time contained in the Q68's clock. The format of the string returned by **DATE\$** is:

"yyyy mmm dd hh:mm:ss"

where	yyyy	is the year 2022, 2023, etc
	mmm	is the month Jan, Feb etc
	dd	is the day 01 to 28, 29, 30, 31
	hh	is the hour 00 to 23
	mm	are the minutes 00 to 59
	ss	are the seconds 00 to 59

DATE will return the date as a floating point number which can be used to store dates and times in a compact form.

If **DATE\$** is used with a numeric parameter then the parameter will be interpreted as a date in floating point form and will be converted to a date string.

syntax:	DATE\$	{get the time from the clock}
	DATE\$ (numeric_expression)	{get time from supplied parameter}
	DATE [(yyyy,m,d,h,m,s)]	

example:	i. PRINT DATE\$	{output the date and time}
	ii. PRINT DATE\$(234567)	{convert 234567 to a date}
	iii. PRINT DATE	{output today's date as a floating point number}
	iv. PRINT DATE (2002,7,23,10,32,15)	{output 23 rd July 2002 at 10:32:15 as a floating point number}

DAY\$ clock

DAY\$ is a function which will return the current day of the week. If a parameter is specified then **DAY\$** will interpret the parameter as a date and will return the corresponding day of the week.

syntax:	DAY\$	{get day from clock}
	DAY\$ (numeric_expression)	{get day from supplied parameter}

example:	i. PRINT DAY\$	{output the day}
	ii. PRINT DAY\$(234567)	{output the day represented by 234567 (seconds)}

DEFine FuNction

END DEFine functions and procedures

DEFine FuNction defines a SuperBASIC function. The sequence of statements between the **DEFine** function and the **END DEFine** constitute the function. The function definition may also include a list of *formal parameters* which will supply data for the function. Both the formal and *actual parameters* must be enclosed in brackets. If the function requires no parameters then there is no need to specify an empty set of brackets.

Formal parameters take their type and characteristics from the corresponding *actual parameters*. The type of data returned by the function is indicated by the type appended to the function identifier. The type of the data returned in the **RETURN** statement must match.

An answer is returned from a function by appending an expression to a **RETURN** statement. The type of the returned data is the same as type of this expression.

A function is activated by including its name in a SuperBASIC expression.

Function calls in SuperBASIC can be recursive; that is, a function may call itself directly or indirectly via a sequence of other calls.

syntax: *formal_parameters* = (expression *[, expression]*)
actual_parameters := (expression *[, expression]*)

type := | \$
 | %
 |

```
DEF FuNction identifier type {formal_parameters}  
  [LOCAL identifier *[, identifier]*]  
  statements  
  RETURN expression  
END DEFine [identifier type]
```

RETURN can be at any position within the procedure body. **LOCAL** statements must precede the first executable statement in the function.

example: 10 DEFine FuNction mean(a, b, c)
 20 LOCAL answer
 30 LET answer = (a + b + c)/3
 40 RETURN answer
 50 END DEFine mean
 60 PRINT mean(1,2,3)

comment: To improve legibility of programs the name of the function can be appended to the **END DEFine** statement. However, the name will not be checked by SuperBASIC.

DEFine PROCedure

END DEFine functions and procedures

DEFine PROCedure defines a SuperBASIC procedure. The sequence of statements between the **DEFine PROCedure** statement and the **END DEFine** statement constitutes the procedure. The procedure definition may also include a list of *formal parameters* which will supply data for the procedure. The *formal parameters* must be enclosed in brackets for the procedure definition, but the brackets are not necessary when the procedure is called. If the procedure requires no parameters then there is no need to include an empty set of brackets in the procedure definition.

Formal parameters take their type and characteristics from the corresponding *actual parameters*.

Variables may be defined to be **LOCal** to a procedure. Local variables have no effect on similarly named variables outside the procedure. If required, local arrays should be dimensioned within the **LOCAl** statement.

The procedure is called by entering its name as the first item in a SuperBASIC statement together with a list of actual parameters. Procedure calls in SuperBASIC are recursive that is, a procedure may call itself directly or indirectly via a sequence of other calls.

It is possible to regard a procedure definition as a command definition in SuperBASIC; many of the system commands are themselves defined as procedures.

syntax: *formal_parameter* := (*expression* *[, *expression*]*)
actual_parameters := *expression* *[, *expression*]*

```
DEFine PROCedure identifier [formal_parameters]  
  [LOCAl identifier *[, identifier]*]  
  statements  
  [RETurn]  
END DEFine [identifier]
```

RETURN can appear at any position within the procedure body. If present the **LOCAl** statement must be before the first executable statement in the procedure. The **END DEFine** statement will act as an automatic return.

example: i. **100 DEFine PROCedure start_screen**
 110 WINDOW 100,100,10,10
 120 PAPER 7 : INK 0 : CLS
 130 BORDER 4,255
 140 PRINT "Hello Everybody"
 150 END DEFine
 160 start_screen

ii. **100 DEFine PROCedure slow_scroll(scroll_limit)**
 110 LOCAl count
 120 FOR count =1 TO scroll
 130 SCROLL 2
 140 END FOR count
 150 END DEFine slow_scroll
 160 slow_scroll 20

comment: To improve legibility of programs the name of the procedure can be appended to the **END DEFine** statement. However, the name will not be checked by SuperBASIC.

DEG maths functions

DEG is a function which will convert an angle expressed in radians to an angle expressed in degrees.

syntax: **DEG**(*numeric_expression*)

example: **PRINT DEG(PI/2)** {will print 90}

DELETE directory devices

DELETE will remove a file from the directory of the directory device specified.

syntax: **DELETE** *name* {delete one file}

example: i. **DELETE flp1_old_data**
ii. **DELETE win1_letter_file**

DIM arrays

Defines an array to SuperBASIC. *String*, *integer* and *floating point* arrays can be defined. String arrays handle fixed length strings and the final *index* is taken to be the string length.

Array indices run from 0 up to the maximum index specified in the **DIM** statement; thus **DIM** will generate an array with one more element in each dimension than is actually specified.

When an array is specified it is initialised to zero for a numeric array and zero length strings for a string array.

syntax: *index* := *numeric_expression*
array := *identifier*(*index* *[, *index*]*)

DIM *array* *[, *array*] *

example: i. **DIM string_array\$(10,10,50)**
ii. **DIM matrix(100,100)**

DIMN arrays

DIMN is a function which will return the maximum size of a specified dimension of a specified array. If a dimension is not specified then the first dimension is assumed. If the specified dimension does not exist or the identifier is not an array then zero is returned.

syntax: *array* := *identifier*
dimension := *numeric_expression* {1 for dimension 1, etc.}

DIMN(*array* [, *dimension*])

example: consider the array defined by: **DIM a(2,3,4)**
i. **PRINT DIMN(A,1)** {will print 2}
ii. **PRINT DIMN(A,Z)** {will print 3}
iii. **PRINT DIMN(A,3)** {will print 4}
iv. **PRINT DIMN(A)** {will print 2}
v. **PRINT DIMN(A,4)** {will print 0}

DIR directory devices

DIR will obtain and display in the *window* attached to the specified or default *channel*, the directory of the disk drive in the specified directory device.

syntax: **DIR** *device*

The device specification must be a valid directory device

The directory format output by **DIR** is as follows:

format:= disk format operating system QDOS or MSDOS
density:= formatting density SD, DD, or HD
free_sectors:= the number of free sectors
available_sectors:= the maximum number of sectors on this disk drive
file_name:= a SuperBASIC file name

screen format: *Volume name* *format* *density*
 free_sectors | *available_sectors* **sectors**
 file_name

 file_name

example: i. **DIR** flp1_
 ii. **DIR** "dev2_"
 iii. **DIR** "win" & hard_drive_number\$ & "_"

screen format: **BASIC QDOS HD**
 183 / 221 sectors
 demo_1
 demo_1_old
 demo_2

DISP_MODE graphics device 2

DISP_MODE sets the Q68's display mode. There are 3 different screen modes available in Minerva4Q68, presenting different screen sizes and colours.

The available Q68 screen modes are:

Mode 0 QL 8 colour mode
 The standard QL 256 x 256 pixels mode in 8 colours. In this mode you can also set mode 4, with the usual **MODE** keyword. This is then equivalent to setting **DISP_MODE 1**.

Mode 1 QL 4 colour mode
 The standard QL 512 x 256 pixels mode in 4 colours. In this mode you can also set mode 8, with the usual **MODE** keyword. This is then equivalent to setting **DISP_MODE 0**.

Mode 4 Large QL Mode 4
 1024 x 768 pixels in QL 4 colours mode (there is no mode 8 in this display mode).

syntax: *mode*:= *numeric_expression*

DISP_MODE *mode* {0 to 7}

example: **DISP_MODE 1** {sets a 512 x 256 QL mode 4 screen display}

note: The more colours that are displayed, and the higher the resolution. The slower the Q68 will become.

DIV operator

DIV is an operator which will perform an integer divide.

syntax: *numeric_expression DIV numeric_expression*

example: i. **PRINT 5 DIV 2** {will output 2}
 ii. **PRINT -5 DIV 2** {will output -3}

DLINE BASIC

DLINE will delete a single line or a range of lines from a SuperBASIC program.

syntax: *range:=* | *line_number TO line_number* (1)
 | *line_number TO* (2)
 | **TO** *line_number* (3)
 | *line_number* (4)

DLINE *range*[,range]**

where (1) will delete a range of lines
 (2) will delete from the specified line to the end
 (3) will delete from the start to the specified line
 (4) will delete the specified line

example: i. **DLINE 10 TO 70, 80, 200 TO 400**
 {will delete lines 10 to 70 inclusive, line 80 and lines 200 to 400 inclusive}

 ii. **DLINE**
 {will delete nothing}

DO program

DO will execute a series of SuperBASIC commands from file.

The commands should be 'direct': any lines with line numbers will be merged into the current SuperBASIC program. The file should not contain any of the following commands. **RUN, LRUN, MRUN, MERGE, SAVE, LOAD, STOP, NEW, CLEAR, CONTINUE, RETRY** or **GOTO**.

A **DO** file should be able to invoke SuperBASIC procedures without harmful effect.

syntax: **DO** *name*

comment: A **DO** file can contain in line clauses:

FOR i=1 to 20: PRINT 'This is a DO file'

If you try to **RUN** a BASIC program from a **DO** file, then the file will be left open.
Likewise, if you put direct commands in a file that is MERGED, then the file will be left open.

EDIT

The **EDIT** command enters the SuperBASIC line editor.

The **EDIT** command is closely related to the **AUTO** command, the only difference being in their defaults. **EDIT** defaults to a line increment of zero and thus will edit a single line unless a second parameter is specified to define a line increment.

If the specified line already exists then the line number is displayed and editing can be started. If the line does not exist then the line number is displayed and the line can be entered.

The cursor can be manipulated within the edit line using the standard Minerva4Q68QL keystrokes.

→	cursor right	
←	cursor left	
↑	cursor up	same as ENTER but automatically gives previous existing line to edit next
↓	cursor down	same as ENTER but automatically gives next existing line to edit next
CTRL →	delete character right	
CTRL ←	delete character left	
ALT ←/→	move to start/end of current line	
TAB	move along to 8th character from start of buffer	
SHIFT-TAB	move back in the same steps as above	
CTRL-ALT ←	deletes to start of current visible line	
CTRL-ALT →	deletes from current character to total end of line	
ESC ape	behaves like CTRL/SPACE (Break)	
SHIFT-ENTER	behaves pretty much like ENTER	
SHIFT-SPACE	behaves pretty much like SPACE	

When the line is correct pressing ENTER will enter the line into the program.

If an increment was specified then the next line in the sequence will be edited otherwise edit will terminate.

syntax: *line_number:= numeric_expression*
increment:= numeric_expression

EDIT *line_number* [,*increment*]

example: i. **EDIT 10** {edit line 10 only}
ii. **EDIT 20,10** {edit lines 20, 30 etc.}

EOF devices

The **EOF** function will determine if an end of file condition has been reached on a specified channel. If **EOF** is used without a channel specification then **EOF** will determine if the end of a program's embedded data statements has been reached.

If an end of file condition cannot be determined immediately, **EOF** will wait a certain amount of time before returning.

syntax: **EOF** [(*channel*)]

example: i. **IF EOF(#6) THEN STOP**
ii. **IF EOF THEN PRINT "Out of data"**

ERLIN

ERNUM error handling

ERLIN is a function that will return the line number where an error has occurred.

ERNUM is a function that will return the error number.

ERLIN and **ERNUM** should only be used as direct commands from the keyboard, or within a **WHEN ERROR** clause.

syntax: **ERLIN**
 ERNUM

example: i. **PRINT ERLIN**
 ii. **last_error = ERNUM**

EXEC

EXEC_W multitasking

EXEC and **EXEC_W** will load a sequence of programs and execute them in parallel.

EXEC will return to the command processor after all processes have started execution, **EXEC_W** will wait until all the processes have terminated before returning.

EXEC and **EXEC_W** have been extended to allow channel numbers and an optional string to be set up for the job. This is a strictly checked subset of the TK2 standard.

