

QBasic

en.wikibooks.org

June 11, 2019

On the 28th of April 2012 the contents of the English as well as German Wikibooks and Wikipedia projects were licensed under Creative Commons Attribution-ShareAlike 3.0 Unported license. A URI to this license is given in the list of figures on page 93. If this document is a derived work from the contents of one of these projects and the content was still licensed by the project under this license at the time of derivation this document has to be licensed under the same, a similar or a compatible license, as stated in section 4b of the license. The list of contributors is included in chapter Contributors on page 89. The licenses GPL, LGPL and GFDL are included in chapter Licenses on page 97, since this book and/or parts of it may or may not be licensed under one or more of these licenses, and thus require inclusion of these licenses. The licenses of the figures are given in the list of figures on page 93. This PDF was generated by the \LaTeX typesetting software. The \LaTeX source code is included as an attachment (`source.7z.txt`) in this PDF file. To extract the source from the PDF file, you can use the `pdfdetach` tool including in the `poppler` suite, or the <http://www.pdfplabs.com/tools/pdftk-the-pdf-toolkit/> utility. Some PDF viewers may also let you save the attachment to a file. After extracting it from the PDF file you have to rename it to `source.7z`. To uncompress the resulting archive we recommend the use of <http://www.7-zip.org/>. The \LaTeX source itself was generated by a program written by Dirk Hünninger, which is freely available under an open source license from http://de.wikibooks.org/wiki/Benutzer:Dirk_Huenniger/wb2pdf.

Contents

1	QBasic/Print version	3
2	Introduction	5
3	Basic Input	7
3.1	6INPUT.BAS	7
3.2	INPUT # and LINE INPUT	8
4	Text Output	9
4.1	Your first QBasic program: 1HELLO.BAS	9
4.2	1HELLO.BAS	9
4.3	PRINT	9
4.4	2HELLO.BAS	9
4.5	PRINT, Commas, Semicolons and CLS	10
4.6	3HELLO.BAS	10
4.7	Variables	10
4.8	4FACE.BAS	10
4.9	LOCATE statement	11
4.10	5FACE.BAS	11
4.11	COLOR statement	11
4.12	Color by Number	12
4.13	Summary	12
5	Basic Math	13
5.1	Equation Setup	13
5.2	7MATH.BAS	13
5.3	Understanding 7MATH.BAS	14
5.4	8MATH.BAS	14
5.5	Understanding 8MATH.BAS	14
5.6	9TIP.BAS	14
5.7	Tip Calculator	15
5.8	10OROP.BAS	15
5.9	Parentheses and Order of Operations	16
5.10	Random Numbers	16
5.11	11RND.BAS	16
5.12	More on RND	16
6	Flow Control	19
6.1	Conditional execution	19
6.2	True or False	19

6.3	IF	19
6.4	IF...THEN...ELSE	20
6.5	13 ELSEIF	20
6.6	FOR...NEXT	20
6.7	14FOR.BAS	21
6.8	WHILE...WEND	21
6.9	15WHILE.BAS	21
6.10	DO...LOOP	21
6.11	12IF.BAS	22
6.12	SELECT CASE	22
7	Advanced Input	25
7.1	INKEY\$	25
7.2	The keyboard buffer	25
7.3	Scancodes	26
8	Subroutines and Functions	27
8.1	Purpose	27
8.2	Procedure vs. Function	27
8.3	GOTO and GOSUB	27
8.4	ON ERROR	28
8.5	Declaring a subroutine	28
8.6	Declaring a function	29
9	Arrays and Types	31
9.1	Built-in Types	31
9.2	User-defined type	31
9.3	Array	32
9.4	Multidimensional array	32
9.5	Non-zero base	32
10	Files	33
10.1	The OPEN statement	33
10.2	Input and output	34
10.3	Reading the file from disk	34
10.4	Writing a file to the disk	35
11	Advanced Text Output	37
11.1	Cursor manipulation	37
11.2	Color	37
11.3	Formatted printing	37
12	Sound	39
12.1	BEEP	39
12.2	SOUND	39
12.3	PLAY	40
12.4	Simple Musical Keyboard In Qbasic	40
13	Graphics	43

14	Advanced Graphics	47
14.1	Animation	47
15	3d Graphics	51
15.1	Simple 3D Box	51
16	Images	53
16.1	WARNING	53
16.2	Simple image	53
16.3	Secondary use of <code>_newimage</code> for setting screen details	54
17	What Next	57
17.1	Websites	57
17.2	Further in programming	57
18	Sample Programs	59
18.1	Calculator	59
18.2	Basic Game	60
18.3	Clock	62
18.4	Binary Coder	65
19	Appendix	67
19.1	Commands	67
20	Contributors	89
	List of Figures	93
21	Licenses	97
21.1	GNU GENERAL PUBLIC LICENSE	97
21.2	GNU Free Documentation License	98
21.3	GNU Lesser General Public License	99

1 QBasic/Print version



This is the print version¹ of QBasic²

You won't see this message or any elements not part of the book's content when you print or preview³ this page.

1 https://en.wikibooks.org/wiki/Help:Print_versions

2 <https://en.wikibooks.org/wiki/QBasic>

3 http://en.wikibooks.org/w/index.php?title=QBasic/Print_version&printable=yes

2 Introduction

Microsoft QuickBasic and QBasic (Quick Beginners All-purpose Symbolic Instruction Code) are very popular programming languages for beginners. While its lack of power makes it unsuitable for many of today's applications, it is an invaluable learning tool. This wikibook will cover the usage of QBasic and most of its functions. While most of the text discussed will work under QBasic, that version of the Basic interpreter has limitations and certain advanced commands will only work under QuickBasic. One of the best Qbasic IDEs (Integrated Development Environment) is QB64. There are certain functions that work only in QB64. [View this Wikibook in full book view.](#)

3 Basic Input

The **INPUT** command is used to gather input from the user. This section will attempt to teach you how to gather input upon request from the user. For real-time input, see QBasic/Advanced Input¹. Here is the syntax of the input command:

```
INPUT [text to user]; [variable] Question mark added
```

OR

```
INPUT [text to user], [variable] No question mark added
```

Example:

```
INPUT What is your name; name$
```

OR

```
INPUT What is your age, age
```

When a semicolon (;) is used after the text output to the user, a question mark (?) and 'space' are added to the output. When a comma (,) is used no question mark is added. If a string is specified (eg 'name\$'), anything the user enters before pressing the 'return' key will be accepted. If a numeric variable (eg 'age') is specified, the user must enter a number (if any non-numeric key is entered, the error message "Redo from start" will be output and the INPUT command rerun)

3.1 6INPUT.BAS

```
1 CLS
2 INPUT What is your name; name$
3 PRINT Hello, ; name$; !
4 INPUT How old are you; age
5 INPUT what is your best computer game?, game$
6 PRINT name;; name$
7 PRINT age;; age; years old
8 PRINT best game;; game$
```

Please note: In the PRINT command, the (;) function *concatenates* (joins) the contents of the string variables with the text between the quotes (" "). Note the use of spaces so that the final printed text reads properly. If a numerical variables is specified within the PRINT command, an additional space is automatically added both before and after the number. **See also:** LINE INPUT command to read a line of text from a file (and place the result in a string variable) or to input a series of variables (in which case any commas found will be treated as delimiters between fields).

¹ https://en.wikibooks.org/wiki/QBasic/Advanced_Input

3.2 INPUT # and LINE INPUT

INPUT # uses an open file stream to collect data from the file itself. The file may be a data file, a bitmap, or a text file. The syntax is:

```
INPUT #file_stream, variable1 ; variable2$ more variables can be taken.
```

LINE INPUT is used to collect an entire line of a text file. Syntax:

```
LINE INPUT 1,file_line 1 is the file stream number. Can be any other number too.
```

WARNING: If input is taken beyond the file end, the error : "Input past end of file " is issued. You can use LOF and EOF functions to prevent errors. (LOF stands for LENGTH OF FILE while EOF stands for END OF FILE)

4 Text Output

Text Output

4.1 Your first QBasic program: 1HELLO.BAS

*The following paragraph requires a computer with QBasic installed*To begin, copy everything from the program below ("PRINT "Hello World") into a text editor or into the QBasic **IDE** (Integrated Development Interface) itself and save it as "1HELLO.BAS". Next open the file in QBasic (unless you used QBasic IDE in which case it is already open) and press F5. *Optionally you can use the "RUN" menu located on the menu bar at the top of the IDE window.* This will execute (run) the program. The words "Hello World" should appear on the upper left hand side of the screen. You have just executed your first QBasic program. If you press F5 again, another line of text saying "Hello World" will appear on the upper left hand side of the screen pushing the first one down to the second row of the screen. You can follow the same procedure for the rest of the example programs in this wikibook.

4.2 1HELLO.BAS

```
PRINT Hello World
```

4.3 PRINT

PRINT is QBasic's text output function. It is the command that we will be exploring through this section. PRINT is a QBasic function that requires arguments. The argument in the Hello World program we just ran were the words "Hello World". So, PRINT is the function and "Hello World" is the argument we *pass-to* the function. PRINT [Text to screen] **note, for a short cut, just use a question mark "?" in place of the command "PRINT". Likewise you can use a single quote "' in place of the key word REM to insert comments in your code*

4.4 2HELLO.BAS

```
PRINT This line will be erased
CLS
PRINT Hello;
PRINT World,
PRINT Hello Jupiter
```

```
PRINT Good Bye,,For; Now
PRINT 1,2,3,4,5

PRINT "Hello daddy"
```

4.5 PRINT, Commas, Semicolons and CLS

This is what the program output should look like:

```
Hello World   Hello Jupiter
Good Bye           For Now
1             2             3             4             5
```

The first line of 2HELLO.BAS outputs "This line will be erased." to the screen. However, in the second line, the CLS command clears the screen immediately after. So, it will only flash momentarily. The text "Hello Jupiter" should line up with '2' under it. More than one comma can be used consecutively. In this example, after "Good Bye" two commas are used to move "For Now" over two tab columns. "For Now" should line up with '3'. My final statement on this topic is to play around with it. Try using commas and semicolons in a program.

4.6 3HELLO.BAS

```
CLS
hello$ = Hello World
number = 12
PRINT hello$, number
```

4.7 Variables

Variables are used to store information. They are like containers. You can put information in them and later change the information to something else. In this first example they may not seem very useful but in the next section (Input) they will become very useful.

In this example we use two types of variables - string variables and numeric variables. A string variable holds words, a string of characters (a character is a letter, digit or symbol). In this case the characters are letters. A string variable is denoted by ending the name of the variable with a dollar sign. The string variable in this program is hello\$. What ever you set hello\$ equal to will be displayed in the PRINT statement. The numeric variable is number. Numeric variables do not have a special ending like string variables.

4.8 4FACE.BAS

```
CLS
LOCATE 14, 34      position the left eye
PRINT <=>         draw the left eye
```

```

LOCATE 14, 43    position the right eye
PRINT <=>       draw the right eye
LOCATE 16, 39    position the nose
PRINT o|o       draw the nose
LOCATE 18, 36    position the mouth
PRINT \_____/   draw the mouth
LOCATE 19, 42    the bottom
PRINT The Face by QBasic

```

4.9 LOCATE statement

LOCATE allows you to position the cursor for the next piece of text output. Contrary to Cartesian coordinates which read (X,Y), the locate statement is LOCATE Y,X. In this case Y is the distance down from the top of the screen and X is the distance from the left side of the screen. The reason that LOCATE does not follow the standard coordinate system is that it is not necessary to include the X portion, you can use the format LOCATE Y which just specifies the line to start on. LOCATE[row,column]

LOCATE[row]

4.10 5FACE.BAS

```

CLS

LOCATE 14, 34
COLOR 9
PRINT <=>

LOCATE 14, 43
PRINT <=>

COLOR 11
LOCATE 16, 39
PRINT o|o

COLOR 4
LOCATE 18, 36
PRINT \_____/

COLOR 20
LOCATE 19, 42
PRINT U

LOCATE 1, 1
COLOR 16, 1
PRINT Hello World

```

4.11 COLOR statement

The program 5FACE.BAS is broken into sections to make it easier to read. This is an example of a good programming habit. Each three line piece of code each piece of code specifies what color it's part of the face should be, where it should be and what it should

look like. The order of the position and the color is unimportant. The new statement COLOR allows you to change the color of the text. Once changed, all output will be in the new color until COLOR or CLS is used.

COLOR [foreground]

COLOR [foreground],[background]

The colors are designated by numbers which will be discussed in the next section.

4.12 Color by Number

There are 16 colors (in screen mode 0), numbered from 0 to 15.