All the channels must already be open. They are passed to the job, along with the specified string, in the standard QDOS manner.

syntax: *program:= device_filename*
 channel_no:= numeric_expression
 string:= string_expression

 EXEC *program* *[,#*channel_no*]* [;*string*]
 EXEC_W *program* *[,#*channel_no*]* [;*string*]

example: i. **EXEC win1_communications**
 ii. **EXEC_W win1_printer_process,#4,#5,"Title"**

EXIT repetition

EXIT will continue processing after the **END** of the named **FOR** or **REPEAT** structure.

syntax: **EXIT** *identifier*

example: i. **100 REM start Looping**
 110 LET count = 0
 120 REPEAT Loop
 130 LET count = count +1
 140 PRINT count
 150 IF count = 20 THEN EXIT Loop
 160 END REPEAT loop
 {the loop will be exited when count becomes equal to 20}

 ii. **100 FOR n =1 TO 1000**
 110 REM program statements
 120 REM program statements
 130 IF RND >.5 THEN EXIT n
 140 END FOR n
 {the loop will be exited when a random number greater than 0.5
 is generated}

EXP maths functions

EXP will return the value of e raised to the power of the specified parameter.

syntax: **EXP** (*numeric_expression*) {range -500..500}

example: i. **PRINT EXP(3)**
 ii. **PRINT EXP(3.141592654)**

FILL graphics

FILL will turn *graphics fill* on or off. **FILL** will fill any non-re-entrant shape drawn with the *graphics* or *turtle graphics* procedures as the shape is being drawn. Re-entrant shapes must be split into smaller non-re-entrant shapes.

When you have finished filling, **FILL 0** should be called.

syntax: *switch:= numeric_expression* {range 0..1}

FILL [*channel*,] *switch*

example: i. **FILL 1:LINE 10,10 TO 50,50 TO 30,90 TO 10,10:FILL 0**
 {will draw a filled triangle}
 ii. **FILL 1:CIRCLE 50,50,20:FILL 0**
 {will draw a filled circle}

FILL\$ string arrays

FILL\$ is a function which will return a string of a specified length filled with a repetition of one or two characters.

syntax: **FILL\$** (*string_expression*, *numeric_expression*)

The string expression supplied to **FILL\$** must be either one or two characters long.

example: i. **PRINT FILL\$("a",5)** {will print aaaaa}
 ii. **PRINT FILL\$("oO",7)** {will print oOoOoOo}
 iii. **LET a\$ = a\$ & FILL\$(" ",10)**

FLASH windows

FLASH turns the flash state on and off. **FLASH** is only effective in the *low resolution* mode of **DISP_MODE 0**. **FLASH** will be effective in the window attached to the specified or default channel.

syntax: *switch:= numeric_expression* {range 0..1}

FLASH [*channel*,] *switch*

where: switch = 0 will turn the flash off
 switch = 1 will turn the flash on

example: **100 PRINT "A ";**
 110 FLASH 1
 120 PRINT "flashing ";
 130 FLASH 0
 140 PRINT "word"

warning: Writing over part of a flashing character can produce spurious results and should be avoided.

FOR

END FOR repetition

The **FOR** statement allows a group of SuperBASIC statements to be repeated a controlled number of times. The **FOR** statement can be used in both a long and a short form.

NEXT and **END FOR** can be used together within the same **FOR** loop to provide a *loop epilogue*, i.e. a group of SuperBASIC statements which will not be executed if a loop is exited via an **EXIT** statement but which will be executed if the **FOR** loop terminated normally.

define: *for_item*:= | *numeric_expression*
 | *numeric_exp TO numeric_exp*
 | *numeric_exp TO numeric_exp STEP numeric_exp*

 for_list:= *for_item* *[, *for_item*] *

SHORT: The **FOR** statement is followed on the same logical line by a sequence of SuperBASIC statements. The sequence of statements is then repeatedly executed under the control of the **FOR** statement. When the **FOR** statement is exhausted, processing continues on the next line. The **FOR** statement does not require its terminating **NEXT** or **END FOR**. Single line **FOR** loops must not be nested.

syntax: **FOR** *variable* = *for_list* : *statement* *[: *statement*]*

example: i. **FOR** *i* = 1, 2, 3, 4 **TO** 7 **STEP** 2 : **PRINT** *i*
 ii. **FOR** *element* = *first* **TO** *last* : **LET** *buffer* (*element*) = 0

LONG: The **FOR** statement is the last statement on the line. Subsequent lines contain a series of SuperBASIC statements terminated by an **END FOR** statement. The statements enclosed between the **FOR** statement and the **END FOR** are processed under the control of the **FOR** statement.

syntax: **FOR** *variable* = *for_list*
 statements
 END FOR *variable*

example: 100 **INPUT** "data please" ! *x*
 110 **LET** *factorial* = 1
 120 **FOR** *value* = *x* **TO** 1 **STEP** -1
 130 **LET** *factorial* = *factorial* * *value*
 140 **PRINT** *x* !!!! *factorial*
 150 **IF** *factorial* > IE20 **THEN**
 160 **PRINT** "Very Large number"
 170 **EXIT** *value*
 180 **END IF**
 190 **END FOR** *value*

FORMAT directory devices

FORMAT will format and make ready for use the directory device contained in the specified drive.

The specified device is the drive (physical or virtual) to be used for formatting and an identifier part used as the medium or volume name for the drive.

FORMAT will write the number of good sectors and the total number of sectors available on the directory device to the default or on the specified channel.

The examples below assume that extra device drivers have been installed. As microdrives and floppy disks are not natively supported by the Q68.

syntax: *device:= device_name | name*
 | *number*

FORMAT [*channel*,] *device*

example: i. **FORMAT flp2_data_disk**
 ii. **FORMAT mdv1_games**

FORMAT can be used to reinitialise a used directory device. However all data contained on that device will be lost.

warning: Use SMSQ/E to format the WIN drives in Minerva4Q68, as the system will likely hang if you try to format them.

FREE_FMEM memory management

The function **FREE_FMEM** will return the amount of free memory available in the 'Fast Memory' area.

syntax: **FREE_FMEM**

example: **PRINT FREE_FMEM**

note: In a freshly booted system **FREE_FMEM** should return around 10 Kilobytes.

GOSUB

For compatibility with other BASICs, SuperBASIC supports the **GOSUB** statement. **GOSUB** transfers processing to the specified line number; a **RETurn** statement will transfer processing back to the statement following **GOSUB**.

The line number specification can be an expression.

syntax: **GOSUB** *line_number*

example: i. **GOSUB 100**
 ii. **GOSUB 4*select_variable**

comment: The control structures available in SuperBASIC make the **GOSUB** statement redundant.

GOTO

For compatibility with other BASICs, SuperBASIC supports the **GOTO** statement. **GOTO** will unconditionally transfer processing to the statement number specified. The statement number specification can be an expression.

syntax: **GOTO** *line_number*

example: i. **GOTO program_start**
 ii. **GOTO 9999**

comment: The control structures available in SuperBASIC make the **GOTO** statement redundant.

IF THEN ELSE END IF

The **IF** statement allows conditions to be tested and the outcome of that test to control subsequent program flow.

The **IF** statement can be used in both a long and a short form:

SHORT: The **THEN** keyword is followed on the same logical line by a sequence of SuperBASIC keyword. This sequence of SuperBASIC statements may contain an **ELSE** keyword. If the expression in the **IF** statement is true (evaluates to be non-zero), then the statements between the **THEN** and the **ELSE** keywords are processed. If the condition is false (evaluates to be zero) then the statements between the **ELSE** and the end of the line are processed.

If the sequence of SuperBASIC statements does not contain an **ELSE** keyword and if the expression in the **IF** statement is true, then the statements between the **THEN** keyword and the end of the line are processed. If the expression is false then processing continues at the next line.

syntax: *statements := statement *[: statement]**

IF *expression* **THEN** *statements* [:**ELSE** *statements*]

example: i. **IF a=32 THEN PRINT "Limit" : ELSE PRINT "OK"**
 ii. **IF test >maximum THEN LET maximum = test**
 iii. **IF "1"+1=2 THEN PRINT "coercion OK"**

LONG 1: The **THEN** keyword is the last entry on the logical line. A sequence of SuperBASIC statements is written following the **IF** statements. The sequence is terminated by the **END IF** statement. The sequence of SuperBASIC statements is executed if the Expression contained in the **IF** statement evaluates to be non zero. The **ELSE** keyword and second sequence of SuperBASIC statements are optional.

LONG 2: The **THEN** keyword is the last entry on the logical line. A Sequence of SuperBASIC statements follows on subsequent lines, terminated by the **ELSE** keyword. If the expression contained in the **IF** statement evaluates to be non zero then this first sequence of SuperBASIC statements is processed. After the **ELSE** keyword a second sequence of SuperBASIC statements is entered, terminated by the **END IF** keyword. If the expression evaluated by the **IF** statement is zero then this second sequence of SuperBASIC statements is processed.

syntax: **IF** *expression* **THEN**
 statements
 [**ELSE**
 statements]
 END IF

example: **100 LET Limit =10**
 110 INPUT "Type in a number" ! number
 120 IF number > limit THEN
 130 PRINT "Range error"
 140 ELSE
 150 PRINT "Inside Limit"
 160 END IF

comment: In all three forms of the **IF** statement the **THEN** is optional. In the short form it must be replaced by a colon to distinguish the end of the **IF** and the start of the next statement. In the long form it can be removed completely.

nesting: **IF** statements may be nested as deeply as the user requires (subject to available memory). However, confusion may arise as to which **ELSE**, **END IF** etc, matches which **IF**. SuperBASIC will match nested **ELSE** statements etc, to the closest **IF** statement, for example:

```
100 IF a = b THEN
110  IF c = d THEN
120    PRINT "error"
130  ELSE
140    PRINT "no error"
150  END IF
160 ELSE
170  PRINT "not checked"
180 END IF
```

The **ELSE** at line 130 is matched to the second **IF**. The **ELSE** at line 160 is matched with the first **IF** (at line 100).

INK windows

This sets the current ink colour, i.e. the colour in which the output is written. **INK** will be effective for the window attached to the specified or default *channel*.

syntax: **INK** [*channel*,] *colour*

example: i. **INK 5**
 ii. **INK 6,2**
 iii. **INK #2,255**

INKEY\$

INKEY\$ is a function which returns a single character input from either the specified or default *channel*.

An optional timeout can be specified which can wait for a specified time before returning, can return immediately or can wait forever. If no parameter is specified then **INKEY\$** will return immediately.

syntax: **INKEY\$** [(*channel*)
 |(*channel*, *time*)
 |(*time*)]

where: *time* = 1..32767 {wait for specified number of frames.
 In the UK 50 Frames = 1 Second
 In the US 60 Frames = 1 Second}
 time = -1 {wait forever}
 time = 0 {return immediately}

example: i. **PRINT INKEY\$** {input from the default channel}
 ii. **PRINT INKEY\$(#4)** {input from channel 4}
 iii. **PRINT INKEY\$(50)** {wait for 50 frames then return anyway}
 iv. **PRINT INKEY\$(0)** {return immediatly (poll the keyboard)}
 v. **PRINT INKEY\$(#3,100)** {wait for 100 frames for an input from channel 3 then
 return anyway}

comment: If no character was available when **INKEY\$** times out, then a Null (**CHR\$(0)**) will be returned.

INPUT

INPUT allows data to be entered into a SuperBASIC program directly from the Q68's keyboard by the user. SuperBASIC halts the program until the specified amount of data has been input; the program will then continue. Each item of data must be terminated by the **ENTER** key.

INPUT will input data from either the specified or the default *channel*.

If input is required from a particular console channel the cursor for the window connected to that channel will appear and start to flash.

INPUT accepts enhanced movement keys as follows:

ALT ←/→	move to start/end of current line
TAB	move along to 8th character from start of buffer
SHIFT-TAB	move back in the same steps as above
CTRL-ALT ←	deletes to start of current visible line
CTRL-ALT →	deletes from current character to total end of line
ESCape	behaves like CTRL/SPACE (Break)
SHIFT-ENTER	behaves pretty much like ENTER
SHIFT-SPACE	behaves pretty much like SPACE

syntax: *separator* := |!
 |,
 |\n
 |;
 | **TO**

prompt := [*channel*,] *expression separator*

INPUT [*prompt*] [*channel*] *variable* *[,*variable*]*

example: i. **INPUT** ("Last guess "& guess & "New guess?") ! guess
 ii. **INPUT** "What is your guess?"; guess
 iii. **100 INPUT** "array size?" ! Limit
 110 DIM array(limit-1)
 120 FOR element = 0 to Limit-1
 130 INPUT ("data for element" & element) array(element)
 140 END FOR element
 150 PRINT array
 iv. **INPUT#3,x\$**

INSTR operator

INSTR is an operator which will determine if a given substring is contained within a specified string. If the string is found then the substring's position is returned. If the string is not found then **INSTR** returns zero.

Zero can be interpreted as false, i.e. the substring was not contained in the given string. A non zero value, the substrings position, can be interpreted as true, i.e. the substring was contained in the specified string.

syntax: *string_expression INSTR string_expression*

example: i. **PRINT** "a" **INSTR** "cat" {will print 2}
 ii. **PRINT** "CAT" **INSTR** "concatenate" {will print 4}
 iii. **PRINT** "x" **INSTR** "eggs" {will print 0}

INT maths functions

INT will return the integer part of the specified floating point expression.

syntax: **INT** (*numeric_expression*)

example: i. **PRINT INT(X)**
 ii. **PRINT INT(3.141592654/2)**

KBTABLE

KBTABLE will set the keyboard layout to be used.

syntax: *country_code:= numeric_expression*

KBTABLE *country_code*

example: i. **KBTABLE 44** {keyboard table set to English}
 ii. **KBTABLE 1** {keyboard table set to US}
 iii. **KBTABLE 49** {keyboard table set to German}

note: Currently only US, English, and German keyboards are supported.

KEYROW

KEYROW is a function which looks at the instantaneous state of a row of keys (the table below shows how the keys are mapped onto a matrix of 8 rows by 8 columns). **KEYROW** takes one parameter, which must be an integer in the range 0 to 7: this number selects which row is to be looked at. The value returned by **KEYROW** is an integer between 0 and 255 which gives a binary representation indicating which keys have been depressed in the selected row.

Since **KEYROW** is used as an alternative to the normal keyboard input mechanism using **INKEY\$** or **INPUT**, any character in the keyboard type-ahead buffer are cleared by **KEYROW**: thus key depressions which have been made before a call to **KEYROW** will not be read by a subsequent **INKEY\$** or **INPUT**.

Note that multiple key depressions can cause surprising results. In particular, if three keys at the corner of a rectangle in the matrix are depressed simultaneously, it will appear as if the key at the fourth corner has also been depressed. The three special keys **CTRL**, **SHIFT** and **ALT** are an exception to this rule, and do not interact with other keys in this way.

syntax: *row:= numeric_expression* {range 0..7}

KEYROW (*row*)

example: **100 REMark run this program and press a few keys**
110 REPEAT loop
120 CURSOR 0,0
130 FOR row = 0 to 7
140 PRINT row !!! KEYROW(row) ;" "
150 END FOR row
160 END REPEAT loop

KEYBOARD MATRIX

COLUMN								
ROW	1	2	4	8	16	32	64	128
7	SHIFT	CTRL	ALT	X	V	/	N	,
6	8	2	6	Q	E	0	T	U
5	9	W	I	TAB	R	-	Y	O
4	L	3	H	1	A	P	D	J
3	[CAPS	K	S	F	=	G	;
2]	Z	.	C	B	`	M	'
1	C/R	left	up	ESC	right		SPC	down
0	F4	F1	5	F2	F3	F5	4	7

LBYTES devices, directory devices

LBYTES will load a data file into memory at the specified start address.

syntax: *start_address:= numeric_expression*
device:= filename

LBYTES *device* ,*start_address*

example: i. **LBYTES flp1_screen, SCR_BASE**
{load a screen image}
ii. **LBYTES win1_program, start_address**
{load a program at a specified address}

LEN string arrays

LEN is a function which will return the length of the specified string *expression*.

syntax: **LEN**(*string_expression*)

example: i. **PRINT LEN("LEN will find the length of this string")**
 ii. **PRINT LEN(output_string\$)**

LET

LET starts a SuperBASIC assignment statement. The use of the **LET** keyword is optional. The assignment may be used for both string and numeric assignments. SuperBASIC will automatically convert unsuitable data types to a suitable form wherever possible.

syntax: [**LET**] *variable* = *expression*

example: i. **LET a = 1 + 2**
 ii. **LET a\$ = "12345"**
 iii. **LET a\$ = 6789**
 iv. **b\$ = test_data**

LINE LINE_R

LINE allows a straight line to be drawn between two points in the *window* attached to the default or specified channel. The ends of the line are specified using the *graphics coordinate system*.

Multiple lines can be drawn with a single **LINE** command.

The normal specification requires specifying the two end points for a line. These end points can be specified either in absolute coordinates (relative to the *graphics origin*) or in relative coordinates (relative to the *graphics cursor*). If the first point is omitted then a line is drawn from the graphics cursor to the specified point. If the second point is omitted then the graphics cursor is moved but no line is drawn.

LINE will always draw with absolute coordinates, i.e. relative to the *graphics origin*, while **LINE_R** will always draw relative to the graphics cursor.

syntax: *x:= numeric_expression*
 y:= numeric_expression
 point:= x,y

parameter_2:= | **TO point** (1)
 | **,point TO point** (2)

parameter_1:= | **TO point, angle** (1)
 | **TO point** (2)
 | *point* (3)

LINE [*channel*,] *parameter_1* *[, *parameter_2*]*
LINE_R [*channel*,] *parameter_1* *[,*parameter_2*]*

Where (1) will draw from the specified point to the next specified point
 (2) will draw from the last point plotted to the specified point
 (3) will move to the specified point, - no line will be drawn

example: i. **LINE 0,0 TO 0, 50 TO 50,0 TO 50,0 TO 0,0** {a square}
 ii. **LINE TO 0.75, 0.5** {a line}
 iii. **LINE 25,25** {move the graphics cursor}

LIST

LIST allows a SuperBASIC line or group of lines to be listed on a specific or default *channel*.

LIST is terminated by **CTRL - SPACE**.

syntax: *line:=* | *line_number TO line_number* (1)
 | *line_number TO* (2)
 | *TO line_number* (3)
 | *line_number* (4)
 | (5)

LIST [*channel*,] *line**[,*line*]*

where (1) will list from the specified line to the specified line
 (2) will list from the specified line to the end
 (3) will list from the start to the specified line
 (4) will list the specified line
 (5) will list the whole program

example: i. **LIST** {list all lines}
 ii. **LIST 10 TO 300** {list lines 10 to 300}
 iii. **LIST 12,20,50** {list lines 12,20 and 50 only}

If **LIST** output is directed to a channel opened as a printer channel then **LIST** will provide hard copy.

LN

LOG10 maths functions

LN will return the natural logarithm of the specified argument. **LOG10** will return the common logarithm. There is no upper limit on the parameter other than the maximum number the computer can store.

syntax: **LOG10** (*numeric_expression*) {range greater than zero}
 LN (*numeric_expression*) {range greater than zero}

example: i. **PRINT LOG10(20)**
 ii. **PRINT LN(3.141592654)**

LOAD devices, directory devices

LOAD will load a SuperBASIC program from any Q68 device. **LOAD** automatically performs a **NEW** before loading another program, and so any previously loaded program will be cleared by **LOAD**.

If a line input during a load has incorrect SuperBASIC syntax, the word **MISTAKE** is inserted between the line number and the body of the line. Upon execution, a line of this sort will generate an error

syntax: **LOAD** *device*

example: i. **LOAD "flp2_test_program"**
 ii. **LOAD ram1_guess**
 iv. **LOAD ser1_e**

LOCaI functions and procedures

LOCaI allows *identifiers* to be defined to be **LOCaI** to a *function or procedure*. Local identifiers only exist within the function or procedure in which they are defined, or in procedures and functions called from the function or procedure in which they are defined.

They are lost when the function or procedure terminates. Local identifiers are independent of similarly named identifiers outside the defining function or procedure. *Arrays* can be defined to be local by dimensioning them within the **LOCaI** statement.

The **LOCaI** statement must precede the first executable statement in the function or procedure in which it is used.

syntax: **LOCaI** *identifier* *[, *identifier*]*

example: i. **LOCaI** a,b,c(10,10)
 ii. **LOCaI** temp_data

comment: Defining variables to be **LOCaI** allows variable names to be used within functions and procedures without corrupting meaningful variables of the same name outside the function or procedure.

LRUN devices, directory devices

LRUN will load and run a SuperBASIC *program* from a specified *device*. **LRUN** will perform **NEW** before loading another program and so any previously stored SuperBASIC program will be cleared by **LRUN**.

If a line input during a loading has incorrect SuperBASIC syntax, the word **MISTAKE** is inserted between the line number and the body of the line. Upon execution, a line of this sort will generate an error.

syntax: **LRUN** *device*

example: i. **LRUN** flp2_TEST
 ii. **LRUN** ram1_game

MACHINE Minerva

MACHINE will return the machine type that Minerva is running on

syntax: **MACHINE**

example: **PRINT MACHINE**

comment: **MACHINE** will return 18 for the Q68.

MERGE devices, directory devices

MERGE will load a *file* from the specified *device* and interpret it as a SuperBASIC program. If the new file contains a line number which doesn't appear in the program already in the Q68 then the line will be added. If the new file contains a replacement line for one that already exists then the line will be replaced. All other old program lines are left undisturbed.

If a line input during a **MERGE** has incorrect SuperBASIC syntax, the word **MISTAKE** is inserted between the line number and the body of the line. Upon execution, a line of this sort will generate an error.

syntax: **MERGE** *device*

example: i. **MERGE** win1_overlay_program

MOD operators

MOD is an operator which gives the modulus, or remainder; when one integer is divided by another.

syntax: *numeric_expression* **MOD** *numeric_expression*

example: i. **PRINT 5 MOD 2** {will print 1}
 ii. **PRINT 5 MOD 3** {will print 2}

MODE windows

MODE sets the resolution of the screen and the number of solid colours which it can display. **MODE** will clear all *windows* currently on the screen, but will preserve their position and shape. Changing to low resolution mode (8 colour) will set the minimum character size to 2,0.

In Minerva4Q68 **MODE** allows the use of single and dual screens. If **F1** or **F2** is pressed at startup then single screen mode is activated where the **MODE** command takes one parameter. If **F3** or **F4** is pressed at startup then dual screen mode is activated. The original one parameter call is exactly as before. The new form being:

MODE screen mode, display type

The screen mode accepts the normal 4, 8, etc, but with a few additions (see below).

Display type is simple, 0 for monitor, 1 for 625-line TV, and 2 for 525-line TV, or (usually) to leave the display type alone. Very little software uses the display type record, as old versions of SuperBASIC smashed it!

Screen mode is *very* complex...

The two screens are known as Screen0 and Screen1. Each screen may be in 4-colour mode or 8-colour mode, or blank. Each job now has a *default* screen on which any new windows will be opened - it can change this default to have windows open in both screens. The screen which is not the default for the current job (the one calling the new **MODE**) is the *other* screen. Note that it is not necessarily the case that the default screen for a job is the same one the user is looking at (the *displayed* screen).

And the user can't necessarily see the displayed screen, it might be blank.

toggling: **MODE 64+n, -1**

toggles various attributes of the display, where:

n Toggles...

- ```

1 other screen from visible to blank
2 default screen from visible to blank
4 other screen from 4-colour to 8-colour
8 default screen from 4-colour to 8-colour
16 displayed screen from Screen0 to Screen1
32 default screen from Screen0 to Screen1

```

You can add together the various values for *n* to combine effects, so ***n=12=8+4*** will toggle the colour modes of both screens. Note: a change to the default screen takes place before any of the others. Also, none of these calls do the “forced re-draw” imposed by the one parameter version of the **MODE** call.

setting: **MODE** -16384+128+k\*n+c, -1

sets or resets display attributes. The values of n are as above.

## k Sets...

- 0 ... to the *from* column above  
1 ... to the *to* column above  
257 toggles, as above

Values of k can't be combined. The c portion controls which screen is force re-drawn:

**c** **Re-draws...**

- |        |                |
|--------|----------------|
| -16384 | other screen   |
| 32768  | default screen |

You can add the c values to re-draw both screens.

So...

|                      |                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>MODE 80, -1</b>   | toggles the <i>displayed</i> screen (80=64+16)                                                                                      |
| <b>MODE 96, -1</b>   | makes subsequent <b>OPENS</b> happen on what was the <i>other</i> screen (96=64+32)                                                 |
| <b>MODE 112 ,-1</b>  | does both the above simultaneously (112=64+32+16)                                                                                   |
| <b>MODE 16560,-1</b> | sets default to Screen1, displays it in 4-colour mode, and force re-draws all windows in it:<br>(16560= -16384+128+1*(32+16)+32768) |

syntax: *screen\_mode:= numeric\_expression*  
*display type:= numeric\_expression*

**MODE** *screen\_mode* [,*display\_type*]

where: 8 or 256 will select low resolution mode  
4 or 512 will select high resolution mode

example: i. **MODE 256**  
ii. **MODE 4**

## **MOVE** turtle graphics

**MOVE** will move the graphics turtle in the *window* attached to the default or specified *channel* a specified distance in the current direction. The direction can be specified using the **TURN** and **TURNTO** commands. The graphics scale factor is used in determining how far the turtle actually moves. Specifying a negative distance will move the turtle backwards.

The turtle is moved in the window attached to the specified or default *channel*.

syntax:     *distance:= numeric\_expression*

**MOVE** [*channel*,] *distance*

example: i. **MOVE #2,20**        {move the turtle in channel 2 20 units forwards}  
          ii. **MOVE -50**         {move the turtle in the default channel 50 units backwards}

## **MRUN** devices, directory devices

**MRUN** will interpret a *file* as a SuperBASIC program and merge it with the currently loaded program.

If used as *direct command* **MRUN** will run the new program from the start. If used as a program statement **MRUN** will continue processing on the line following **MRUN**.