0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Purple	13	Light Purple
6	Brown/Orange	14	Yellow (Light Orange)
7	Light Grey (White)	15	White (Light White)

If you look carefully at this chart you can see that there are 8 main colors (0 through 7) and then those colors repeat, each in a lighter shade. You may also notice that the colors act as a combination of binary values (where blue=1, green=2, red=4, etc.) This makes it much easier to memorize the color scheme. Blinking colors are also available: at 16, the colors start over again with blinking black and extend through 31 (blinking white). However, the blinking option is not available for the background, only for the text (foreground). Add 16 to the color you wish to blink. eg: 2+16=18 - Blinking Green, 4+16=20 - Blinking Red. It is possible to switch the blinking foreground text with an intense background, but this task is beyond the scope of this QBasic textbook, and may not work when MS Windows displays the console in a windowed mode.

4.13 Summary

In this section we looked at several methods to manipulate text output. All centered around the PRINT statement. LOCATE and COLOR modified where the text was displayed and how it looked. We used CLS to clear the screen and gave a brief introduction to variables which will be expanded upon in later sections.

5 Basic Math

There are six numerical variables within QBasic:

Type	Minimum	Maximum
Integer	-32,768	32,767
Long Integer	-2,147,483,648	2,147,483,647
Float	-3.37x10 ³⁸	3.37x10 ³⁸
Double	-1.67x10 ³⁰⁸	1.67x10 ³⁰⁸
64-bit Integer	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
64-bit Float	±1.18E-4932	±1.18E+4932

Please note that Integer and Float type variables for 64-bit are available only in QB64. A lot of programming is math. Don't let this scare you: a lot of the math is simple, but it's still math. In this section, we will look at doing some basic math (the same stuff you learned in the 3rd grade) and manipulating numbers.

5.1 Equation Setup

In QBasic an equation has a basic setup a right side and a left side. For instance X=5, as you can probably figure out, this sets the variable X to 5. But we can use variables on the right side too. Y=X*10 would set Y equal to 10 times X, in this situation, 50. In this next program I will show several equations to give you a feel for math.

5.2 7MATH.BAS

```
CLS

Set a-d to initial values
a = 10
b = 6
c = 3.1415
d = 3.333333

e = a + b
PRINT a; +; b; =; e

f = c * d
PRINT c; *; d; =; f

g = b - c
PRINT b; -; c; =; g

h = b / d
PRINT b; /; d; =; h
```

```
i = INT(d)
PRINT Remove the decimal from ; d; =; i
```

5.3 Understanding 7MATH.BAS

The most important thing you can take away from this is the setup for math equations. I think you can figure out what all the symbols are and what they do, but QBasic is picky about equations. For 'e=a+b', if you try 'a+b=e' it will not work. The final thing I would like to address in 7MATH.BAS is the INT() function. As far as vocabulary, a function is something that takes in a piece of information and gives you another piece of information back. So PRINT, was a statement, and INT() is a function. The INT() function takes a number and truncates it's decimal, it does not round. So INT(5.1) is 5 and INT(5.999) is still 5. If you want to round a number use CINT().

5.4 8MATH.BAS

```
CLS
INPUT Enter a number: , x
PRINT

x = x + 5
PRINT X is now: ; x

x = x * x
PRINT X is now: ; x

x = x / 5
PRINT X is now: ; x

x = x - 4
PRINT X is now: ; x

x = x / x
PRINT X should be 1: ; x
```

5.5 Understanding 8MATH.BAS

8MATH.BAS shows one simple concept that is very important in programming, but impossible in math. The way that the computer calculates the equation is it does all the math on the right side of the equation and then sticks it in the variable on the left side. So the equation $x=x+5$ makes perfect sense, unlike math where it is a contradiction. Reassigning a value to a variable based on its current value is common and a good way to keep the number of variables down.

5.6 9TIP.BAS

```
CLS
INPUT How much is your bill: , bill
```

```

INPUT What percent tip do you want to give: , tip

tip = tip / 100  change percent to decimal
tip = tip * bill change decimal to money

PRINT
PRINT The tip is; tip; $.
PRINT Pay; tip + bill; $ total.

```

5.7 Tip Calculator

9TIP.BAS calculates your tip and total bill from the bill and percent tip you wish to give. The first three lines clear the screen and get the information from the user. The fifth line changes the tip from a percent to the correct decimal by dividing by 100 (ex. 20%=.2 because $20/100=.2$) the next line takes that percent and turns it into a dollar value by multiplying the decimal value by the bill. So if your bill is \$20.00 and you leave a 20% tip, it multiplies $20*.2$ which is 4 or \$4.00. The last three lines format the output.

This is a good example of a complete program. It collects information from the user, it processes the information and it gives the user feedback. Also, the middle section of the program is a good example of variable conservation. This is subject that will take some practice to get used to. In writing a program, if you use too many variables, it will become difficult to keep track of all of them. If you try and conserve too much, your code may become difficult to understand. You may notice that the program may print more than two decimal places if you enter a bill that is not an exact dollar value. As an exercise, try modifying the program so that it only displays two decimal places - you can use the CINT() function or any other rounding method you intend to use.

5.8 10OROP.BAS

```

ORder of OPerations
CLS
a = 15
b = 10
c = 12.2
d = 1.618

PRINT a * b + c  these two are different
PRINT a * (b + c)

PRINT

PRINT b - c / d  these two are different
PRINT (b - c) / d

PRINT

PRINT a * b - c * d / a + d  these two are the same
PRINT (a * b) - ((c * d) / a) + d

```

5.9 Parentheses and Order of Operations

10OROP.BAS is an example of order of operations and how parentheses can be used to manipulate it. I do not want to go into an indepth explanation of the order of operations here. The best advice I can give is unless you are sure of the order of operations, use parentheses to make sure the equation works how you want. All you need to know about parentheses is that the deepest nested parentheses calculate first. If you wish to know more, there are plenty of algebra resources available. On that note, you may wish to brush up on algebra. While it is not necessary for programming, it can help make programming easier and it can allow you to create more advanced programs.

5.10 Random Numbers

Though we will not go into their use until the next section, I would like to discuss the generation of random numbers. QBasic has a random number statement, RND, that generates a random decimal between 0 and 1. You can think of it as a random percent. At first, this may seem like an odd way to generate random numbers. However, with a little math it is very easy to manipulate this to provide numbers in whatever range you want.

The first step is to multiply RND by a number (the range you want). For instance 'RND*10'. This will return random numbers (decimal numbers) between 0 and 10(both included). So, to pick a random number between zero and ten we would say '(RND*10)'

5.11 11RND.BAS

```
CLS
RANDOMIZE TIMER

PRINT Random number from 0-9:; RND * 10
PRINT

PRINT Random number from 1-10:; (RND * 10) + 1
PRINT

PRINT Random integer from 1-10:; INT(RND * 10) + 1
PRINT

PRINT Random even integer from 50-100:; INT(RND * 25) * 2 + 50
```

5.12 More on RND

A few notes on 11RND.BAS, the second line, RANDOMIZE TIMER, sets it so that the computer uses the current time to pick random number. If you don't do this, it picks the same random number every time (try it, write a one line program, PRINT RND, and run it over and over, your screen will fill up with the same number) this can prove useful for some applications, but not most. Stick RANDOMIZE TIMER in at the top of all your programs that use the RND statement and they will be far less predictable. This program just show

some ways to choose what you want from your random number generator. The last line shows that you can be very specific in what you get. Make sure to run this program several times to see the different results.

6 Flow Control

6.1 Conditional execution

To choose between two or more sections of the program to execute, the IF statement can be used. It is also possible to use the WHILE, DO UNTIL and CASE statements. All of these control conditional execution by using a Boolean logic 'test', the result of which is either TRUE or FALSE. To repeat a section of code for a set number of times, the FOR statement is used. The IF test can be executed in a single line, however it can also be used like the others to control a block of code.

6.2 True or False

Boolean logic is a test that yields one of only two possible results, true or false. The tests are always mathematical in nature .. when two characters (or strings) are 'compared' it is their ASCII codes that are used (thus $a < b$ and $b < A$). The comparison operators used in qbasic are: = true if two variables are equal < true if the first is less than the second =< true if the first is less than or equal to the second > true if the first is greater than the second >= true if the first is greater than or equal to the second <> true if the two are unequal Multiple tests can be linked together in the comparison, using the 'AND', 'OR' and 'NOT' operators. We will cover exactly what these mean later on, but you probably understand the first two already.

6.3 IF

One of the most useful statements in QBasic is the IF statement. It allows you to choose what your program will do depending on the conditions you give it. The next few programs will be taking a look at ways to use the IF statement.

```
IF [conditional] THEN [do this]
```

The single line IF is the simplest example. To execute a block of code, the END IF is used

```
IF [conditional] THEN
  [do this]
  [and do this]
  ...
  [and also do this]
END IF
```

6.4 IF...THEN...ELSE

```
IF [conditional] THEN [do this] ELSE [do that]
```

To choose between two different code blocks, the ELSE statement is used.

```
IF [conditional] THEN
  [do this]
  ..
  [and do this]
ELSE
  [do that]
  ..
  [and also that]
END IF
```

6.5 13 ELSEIF

As an alternative to starting an entirely new IF THEN ELSE statement sequence. You can follow the THEN statement(s) with ELSEIF [conditional] THEN. This does not create a new level of nesting. IF [conditional] THEN

```
[do this]
..
[and do this]
```

```
ELSEIF [conditional] THEN
```

```
[do that]
..
[and also that]
```

```
ELSEIF [conditional] THEN
```

```
[do the other]
..
[and also ...]
```

```
ELSE
```

```
[do this final thing]
```

```
END IF
```

6.6 FOR...NEXT

```
FOR <variable name> = <start value> TO <end value> [STEP <increment>]
  [do this]
  ..
  [and do this]
NEXT
```


<increment> may be + or - and is optional. If omitted the default is +1. The code contained within the FOR loop will always be executed at least once because it is only at the 'NEXT' statement that the value of the variable is checked against the end value. When the NEXT statement executes, the variable is modified by STEP value and compared against the end value. If the variable has not yet exceeded the end value, control is returned to the line following the FOR. You can exit a FOR loop early with the **EXIT FOR** command.

6.7 14FOR.BAS

```
CLS
RANDOMIZE TIMER

num = INT(RND * 20) + 1

FOR count = 1 TO 5
  INPUT Pick a number between 1 and 20: , answer
  IF answer = num THEN PRINT You win after;count;guesses!: END
NEXT
PRINT You lose
```

6.8 WHILE...WEND

```
WHILE <condition is true>
  [do this]
  ..
  [and this]
WEND
```

If the condition is true, the code following the WHILE is executed. When the WEND command is executed, it returns control to the WHILE statement (where the condition is tested again). When the condition evaluates to FALSE, control is passed to the statement following the WEND.

6.9 15WHILE.BAS

```
PRINT Press any key to continue
WHILE INKEY$=
WEND
```

In the example above, you see a press any key prompt that waits until the user presses a key. (The INKEY\$ feature will be described under Advanced Input¹.)

6.10 DO...LOOP

```
DO
  [this]
```

1 https://en.wikibooks.org/wiki/QBasic/Advanced_Input

```
..
[and this]
LOOP WHILE <condition is true> / LOOP UNTIL <condition is true>
```

The **DO...LOOP** construct is a more advanced of the WHILE loop - as with other flow control blocks, it is marked by **DO** and **LOOP** to denote the boundaries. It relies on a conditional statement placed after either DO or LOOP:

```
DO
  a$ = inkey$
LOOP WHILE a$=
```

As an alternative, you can instead replace **WHILE** with **UNTIL** have the loop continue until a specific condition is met:

```
DO
  x=x+1
LOOP UNTIL x >= 10
```

In some versions of BASIC the UNTIL or WHILE condition can follow the DO statement rather than the LOOP statement (pre-test) as apposed to the above shown (post-test).

6.11 12IF.BAS

```
CLS
RANDOMIZE TIMER

num = INT(RND * 100) + 1
DO
  INPUT Pick a number between 1 and 100: , answer

  IF num = answer THEN PRINT You Got It!
  IF num > answer THEN PRINT Too Small
  IF num < answer THEN PRINT Too big
LOOP UNTIL num = answer
PRINT Game Over.
```

6.12 SELECT CASE

```
SELECT CASE <variable expression>
CASE <value>
  [do this]
CASE <value 2>
  [do instead]
...
CASE ELSE
...
END SELECT
```

The select statement is a substitute for repeated use of IF statements. The <variable expression> is evaluated and compared against each CASE <value> in turn. When a CASE <value> is found to match, the [do this] code following is executed. If an EXIT CASE is executed, control passes to the line following the END SELECT, otherwise the next CASE <value> is checked. If no matches are found, the CASE ELSE is executed. Note that <value> may be a number, character or string or logical expression (eg '>0', '<>1').