If a line input during a merge has incorrect SuperBASIC syntax, the word **MISTAKE** is inserted between the line number and the body of the line. Upon execution, a line of this sort will generate an error.

syntax:     **MRUN** *device*

example: i. **MRUN flp1\_chain\_program**

## **NET** network

**NET** allows the network station number to be set. If a station number is not explicitly set then the Q68 assumes station number 1.

Although the Q68 does not natively support the QL network. With some additional hardware and software, the Q68 can be connected to a QL network.

syntax:     *station:= numeric\_expression*        {range 1 to 127}

**NET** *station*

example: i. **NET 63**  
          ii. **NET 1**

comment: Confusion may arise if more than one station on the network has the same station number.

## **NEW**

**NEW** will clear out the old *program*, *variables* and *channels* other than 0,1 and 2.

syntax:     **NEW**

example:   **NEW**

## **NEXT** repetition

**NEXT** is used to terminate, or create a loop *epilogue* in, **REPEAT** and **FOR** loops.

syntax:     **NEXT** *identifier*

The identifier must match that of the loop which the **NEXT** is to control

- example:
- i.   **10 REMark this loop must repeat forever**  
      **11 REPEAT infinite\_loop**  
      **12 PRINT "still looping"**  
      **13 NEXT infinite\_loop**
  
  - ii.  **10 REMark this loop will repeat 20 times**  
      **11 LET limit = 20**  
      **12 FOR index=1 TO Limit**  
      **13 PRINT index**  
      **14 NEXT index**
  
  - iii. **10 REMark this Loop will tell you when a 30 is found**  
      **11 REPEAT Loop**  
      **12 LET number = RND(1 TO 100)**  
      **13 IF number = 30 THEN NEXT Loop**  
      **14 PRINT number; " is 30"**  
      **15 EXIT LOOP**  
      **16 END REPEAT loop**

**in REPEAT:**   If **NEXT** is used inside a **REPEAT - END REPEAT** construct it will force processing to continue at the statement following the matching **REPEAT** statement.

**In FOR:**       The **NEXT** statement can be used to repeat the **FOR** loop with the control variable set at its next value. If the **FOR** loop is exhausted then processing will continue at the statement following the **NEXT**; otherwise processing will continue at the statement after the **FOR**.

## **ON...GOTO** **ON...GOSUB**

To provide compatibility with other BASICs, SuperBASIC supports the **ON GOTO** and **ON GOSUB** statements. These statements allow a variable to select from a list of possible *line numbers* a line to process in a **GOTO** or **GOSUB** statement. If too few line numbers are specified in the list then an error is generated.

syntax:     **ON** *variable* **GOTO** *expression* \*[, *expression*]\*  
              **ON** *variable* **GOSUB** *expression* \*[, *expression*]\*

- example:
- i.   **ON x GOTO 10, 20, 30, 40**
  - ii.  **ON select\_variable GOSUB 1000,2000,3000,4000**

comment: **SELEct** can be used to replace these two BASIC commands.

## OPEN, OPEN\_IN

### OPEN\_NEW devices, directory devices

**OPEN** allows the user to link a logical *channel* to a physical Q68 *device* for I/O purposes.

If the channel is to a directory device then the directory device file can be an existing file or a new file. In which case **OPEN\_IN** will open an already existing directory device file for input and **OPEN\_NEW** will create a new directory device file for output.

Minerva4Q68 will accept a third parameter on these commands. Should it be given, it becomes irrelevant which of the three commands was used, as it overrides that. The command will pass it as the "open type" to the driver (IO.OPEN D3.W).

For directory structured device drivers, this may be one of:

- |   |          |                                                               |
|---|----------|---------------------------------------------------------------|
| 0 | IO.OLD   | (same as doing <b>OPEN</b> in the first place)                |
| 1 | IO.SHARE | (same as doing <b>OPEN_IN</b> )                               |
| 2 | IO.NEW   | (same as doing <b>OPEN_NEW</b> )                              |
| 3 | IO.OVERW | (as TK2's <b>OPEN_OVER</b> , it overwrites any existing file) |
| 4 | IO.DIR   | (as TK2's <b>OPEN_DIR</b> , it opens the directory given)     |

The other thing that uses this "open type" parameter is the "pipe" device, where it requires the QDOS channel number of the source end of the pipe. It is possible, with some effort, to get pipes connected between MultiBasics using:

**OPEN#chan,"pipe\_128" : qdch=PEEK\_W(48\chan\*40+2)**

in one, getting *qdch* across to another, and doing:

**OPEN#chan,"pipe\_128", qdch**      there.

note:      The above facilities are lost when TK2 (or anything else) comes in and replaces the calls.

syntax:    *channel*:= # *numeric\_expression*  
          *open\_type*:= *numeric\_expression*      {0...4}

**OPEN** *channel, device [,open\_type]*  
**OPEN\_IN** *channel, device [,open\_type]*  
**OPEN\_NEW** *channel, device [,open\_type]*

example: i. **OPEN #5, f\_name\$**  
          ii. **OPEN\_IN #9,"flp1\_filename"**  
              {open file flp1\_file\_\_name}  
          iii. **OPEN\_NEW #7,win1\_datafile**  
              {open file win1\_datafile}  
          iv. **OPEN #6,con\_10x20a20x2032**  
              {Open channel 6 to the console device creating a window size 10x20 pixels at  
               position 20,20 with a 32 byte keyboard type ahead buffer.}  
          v. **OPEN #8,"win1\_",4**  
              {opens the directory of win1\_, equivalent of OPEN\_DIR}



## **OVER** windows

**OVER** selects the type of over printing required in the window attached to the specified or default channel. The selected type remains in effect until the next use of **OVER**.

syntax:     *switch:= numeric\_expression*             {range -1..1}

**OVER** [*channel*,] *switch*

where        *switch* = 0 - print ink on *strip*  
              *switch* = 1 - print in ink on transparent *strip*  
              *switch* = -1 - XORs the data on the screen

example: i. **OVER 1**             {set "overprinting"}  
          ii. **10 REMark Shadow Writing**  
              **11 PAPER 7 : INK 0 : OVER 1 : CLS**  
              **12 CSIZE 3,1**  
              **13 FOR i = 0 TO 10**  
              **14 CURSOR i,i**  
              **15 IF i=10 THEN INK 2**  
              **16 PRINT "Shadow"**  
              **17 END FOR i**

## **PAN** windows

**PAN** the entire current window the specified number of pixels to the left or the right. **PAPER** is scrolled in to fill the clear area.

An optional second parameter can be specified which will allow only part of the screen to be panned.

syntax:     *distance:= numeric\_expression*  
           *part:= numeric\_expression*

**PAN** [*channel*,] *distance* [, *part*]

where        *part* = 0 - whole screen (or no parameter)  
              *part* = 3 - whole of the cursor line  
              *part* = 4 - right end of cursor line including the cursor position

If the expression evaluates to a positive value then the contents of the screen will be shifted to the right.

example: i. **PAN #2,50**           {pan left 50 pixels}  
          ii. **PAN -100**          {pan right 100 pixels}  
          iii. **PAN 50.3**         {pan the whole of the current cursor line 50 pixels to the right}

**warning:** If *stipples* are being used or the screen is in low resolution mode then, to maintain the stipple pattern, the screen must be panned in multiples of two pixels.

## PAPER windows

**PAPER** sets a new paper colour (i.e. the colour which will be used by **CLS**, **PAN**, **SCROLL**, etc). The selected paper colour remains in effect until the next use of **PAPER**. **PAPER** will also set the **STRIP** colour

**PAPER** will change the paper colour in the *window* attached to the specified or default *channel*.

syntax: **PAPER** [*#channel*,] *colour*

example: i. **PAPER #3,7** {White paper on channel 3}  
ii. **PAPER 7,2** {White and red stipple}  
iii. **PAPER 255** {Black and white stipple}  
iv. **10 REMark Show colours and stipples**  
**11 FOR colour = 0 TO 7**  
**12 FOR contrast = 0 TO 7**  
**13 FOR stipple = 0 TO 3**  
**14 PAPER colour, contrast, stipple**  
**15 SCROLL 6**  
**16 END FOR stipple**  
**17 END FOR contrast**  
**18 END FOR colour**

## PAUSE

**PAUSE** will cause a program to wait a specified period of time. Delays are specified in units of 20ms in the UK only, otherwise 16.67ms. If no delay is specified, or the delay is -1, then the program will pause indefinitely. Keyboard input will terminate the **PAUSE** and restart program execution.

In Minerva4Q68 **PAUSE** now takes an optional channel number, allowing you to use the procedure from programs which do not have a channel #0.

syntax: *delay:= numeric\_expression*

**PAUSE** [*#channel*,] [*delay*]

example: i. **PAUSE 50** {wait 1 second}  
ii. **PAUSE 500** {wait 10 seconds}  
iii. **PAUSE #4,50**

## PEEK, PEEK\_W

### PEEK\_L SuperBASIC

**PEEK** is a function which returns the contents of the specified memory location. **PEEK** has four forms which will access a byte (8 bits), a word (16 bits), or a long word (32 bits).

**PEEK** may be referenced from the system variables if the first parameter of **PEEK** is preceded by an exclamation mark, then the address of the peek is in the system variables or referenced via the system variables. There are two variations: direct and indirect references.

For direct references, the exclamation mark is followed by another exclamation mark and an offset within the system variables.

For indirect references, the exclamation mark is followed by the offset of a pointer within the system variables, another exclamation mark and an offset from that pointer.

**PEEK** may also be referenced from the SuperBASIC variables if the first parameter of **PEEK** is preceded by a backslash, then the address of the peek is in the SuperBASIC variables or referenced via the SuperBASIC variables. There are two variations: direct and indirect references.

For direct references, the backslash is followed by another backslash and an offset within the SuperBASIC variables.

For indirect references, the backslash is followed by the offset of a pointer within the SuperBASIC variables, another backslash and an offset from that pointer.

syntax:     *address:= numeric\_expression*  
              | *!! numeric\_expression*  
              | *! numeric\_expression ! numeric\_expression*  
              | *\\ numeric\_expression*  
              | *\ numeric\_expression! \ numeric\_expression*

**PEEK**(address)     {byte access}  
**PEEK\_W**(address) {word access}  
**PEEK\_L**(address) {long word access}

example: i. **PRINT PEEK(12245)**     {byte contents of location 12245}  
          ii. **PRINT PEEK\_W(12)**     {word contents of locations 12 and 13}  
          iii. **PRINT PEEK\_L(1000)**   {long word contents of location 1000}  
          iv. **ramt = PEEK\_L (! !\$20)** {find the top of RAM \$20 bytes on from the base of  
                                          the system variables}  
          vi. **job1 = PEEK\_L (!\$68!4)** {find the base address of Job 1 (4 bytes on from base  
                                          of Job table)}  
          vii. **dal = PEEK\_W (\\\$94)**   {find the current data line number}  
          viii. **n6 = PEEK\_W (\\$18!2+6\*8)** {find the name pointer for the 6th name in the  
                                          name table}  
          ix. **nl6 = PEEK (\\$20\n6)**     {...and the length of the name}

comment: **PEEK\_W** will return negative numbers for values above 32768

## PENUP

### PENDOWN turtle graphics

Operates the 'pen' in turtle graphics. If the pen is up then nothing will be drawn. If the pen is down then lines will be drawn as the turtle moves across the screen.

The line will be drawn in the *window* attached to the specified or default *channel*. The line will be drawn in the current ink colour for the channel to which the output is directed.

syntax:    **PENUP** [*channel*]  
          **PENDOWN** [*channel*]

example: i. **PENUP**                {will raise the pen in the default channel}  
          ii. **PENDOWN #2**        {will lower the pen in the window attached to channel 2}

## PI maths function

**PI** is a function which returns the value of  $\pi$ .

syntax:        **PI**

example:       **PRINT PI**

## POINT

### POINT\_R graphics

**POINT** plots a point at the specified position in the *window* attached to the specified or default *channel*. The point is plotted using the *graphics coordinates system* relative to the *graphics origin*. If **POINT\_R** is used then all points are specified relative to the graphics cursor and are plotted relative to each other.

Multiple points can be plotted with a single call to **POINT**.

syntax:    *x:= numeric\_expression*  
          *y:= numeric\_expression*

*parameters:= x,y*

**POINT** [*channel*,] *parameters*\* [,*parameters*]\*

example: i. **POINT 256,128**                        {plot a point at (256,128)}  
          ii. **POINT x,x\*x**                       {plot a point at (x,x\*x)}  
          iii. **10 REPEAT example**  
              **20 INK RND(255)**  
              **30 POINT RND(100),RND(100)**  
              **40 END REPEAT example**

## POKE, POKE\_W POKE\_L SuperBASIC

**POKE** allows a memory location to be changed. For word and long word accesses the specified address must be an even address.

**POKE** has four forms which will access a byte (8 bits), a word (16 bits), or a long word (32 bits).

**POKE** may be referenced from the system variables if the first parameter of **POKE** is preceded by an exclamation mark, then the address of the poke is in the system variables or referenced via the system variables. There are two variations: direct and indirect references.

For direct references, the exclamation mark is followed by another exclamation mark and an offset within the system variables.

For indirect references, the exclamation mark is followed by the offset of a pointer within the system variables, another exclamation mark and an offset from that pointer.