Note also that multiple CASE matches may be found and executed (so, for example, if two CASE <values> are 'CASE >1' and 'CASE >10', a <variable expression> that evaluates to 11 (or more) will result in both CASE >1 and CASE >10 being executed)

```
CLS
PRINT WELCOME
PRINT I HAVE AN ANSWER FOR ANY OF YOUR QUESTIONS
10 INPUT WRITE YOUR QUESTION AND ILL GIVE YOU AN ANSWER , question$
RANDOMIZE TIMER
PRINT
answer = INT(RND * 4 + 1)
SELECT CASE answer
    CASE 1
        PRINT PLEASE REPHRASE YOUR QUESTION.
    CASE 2
        PRINT YOUR QUESTION IS MEANINGLESS.
    CASE 3
        PRINT DO YOU THINK I CAN ANSWER THIS?
    CASE 4
        PRINT THIS QUESTION LOOKS FUNNY.
END SELECT
PRINT
PRINT ENTER ANOTHER QUESTION, K$
GOTO 10
```

If a parameter would be covered by more than one case statement, the first option will take priority.

7 Advanced Input

7.1 INKEY\$

Getting real time information from the user is a little more difficult. To do so, we will use the INKEY\$ command, which checks to see if a user typed a key and provides the keypress to the program. Look at this code and then we will look at it in depth:

```
DO
  LET k$ = INKEY$
LOOP UNTIL k$ <> ""
SELECT CASE k$
  CASE "q"
    QuitProgram
  CASE "c"
    MakeCircle
  CASE "s"
    MakeSquare
END SELECT
```

The first part is the DO-LOOP which constantly polls INKEY\$ for a return value. In the basic use, INKEY\$ returns an empty string if no keys are being pressed and continues with the program. Once a key is pressed, INKEY\$ will return that key immediately.

7.2 The keyboard buffer

What is INKEY\$ doing and how does it work? While the INKEY\$ command looks like it returns the key currently being pressed, this is not the case. It is used by the program to answer the question, "What is IN the KEYboard buffer?" To understand this you will need to understand what a basic buffer is and why it is used. In older systems (not necessarily the IBM PC) a single chip processed keyboard input, and controlled the LED lights for caps lock and number lock. Because a computer does many things at once (e.g., take input from the mouse, crunch numbers, call subroutines, display new information on the screen), it needs to be able to remember what was pressed on the keyboard while it is busy. This chip contained some memory (called a buffer) that allow keeping track of a limited number of keypresses. Within the Dos platform under IBM PCs, the hardware has changed slightly. Instead of a hardware buffer, pressing or releasing a key will interrupt the running program to add a keystroke to a software buffer located in the BIOS. This procedure is usually unnoticed by the user and has minimal impact on system performance. However, this buffer allows for 15 characters - attempting to overflow it when the computer is busy will cause a short beep and drop any further characters. The INKEY\$ command uses this buffer as a FIFO (First In First Out) buffer. As an example let's say you have a game that has a bouncing ball on the screen and a paddle at the bottom. The computer program constantly has to

update the screen to show the movement of the ball. While it does this the program passes by an `INKEY$` command to see what value is returned. If the user has pressed a key since the last time the command was invoked it will return that key. Let's say the ball is moving over to the right and the user needs to press the "R" key to tell the program to move the paddle right. Since the program is busy moving the ball and updating the screen, it does not instantaneously notice that the user has pressed the key. Instead, the key press is stored in the keyboard buffer, and retrieved a few milliseconds (or microseconds) later when the `INKEY$` command is used. In many programs (as above), `INKEY$` will appear nested in a loop. It is requested over and over again. This allows the program to get user input one character at a time. Using our example above, the user may need to press R over and over again until the paddle is under the ball. On the other hand, the user may press R too many times and need to press L to move it left. Because the `INKEY$` command is using a FIFO buffer it will always retrieve the keys pressed in the same order as they were typed. In summary, the `INKEY$` command will always return and remove the first character in the keyboard buffer. Generally speaking, it is used over and over to retrieve every key that has been pressed, and to allow a user to interact with a program in a close approximation to "real time." If there is no key in the keyboard buffer, `INKEY$` it will return an empty string (no character).

7.3 Scancodes

Some keypresses are not associated with an ASCII character. When one of these keys is pressed, `INKEY$` returns a string with two characters = the first character is a null (ASCII code 0), and the second is the raw scan code for the keyboard. A full listing of the scancodes can be found within the QBASIC help file - you can also determine the scan codes by examining the results of `INKEY$` as you press those keys in question.

Note:

Some keys cannot be directly detected by `INKEY$`.

8 Subroutines and Functions

8.1 Purpose

Subroutines and functions are ways to break up your code into reusable 'lumps'. They allow the programmer reuse a large set of common instructions just by calling the appropriate procedure or function. For example, lets say you need to PRINT multiple Tables of values. One way to do this is to just enter all the Table PRINT commands directly into where you need them. However this not only makes the program very large but also makes it harder to debug or change the 'style' of the table. A simpler way is to create a single 'Print Table' procedure and enter all of the PRINT commands there. Then, each time need to print a Table, you would simply 'call' the 'Print Table' procedure with a list of the values to be printed.

8.2 Procedure vs. Function

A procedure does something and does not return anything for the programmer. For example, a procedure might be used to set the screen mode and palette. A function does something and RETURNS a value. For example, if you need to find the average of two values, you might write a function that takes in two numbers and returns the average.

8.3 GOTO and GOSUB

The GOTO and GOSUB statements were the original methods by which functions were created. They were the most common on older basic implementations and are kept around for compatibility reasons; however, their use is not recommended in other programming languages or in large scale projects, both because GOTO's make it harder to 'follow' the program flow and because GOSUB's do not 'isolate' the changes made to any variables. These two commands depend on Labels, which come in one of two forms. The first and older form involves writing line numbers at the beginning of each line (usually in increments of 10). The newer method looks similar to other programming languages, which is a single word followed by a colon. The GOTO statement is simple; it just moves the execution point to a given Label:

The GOSUB statement transfers control to a given Label, however when a RETURN statement is encountered, execution returns to the line following the GOSUB statement. Any changes made within the GOSUB will be to actual variables used within the 'main' code.

8.4 ON ERROR

The ON ERROR allows you to define an error handler for your program; when an error occurs, it immediately jumps to the given label. The control returns once the program reaches a RESUME statement, which can either return control to the same point, the next statement, or any other desired label. Within Qbasic, the error handler cannot be located within any subroutines. As such, any error checking or flags will have to be handled through the use of variables that are shared with the main module.

Note:

While the QBasic documentation states ON ERROR RESUME NEXT is a valid statement, this is incorrect.

NOTE: If your error handling routine does not have a "resume" statement in it (IOW you try to do it all with gotos) error handling will only work once - the next "on error" will be ignored and the program ends as if you had no "on error" statement at all. This problem does not seem to be mentioned in any of the documentation. It took me three hours to figure out why two nearly identical program portions acted so differently.

8.5 Declaring a subroutine

A superior method of declaring a subroutine is using the SUB statement block, because (by default) any new variables used within the subroutine are discarded on exit. Under the QBasic IDE, doing so moves the SUB block to its own section in the window to prevent accidental deletion of the module, and allows the easier organization of the program code. Calling a subroutine is as simple as writing the name of the subroutine (passing any required parameters). If you want, you can use the CALL statement to indicate to other programmers that it is a subroutine.

```
SUB name (params)
{SHARED variables if any}
{code to execute}
...
...
{STATIC variables if any, to be saved for use next time}
END SUB
```

Whilst the Parameters passed into subroutines are passed by 'reference' (i.e. they take on a new name within the SUB), any changes that are made to the values are 'reflected back' into the originals. By default, all other variables used within the SUB are discarded when the END SUB is reached (or an EXIT SUB is executed), except as below :- To 'preserve' the values of variables used within the SUB for re-use on the next CALL, use the STATIC keyword at the end. If you need access to a variable (that has not been passed as a parameter), use the SHARED keyword to define each at the start of the subroutine (a SHARED variable retains its name).

8.6 Declaring a function

A function is a form of subroutine that returns a value. Everything that applies in defining a subroutine also applies to a function. Within the function, the return value is created by using the function name as a variable - the return value is then passed to the calling expression.

```
FUNCTION name (params)
  Shared variable declarations
  name = result
  ...
END FUNCTION
```

Functions are declared in the same way as variables - it returns the variable type it's defined to return, in the same way variables are defined to contain their specified type. By default, it is a number, but appending a dollar sign indicates that it is returning a string. Functions can only be called within an expression; unlike subroutines, they are not a standalone statement.

9 Arrays and Types

Q basic is an IDE(integrated development environment) developed by Microsoft to create, edit, debug and execute basic program.

9.1 Built-in Types

QBasic has five built-in types: INTEGER (%), LONG(&) integer, SINGLE(!) float, DOUBLE(#) float and STRING(\$). QB64 has two more built-in types: `_INTEGER64 (&&)` and `_FLOAT (##)` Implicit declaration is by adding the type character to the end of the variable name (% , &, !, #, \$, &&, ##). See **QBasic/Basic math**¹ for more. Explicit declaration is by using the DIM statement before first use:

```
DIM a AS STRING
DIM b AS INTEGER
DIM c AS LONG
DIM d AS SINGLE
DIM e AS DOUBLE
DIM f AS _INTEGER64 QB64 only
DIM g AS _FLOAT QB64 only
```

If you do not use either implicit or explicit declaration, QBASIC interpreter assumes SINGLE type.

9.2 User-defined type

A user defined type allows you to create your own data structures. Please note that custom types are similar to arrays.

```
TYPE playertype
  name AS STRING
  score AS INTEGER
END TYPE
```

You can then declare variables under this type, and access them:

```
DIM playertype AS playertype
playertype.name = Bob
playertype.score = 92
```

This above example shows how a custom type can be used for maintaining data, say on a player who plays a game.

¹ https://en.wikibooks.org/wiki/QBasic/Basic_Math

9.3 Array

An array is a collection of values stored in a single variable. A STRING is an array of characters (so, for example, `char$(1)` means 1st character in string `char$`). Arrays of numbers should be defined using the DIM instruction (unless you DIM them, they are limited to 10 elements on each dimension). By default, arrays in QBasic are static in size and cannot be changed later in the program. Code that will set up this type of array is as follows:

```
DIM myArray(10) as TYPE this is explaining the datatype to be used during
program execution in array
```

TYPE can be any of the built in QBasic (INTEGER, LONG, SINGLE, DOUBLE, STRING) or user-defined type. If this is not specified, the array takes the Type defined by the variable name suffix - INTEGER (%), LONG(&) integer, SINGLE(!) float, DOUBLE(#), STRING(\$) - or INTEGER if none. WARNING: If your data Type is string, DIM string(10) defines a SINGLE string of 10 characters, NOT 10 strings of arbitrary length ! (10 strings of up to 128 chars each would be defined as DIM string(10,128) By issuing the Meta Command '\$DYNAMIC at the beginning of your program you can cause your arrays to be dynamic:

```
$DYNAMIC
DIM myDynamicArray(5) as INTEGER
REDIM myDynamicArray(10) as INTEGER
```

This is now perfectly legal code. To free up space occupied by an array, use the ERASE statement.

9.4 Multidimensional array

An array isn't restricted to one dimension - it's possible to declare an array to accept two parameters in order to represent a grid of values.

```
DIM housenames(25,25) as STRING
```

You cannot use the REDIM statement to change the number of dimensions on the array, even with dynamic allocation.

9.5 Non-zero base

In most languages, arrays start at the value 0, and count up. In basic, it's possible to index arrays so that they start at any value, and finish at any other value.

```
DIM deltas(-5 TO 5)
```

You can change the default lower bound with the OPTION BASE statement.

10 Files

In this lesson, we will learn how to create and modify files. In doing so, we will create a portion of a text editor to handle reading and writing files to and from disk - the program won't be complete by the end of this chapter, but will be finished within Advanced Text Output. Let's start by setting up our main procedure:

```
$DYNAMIC
ON ERROR GOTO handler  Prepares the error handler
DIM text(50) AS STRING  Used to contain the text file.
maxlines = 50  Contains the current size of the buffer.

DO
  CLS clears the screen
  INPUT Would you like to create a (N)ew file, (L)oad an existing one, or
  (E)xit the program; choice$

  SELECT CASE UCASE$(choice$) UCASE$ converts strings to UPPER CASE
    CASE N New file
    CASE L Load a file
    CASE E Exit
      CLS
      END
  END SELECT
LOOP returns to the top of the program.

handler:
errorflag = ERR  Keep track of the error that occurred.
RESUME NEXT  Proceeds with the next statement.
```

As you can see, we are using `CASE` rather than `IF`. `IF` statements can sometimes work better than case statements, but for now, we want to avoid *spaghetti code* (where there are too many `GOTO`'s). So far, we don't really have much, but it's a start. We've asked the user what they want to do, and finished 1/3 options. Not so shabby when you put it that way!