**POKE** may also be referenced from the SuperBASIC variables if the first parameter of **POKE** is preceded by a backslash, then the address of the poke is in the SuperBASIC variables or referenced via the SuperBASIC variables. There are two variations: direct and indirect references.

For direct references, the backslash is followed by another backslash and an offset within the SuperBASIC variables.

For indirect references, the backslash is followed by the offset of a pointer within the SuperBASIC variables, another backslash and an offset from that pointer.

**POKE** allows more than one value to be **POKEd** at a time. For **POKE\_W** and **POKE\_L**, the address may be followed by a number of values to poke in succession. For **POKE** the address may be followed by a number of values to poke in succession and the list of values may include strings.

```
syntax: address:= numeric_expression
 | !! numeric_expression
 | ! numeric_expression ! numeric_expression
 | \\ numeric_expression
 | \ numeric_expression \ numeric_expression
data:= numeric_expression

POKE address, data [*,data *] {byte access}
POKE_W address, data [*,data *] {word access}
POKE_L address, data [*,data *] {long word access}
```

```
example: i. POKE 12235,0 {set byte at 12235 to 0}
 ii. POKE_L 131072,12345 {set long word at 131072 to 12345}
 iv. POKE_W ! ! $E,3 {set the auto-repeat speed to 3}
 v. POKE ! $B0!2, 'WIN' {change the first three characters of DATA_USE to WIN}
```

**warning:** Poking data into areas of memory used by Minerva can cause the system to crash and data to be lost. Poking into such areas is not recommended.

## **PRINT** devices, directory devices

Allows output to be sent to the specified or default channel. The normal use of **PRINT** is to send data to the Q68 screen.

syntax:     `separator:=` | **!**  
                              | **,**  
                              | **\**  
                              | **;**  
                              | **TO** *numeric\_expression*

*item*:= | *expression*  
          | *channel*  
          | *separator*

**PRINT** *\*[item]\**

Multiple print *separators* are allowed. At least one separator must separate *channel* specifications and *expressions*.

- example: i. **PRINT "Hello World"**  
          {will output Hello World on the default output device (channel 1)}
- ii. **PRINT #5,"data",1,2,3,4**  
      {will output the supplied data to channel 5 (which must have been previously opened)}
- iii. **PRINT TO 20; "This is in column 20"**

**!**       Normal action is to insert a space between items output on the screen. If the item will not fit on the current line a line feed will be generated. If the current print position is at the start of a line then a space will not be output. **!** affects the next item to be printed and therefore must be placed in front of the print item being printed. Also a **;** or a **!** must be placed at the end of a print list if the spacing is to be continued over a series of **PRINT** statements.

**,**       Normal separator, SuperBASIC will tabulate output every 8 columns.

**\**       Will force a new line.

**;**       Will leave the print position immediately after the last item to be printed. Output will be printed in one continuous stream.

**TO**      Will perform a tabbing operation. **TO** followed by a *numeric\_expression* will advance the print position to the column specified by the *numeric\_expression*. If the requested column is meaningless or the current print position is beyond the specified position then no action will be taken.

## PROCESSOR *Minerva*

**PROCESSOR** will return the Motorola MC680x0 family type.

syntax:    **PROCESSOR**

example:   **PRINT PROCESSOR**

comment:   **PROCESSOR** will return 0 for the Q68.

## **RAD** *maths functions*

**RAD** is a function which will convert an angle specified in degrees to an angle specified in radians.

syntax:    **RAD** (*numeric\_expression*)

example:   **PRINT RAD(180)**    {will print 3.141593}

## **RANDOMISE** *maths functions*

**RANDOMISE** allows the random number generator to be reseeded. If a parameter is specified the parameter is taken to be the new seed. If no parameter is specified then the generator is reseeded from internal information.

syntax:    **RANDOMISE** [*numeric\_expression*]

example:   i.   **RANDOMISE**                    {set seed to internal data}  
             ii.   **RANDOMISE 3.2235**        {set seed to 3.2235}

## **RECOL** *windows*

**RECOL** will recolour individual pixels in the window attached to the specified or default *channel* according to some pre-set pattern. Each parameter is assumed to specify, in order, the colour in which each pixel is recoloured, i.e. the first parameter specifies the colour with which to recolour all black pixels, the second parameter blue pixels, etc.

The colour specification must be a solid colour, i.e. it must be in the range 0 to 7.

syntax:    *c0*:= new colour for black  
             *c1*:= new colour for blue  
             *c2*:= new colour for red  
             *c3*:= new colour for magenta  
             *c4*:= new colour for green  
             *c5*:= new colour for cyan  
             *c6*:= new colour for yellow  
             *c7*:= new colour for white

**RECOL** [*channel* ,] *c0, c1, c2, c3, c4, c5, c6, c7*

example:   **RECOL 2,3,4,5,6,7,1,0**        {recolour blue to magenta, red to green, magenta to cyan etc.}

## REMark

**REMark** allows explanatory text to be inserted into a program. The remainder of the line is ignored by SuperBASIC.

syntax:     **REMark** *text*

example:   **REMark This is a comment in a program**

comment:   **REMark** is used to add comments to a program to aid clarity.

## RENUM

**RENUM** allows a group or a series of groups of SuperBASIC line numbers to be changed. If no parameters are specified then **RENUM** will renumber the entire program. The new listing will begin at line 100 and proceed in steps of 10.

If a start line is specified then line numbers prior to the start line will be unchanged. If an end line is specified then line numbers following the end line will be unchanged.

If a start number and stop are specified then the lines to be renumbered will be numbered from the start number and proceed in steps of the specified size.

If a **GOTO** or **GOSUB** statement contains an expression starting with a number then this number is treated as a line number and is renumbered.

syntax:     *start\_line*:=           *numeric\_expression*     {start renumber}  
              *end\_line*:=           *numeric\_expression*     {stop renumber}  
              *start\_number*:=       *numeric\_expression*     {base line number}  
              *step*:=                *numeric\_expression*     {step}

**RENUM** [*start\_line* [TO *end\_line*];] [*start\_number*] [,*step*]

example:   i.   **RENUM**                    {renumber whole program from 100 by 10}  
              ii. **RENUM 100 TO 200**        {renumber from 100 to 200 by 10}

**warning:** No attempt must be made to use **RENUM** to renumber program lines out of sequence, i.e. to move lines about the program. **RENUM** should not be used in a program.



## REPEAT

### END REPEAT repetition

**REPEAT** allows general repeat loops to be constructed. **REPEAT** should be used with **EXIT** for maximum effect. **REPEAT** can be used in both long and short forms:

**short:** The **REPEAT** keyword and loop identifier are followed on the same logical line by a colon and a sequence of SuperBASIC *statements*. **EXIT** will resume normal processing at the next logical line.

syntax: **REPEAT** *identifier* : *statements*

example: **REPEAT** wait : IF INKEY\$ = "" THEN EXIT wait

**long:** The **REPEAT** keyword and the loop identifier are the only statements on the logical line. Subsequent lines contain a series of SuperBASIC *statements* terminated by an

**END**

**REPEAT** statement.

The statements between the **REPEAT** and the **END REPEAT** are repeatedly processed by SuperBASIC.

syntax: **REPEAT** *identifier*  
*statements*  
**END REPEAT** *identifier*

example: **10** LET number = RND(1 TO 50)  
**11** REPEAT guess  
**12** INPUT "What is your guess?", guess  
**13** IF guess = number THEN  
**14** PRINT "You have guessed correctly"  
**15** EXIT guess  
**16** ELSE  
**17** PRINT "You have guessed incorrectly"  
**18** END IF  
**19** END REPEAT guess

comment: Normally at least one statement in a **REPEAT** loop will be an **EXIT** statement.

## REPORT error handling

**REPORT** will report the description of the last error encountered to the specified or default channel. An optional negative error number may be supplied. If so, the error message for this number will be reported.

syntax: *error\_number* := -*numeric\_expression*  
**REPORT** [*#channel*, ] [*error\_number*]

example: **REPORT -1** {display a **Not Complete** error message}

comment: The default channel is #0

## **RESPR** memory management

**RESPR** is a function which will reserve some of the resident procedure space. (For example to expand the SuperBASIC procedure list.)

If resident procedure space is not available, then space will be reserved in the common heap.

syntax:    *space:= numeric\_expression*  
          **RESPR** (*space*)

example:   **PRINT RESPR(1024)**            {will print the base address of a 1024 byte block}

## **RETurn** functions and procedures

**RETurn** is used to force a *function* or *procedure* to terminate and resume processing at the statement after the procedure or function call. When used within a function definition the **RETurn** statement is used to return the function's value.

syntax:    **RETurn** [*expression*]

example: i.   **100 PRINT ack (3,3)**  
              **110 DEFine FuNction ack(m,n)**  
              **120 IF m=0 THEN RETurn n+1**  
              **130 IF n=0 THEN RETurn ack (m-1,1)**  
              **140 RETurn a c k (m-1 ,a c k (m, n-1 ) )**  
              **150 END DEFine**

          ii.   **10 LET warning\_flag =1**  
              **11 LET error\_number = RND(0 TO 10)**  
              **12 warning error\_number**  
              **13 DEFine PROCedure warning(n)**  
              **14 IF warning\_flag THEN**  
              **15    PRINT "WARNING:";**  
              **16    SElect ON n**  
              **17      ON n =1**  
              **18        PRINT "Microdrive full"**  
              **19      ON n = 2**  
              **20        PRINT "Data space full"**  
              **21      ON n = REMAINDER**  
              **22        PRINT "Program error"**  
              **23    END SElect**  
              **24 ELSE**  
              **25    RETurn**  
              **26 END IF**  
              **27 END DEFine**

comment:    It is not compulsory to have a **RETurn** in a procedure. If processing reaches the **END DEFine** of a procedure then the procedure will return automatically.

**RETurn** by itself is used to return from a **GOSUB**.

## **RND** maths function

**RND** generates a random number. Up to two parameters may be specified for **RND**. If no parameters are specified then **RND** returns a pseudo random *floating point* number in the exclusive range 0 to 1. If a single parameter is specified then **RND** returns an integer in the inclusive range 0 to the specified parameter. If two parameters are specified then **RND** returns an integer in the inclusive range specified by the two parameters.

syntax: **RND**( [*numeric\_expression*] [**TO** *numeric\_expression*])

example: i. **PRINT RND** {floating point number between 0 and 1}  
ii. **PRINT RND(10 TO 20)** {integer between 10 and 20}  
iii. **PRINT RND(1 TO 6)** {integer between 1 and 6}  
iv. **PRINT RND(10)** {integer between 0 and 10}

## **RUN** program

**RUN** allows an SuperBASIC program to be started. If a line number is specified in the **RUN** command then the program will be started at that point, otherwise the program will start at the lowest line number.

syntax: **RUN** [*numeric\_expression*]

example: i. **RUN** {run from start}  
ii. **RUN 10** {run from line 10}  
iii. **RUN 2\*20** {run from line 40}

comment: Although **RUN** can be used within a program its normal use is to start program execution by typing it in as a direct command.

## **SAVE** devices, directory devices

**SAVE** will save a SuperBASIC program onto any Q68 device.

when saving to a non directory device, The device name may be replaced with a channel number.

syntax: *line*:= | *numeric\_expression* **TO** *numeric\_expression* (1)  
| *numeric\_expression* **TO** (2)  
| **TO** *numeric\_expression* (3)  
| *numeric\_expression* (4)  
| (5)

**SAVE** *device* \*[,*line*]\*

where (1) will save from the specified line to the specified line  
(2) will save from the specified line to the end  
(3) will save from the start to the specified line  
(4) will save the specified line  
(5) will save the whole program

example: i. **SAVE win1\_program,20 TO 70**  
{save lines 20 to 70 on win1}  
ii. **SAVE ser1**  
{save the entire program on serial channel }  
iii. **OPEN\_NEW#4,pipe\_alpha\_1000**  
**SAVE#4**  
{save the entire program to a channel }

## **SBYTES** devices, directory devices

**SBYTES** allows areas of the Q68 memory to be saved on a Q68 device.

If a channel number of an open channel is supplied in place of a filename, then **SBYTES** will attempt to save the file to the channel.

syntax:    *start\_address:= numeric\_expression*  
          *length:= numeric\_expression*  
          *device:= filename | channel*

**SBYTES** *device, start\_address, length*

example: i. **SBYTES flp1\_screendata, SCR\_BASE, SCR\_LLEN \* SCR\_YLIM**  
          {save screen image on flp1\_test\_program}  
          iii. **SBYTES neto\_3,32768,32678**  
              {save memory 32768 length 32768 bytes on the network}  
          iv. **SBYTES ser1,0,32768**  
              {save memory 0 length 32768 bytes on serial channel 1}

## **SCALE** graphics

**SCALE** allows the scale factor used by the graphics procedures to be altered. A scale of 'x' implies that a vertical line of length 'x' will fill the vertical axis of the *window* in which the figure is drawn. A scale of 100 is the default. **SCALE** also allows the origin of the coordinate system to be specified. This effectively allows the window being used for the graphics to be moved around a much larger graphics space.

syntax:    *x:=numeric\_expression*  
          *y:=numeric\_expression*  
  
          *origin:= x,y*  
          *scale:= numeric\_expression*

**SCALE** [*channel*,] *scale, origin*

example: i. **SCALE 0.5,0.1,0.1**            {set scale to 0.5 with the origin at 0.1,0.1}  
          ii. **SCALE 10,0,0**                {set scale to 10 with the origin at 0,0}  
          iii. **SCALE 100,50,50**            {set scale to 100 with the origin at 50,50}

## SCROLL windows

**SCROLL** scrolls the window attached to the specified or default *channel* up or down by the given number of pixels. *Paper* is scrolled in at the top or the bottom to fill the clear space.

An optional third parameter can be specified to obtain a part screen scroll.

syntax:    *part*:=        *numeric\_expression*  
          *distance*:=    *numeric\_expression*

where        *part* = 0 - whole screen (default is no parameter)  
              *part* = 1 - top excluding the cursor line  
              *part* = 2 - bottom excluding the cursor line

**SCROLL** [*channel*,] *distance* [, *part*]

If the distance is positive then the contents of the screen will be shifted down.

example: i. **SCROLL 10**        {scroll down 10 pixels}  
          ii. **SCROLL -70**       {scroll up 70 pixels}  
          iii. **SCROLL -10,2**    {scroll the lower part of the window up 10 pixels}

## SCR\_BASE

### SCR\_LLEN windows

**SCR\_BASE** will return the base address of the screen attached to the specified or default channel.

**SCR\_LLEN** will return the line length in bytes of the screen attached to the specified or default channel.

syntax:    **SCR\_BASE** [(#*channel*)]  
          **SCR\_LLEN** [(#*channel*)]

example: i. **PRINT SCR\_BASE**  
          ii. **PRINT SCR\_LLEN (#1)**

comment: In current versions, the values returned are the same for all screen channels.

## SCR\_XLIM

### SCR\_YLIM windows

**SCR\_XLIM** will return the maximum number of pixels across the screen (+1), available for the screen attached to the specified, or default channel.

**SCR\_YLIM** will return the maximum number of pixels down the screen (+1), available for the screen attached to the specified, or default channel.

syntax:    **SCR\_XLIM** [(#*channel*)]  
          **SCR\_YLIM** [(#*channel*)]

example: i. **PRINT SCR\_XLIM**  
          ii. **PRINT SCR\_YLIM( #1)**

comment: The values returned are not the same as the current window size, but they defines the maximum size that a window can be. **SCR\_XLIM** and **SCR\_YLIM** should only be called for a primary window, usually #0 the default channel, for an SuperBASIC job.

## **SDATE** clock

The **SDATE** command allows the Q68's clock to be reset.

Note that **SDATE** does not set the battery backed clock in the Q68. To set the battery backed real time clock. First set the date and time with **SDATE**, then execute the clock utility program named 'Q68SETRTC'

syntax:    *year:=            numeric\_expression*  
          *month:=        numeric\_expression*  
          *day:=            numeric\_expression*  
          *hours:=        numeric\_expression*  
          *minutes:=     numeric\_expression*  
          *seconds:=     numeric\_expression*

**SDATE** *year, month, day, hours, minutes, seconds*

example: i. **SDATE 1984,4,2,0,0,0**  
          ii. **SDATE 1984,1,12,9,30,0**  
          iii. **SDATE 1984,3,21,0,0,0**

## **SElect**

### **END SElect** conditions

**SElect** allows various courses of action to be taken depending on the value of a variable.

define:    *select\_variable:= | numeric\_variable | integer\_variable | string\_variable*  
          *select\_item:=        | expression*  
                              *| expression TO expression*  
          *select\_list:=        | select\_item \*[, select\_item]\**

**long:**        Allows multiple actions to be selected depending on the value of a *select\_variable*.  
          The select variable is the last item on the logical line. A series of SuperBASIC

*statements*

follows, which is terminated by the next **ON** statement or by the **END SElect** statement. If the select item is an expression then a check is made within approximately 1 part in  $10^{-7}$ , otherwise for expression **TO** expression the range is tested exactly and is inclusive. The **ON REMAINDER** statement allows a, "catch-all" which will respond if no other select conditions are satisfied.

syntax:        **SElect ON** *select\_variable*  
                  \*[[**ON** *select\_variable*] = *select\_list*  
                              *statements*] \*  
                  [**ON** *selectvariable*] = **REMAINDER**  
                              *statements*  
          **END SElect**

example:    100 LET error\_number = RND(1 TO 10)  
              110 SElect ON error\_number  
              120    ON error\_number =1  
              130        PRINT "Divide by zero"  
              140        LET error\_number = 0  
              150    ON error\_number = 2  
              160        PRINT "File not found"  
              170        LET error\_number = 0  
              180    ON error\_number = 3 TO 5  
              190        PRINT "Microdrive file not found"  
              200        LET error\_number = 0  
              210    ON error\_number = REMAINDER  
              220        PRINT "Unknown error"  
              230 END SElect

If the select variable is used in the body of the **SElect** statement then it must match the select variable given in the select header.

**Short:** The short form of the **SElect** statement allows simple single line selections to be made. A sequence of SuperBASIC statements follows on the same logical line as the **SElect** statement. If the condition defined in the select statement is satisfied then the sequence of SuperBASIC statements are processed.

syntax:       **SElect ON** *select\_variable* = *select\_list* : *statement* \*[: *statement*] \*

example:    i.   **SElect ON** test data =1 TO 10 :  
              **PRINT** "Answer within range"  
          ii.   **SElect ON** answer = 0.00001 TO 0.00005 :  
              **PRINT** "Accuracy OK"  
          iii.   **SElect ON** a =1 TO 10 : **PRINT** a ! "in range"

comment: The short form of the **SElect** statement allows ranges to be tested more easily than with an **IF** statement. Compare example ii. above with the corresponding **IF** statement.

## **SER\_BUFF** devices

**SER\_BUFF** specifies the size of the transmit buffer and (optionally) the receive buffer.

This allows you to configure the transmit and receive buffers, like SMSQ/E's **SER\_BUFF** command. Its behaviour is somewhat different in that the default sizes are 16384 bytes for the receive buffer and 1024 bytes for the transmit buffer, and that dynamic-sized transmit buffers are not possible (they will usually fail when flow control has to be asserted, e.g. with Zmodem transfers).

syntax:    *input\_buff*:= *numeric\_expression*  
          *output\_buff*:= *numeric\_expression*

**SER\_BUFF** *output\_buff*, *input\_buff*

example:   i.   **SER\_BUFF** 200                {200 byte output buffer on SER1}  
          ii.   **SER\_BUFF** 50,80            {50 byte output buffer, 80 byte input buffer on SER1}

## **SER\_CLEAR** devices

**SER\_CLEAR** will clear both input and output buffers of the serial port.

syntax:    **SER\_CLEAR**

example:   i.   **SER\_CLEAR**                        {clear input and output of SER1}

## SER\_FLOW devices

**SER\_FLOW** specifies the flow control for the port by a one-letter parameter *I*, *H*, or *X*, where:

- I: stands for no flow-control (i.e. send at full speed, ignore XON/XOFFs sent by the remote)
- X: use XON/XOFF flow control when sending and receiving. This is not transparent to the data; if your data contains either of these characters (11H and 13H) this will disrupt transfers. Only use it when sending plain text data.
- H: Use XON/XOFF flow control but escape these characters in the data stream (using DLE, so XON will be sent as 10H followed by 'Q' and XOFF as 10H followed by 'S'). Using this technique, full 8-bit data transfers will be possible whilst still providing flow control (equivalent to using the *H* option with DTR/CTS handshake on the original QL's SER ports). This is an implementation-specific extension to the protocol and will only work between two Q68s using the SER driver, or a QIMSI serial port using SER4.

These options may also be specified when opening a channel to the port; e.g. OPEN#3,ser1x will open the serial port and use XON/XOFF flow control.

syntax: *hand\_shake*:= H | X | I {Hardware, XON/XOFF, or Ignore}

**SER\_FLOW** *hand\_shake*

example: i. **SER\_FLOW X** {visible XON/XOFF handshaking}  
ii. **SER\_FLOW H** {blind XON/XOFF handshaking}

note: Hardware flow control is not available in the Q68.

## SER\_ROOM devices

**SER\_ROOM** specifies the receive buffer's threshold for asserting flow control. When the receive buffer has been filled up to the point where there are less than *threshold* bytes free, an XOFF is sent to the remote.

This threshold is also affected by **SER\_BUFF**, which sets it to 25 percent of the receive buffer size. When the receive buffer has been emptied to 75 percent of its size, an XON character will be sent to the remote.

syntax: *threshold*:= *numeric\_expression*

**SER\_ROOM** *threshold*

example: **SER\_ROOM 100** {send an XOFF when the receive buffer has less than 100 bytes free}

comment: **SER\_ROOM** will not usually be required as **SER\_BUFF** also sets **SER\_ROOM** to one quarter of the buffer size. You will not succeed in setting **SER\_ROOM** to greater than **SER\_BUFF**, however, as **SER\_ROOM** will always ensure that the buffer is at least twice the size of the spare room.



## **SER\_USE** devices

**SER\_USE** specifies a name for the serial port. The *name* should be a four-character device name.

It main use is to substitute an existing SER port such as ser2 or ser3, so existing software using these names will be able use the port. In addition, the STX and SRX transmit-only and receive-only devices will have their last character modified as well, so entering **SER\_USE SER2** will change these names to STX2 and SRX2 respectively.

syntax:    **SER\_USE** [ *name* ]

example: i.    **SER\_USE SER2**                    {from now on, when you open ser2, you open a ser1 port}  
             ii.   **SER\_USE SER1**                   {sets you back to normal}  
             iii.   **SER\_USE**                        { ..as does this}

## **SEXEC** job creation

Will save an area of memory in a form which is suitable for loading and executing with the **EXEC** command.

The data saved should constitute a machine code program.

The optional parameters default to zero, except *type* which defaults to 1.

The *length* parameter is the length of area to be saved from the *start\_address*.

The *data\_space* parameter is the size of the data area that is required by the program.

The *extra* parameter is what the TK2 **FXTRA** function returns; but unfortunately TK2 replaces **SBYTES** and **SEXEC** to get default directories working.

The *type* parameter actually allows you to set both the file type (bottom byte) and the file access key (next byte up), though this latter byte has never been used by anything except Toolkit 3.

syntax:    *device*:=            *filename*  
            *start\_address*:= *numeric\_expression* {start of area}  
            *length*:=           *numeric\_expression*  
            *data\_space*:=      *numeric\_expression*  
            *extra*:=            *numeric\_expression*  
            *type*:=             *numeric\_expression*  
  
            **SEXEC** *device*, *start\_address* [, *length* [, *data\_space* [, *extra* [, *type*]]]]

example:   **SEXEC flp1\_program,262144,3000,500**

The QDOS, and Minerva system documentation should be read before attempting to use this command.

## **SIN** maths function

**SIN** will compute the sine of the specified parameter.

syntax:    *angle*:= *numeric\_expression* {range -10000..10000 in radians}

**SIN**(*angle*)

example: i.    **PRINT SIN(3)**  
             ii.   **PRINT SIN(3.141592654/2)**

## SLUG

**SLUG** will delay all subsequent reads of the keyboard by a supplied amount in thousandths of a second (milliseconds). This is to allow some programs which are too fast in the Q68 to be slowed down.

syntax:    **SLUG** *numeric\_expression*                      {0 to 255}

example:   **SLUG 15**                                              {add a 15 thousandths of a second delay}

## SQRT maths function

The **SQRT** function will compute the square root of the specified argument. The argument must be greater than or equal to zero.

syntax:    **SQRT** (*numeric\_expression*)                      {range >= 0}

example:   i.   **PRINT SQRT(3)**                                  {print square root of 3}  
             ii.   **LET C = SQRT(a^2+b^2)** {let c become equal to the square root of  $a^2 + b^2$ }

## STOP SuperBASIC

**STOP** will terminate execution of a program and will return SuperBASIC to the *command interpreter*.

syntax:    **STOP**

example:   i.   **STOP**  
             ii.   **IF n = 100 THEN STOP**

You may **CONTINUE** after **STOP**.

comment:   The last executable line of a program will act as an automatic stop.

## STRIP windows

**STRIP** will set the current strip colour in the window attached to the specified or default *channel*. The strip colour is the background colour which is used when **OVER 1** is selected. Setting **PAPER** will automatically set the strip colour to the new **PAPER** colour.

syntax:    *colour:= numeric\_expression*                      {range 0 ... 255}

**STRIP** [*channel*,] *colour*

example:   i.   **STRIP 7**                                              {set a white strip}  
             ii.   **STRIP 0,4,2**                                          {set a black and green stipple strip}

comment:   The effect of **STRIP** is rather like using a highlighting pen.