10.1 The `OPEN` statement

The `open` statement allows either reading or writing information from the disk. In general, the `open` statement follows this pattern:

```
OPEN file$ FOR INPUT AS 1
OPEN file$ FOR OUTPUT AS 2
```

The `file$` determines the filename to use. The `FOR` portion indicates how the file will be accessed or operated - it may be `APPEND`, `BINARY`, `INPUT`, `OUTPUT`, and `RANDOM`. The `AS #` is the identifier used for the file handle in question - this may be a variable if desired.

Note:

If you allow the user to enter a filename that does not exist on disk, you need to implement error handling using `ON ERROR` to react to this situation.

10.2 Input and output

When you need to access or write content to a file handle, the PRINT and INPUT statements expect a file handle to appear as the first parameter:

```
INPUT #1, a$
PRINT #2, a$
```

In some cases, you need to detect if you are going to reach the end of file - this is performed by the EOF function, which accepts a filehandle that takes input.

10.3 Reading the file from disk

We will now add a subroutine to read the complete file from disk, as lines of text, into an string array called text(). It is also possible to read a data file full of numerical values (and input these into a number array), however that is a different topic. Note the code that finds the file 'size', by reading lines one at a time until the End Of File is reached, and the use of 'SEEK' to 'rewind' to the beginning again.

```
SUB LoadFile
  SHARED filename$
  SHARED lines, maxlines
  SHARED text() AS STRING
  SHARED errorflag

  INPUT Enter filename: ; filename$

  OPEN filename$ FOR INPUT AS 1
  IF errorflag <> 0 THEN
    errorflag = 0
    CLOSE
    PRINT File not found - press return to continue.
    INPUT , a$
    EXIT SUB
  END IF

  Count the number of lines.
  lines = 0
  DO WHILE NOT EOF(1)
    LINE INPUT #1, l$
    lines = lines + 1
  LOOP

  Allocate enough space for input.
  IF maxlines > lines THEN
    REDIM text(lines + 25) AS STRING
    maxlines = lines + 25
  END IF
  SEEK #1, 1 Rewind to the beginning of the file.

  Read the lines into the buffer
  FOR cline = 1 TO lines
    LINE INPUT #1, text(cline)
  NEXT
  CLOSE 1
  errorflag = 0
END SUB
```

The example above treats the file as type=text. If the file contains numbers (for example, a data array of N integers per line x M lines) these can be read (input #) one at a time, directly into a numeric array. Input will read the numbers one at a time, 'stopping' after each is input. Numbers can be separated by 'anything' (so lines of text will be skipped).

10.4 Writing a file to the disk

The function for writing a file to disk is easier:

```
SUB SaveFile (outfile$)
  SHARED filename$
  SHARED lines, maxlines
  SHARED text() AS STRING
  SHARED errorflag

  IF outfile$ = THEN
    LOCATE 1, 1
    INPUT Enter filename: ; outfile$
  END IF

  OPEN outfile$ FOR OUTPUT AS 1
  IF errorflag <> 0 THEN
    errorflag = 0
    CLOSE
    PRINT Couldnt save file - press return to continue.
    INPUT , a$
    EXIT SUB
  END IF

  Write each line to the file
  FOR cline = 1 TO lines
    PRINT #1, text(cline)
  NEXT

  CLOSE 1
  errorflag = 0
  filename$ = outfile$

END SUB
```

10.4.1 Part 2: New file

In order to create a new file, you have to open it for OUTPUT, then close it. Example:

```
OPEN NEWFILE FOR OUTPUT AS #1
CLOSE #1
```

NOTE: If you accidentally open an existing file, all of its contents will be overwritten!

11 Advanced Text Output

11.1 Cursor manipulation

As you try to write your text editor, you may realize that you will need to place the cursor in a given location on the screen. This is performed using the `LOCATE` statement.

Note:

Printing any character in the bottom-right corner of the screen will cause the display to scroll.

11.2 Color

To change the current printing color, use the `COLOR` statement.

```
COLOR 7,0 Uses the default white on black.  
COLOR 15,0 Bright white on black.  
COLOR 0,1 Black on blue  
Color 14,0 Bright yellow.
```

This can be used for title or status bars at the bottom.

11.3 Formatted printing

The `PRINT USING` statement allows you to output strings or numbers in a specified format. With this statement, you can write numbers out to specified decimal places or perform advanced output. The most common format specifiers would be `#` and `.`, which reserve space for digits and decimal points respectively. You may also use the underscore to ensure that a given character is printed literally. Note - `PRINT USING` is unable to add 'leading zeros' to a number (if you specify eg 3 digits (`###`) a two digit number will be output with leading spaces).

12 Sound

QBasic has three ways of making sounds.

- BEEP command
- SOUND command
- PLAY command

12.1 BEEP

Earlier used as `PRINT CHR$(07)` the now available BEEP command makes a beep noise. It is commonly used to attract attention, when some important message is being displayed.

```
Syntax example
BEEP
PRINT CHR$(07)
```

The two commands , as can be seen by running this program on a computer, are exactly the same.

12.2 SOUND

The SOUND command produces sound of a specific frequency for a specific duration from the PC Speaker. Only one sound can be played at a time. If multiple SOUND statements are issued sequentially, every statement after the first will not execute until after the previous one finishes.

12.2.1 Syntax

```
SOUND f, d
```

- f - Frequency in Hertz
- d - Duration in ticks

12.2.2 Example

```
SOUND 100 , 20
```

Plays a 100 hertz wave for 20 ticks. One second is about 18.2 ticks, so 20 ticks are about 1.1 seconds. The lowest frequency allowed by QBasic is 37 Hz, which is roughly a D in the 2nd octave. The highest frequency is 32 767 Hz, but this tone can not be heard, because the normal human hearing range ends at 20 000 Hz. A secondary function of the SOUND command is to use it to control time in a program.

```
For x% = 1 TO 10
  Print x%
  Sound 32000,18.2
NEXT
```

This program will print the numbers 1 to 10, with a 1 second delay between each number.

12.3 PLAY

The PLAY command is for playing musical notes,octave. It can only play one note at a time. More than that the play command can play a complex stored "song" This is accomplished through the use of a song string. For string command detail, see QBasic/Appendix#PLAY¹.

12.4 Simple Musical Keyboard In Qbasic

```
rem Music keyboard
do
note$ = inkey$
select case ucase$(note$)
case A
Play A

case B
Play B

case C
Play C

case D
Play D

case E
Play E

case F
Play F

case G
Play G

end select

loop until ucase$(note$) = Q

end
```

¹ <https://en.wikibooks.org/wiki/QBasic/Appendix#PLAY>

This code uses a select case command to check values of note\$. Ucase\$ is used to maintain that there be no difference if Caps Lock is applied. The play command is used to play different notes. The other features of play have not been used here.

13 Graphics

What can QBasic do in the way of Graphics

QBasic is not that much graphically good, however many good programs can be created with it. Commands like PSET, CIRCLE, LINE, etc., are used to draw graphics in Qbasic. One of the good programs created using QBasic is **SYMMETRIC ICON** and **SYMMETRIC FRACTALS**.

The PSET Command

The PSET command lets the programmer display pixels on the screen. Before you type the command in, you must make sure that a SCREEN command is in. Look at this example:

```
SCREEN 13
PSET (1,1), 43
```

This command will display one yellow pixel at the coordinates 1, 1. The coordinates are X and Y coordinates, like any other mathematical situation. So the PSET command has to have this layout to function properly:

```
PSET ([X coordinate], [Y coordinate]), [Colour of Pixel]
```

Remember that X coordinates are those that go left to right on the screen and Y coordinates are those that go Up to Down on the screen.

The LINE Command

This simple command displays a line. It can be in a command like this:

```
SCREEN 13
LINE (16, 24) - (40, 30), 1 BF
```

This simple program will input a line on the screen. For an in depth knowledge of what the numbers mean, look at this layout:

```
LINE ([coloum 1], [row 1]) - ([coloum 2], [row 2]), [Colour Number]
'1 Being the start of the Line
'2 Being the End of the line
```

Like the PSET command you MUST put a SCREEN command.

The CIRCLE Command

```
CIRCLE (100, 100), 25, 4,0,3.14
```

This displays a circle at the coordinates. The layout of the function is as follows:

```
CIRCLE ([X Coordinate], [Y Coordinate]), [Radius], [Colour Number]
```

Remember to put a SCREEN command at the front.

The PAINT command

To use PAINT, there must be a SCREEN command declared. The command has coordinates that tells QBasic where to start. The color number specifies the color to paint with, and the bordercolor tells the PAINT command that it should not paint further when it encounters a pixel of that color. In almost all cases, you will need to use the bordercolor parameter, simply because without it, PAINT will cover the entire screen.

```
PAINT ([X Coordinate],[Y Coordinate]), [Color Number], [Border Color]1,2,3,4,5,5
```

Making An Image Using the DATA command

Graphics using this command can either be made by using a Graphics Editor or by using the DATA command Manually. The DATA command is a way of inputting information into QBasic and being read by the READ command. Remember that the DATA command cannot be used in subroutines or functions. Look at this example:

```
SCREEN 7
FOR y = 1 TO 10
FOR x = 1 TO 10
READ z
PSET (x, y), z
NEXT
NEXT
DATA 04, 04, 04, 04, 04, 04, 04, 04, 04, 04
DATA 04, 00, 00, 00, 00, 00, 00, 00, 00, 04
DATA 04, 00, 00, 00, 00, 00, 00, 00, 00, 04
DATA 04, 00, 00, 00, 00, 00, 00, 00, 00, 04
DATA 04, 00, 00, 00, 00, 00, 00, 00, 00, 04
DATA 04, 00, 00, 00, 00, 00, 00, 00, 00, 04
DATA 04, 00, 00, 00, 00, 00, 00, 00, 00, 04
DATA 04, 00, 00, 00, 00, 00, 00, 00, 00, 04
DATA 04, 04, 04, 04, 04, 04, 04, 04, 04, 04
```

The FOR commands declare the amount of pixels there are to be read. On this particular program, there are pixels of 10 by 10 (100) so we put:

```
FOR x = 1 TO 10
FOR y = 1 TO 10
```


We have now declared the x and y planes. You can change these values if you want a smaller or bigger picture Bitmap. The READ command reads the DATA commands and declares the information gathered as z.

```
READ z
```

The PSET reads the planes and the DATA read and a bitmap.

```
PSET (x, y), z
```

It works like the first example on this page, except it is reading more than one pixels

14 Advanced Graphics

14.1 Animation

14.1.1 Basic Movement

Animation is basically graphics that changes over a fixed period of time. In this we will be using a do-loop .

```
SCREEN 7    we need to use a graphics enabled screen mode
animation   calling the sub

SUB animation
  SCREEN 7
  x = 10 set first x- coordinate
  y = 10 set first y-coordinate
  DO
    CLS going back to a blank screen so that the previous rectangle is
    erased
    x = x + 3 setting increment of coordinate x
    y = y + 3 setting increment of coordinate y

    LINE (x, y)-(x + 5, y) drawing lines
    LINE (x, y + 5)-(x + 5, y + 5)
    LINE (x, y)-(x, y + 5)
    LINE (x + 5, y)-(x + 5, y + 5)

    SLEEP 2

  LOOP UNTIL INKEY$ <>

END SUB
```

Explanation:

1. We have switched from the default qbasic text-only screen to one which enables graphics.
2. We have called the sub which creates the animation.
3. We have begun the do-loop until. This enables the animation to run until the user ends it by pressing a key.
4. We have set an increment of the coordinates. This allows the box to be drawn on a new position rather than the same one. If it movement in only one direction was wished, we had to set the increment only in one variable.
5. We have drawn lines from each coordinate to another. Note that each time one coordinate remains fixed while the others change.(In this I refer to the two coordinate sets, the first starting and the ending one)

6. We have issued a sleep command . This stops execution for 2 seconds . Without this the do-loop will execute more quickly than we want , and the animation will be very short-lived.
7. By using RND for the variables, you can create a randomized ,unpredictable animation.

14.1.2 Mouse-Control

In this step, we will use the QB64 inbuilt `_mousehide`, `_mousex`, `_mousey`, `_mouseinput` and `_mousebutton` commands to control the mouse input.

WARNING

These Functions only work in QB64!

```
_mousehide
screen 7

mousetrack

sub mousetrack
do while _mouseinput
cls

X = _mousex
Y = _mousey

LINE (X - 10, Y)-(X + 10, Y), 15
LINE (X, Y - 10)-(X, Y + 10), 15

    IF _MOUSEBUTTON(1) THEN
        IF X > A AND X < A + 25 AND Y > B AND Y < B + 25 THEN
            message$ = yes
            goto action
        END IF
    END IF

loop until inkey$ <>
```

1. Here the first function `"_mousehide"` prevents the default pointer mouse format to be displayed on the screen
2. `Mouseinput` function retrieves mouse information.
3. The next functions `"_mousex"` and `"_mousey"` hold the current x and y coordinates of the mouse.
4. The lines draw a basic trigger .
5. The `"_mousebutton"` function returns the value of the mouse button pressed, `"1"` signifies the left button being pressed.

6. If the mouse button event has taken place within a certain enclosed area, a message in the form of "message\$" is issued. This can be used later on.
7. The procedure, if the previous condition has been fulfilled , goes to line label "action" where any commands to be executed may lie.
8. Else , the process loops , until a "loop until" condition has been met. It can also be something other than the one given above.

14.1.3 Usage

These graphics and animations can be used to create a complete game. Instead of the "Mouse" functions, you could use the "Inkey\$" command to issue various scenarios and cases, each with a complete code to decide what happens next.

Tip

Instead of making games which do not contain any user information, you could use ".txt" files to store information. This information can be later retrieved to make a game with a complete "Career" option.

15 3d Graphics

15.1 Simple 3D Box

3 Dimension or 3D graphics in Qbasic is nothing , but including an additional 'z' axis and to extend the 2 dimensional structure along that axis. This can be achieved by drawing a box, or the structure you want , each time at a new x and y position. It is quite like animation, except, we do not erase the structure after it being drawn , and there is no need of any intermediate pause. You can better understand by looking at the 3d box program given below.

```
Redo:
cls
screen 1
Input Enter the X-position?;k  entering coordinates of the screen from where
to start.
Input Enter the Y-position?;l   this also determines the size of the box.
color 1
for i = 1 to 50 step 2  rem box  step to ensure space between the boxes, make
it one to eliminate the space. The 50 number sets the extension of the box along
the z axis
  a = k+i:b = l+i          this for-next loop draws the box over and over
again, each with incremented values of k and l.
  line (a,a)-(b,a)
  line (a,b)-(b,b)
  line (a,a)-(a,b)
  line (b,a)-(b,b)
next                      rem diagonals
line (k,k)-(a,a)         the four diagonals to the structure , which make
it more realistic
line (l,l)-(b,b)
line (l,k)-(b,a)
line (k,l)-(a,b)
InputDo you want to redo? (Y/N);ans$
if ucase$(ans$)=Y then goto Redo
end
```


16 Images

16.1 WARNING

NONE of these functions work in IDEs other than QB64.

16.2 Simple image

```
rem image_display
cls
Dim Image as long
x = 1000 resolution
y = 1000
Image = _loadimage(TEST.jpg) loading the image
screen _newimage(x,y,32) putting screen sizes
_putimage (0,0),Image putting image
```

So, you were most probably expecting miles of code. And there you have it, all you need to display an image in QB64! So, what does this astoundingly simple piece of code even do? A step by step explanation:

1. We have DIMed our variable Image as a long value. This is because the image handle returned by the `_loadimage` function is a long value.
2. x and y are our variables. They hold the values we need to put as the resolution. Thus , for a 800 x 900 image, x = 800 , y = 900.
3. The image variable Image is next being put the handle values of the image "TEST.jpg". A healthy warning: Keep the image in the folder of the QB64 IDE. Or else , the function wont work!
4. Next, we have resized the screen to fit the image. The `newimage` function requires three parameters, the resolution parameters and the color mode parameters. Here we have used 32 bit color modes, you can also use 256 bit pallete color modes.
5. Lastly, we put the image using `_putimage`, which takes the Image variable(our image handle) as its parameter.

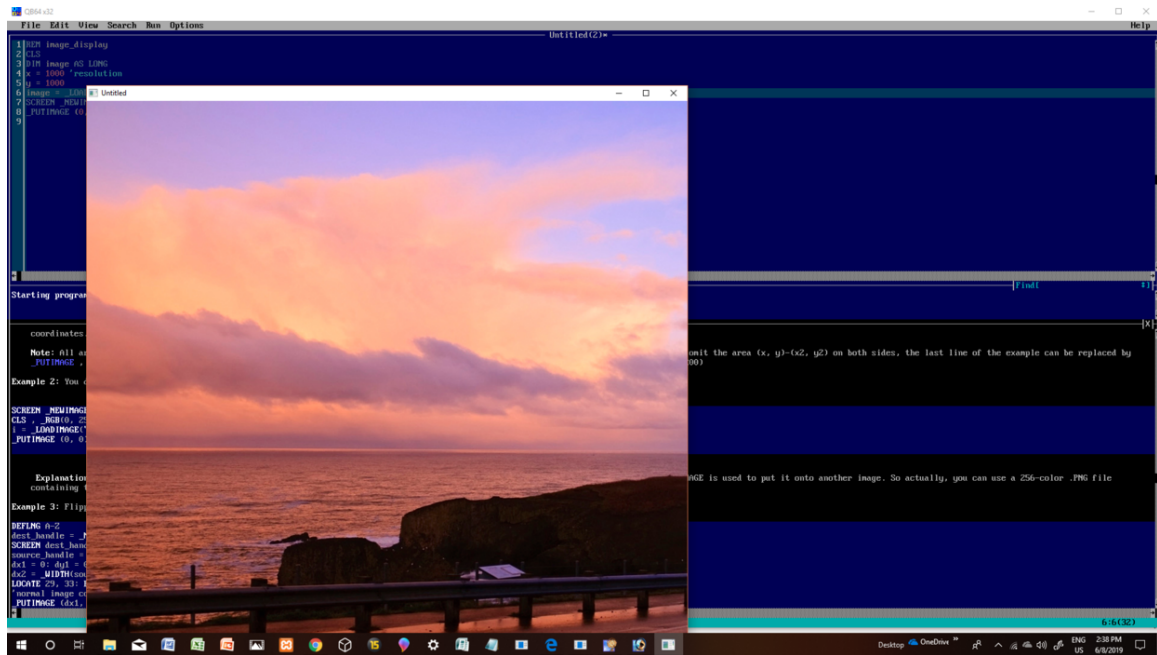


Figure 2 an example of images in qbasic

16.3 Secondary use of `_newimage` for setting screen details

Well, you must be thinking , all these commands must be used in this exact same order. Nah, that isnt the case. You can use `_newimage` solo, to set the screen details, like shown below:

```
screen _Newimage(1000,1000,256)
```

This code sets the screen to a massive 1000 x 1000 resolution , with 256 bit palette color modes!

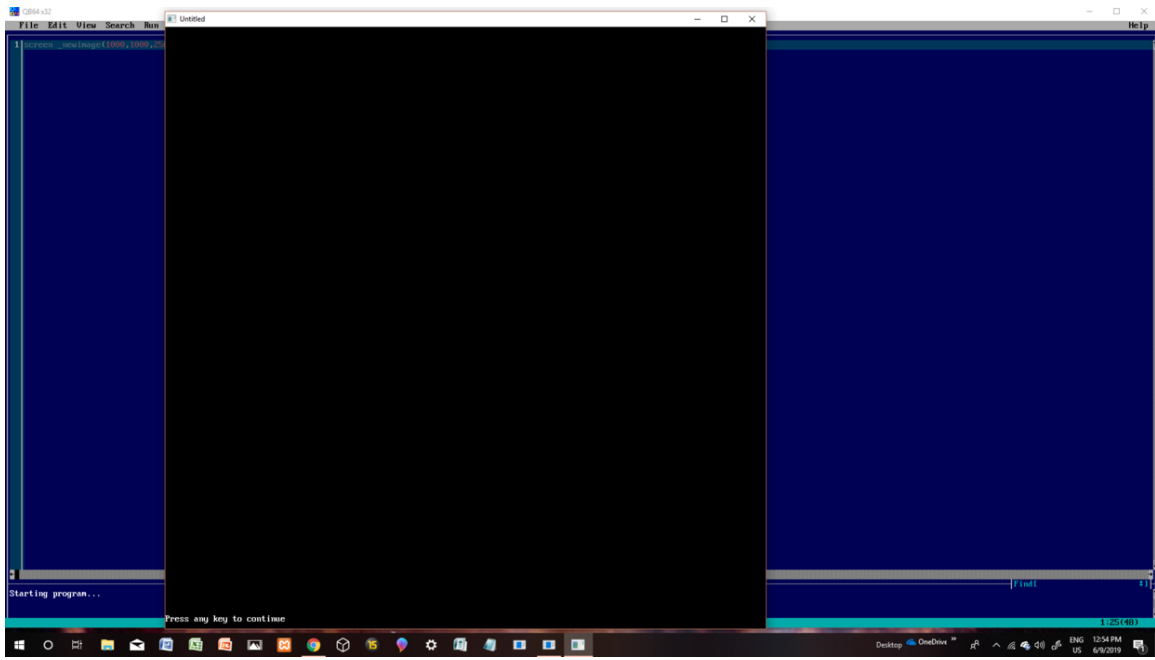


Figure 3

17 What Next

17.1 Websites

There are many good websites out there on the topic of QBasic. This is a list of the best sites: Petes QBasic Site¹: This site is mostly aimed at people thinking about programming video games in QBasic. The QBasic Page²: This is a good site for getting source codes and programs for QBasic. QBasic News³: The most recent and up to date news on the QBasic community. QBasic Programming for Kids⁴: A good site for young people to start programming. Qb64 ⁵ : All you need to know about QB64. Also, the QB64 IDE has an inbuilt help page, giving essential help when you don't know how to use a command. The only drawback is , it does not show any help unless you select the command and right click on it, and click on "HELP ON" which means, you atleast have to know the name of the command before you seek help on it.

17.2 Further in programming

If you are learning QBasic, chances are that you are still new to programming. It is a diverse world, explore it! A list of extremely good programming languages to continue further:

- *C++*⁶
- *HTML*⁷
- *Java*⁸
- *Javascript*⁹
- *CSS*¹⁰

1 <http://www.petesqbsite.com/>
2 <http://www.qbasic.com/>
3 <http://www.qbasicnews.com/>
4 <http://www.tedfelix.com/qbasic/>
5 <http://www.QB64.org/>
6 https://en.wikibooks.org/wiki/C%2B%2B_Programming
7 <https://en.wikibooks.org/wiki/HTML>
8 https://en.wikibooks.org/wiki/Java_Programming
9 <https://en.wikibooks.org/wiki/Javascript>
10 https://en.wikibooks.org/wiki/Cascading_Style_Sheets

18 Sample Programs

18.1 Calculator

This program can be used to make a simple, functioning calculator, very simply.

```
Rem calculator
cls
10
print input first operand
input a
print select operation
print addition(a)
print subtraction(s)
print multiplication(m)
print division(d)
print exponentification(e)
print rooting(r)
print Quit(q)
do
next$ = inkey$
loop until next$ <>
gosub input_var2

select case next$
case a
c = a + b
print sum is:;c
case s
c = a - b
print Difference is:;c
case m
c= a*b
print Product is :;c
case d
c = a/b
print Quotient is:;c
case e
c = a^b
print Exponentification is:c
case r
c = a^ 1/b
print Root is:;c
case q
end
end select
sleep 3
goto 10
```

```
sub input_var
```

```
input enter second operand;b
end sub
```

For reference goto Basic Math¹

18.2 Basic Game

Uses animation to make a simple game.

```
SCREEN 7
COLOR 15, 1
```

```
_MOUSEHIDE
CLS
LOCATE 5, 1
PRINT GUNSHOTS
```

```
DO
    NEXT$ = INKEY$
LOOP UNTIL NEXT$ <>
```

```
CLS
LOCATE 5, 1
PRINT In this game, you require to bring
PRINT
PRINT the crosshairs to the box
PRINT
PRINT , which is the target ,
PRINT
PRINT and click to shoot it.
PRINT
PRINT In this game , you control
PRINT
PRINT the crosshairs with your mouse.
PRINT
PRINT You will be given a
PRINT
PRINT fixed number of tries.
PRINT
PRINT The number of times you hit the target,
PRINT
PRINT you will be given a point
```

```
DO
    NEXT$ = INKEY$
LOOP UNTIL NEXT$ <>
```

```
CLS
LOCATE 5, 1
PRINT Get Ready!
```

```
DO
```