## TAN maths functions

The **TAN** function will compute the tangent of the specified argument. The argument must be in the range -30000 to 30000 and must be specified in radians.

syntax:    **TAN** (*numeric\_expression*)                      {range -30000..30000}

example:   i.   **PRINT TAN(3)**                                          {print tan 3}  
             ii.   **PRINT TAN(3.141592654/2)**                      {print tan  $\pi/2$ }

## TRA

**TRA** allows you to set up a translation table for a printer or system messages. The command accepts 1 or two parameters:

The *serial\_table* parameter controls the serial i/o translation tables:

|         |                                            |
|---------|--------------------------------------------|
| <= -2   | turn on translation                        |
| = -1    | no change                                  |
| = 0     | turn off translation                       |
| = 1     | set default i/o tables and turn it on      |
| = other | set user defined i/o tables and turn it on |

The *message\_table* parameter controls the message table:

|         |                             |
|---------|-----------------------------|
| <= 0    | no change                   |
| = 1     | set default system messages |
| = other | set user defined messages   |

User defined translation tables must be specified by an even address, at which the first word is \$4AFB. The translation tables then contain two words, giving the offset to the input and output translation tables. These may be given as zero to turn each off, or should point at a 256 byte direct translate list. Only 128 bytes are needed if only 7-bit data is being used.

The message table continues with a count of the number of messages in the table and then their offsets. Each message must be on an even address, prefixed by its character count word.

syntax:    *serial\_table:= numeric\_expression*  
          *message\_table:= numeric\_expression*

**TRA** *serial\_table* [, *message\_table* ]  
**TRA** , *message\_table*

|          |                               |                                              |
|----------|-------------------------------|----------------------------------------------|
| example: | i. <b>TRA 0,0</b>             | {translates off, table unchanged}            |
|          | ii. <b>TRA 1</b>              | {set to use default serial table}            |
|          | iii. <b>TRA 234000</b>        | {set user defined serial table}              |
|          | iv. <b>TRA ,235000</b>        | {set user defined message table}             |
|          | v. <b>TRA 234000 , 235000</b> | {set user defined serial and message tables} |

## TURN

### **TURNTO** turtle graphics

**TURN** allows the heading of the 'turtle' to be turned through a specified angle while **TURNTO** allows the turtle to be turned to a specific heading.

The turtle is turned in the *window* attached to the specified or default *channel*.

The angle is specified in degrees. A positive number of degrees will turn the turtle anti-clockwise and a negative number will turn it clockwise.

Initially the turtle is pointing at 0°, that is to the right hand side of the window.

syntax:    *angle:= numeric\_expression* {angle in degrees}

**TURN** [*channel*,] *angle*  
**TURNTO** [*channel*,] *angle*

|          |                     |                       |
|----------|---------------------|-----------------------|
| example: | i. <b>TURN 90</b>   | {turn through 90° }   |
|          | ii. <b>TURNTO 0</b> | {turn to heading 0° } |

## **UNDER windows**

Turns underline either on or off for subsequent output lines. Underlining is in the current **INK** colour in the *window* attached to the specified or default *channel*.

syntax:     *switch*:= *numeric\_expression*             {range 0..1}

**UNDER** [*channel*,] *switch*

example: i. **UNDER 1**             {underlining on}  
          ii. **UNDER 0**             {underlining off}

## **VER\$ SuperBASIC**

**VER\$** will return system version information.

**VER\$** without parameters, or with a parameter of 0 will return the SuperBASIC version.  
A parameter of 1 will return the Minerva version number, a parameter of -1 will return the job ID,  
and a parameter of -2 will return the address of the system variables.

syntax:     **VER\$** [ ( *numeric\_expression* ) ]

example: i. **PRINT ver\$**             {prints JSL1 (SuperBASIC version ID)}  
          ii. **PRINT ver\$(0)**         {also prints JSL1 (SuperBASIC version ID)}  
          iii. **PRINT ver\$(1)**         {prints 1.98 (or later Minerva version number)}  
          iv. **PRINT ver\$(-1)**         {print the Job ID (0 for initial SuperBASIC)}  
          v. **PRINT ver\$(-2)**         {prints the address of the system variables}

note:       The Minerva version number returned is not the Minerva4Q68 version number.

## WHEN ERROR

### END WHEN error handling

Error handling is invoked by a **WHEN ERROR** clause. Unlike procedure and function definitions, these clauses are static. The error handling within a **WHEN ERROR** clause is set up when the clause is executed, but is only actioned **WHEN** an **ERROR** occurs. This means that a program may have more than one **WHEN ERROR** clause. As each one is executed, the error processing within that clause replaces the previously defined error processing.

The clause is opened with a **WHEN ERROR** statement, and closed with an **END WHEN** statement. Within the clause there may be any normal type of statement. (Although it might be better to avoid calling SuperBASIC functions or procedures!) A **WHEN ERROR** clause is exited by a **STOP**, **CONTINUE**, **RETRY**, **RUN**, **LOAD** or **LRUN** command. Furthermore **RUN**, **NEW**, **CLEAR**, **LOAD**, **LRUN**, **MERGE** and **MRUN** will reset the error processing.

syntax: **WHEN ERROR**

There are some additional facilities intended for use within **WHEN ERROR** clauses.

#### ERROR functions

These functions correspond to each of the system error codes

|               |                      |               |                        |
|---------------|----------------------|---------------|------------------------|
| <b>ERR_NC</b> | Not Complete,        | <b>ERR_NJ</b> | Invalid Job,           |
| <b>ERR_OM</b> | Out of Memory,       | <b>ERR_OR</b> | Out of Range,          |
| <b>ERR_BO</b> | Buffer Full,         | <b>ERR_NO</b> | Channel not Open,      |
| <b>ERR_NF</b> | Not Found,           | <b>ERR_EX</b> | Already Exists,        |
| <b>ERR_IU</b> | In Use,              | <b>ERR_EF</b> | End of File,           |
| <b>ERR_DF</b> | Drive Full,          | <b>ERR_BN</b> | Bad Name,              |
| <b>ERR_TE</b> | Transmit Error,      | <b>ERR_FF</b> | Format Failed,         |
| <b>ERR_BP</b> | Bad Parameter,       | <b>ERR_FE</b> | Bad or Changed Medium, |
| <b>ERR_XP</b> | Error in Expression, | <b>ERR_OV</b> | Overflow,              |
| <b>ERR_NI</b> | Not Implemented,     | <b>ERR_RO</b> | Read Only,             |
| <b>ERR_BL</b> | Bad line             |               |                        |

and return the value TRUE if the error, which caused the **WHEN ERROR** clause to be invoked, is of that type.

example: i. **10 WHEN ERROR**  
**20 IF ERR\_BP THEN PRINT "Bad Parameter error"**  
**30 IF ERR\_OV THEN PRINT "An Overflow has occurred"**  
**40 IF ERR\_NO THEN PRINT "Channel is not open"**  
**50 END WHEN**

ii. **10 WHEN ERROR :GO TO 2000**  
**100 PRINT "before"**  
**110 PRINT 10/0** {generate an error}  
**120 PRINT "after"**  
**1000 END WHEN**  
**1010 STOP**  
**2000 IF ERR\_OV THEN PRINT "it's overflowed on line ";ERLIN**

## **WIDTH** devices

**WIDTH** allows the default width for non-console based devices to be specified, for example printers.

syntax: *line\_width:= numeric\_expression*

**WIDTH** [*channel*,] *line\_width*

example: i. **WIDTH 80** {set the device width to 80}  
ii. **WIDTH #6,72** {set the width of the device attached to channel 6 to 72}

## **WINDOW** windows

Allows the user to change the position and size of the *window* attached to the specified or default channel. Any borders are removed when the window is redefined.

Minerva4Q68 allows the border size and colour to be specified as the window is moved.

Coordinates are specified using the *pixel system* relative to the screen origin.

syntax: *width:= numeric\_expression*  
*depth:= numeric\_expression*  
*x:= numeric\_expression*  
*y:= numeric\_expression*  
*size:= numeric\_expression*  
*colour:= numeric\_expression* {range 0 ... 255}

**WINDOW** [#*channel*,] *width*, *depth*, *x*, *y* [\ *size* [, *colour*]]

example: i. **WINDOW 30, 40, 10, 10** {window 30x40 pixels at 10,10}  
ii. **WINDOW 100, 50, 0, 0 \ 4** {window with a 4 pixel border}  
iii. **WINDOW 100, 50, 10, 20 \ 4, 2** {window with a 4 pixel border in red}

## A

|                      |       |
|----------------------|-------|
| ABS.....             | 3     |
| Absolute values..... | 3     |
| ACOS.....            | 3     |
| ACOT.....            | 3     |
| ADATE.....           | 3     |
| ALFM.....            | 4     |
| ARC.....             | 4     |
| ARC_R.....           | 4     |
| Arccosine.....       | 3     |
| Arccotangent.....    | 3     |
| Arcsine.....         | 3     |
| Arctangent.....      | 3     |
| Arrays.....          |       |
| DIM.....             | 19    |
| DIMN.....            | 19    |
| ASIN.....            | 3     |
| Assignment.....      | 33    |
| AT.....              | 5     |
| ATAN.....            | 3     |
| AUTO.....            | 5, 22 |

## B

|                |   |
|----------------|---|
| BAUD.....      | 6 |
| Baudrates..... | 6 |
| BEEP.....      | 7 |
| BEEPING.....   | 7 |
| BLOCK.....     | 8 |
| BORDER.....    | 8 |

## C

|                 |    |
|-----------------|----|
| CALL.....       | 9  |
| CARD_INIT.....  | 9  |
| Channel.....    |    |
| CLOSE.....      | 11 |
| Character.....  |    |
| CODE.....       | 11 |
| size.....       | 13 |
| CHR\$.....      | 9  |
| CIRCLE.....     | 10 |
| CIRCLE_R.....   | 10 |
| CLEAR.....      | 10 |
| screen.....     | 11 |
| SuperBASIC..... | 10 |
| window.....     | 11 |
| CLOCK.....      |    |
| ADATE.....      | 3  |
| DATE.....       | 16 |
| DATE\$.....     | 16 |
| DAY\$.....      | 16 |
| SDATE.....      | 54 |
| CLOSE.....      | 11 |
| Closing.....    |    |
| channels.....   | 11 |
| CLS.....        | 11 |
| CODE.....       | 11 |
| Colour.....     |    |
| INK.....        | 29 |
| MODE.....       | 36 |
| PAPER.....      | 42 |

|                     |    |
|---------------------|----|
| RECOL.....          | 47 |
| recolour.....       | 47 |
| Comments.....       | 48 |
| Communications..... |    |
| baud rates.....     | 6  |
| Conditions.....     |    |
| IF.....             | 27 |
| SELEct.....         | 54 |
| CONTINUE.....       | 12 |
| COPY.....           | 12 |
| COPY_N.....         | 12 |
| COS.....            | 12 |
| cosine.....         | 12 |
| COT.....            | 13 |
| cotangent.....      | 13 |
| CSIZE.....          | 13 |
| CURSOR.....         | 14 |

## D

|                        |    |
|------------------------|----|
| DATA.....              | 14 |
| structures.....        | 19 |
| DATE.....              | 16 |
| DATE\$.....            | 16 |
| DAYS\$.....            | 16 |
| DEFine.....            |    |
| FuNction.....          | 17 |
| PROCedure.....         | 18 |
| DEG.....               | 19 |
| Degrees.....           | 19 |
| Delay.....             | 42 |
| DELETE.....            | 19 |
| files.....             | 19 |
| lines.....             | 21 |
| Devices.....           |    |
| directory.....         | 20 |
| LBYTES.....            | 32 |
| LOAD.....              | 34 |
| load and run.....      | 35 |
| LRUN.....              | 35 |
| MERGE.....             | 36 |
| merge and run.....     | 38 |
| MRUN.....              | 38 |
| NET.....               | 38 |
| OPEN.....              | 40 |
| open for input.....    | 40 |
| open new.....          | 40 |
| OPEN_IN.....           | 40 |
| OPEN_NEW.....          | 40 |
| RUN.....               | 51 |
| SAVE.....              | 51 |
| SBYTES.....            | 52 |
| DIM.....               | 19 |
| Dimension arrays.....  | 19 |
| DIMN.....              | 19 |
| DIR.....               | 20 |
| Directory.....         | 20 |
| Directory devices..... |    |
| COPY.....              | 12 |
| copying.....           | 12 |
| DELETE.....            | 19 |
| deleting files.....    | 19 |
| FORMAT.....            | 26 |

|                              |    |
|------------------------------|----|
| LOAD.....                    | 34 |
| loading SBASIC programs..... | 34 |
| SAVE.....                    | 51 |
| saving SBASIC programs.....  | 51 |
| DISP_MODE.....               | 20 |
| Display directory.....       | 20 |
| DIV.....                     | 21 |
| DLINE.....                   | 21 |
| DO.....                      | 21 |
| Documentation.....           | 48 |
| Dots.....                    | 44 |

## E

|                     |        |
|---------------------|--------|
| EDIT.....           | 22     |
| ELLIPSE.....        | 10     |
| ELLIPSE_R.....      | 10     |
| ELSE.....           | 27     |
| END.....            |        |
| DEFine.....         | 17, 18 |
| FOR.....            | 25     |
| IF.....             | 27     |
| REPeat.....         | 49     |
| SELection.....      | 54     |
| WHEN.....           | 61     |
| EOF.....            | 22     |
| Equals.....         | 33     |
| ERLIN.....          | 23     |
| ERNUM.....          | 23     |
| Errors.....         |        |
| CONTINUE.....       | 12     |
| RETRY.....          | 12     |
| EXEC.....           | 23     |
| EXEC_W.....         | 23     |
| EXIT.....           | 23     |
| with FOR.....       | 23     |
| with REPeat.....    | 23     |
| EXP.....            | 24     |
| Exponentiation..... | 24     |
| EXT.....            | 39     |