¹ https://en.wikibooks.org/wiki/QBasic/Basic_Math


```
NEXT$ = INKEY$
LOOP UNTIL NEXT$ <>

10
A = INT(RND * 100)
B = INT(RND * 100)

DO: K$ = INKEY$
  20

  DO WHILE _MOUSEINPUT
    CLS

    IF TRY_COUNT > 30 THEN
      CLS
      LOCATE 10, 1
      PRINT Remarks:
      IF POINT_COUNT < 10 THEN PRINT OH NO! NICE TRY!
      IF POINT_COUNT > 10 AND POINT_COUNT < 16 THEN PRINT GOOD WORK!
      IF POINT_COUNT > 15 AND POINT_COUNT < 21 THEN PRINT GREAT!
      IF POINT_COUNT > 20 AND POINT_COUNT < 26 THEN PRINT AMAZING!
      END
    END IF

    SECOND = VAL(RIGHT$(TIME$, 2))
    IF PREVSEC <> SECOND THEN
      COUNT = COUNT + 1
    END IF

    LOCATE 25, 25
    PRINT POINT_COUNT

    X = _MOUSEX: Y = _MOUSEY
    LINE (X - 10, Y)-(X + 10, Y), 15
    LINE (X, Y - 10)-(X, Y + 10), 15

    LINE (A, B)-(A + 25, B), 15
    LINE (A, B + 25)-(A + 25, B + 25), 15
    LINE (A, B)-(A, B + 25), 15
    LINE (A + 25, B)-(A + 25, B + 25), 15
    PAINT (A, B), (1), 15

    IF _MOUSEBUTTON(1) THEN
      IF X > A AND X < A + 25 AND Y > B AND Y < B + 25 THEN
        POINT_COUNT = POINT_COUNT + 1
        TRY_COUNT = TRY_COUNT + 1
        GOTO 10
      END IF
    END IF

    IF COUNT > 1 THEN
      COUNT = 0
      TRY_COUNT = TRY_COUNT + 1
```

```
        GOTO 10

    END IF

    PREVSEC = SECOND
    GOTO 20

LOOP
LOOP
```

For reference , goto [Advanced Graphics](#)²

18.3 Clock

A clock which is quite like a digital clock,with no hands.Use draw to make them if you want.

```
REM Clock
SCREEN 7

CLS
start:
SCREEN 7
_FONT 16
LOCATE 1, 5
PRINT CLOCK
PRINT -----

LINE (50, 50)-(100, 100), 1, BF
LOCATE 9, 5
PRINT TIME
LOCATE 10, 5
PRINT CONTROL
LINE (150, 50)-(200, 100), 2, BF
LOCATE 9, 18.5
PRINT STOP WATCH

DO

    exit$ = INKEY$
    IF exit$ = e OR exit$ = E THEN
        CLS
        SCREEN 7
        COLOR 2, 1
        LOCATE 5, 5
        PRINT YOU HAVE ABORTED THE CLOCK

        WHILE close_count <> 10
            close_count = close_count + 1
            LOCATE 7, 5
            PRINT APPLICATION ;
            IF close_count MOD 2 = 1 THEN
```

² https://en.wikibooks.org/wiki/QBasic/Advanced_Graphics

```
        PRINT CLOSING >>>
    ELSE
        PRINT CLOSING >>>
    END IF

    SLEEP 1
WEND

CLS
SCREEN 7
COLOR 10, 0
END
END IF

Mouser mx, my, mb
IF mb THEN
    IF mx >= 50 AND my >= 50 AND mx <= 100 AND my <= 100 THEN button down
        DO WHILE mb wait for button release
            Mouser mx, my, mb
            LOOP
            verify mouse still in box area
            IF mx >= 50 AND my >= 50 AND mx <= 100 AND my <= 100 THEN
                GOTO proccess
            END IF
        END IF
    END IF
END IF

Mouser mx, my, mb
IF mb THEN
    IF mx >= 150 AND my >= 50 AND mx <= 200 AND my <= 100 THEN button down
        DO WHILE mb wait for button release
            Mouser mx, my, mb
            LOOP
            verify mouse still in box area
            IF mx >= 150 AND my >= 50 AND mx <= 200 AND my <= 100 THEN
                time_control = 1
                GOTO proccess
            END IF
        END IF
    END IF
END IF
LOOP

proccess:
IF time_control = 0 THEN
    time_enter:
    LOCATE 12, 6
    INPUT enter time; t
    IF t > 1800 THEN
        mistake = mistake + 1
        IF mistake > 3 THEN
            PRINT BLOCKED
        END
    END IF
END IF
```

Sample Programs

```
        GOTO time_enter
    END IF
END IF

Mouser mx, my, mb
IF mb THEN
    IF mx >= 150 AND my >= 50 AND mx <= 200 AND my <= 100 THEN button down
        DO WHILE mb wait for button release
            Mouser mx, my, mb
        LOOP
        verify mouse still in box area
        IF mx >= 150 AND my >= 50 AND mx <= 200 AND my <= 100 THEN
            time_control = 1
            GOTO proccess
        END IF
    END IF
END IF

WHILE INKEY$ <>
    SLEEP 1
    count = count + 1
    tc = tc + 1
    BEEP
    CLS
    LOCATE 1, 5
    PRINT CLOCK
    PRINT -----

    IF time_control = 1 THEN
        LINE (150, 50)-(200, 100), 2, BF
    END IF
    LOCATE 3, 5
    PRINT CURRENT TIME:; TIME$
    LOCATE 5, 5
    PRINT MINUTES:; minute
    LOCATE 6, 5
    PRINT SECONDS:; count
    IF count = 60 THEN
        count = 0
        minute = minute + 1
    END IF

    IF time_control = 0 THEN
        LOCATE 8, 5
        PRINT TIME LEFT:; (t - tc) \ 60; ;; (t - tc) MOD 60
        IF tc = t THEN
            BEEP
            BEEP
            BEEP
            BEEP
        END IF
    END IF

    IF time_control = 1 THEN
        Mouser mx, my, mb
        IF mb THEN
            IF mx >= 150 AND my >= 50 AND mx <= 200 AND my <= 100 THEN button
down
                DO WHILE mb wait for button release
                    Mouser mx, my, mb
                LOOP
                verify mouse still in box area
                IF mx >= 150 AND my >= 50 AND mx <= 200 AND my <= 100 THEN
                    END
                END IF
            END IF
        END IF
    END IF
END IF
```

```

        END IF
        LOCATE 10, 10
        PRINT PRESS BUTTON TO END
    END IF

WEND
GOTO start

SUB Mouser (x, y, b)
    mi = _MOUSEINPUT
    b = _MOUSEBUTTON(1)
    x = _MOUSEX
    y = _MOUSEY
END SUB

```

This is a little logical combination of all the chapters you have read so far.

18.4 Binary Coder

No, this is NOT a binary decoder, but a binary Coder. This takes any decimal system number and converts it to binary. Run this program to see for yourself.

```

REM binary
SCREEN 7
COLOR 1, 2

_FONT 16
LOCATE 7, 10
PRINT Binary Coder

SLEEP 5

start:

CLS
LOCATE 1, 1
PRINT Binary coder
PRINT -----
PRINT
PRINT
PRINT
PRINT

INPUT Enter Decimal number; a
CLS
LOCATE 1, 1
PRINT Binary coder
PRINT -----
PRINT
PRINT
WHILE a <> 0
    PRINT a MOD 2;
    IF a MOD 2 = 1 THEN
        a = a \ 2
    ELSE a = a / 2
    END IF
WEND
PRINT
PRINT

```

```
PRINT Binary code is reversed  
WHILE INKEY$ <>  
WEND  
GOTO start
```

Just the trouble is: the binary code is reversed. You might have guessed it by looking at the last PRINT statement. I still haven't figured out how to reverse it, so I guess you have to do it yourself. And , the WHILE loop has a print statement with the semicolon at the end. That is used to ensure that the next number comes after it, not on the next line.

19 Appendix

Q Basic uses REM to insert descriptions and comments in source program

19.1 Commands

ABS()

N = ABS(expression returning a numerical value)

Returns the 'absolute' value of the expression, turning a negative to a positive (e.g. -4 to 4)

```
PRINT ABS(54.345) 'This will print the value ABS now as it is (54.345)
```

```
PRINT ABS(-43) 'This will print the value as (43)
```

.

ACCESS

```
OPEN "file.txt" FOR APPEND ACCESS WRITE
```

This sets the access of a file that has been declared into the program. There are three settings that the programmer can set. These are:

READ - Sets up the file to be read only, no writing.

WRITE - Writes only to the file. Cannot be read.

READ WRITE - Sets the file to both of the settings above.

APPEND means 'add' to the 'end of' the existing "file.txt". If omitted, any existing "file.txt" is over-written without warning

ASC()

```
PRINT ASC("t") 'Will print 116
```

Prints the ASCII code number of the character found within the brackets. If the programmer put in a string into brackets, only the first character of the string will be shown.

ATN()

ATN(expression)

Part of the inbuilt trigonometry functions. An expression that evaluates to a numeric value is converted to its Arc-Tangent.

```
angle = ATN( B )
angle2 = ATN( 23 / 34 )
```

BEEP

BEEP

The BIOS on the motherboard is instructed to emit a "Beep" sound from the PC 'speaker'. See also SOUND and PLAY. An older, outdated alternative is to use : PRINT CHR\$(07) This was replaced later by the BEEP command.

CASE

```
SELECT CASE [variable]
CASE [value]: [command]
CASE ELSE: [command]
END SELECT
```

Use this when using multiple values in your program and assigning them separate paths. This is an example of a program with no CASE commands that assigns different paths to values:

```
10 PRINT "1. Print 'path'"
PRINT "2. Print 'hello'"
PRINT "3. Quit"
INPUT "Enter a choice: "; a$
IF a$ = "1" THEN
PRINT "path"
GOTO 10
IF a$ = "2" THEN
PRINT "hello"
GOTO 10
IF a$ = "3" THEN END
PRINT "That is not a valid choice"
GOTO 10
```

This is what a program looks like with the CASE command:

```
PRINT "1. Print 'path'"
PRINT "2. Print 'Hello'"
PRINT "3. Quit"
INPUT "Enter a choice: "; a$
SELECT CASE a$
CASE "1":
PRINT "path"
CASE "2":
PRINT "Hello"
CASE "3":
END
```



```
CASE ELSE:  
PRINT "That is not a valid choice"  
END SELECT
```

CHAIN

```
CHAIN {filename.format (parameters list)}
```

This 'calls' (transfers execution) to another program (typically another .bas file, a command (.bat or .cmd) script or an executable (.com, .exe)). Values may be passed directly in the invoking command (to files accepting command line parameters). Values may be passed to another basic script by using the 'COMMON' statement before the CHAIN command. Control returns to the basic interpreter when the 'chained' program terminates.

CHDIR

```
CHDIR [directory name]
```

This is used for changing a directory. Used in conjunction with MKDIR and RMDIR. The directory name is declared exactly like in DOS PROMPT. For example:

```
CHDIR "c:/Program Files/QBasic_Files/Testing_Files"
```

To use this command, the programmer will have to know how MS DOS and COMMAND PROMPT work.

CHR\$()

This returns the string character symbol of an ASCII code value.

```
name$ = CHR$([ascii character code])
```

Often used to 'load' characters into string variables when that character cannot be typed (e.g. the Esc key or the F{n} (Function Keys) or characters that would be 'recognised' and acted upon by the QBASIC interpreter (e.g. ", the double quote). Here is a list of some character codes :-

```
07 Beep (same as BEEP)  
08 Backspace  
09 Tab  
27 Esc  
34 Double quote  
72 Up Arrow  
75 Left Arrow  
77 Right Arrow  
80 Down Arrow
```

CINT()

This rounds the contents of the brackets to the nearest integer.

```
PRINT CINT(4573.73994596)
```

4574

CIRCLE

```
CIRCLE ([X Coordinate], [Y Coordinate]), [Radius],  
[Colour], [Start], [End], [Aspect]
```

Lets the programmer display a circle. Like all graphics commands, it must be used with the SCREEN command.

CLEAR

```
CLEAR
```

Resets all variables, strings, arrays and closes all files. The reset command on QBasic.

CLOSE

```
CLOSE
```

Closes all open files

```
CLOSE #2
```

Closes only the file opened as data stream 2. Other files remain open

CLS

```
CLS
```

Clears the active screen. Erases all text, graphics, resets the cursor to the upper left (1,1), and also applies the current background color (this has to be set using the COLOR command) to the whole screen.