## F

|                     |    |
|---------------------|----|
| Fast Memory.....    |    |
| ALFM.....           | 4  |
| FREE_FMEM.....      | 26 |
| Files.....          |    |
| COPY.....           | 12 |
| DELETE.....         | 19 |
| DIR.....            | 20 |
| directory.....      | 20 |
| LOAD.....           | 34 |
| load and run.....   | 35 |
| LRUN.....           | 35 |
| MERGE.....          | 36 |
| merge and run.....  | 38 |
| MRUN.....           | 38 |
| OPEN.....           | 40 |
| open for input..... | 40 |
| open new.....       | 40 |
| OPEN_IN.....        | 40 |
| OPEN_NEW.....       | 40 |
| PRINT.....          | 46 |
| RUN.....            | 51 |

|                |    |
|----------------|----|
| FILL.....      | 24 |
| FILL\$.....    | 24 |
| FLASH.....     | 24 |
| FN.....        | 17 |
| FOR.....       | 25 |
| with EXIT..... | 25 |
| with NEXT..... | 25 |
| FORMAT.....    | 26 |
| FREE_FMEM..... | 26 |
| FuNction.....  | 17 |
| DEFine.....    | 17 |
| RETurn.....    | 17 |

## G

|                 |    |
|-----------------|----|
| GOSUB.....      | 26 |
| GOTO.....       | 27 |
| Graphics.....   |    |
| ARC.....        | 4  |
| ARC_R.....      | 4  |
| CIRCLE.....     | 10 |
| CIRCLE_R.....   | 10 |
| ELLIPSE.....    | 10 |
| ELLIPSE_R.....  | 10 |
| FILL.....       | 24 |
| fill shape..... | 24 |
| LINE.....       | 33 |
| LINE_R.....     | 33 |
| POINT.....      | 44 |
| POINT_R.....    | 44 |
| Turtle.....     |    |
| FILL.....       | 24 |
| MOVE.....       | 38 |
| PENDOWN.....    | 44 |
| PENUP.....      | 44 |
| SCALE.....      | 52 |
| TURN.....       | 59 |
| TURNTO.....     | 59 |

## H

|                   |    |
|-------------------|----|
| Highlighting..... | 58 |
|-------------------|----|

## I

|                     |    |
|---------------------|----|
| I/O.....            |    |
| INKEY\$.....        | 29 |
| keyboard input..... | 32 |
| KEYROW.....         | 32 |
| printing.....       | 46 |
| IF.....             | 27 |
| nesting.....        | 28 |
| INK.....            | 29 |
| INKEY\$.....        | 29 |
| INPUT.....          | 30 |
| INSTR.....          | 30 |
| INT.....            | 31 |
| Integer divide..... | 21 |

## J

|           |    |
|-----------|----|
| Jump..... | 27 |
|-----------|----|



## K

|                     |        |
|---------------------|--------|
| KBTABLE.....        | 31     |
| Keyboard input..... | 29, 32 |
| KEYROW.....         | 32     |

## L

|                        |     |
|------------------------|-----|
| LBYTES.....            | 32  |
| LEN.....               | 33  |
| Length of strings..... | 33  |
| LET.....               | 33  |
| LINE.....              | 33  |
| delete.....            | 21  |
| DLINE.....             | 21  |
| numbering.....         | 5   |
| RENUM.....             | 48  |
| renumbering.....       | 48  |
| LINE_R.....            | 33  |
| LIST.....              | 34  |
| LN.....                | 34  |
| LOAD.....              | 34  |
| Load and run.....      | 35  |
| LOCaL.....             | 35  |
| Local variables.....   | 35  |
| in functions.....      | 17  |
| in procedures.....     | 18  |
| LOG10.....             | 34  |
| Logarithm.....         | 34  |
| Loop epilogue.....     | 25  |
| Loop repetition.....   | ... |
| EXIT.....              | 23  |
| EXT.....               | 39  |
| FOR.....               | 25  |
| NEXT.....              | 39  |
| REPeat.....            | 49  |
| LRUN.....              | 35  |

## M

|                       |     |
|-----------------------|-----|
| MACHINE.....          | 35  |
| Machine code.....     | 9   |
| CALL.....             | 9   |
| EXEC.....             | 23  |
| EXEC_W.....           | 23  |
| loading.....          | 23  |
| RESPR.....            | 50  |
| saving.....           | 57  |
| SEXEC.....            | 57  |
| Maths functions.....  | ... |
| ABS.....              | 3   |
| absolute value.....   | 3   |
| ACOS.....             | 3   |
| ACOT.....             | 3   |
| arc cosine.....       | 3   |
| arc cotangent.....    | 3   |
| arc sine.....         | 3   |
| arc tangent.....      | 3   |
| ASIN.....             | 3   |
| ATAN.....             | 3   |
| common logarithm..... | 34  |
| COS.....              | 12  |
| cosine.....           | 12  |
| COT.....              | 13  |

|                         |     |
|-------------------------|-----|
| cotangent.....          | 13  |
| EXP.....                | 24  |
| exponentiation.....     | 24  |
| INT.....                | 31  |
| integer part.....       | 31  |
| LN.....                 | 34  |
| LOG10.....              | 34  |
| natural logarithm.....  | 34  |
| RAD.....                | 47  |
| radians conversion..... | 47  |
| SIN.....                | 57  |
| sine.....               | 57  |
| SQRT.....               | 58  |
| square root.....        | 58  |
| TAN.....                | 58  |
| tangent.....            | 58  |
| MERGE.....              | 36  |
| Merge and run.....      | 38  |
| MOD.....                | 36  |
| MODE.....               | 36  |
| modulus.....            | 36  |
| MOVE.....               | 38  |
| MRUN.....               | 38  |
| Multitasking.....       | ... |
| PAUSE.....              | 42  |
| SEXEC.....              | 57  |

## N

|                  |    |
|------------------|----|
| NET.....         | 38 |
| Networks.....    | 38 |
| NEW.....         | 38 |
| NEXT.....        | 39 |
| with FOR.....    | 25 |
| with REPeat..... | 49 |

## O

|                         |     |
|-------------------------|-----|
| ON...GOSUB.....         | 39  |
| ON...GOTO.....          | 39  |
| OPEN.....               | 40  |
| channel.....            | 40  |
| serial port.....        | 40  |
| window.....             | 40  |
| Open existing file..... | 40  |
| Open new file.....      | 40  |
| OPEN_DIR.....           | 40  |
| OPEN_IN.....            | 40  |
| OPEN_NEW.....           | 40  |
| OPEN_OVER.....          | 40  |
| Operators.....          | ... |
| INSTR.....              | 30  |
| MOD.....                | 36  |
| OVER.....               | 41  |
| Overprinting.....       | 41  |

## P

|                 |        |
|-----------------|--------|
| PAN.....        | 41     |
| PAPER.....      | 42     |
| Parameters..... | 17, 18 |
| PAUSE.....      | 42     |
| PEEK.....       | 43     |
| PEEK_L.....     | 43     |

|                      |    |
|----------------------|----|
| PEEK_W.....          | 43 |
| PENDOWN.....         | 44 |
| PENUP.....           | 44 |
| PI.....              | 44 |
| Plotting points..... | 44 |
| POINT.....           | 44 |
| POINT_R.....         | 44 |
| POKE.....            | 45 |
| POKE_L.....          | 45 |
| POKE_W.....          | 45 |
| PRINT.....           | 46 |
| OVER.....            | 41 |
| UNDER.....           | 60 |
| Printout.....        | 46 |
| PRocedures.....      |    |
| DEFine.....          | 17 |
| LOCal.....           | 35 |
| RETurn.....          | 50 |
| PROCESSOR.....       | 47 |
| Programs.....        |    |
| CONTINUE.....        | 12 |
| RETRY.....           | 12 |
| RUN.....             | 51 |
| SAVE.....            | 51 |

## R

|                     |       |
|---------------------|-------|
| RAD.....            | 47    |
| Random numbers..... | 47    |
| RANDOMISE.....      | 47    |
| READ.....           | 14    |
| RECOL.....          | 47    |
| REMark.....         | 48    |
| RENUM.....          | 48    |
| Renumber lines..... | 48    |
| REPeat.....         | 49    |
| EXIT.....           | 23    |
| NEXT.....           | 39    |
| REPetition.....     |       |
| FOR.....            | 25    |
| NEXT.....           | 39    |
| REPORT.....         | 49    |
| Reset clock.....    | 3, 54 |
| Resolution.....     | 36    |
| RESPR.....          | 50    |
| RESTORE.....        | 14    |
| RETRY.....          | 12    |
| RETurn.....         | 50    |
| with FuNction.....  | 17    |
| with PROCedure..... | 18    |
| RND.....            | 51    |
| RS-232-C.....       | 6     |
| RUN.....            | 51    |
| load and run.....   | 35    |
| LRUN.....           | 35    |

## S

|               |    |
|---------------|----|
| SAVE.....     | 51 |
| SBYTES.....   | 52 |
| SCALE.....    | 52 |
| SCR_BASE..... | 53 |
| SCR_LLEN..... | 53 |
| SCR_XLIM..... | 53 |

|                             |        |
|-----------------------------|--------|
| SCR- YLIM.....              | 53     |
| Screen.....                 |        |
| BLOCK.....                  | 8      |
| BORDER.....                 | 8      |
| character size.....         | 13     |
| clear.....                  | 11     |
| FLASH.....                  | 24     |
| INK.....                    | 29     |
| MODE.....                   | 36     |
| output.....                 | 46     |
| OVER.....                   | 41     |
| overprinting.....           | 41     |
| PAN.....                    | 41     |
| PAPER.....                  | 42     |
| PRINT.....                  | 46     |
| RECOL.....                  | 47     |
| recolouring.....            | 47     |
| SCROLL.....                 | 53     |
| STRIP.....                  | 58     |
| UNDER.....                  | 60     |
| underlining.....            | 60     |
| WINDOW.....                 | 62     |
| SCROLL.....                 | 53     |
| SD card.....                |        |
| CARD_INIT.....              | 9      |
| SDATE.....                  | 54     |
| SElect.....                 | 54     |
| SER_BUFF.....               | 55     |
| SER_CLEAR.....              | 55     |
| SER_FLOW.....               | 56     |
| SER_ROOM.....               | 56     |
| SER_USE.....                | 57     |
| Setting clock.....          | 54     |
| Setting station number..... | 38     |
| SEXEC.....                  | 57     |
| Shapes.....                 |        |
| ARC.....                    | 4      |
| CIRCLE.....                 | 10     |
| ELLIPSE.....                | 10     |
| FILL.....                   | 24     |
| LINE.....                   | 33     |
| SIN.....                    | 57     |
| Sine.....                   | 57     |
| Size of characters.....     | 13     |
| SLUG.....                   | 58     |
| Sound.....                  |        |
| BEEP.....                   | 7      |
| BEEPING.....                | 7      |
| SQRT.....                   | 58     |
| Square root.....            | 58     |
| Starting programs.....      | 35, 51 |
| Station number.....         | 38     |
| STOP.....                   | 58     |
| Strings.....                |        |
| CHR\$.....                  | 9      |
| FILL\$.....                 | 24     |
| INSTR.....                  | 30     |
| LEN.....                    | 33     |
| length.....                 | 33     |
| STRIP.....                  | 58     |
| Subroutines.....            | 17, 18 |

## T

|                      |    |
|----------------------|----|
| TAN.....             | 58 |
| Tangent.....         | 58 |
| THEN.....            | 27 |
| Time.....            |    |
| clock adjust.....    | 3  |
| clock reset.....     | 54 |
| date.....            | 16 |
| PAUSE.....           | 42 |
| TRA.....             | 59 |
| TURN.....            | 59 |
| TURNT0.....          | 59 |
| Turtle graphics..... |    |
| FILL.....            | 24 |
| MOVE.....            | 38 |
| PENDOWN.....         | 44 |
| PENUP.....           | 44 |
| TURN.....            | 59 |
| TURNT0.....          | 59 |

## U

|                         |    |
|-------------------------|----|
| Unconditional jump..... | 27 |
| UNDER.....              | 60 |
| Underlining.....        | 60 |

## V

|                      |    |
|----------------------|----|
| Value absolutes..... | 3  |
| VER\$.....           | 60 |

## W

|                     |       |
|---------------------|-------|
| WHEN ERRor.....     | 61    |
| WIDTH.....          | 62    |
| WINDOW.....         | 62    |
| Windows.....        |       |
| AT.....             | 5     |
| BLOCK.....          | 8     |
| Character size..... | 13    |
| clear.....          | 11    |
| CSIZE.....          | 13    |
| cursor control..... | 5, 14 |
| FILL.....           | 24    |
| FLASH.....          | 24    |
| INK.....            | 29    |
| MODE.....           | 36    |
| OVER.....           | 41    |
| overprinting.....   | 41    |
| PAN.....            | 41    |
| PAPER.....          | 42    |
| print position..... | 5     |
| SCROLL.....         | 53    |
| STRIP.....          | 58    |
| UNDER.....          | 60    |
| underlining.....    | 60    |