COLOR

```
COLOR [Text Colour], [Background Colour]
```

This lets you change the colour of the text and background used when next 'printing' to the current output window. It can be done like this:

```
COLOR 14, 01
PRINT "Yellow on Blue"
```

You have a choice of sixteen colours:

00: Black	08: Dark Grey
01: Dark Blue	09: Light Blue
02: Dark Green	10: Light Green
03: Dark Cyan	11: Light Cyan
04: Dark Red	12: Light Red
05: Dark Purple	13: Magenta
06: Orange Brown	14: Yellow
07: Grey	15: White

These values are the numbers that you put in the COLOR command. Note Only screen modes 0, 7, 8, 9, 10 support a background color. To 're-paint' the whole screen in a background colour, use the CLS command.

COMMON

Declares a variable as 'global', which allows its value to be accessed across multiple QBasic programs / scripts (see also the CHAIN command)

```
COMMON SHARED [variablename]
```

Each program that declares 'variablename' as COMMON will share the same value. NOTE. All COMMON statements must appear at the start of the program (i.e. before any executable statements).

CONST

Fixes a variable so it can not be changed within the program.

```
CONST (name) {AS (type = INTEGER / STRING)} = (expression or value)
```

For example :-

```
CONST PI = 3.14159265
```

Assigns the value 3.14159265 to PI.

```
CONST PI2 = 2 * PI
```

PI must be assigned a value before it is used to calculate PI2. Typically all CONST are declared at the beginning of a program.

DATA

```
DATA [constant]
```

Use in conjunction with the READ and RESTORE command. Mostly used in programs dealing with graphics, this command lets QBasic read a large number of constants. The READ command accesses the data while the RESTORE command "refreshes" the data, allowing it to be used again.

DATE\$

A system variable that always contains the current date as a string in mm-dd-yyyy format. Use it like this:

```
a$ = DATE$
```

DEST(Only QB64!)

`_DEST` sets the current write-to page or image. `_DEST image_handle` sends the destination image to an image of handle stored in long variable `image_handle`. `_DEST 0` sends the destination image to the current screen being used.

DIM

This is used to declare an array (early versions of QBasic required all variables to be defined, not just arrays greater than 10)

```
DIM [Array Name] ([count],[count], ..)
```

The Array name can be of any type (Integer, Double, String etc). If not declared, single precision floating point is assumed. Strings can be 'declared' using \$ sign (Integers with the '%' sign). The QBASIC interpreter tolerates numeric arrays of up to 10 count without these needing to be declared. NOTE Early versions of QBasic did not explicitly set the contents of an array to zero (see CLEAR command)

```
DIM table%(100,2)
```

Create an integer array called `table%` containing $100 \times 2 = 200$ entries

```
DIM form$(5)
```

Create a string array called `form$` containing 5 strings

```
DIM quotient(20) AS DOUBLE
```

Create an array called `quotient` that contains 20 double precision numbers

DO .. LOOP

```
DO  
[program]
```

```
LOOP UNTIL [test condition becomes TRUE]
```

Used to create a loop in the program. The [condition] is tested only after the [program] code is executed for the first time (see also WHILE). For example:

```
num$ = 1
sum$ = 0
DO
sum$ = 2 * num$
PRINT sum$
num$ = num$ + 1
LOOP UNTIL num$ = 13
```

This does not work But the following does

```
num = 1
sum = 0
DO
sum = 2 * num
PRINT sum
num = num + 1
LOOP UNTIL num = 13
```

DRAW

```
DRAW "[string expression]"
```

Used to draw a straight line from the current 'cursor' position in the current colour. DRAW defines the direction (up, down etc.) and the length of the line (in pixels). For example:-

```
SCREEN 7
PSET (50, 50), 4
DRAW "u50 r50 d50 l50"
```

The letter in front of each number is the direction:

```
U = Up      E = Upper-right
D = Down    F = Lower-right
L = Left    G = Lower-left
R = Right   H = Upper-left
```

The drawing 'cursor' is left at the position where the line ends. u50 draws from 50,50 upwards ending at 50,0 r50 draws from 50,0 to the right, ending at 100,0 d50 draws from 100,0 downwards, ending at 100,50 l50 draws from 100,50 to the left, ending at 50,50 The example shown will thus draw a red 'wire frame' square. See also LINE and CIRCLE commands. Note: Pixels are square. The diagonal from 0,0 to 100,100 will be $100 * \text{root}(2)$ pixels long (i.e. 141)

END

```
END
```

Signifies the end of the program. When QBasic sees this command it usually comes up with a statement saying: "Press Any Key to Continue".

END IF

```
END IF
```

Ends the program if a condition is reached.

ENVIRON

```
ENVIRON [string expression]
```

NOTE: If you are running QBasic on a Windows system, you will not be able to use this command. This command helps you set an environment variable for the duration of the session. On exit from the QBasic.exe interpreter, the variables revert to their original values.

EOF()

This checks if there are still more data values to be read from the file specified in (). EOF() returns a boolean / binary value, a one or zero. 0 if the end of file has not been reached, 1 if the last value in the file has been read (see also LINE INPUT)

```
OPEN File.txt FOR INPUT AS #1
DO
INPUT #1, text$
PRINT text$
LOOP UNTIL EOF(1)
END
```

Note that, since the INPUT is executed before UNTIL is reached, File.txt must contain at least one line of text - if the file is empty, you will receive an 'ERROR (62) Input past end of file'.

ERASE

```
ERASE [arrayname] [,]
```

Used to erase all dimensioned arrays.

ERROR

System variable holding a numeric value relating to the processing of the previous line of code. If the line completed without error, ERROR is set to 0. If the line failed, ERROR is set to one of the values shown below. Most commonly used to redirect program flow to error handling code as in :-

```
ON ERROR GOTO [line number / label]
```

If ERROR is non=zero, program flow jumps to the line number or label specified. If ERROR is zero, program flow continues with the next line below. To manually test your program and check to see if the error handling routine runs OK, ERROR can be set manually :-

```
ERROR [number]
```

Set ERROR = number The error numbers are as follows:

1 NEXT without FOR	39 CASE ELSE expected
2 Syntax Error	40 Variable required
3 RETURN without GOSUB	50 FIELD overflow
4 Out of DATA	51 Internal error
5 Illegal function call	52 Bad file name or number
6 Overflow	53 File not found
7 Out of memory	54 Bad file mode
8 Label not defined	55 File already open
9 Subscript out of range	56 FIELD statement active
10 Duplicate definition	57 Device I/O error
11 Division by zero	58 File already exists
12 Illegal in direct mode	59 Bad record length
13 Type mismatch	61 Disk full
14 Out of string space	62 Input past end of file
16 String formula too complex	63 Bad record number
17 Cannot continue	64 Bad file name
18 Function not defined	67 Too many files
19 Yes RESUME	68 Device unavailable
20 RESUME without error	69 Communication-buffer overflow
24 Device timeout	70 Permission denied
25 Device Fault	71 Disk not ready
26 FOR without NEXT	72 Disk-media error
27 Out of paper	73 Advanced feature unavailable
29 WHILE without WEND	74 Rename across disks
30 WEND without WHILE	75 Path/File access error
33 Duplicate label	76 Path not found
35 Subprogram not defined	
37 Argument-count mismatch	
38 Array not defined	

Note that ERROR is set when execution fails, not when the code is 'read' - so, for example, a 'divide by 0' will be found before the result is assigned to a non-existent array variable or written to a non-existent file.

EXIT

Allows the immediate exit from a subroutine or a loop, without processing the rest of that subroutine or loop code

```
EXIT DEF
```

Exits from a DEF FN function.

```
EXIT DO
```

Exits from a DO loop, execution continues with the command directly after the LOOP command

EXIT FOR

Exits from a FOR loop, execution continues with the command directly after the NEXT command

EXIT FUNCTION

Exits a FUNCTION procedure, execution continues with the command directly after the function call

EXIT SUB

Exits a SUB procedure.

FOR .. NEXT

```
FOR [variable name] = [start value] TO [end value] {STEP n}  
[program code]  
NEXT [variable name]
```

The variable is set to the [start value], then program code is executed and at the Next statement the variable is incremented by 1 (or by the STEP value, if any is specified). The resulting value is compared to the [end value] and **if not equal** program flow returns to the line following the FOR statement. For example:

```
FOR a = 200 TO 197 STEP-1  
PRINT a  
NEXT a
```

200 199 198 Care must be taken when using STEP, since it is quite possible to step past the (end value) with the result that the FOR loop will run 'for ever' (i.e. until the user aborts the interpreter or an error occurs), for example :-

```
FOR a = 200 TO 197 STEP-2  
PRINT a  
NEXT a
```

200 198 196 194 192 ... 0 -2 -4 ... -32768 ERROR overflow

GOSUB

```
GOSUB [subroutine line number / label]
```

Command processing jumps to the subroutine specified. When the RETURN command is encountered, processing returns to this point and continues with the line below the GOSUB.

IF

```
IF [variable or string] [operator] [variable or string] THEN [command] {ELSE  
[command]}
```


Compares variables or strings. For example, if you wanted to examine whether or not a user-entered password was the correct password, you might enter:

```
IF a$ = "password" THEN PRINT "Password Correct"
```

Where a\$ is the user entered password. Some operators include:

"="- equal to

"<"- less than (only used when variable or string is a number value)

">"- greater than (only used when variable or string is a number value)

"<>"- does not equal

"<=" - less than or equal to (only used when variable or string is a number value)

">=" - greater than or equal to (only used when variable or string is a number value)

One can also preform actions to number values then compare them to other strings or variables using the if command, such as in the below examples:

```
IF a+5 = 15 THEN PRINT "Correct"
```

```
IF a*6 = b*8 THEN PRINT "Correct"
```

INCLUDE (QUICKbasic Only)

QUICKBasic supports the use of include files via the \$INCLUDE directive:

(Note that the Qbasic interpreter does NOT support this command.)

```
'$INCLUDE: 'foobar.bi'
```

Note that the include directive is prefixed with an apostrophe, dollar, and that the name of the file for inclusion is enclosed in single quotation mark symbols.

INKEY\$

```
[variable] = INKEY$
```

This is used when you want a program to function with key input from the keyboard. Look at this example on how this works:

```
a$ = INKEY$
PRINT "Press Esc to Exit"
END IF a$ = CHR$(27)
```

You can use this in conjunction with the CHR\$ command or type the letter (e.g. A).

INPUT

```
INPUT [String Literal] [,or:] [Variable]
```

Displays the String Literal, if a semi colon follows the string literal, a question mark is displayed, and the users input until they hit return is entered into the variable. The variable can be a string or numeric. If a user attempts to enter a string for a numeric variable, the program will ask for the input again. The String Literal is option. If the string literal is used, a comma (,) or semicolon (;) is necessary.

INPUT

```
INPUT #n [String Literal] [,or:] [Variable]
```

Reads a string / value from the specified file stream (see also LINE INPUT #)

```
INPUT #1, a$, b$, n, m
```

Reads 4 values from the file that is OPEN as #1. a\$ is assigned all text until a ',' (comma) or end of line is reached, b\$ the next segment of text, then two numeric values are interpreted and assigned to n and m. Note that, within the file, numbers can be separated by 'anything' - so, if a number is not found (for 'n' or 'm') on the current 'line' of the file, the rest of the file will be searched until a number is found. Input is then left 'pointing' at the position in the file after the last number required to satisfy the input statement (see also 'seek #' command)

INSTR

```
INSTR (start%, Search$, Find$)
```

Returns the character position of the start of the first occurrence of Find\$ within Search\$, starting at character position 'start%' in Search\$. If Find\$ is not found, 0 is returned. start% is optional (default = 1, the first character of Search\$)

```
Pos = INSTR ("abcdefghi", "de")
```

returns 4

LEFT\$()

```
A$ = LEFT$(B$,N)
```

A\$ is set to the N left most characters of B\$.

```
A$ = LEFT$("Get the start only",6)
```

returns "Get th" See also RIGHT\$(), MID\$().

LET

```
LET [variable] = [value]
```

Early versions of the QBasic.exe command interpreter required use of the 'LET' command to assign values to variables. Later versions did not.

```
LET N = 227 / 99
LET A$="a line of simple text"
```

is equivalent to :-

```
N = 227 / 99
A$="a line of simple text"
```

LINE

```
LINE ([X], [Y]) - ([X], [Y]), [Colour Number]
```

Used for drawing lines in QBasic. The first X and Y are used as coordinates for the beginning of the line and the second set are used for coordinating were the end of the line is. You must put a SCREEN command at the beginning of the program to make it work. Note. When in SCREEN 13, the Colour Number == the Palette number

LINE INPUT #

```
LINE INPUT #1, a$
```

Reads a complete line as text characters from the file OPEN as stream #1 and places it in a\$. To find the 'end of line', the QBasic interpreter searches for the 'Carriage Return' + 'Line Feed' (0x0D, 0x0A) characters. When reading text files created on UNIX/LINUX systems (where the 'Line feed' 0x0A is used on it's own to signify 'end of line'), LINE INPUT will not recognise the 'end of line' and will continue to input until the end of file is reached. For files exceeding 2k characters, the result is an "Out of String Space" Error as a\$ 'overflows'. One solution is to use a text editor able to handle UNIX files to open and 'save as' before attempting to process the file using QBasic.

LOADIMAGE (QB64 Only)

(NOTE! The commands in this section refer to a third-party program called "QB64". Neither QUICKbasic nor Qbasic support `_LOADIMAGE`, `_NEWIMAGE`, OR `_PUTIMAGE` commands. Both QUICKbasic and Qbasic have a "SCREEN" command, but it works differently in those languages than in QB64.) `_LOADIMAGE("image.jpg")` Shows an image. Must be used with the commands `SCREEN`, `_NEWIMAGE` and `_PUTIMAGE`. Example: `DIM rabbit AS LONG SCREEN _NEWIMAGE(800, 600, 32) rabbit = _LOADIMAGE("rabbit.jpg") _PUTIMAGE (100,100), rabbit`

LOOP

```
DO
[Program]
LOOP UNTIL [condition]
```

Used to create a loop in the program. This command checks the condition after the loop has started. This is used in conjunction with the DO command.

LPRINT

```
LPRINT [statement or variable]
```

Prints out text to a printer. The LPRINT command expects a printer to be connected to the LPT1(PRN) port. If a printer is not connected to LPT1, QBasic displays a "Device fault" error message. If your printer is connected to a COM port instead, use the MS-DOS MODE command to redirect printing from LPT1 to COMx (for example, to redirect to COM1, use the following command:

```
MODE LPT1=COM1
```

If you need to cancel the redirection when finished, use the following command:

```
MODE LPT1
```

MID\$

```
a$=MID$(string$,start%[,length%])
MID$(string$,start%[,length%])=b$
```

In the first use, a\$ is set to the substring taken from string\$ starting with character start% taking Length% characters. If length% is omitted, the rest of the line (i.e. start% and all the characters to the right) are taken. In the second use, length% characters of string\$ are replaced by b\$ starting at start%. If length% is omitted, the rest of the line is replaced (i.e. start% and all the characters to the right) See also LEFT\$ RIGHT\$ LEN

MOD

```
a MOD b
```

Returns the remainder of an integer divide of a by b For example, 10 MOD 3 returns 1

NEWIMAGE(Only QB64!)

`_NEWIMAGE` is used to set a long variable as the screen dimensions, or can be used with the `SCREEN` command (See later in Appendix) to directly set the screen details. It is very useful as you can enlarge the `SCREEN` mode '13' which has RGB color settings, if you find the default size too small. Syntax: `_NEWIMAGE(width,length,screen_mode)`

- width and length are long variables, while screen_mode is the screen mode format you wish to change.

It is also used to prepare the window screen surface for the image you want to put (first load it using `LOADIMAGE`).

OPEN

```
OPEN "[path]\8.3 file name.ext]" (FOR {INPUT/OUTPUT} AS #{n})
```

This opens a file. You have to give the DOS file name, for example:

```
OPEN "data.txt" FOR INPUT AS #1
```

Opens the existing file data.txt for reading as data stream #1. Since no path is specified, the file must be in the same folder as the QBASIC.exe - if not, processing halts with a 'file not found' error

```
OPEN "C:\TEMP\RUN.LOG" FOR OUTPUT AS #2
```

Opens an empty file named RUN.LOG in the C:\TEMP folder for writing data stream #2. Any existing file of the same name is replaced.

PALETTE

```
PALETTE[palette number, required colour]
```

For VGA (`SCREEN` mode 13) only, sets the Palette entry to a new RGB color. The palette number must be in the range 1-256. The required colour is a LONG integer created from the sum of (required Blue * 65536) + (required Green * 256) + required Red.

RANDOMIZE

```
RANDOMIZE TIMER  
A = INT((RND * 100)) + 1
```

RANDOMIZE will set the seed for QBasic's random number generator. With QBasic, it's standard to simply use RANDOMIZE TIMER to ensure that the sequence remains the same for each run. The example is a mathematical operation to get a random number from 1 to 100. INT stands for Integer, RND for Random and "*" stands for the limit upto which the random number is to be chosen. The "+ 1" is just there to ensure that the number chose is from 1 to 100 and not 0 to 99. Note: Subsequent calls of this function do not guarantee the same sequence of random numbers.

READ

```
READ [Variable]
```

Used in conjunction with the DATA command, this command lets QBasic read data. This is mostly used when dealing with large quantities of data like bitmaps.

REM or '

```
REM {comments}  
' {comments}
```

When the interpreter encounters REM or "'" (a single quote) at the start of a line, the rest of the line is ignored

RETURN

```
RETURN
```

Signifies that it is the end of a subroutines

RND

```
RANDOMIZE TIMER  
A = INT((RND * 100)) + 1
```

RND will provide a random number between 0 and 1. The example is a mathematical operation to get a random number from 1 to 100. RANDOMIZE TIMER will set the initial seed to a unique sequence. INT stands for Integer, RND for Random and "*" stands for the limit upto which the random number is to be chosen. The "+ 1" is just there to ensure that the number chose is from 1 to 100 and not 0 to 99. Internally, the seed a 24-bit number, iterated in the following method: $\text{rnd_seed} = (\text{rnd_seed} * 16598013 + 12820163) \text{ MOD } 2^{24}$

PLAY

```
PLAY "[string expression]"
```

Used to play notes and a score in QBasic on the PC speaker. The tones are indicated by letters A through G. Accidentals are indicated with a "+" or "#" (for sharp) or "-" (for flat) immediately after the note letter. See this example:

```
PLAY "C C# C C#"
```

Whitespaces are ignored inside the string expression. There are also codes that set the duration, octave and tempo. They are all case-insensitive. PLAY executes the commands or notes the order in which they appear in the string. Any indicators that change the properties are effective for the notes following that indicator.

Ln Sets the duration (length) of the notes. The variable n does not indicate an actual duration amount but rather a note type; L1 - whole note, L2 - half note, L4 - quarter note, etc. (L8, L16, L32, L64, ...). By default, n = 4. For triplets and quintets, use L3, L6, L12, ... and L5, L10, L20, ... series respectively. The shorthand notation of length is also provided for a note. For example, "L4 CDE L8 FG L4 AB" can be shortened to "L4 CDE F8G8 AB". F and G play as eighth notes while others play as quarter notes.

On Sets the current octave. Valid values for n are 0 through 6. An octave begins with C and ends with B. Remember that C- is equivalent to B.

< > Changes the current octave respectively down or up one level.

Nn Plays a specified note in the seven-octave range. Valid values are from 0 to 84. (0 is a pause.) Cannot use with sharp and flat. Cannot use with the shorthand notation neither.

MN Stand for Music Normal. Note duration is 7/8ths of the length indicated by Ln. It is the default mode.

ML Stand for Music Legato. Note duration is full length of that indicated by Ln.

MS Stand for Music Staccato. Note duration is 3/4ths of the length indicated by Ln.

Pn Causes a silence (pause) for the length of note indicated (same as Ln).

Tn Sets the number of "L4"s per minute (tempo). Valid values are from 32 to 255. The default value is T120.

. When placed after a note, it causes the duration of the note to be 3/2 of the set duration. This is how to get "dotted" notes. "L4 C#." would play C sharp as a dotted quarter note. It can be used for a pause as well.

MB MF Stand for Music Background and Music Foreground. MB places a maximum of 32 notes in the music buffer and plays them while executing other statements. Works very well for games. MF switches the PLAY mode back to normal. Default is MF.

PRINT

```
PRINT [Argument] [,or;] [Argument]...
```

Displays text to the screen. The Argument can be a string literal, a string variable, a numeric literal or a numeric variable. All arguments are optional.

```
PRINT #[n] [,or;] [Argument]...
```

Saves data to the file that is 'OPEN FOR OUTPUT AS #[n]' or we can use ? symbol for print command

PSET

PSET ([X coordinate],[Y coordinate]), [Pixel Colour]

This command displays pixels, either one at a time or a group of them at once. For the command to work, the program must have a SCREEN command in it.

SCREEN

SCREEN [Screen Mode Number]

This command is used for displaying graphics on the screen. There are ten main types of screen modes that can be used in QBasic depending on the resolution that you want. Here is a list of what screen modes you can choose from: SCREEN 0: Textmode, cannot be used for graphics. This the screen mode that text based programs run on. SCREEN 1: 320 x 200 Resolution. Four Colours SCREEN 2: 640 x 200 Resolution. Two Colours (Black and White) SCREEN 7: 320 x 200 Resolution. Sixteen Colours SCREEN 8: 640 x 200 Resolution. Sixteen Colours SCREEN 9: 640 x 350 Resolution. Sixteen Colours SCREEN 10: 640 x 350 Resolution. Two Colours (Black and White) SCREEN 11: 640 x 480 Resolution. Two Colours SCREEN 12: 640 x 480 Resolution. Sixteen Colours SCREEN 13: 320 x 200 Resolution. 256 Colours. (Recommended) Note. In SCREEN 13 you have a colour Palette of 256 colours. The PALETTE is pre-set by Windows however you can change the RGB values using the PALETTE command.

SEEK

SEEK #[file number], 1

Repositions the 'input #' pointer to the beginning of the file.

SGN

SGN(expression yielding a numeric value)

Yields the 'sign' of a value, -1 if < 0, 0 if 0, 1 if > 0

SHELL

The 'SHELL' command is used in Qbasic to issue a command to Command Prompt/Windows Shell . The 'Shell' command is used along with a string that contains commands that would be understood by any of the above software. The string enclosed commands are much like that of MS-DOS Example: SHELL can be used with a

'DIR' command to make a directory of files in a certain folder or path.

SLEEP

SLEEP [n]

Execution is suspended for n seconds

SOUND

SOUND [frequency], [duration]

Unlike the BEEP command, this produces a sound from the PC speakers that is of a variable frequency and duration. The frequency is measured in Hertz and has a range from 37 to 32767. Put in one of these numbers in the frequency section. The duration is clock ticks that is defaulted at 18.2 ticks per second.

STR\$

Converts a numeric value into a text (string) character

A\$ = STR\$(expression yielding a numeric value)

The numeric value is converted into text characters and placed into A\$. Use to convert numbers into a text string. WARNINGS. 1) If the result is positive, a leading 'space' is added (STR\$(123) = " 123" and not "123" as might be expected). If the result is negative, instead of a space you get a '-' (minus sign), i.e. STR\$(-123) = "-123" and not "-123" as might be expected from the positive behaviour. 2) When converting a float (numb!, numb#) less than 0.1, the string value may be rendered in 'scientific notation', with 'D' used rather than '*10^' (for example "5.nnnnnnD-02" rather than ".05nnnnnn" or "5.nnnnnn*10^-02"). This only occurs when the number of significant digits needs to be preserved (so .03000000 is rendered as ".03", whilst .030000001 becomes "3.0000001D-02"), again perhaps not what you might expect. See also CHR\$ for converting an ascii value into a string character. See also LEFT\$, MID\$, RIGHT\$ for extracting sub-strings from a line of text.

SYSTEM

SYSTEM

The .bas exits, the QBasic.exe interpreter is closed and 'control' passes to the Command Window c:\ prompt (or next line of a calling .cmd script etc.) NOTE!: This only works when you start your program at the command prompt using the "/run" parameter! (EX: "Qbasic /run MyProg.bas") Otherwise, Qbasic assumes you opened your program to make changes, and thus "SYSTEM" drops you back at the editor screen.

THEN

```
[Command] [variable] = [value] THEN GOTO [line command value]
```

Used in conjunction with the GOTO or IF condition commands. It tells the computer what to do if a certain condition has been met.

TO

```
[Command] [Variable] = [Value] TO [Value]
```

Usually used to input a number of variables.

```
FOR a = 400 TO 500
PRINT a
NEXT a
```

This example will print all numbers from 400 to 500. Instead of declaring all values separately, we can get them all declared in one go.

USING

```
USING "format";
```

Used to format the output of data from PRINT commands. Normally, the QBasic interpreter will print a number as 8 characters with as many leading spaces as necessary. To change this behaviour, the USING command can be used to format the output. For example

..

```
IF n > 99 THEN PRINT #1, USING "###"; n; ELSE IF n > 9 AND n<=99
```

```
THEN PRINT #1, USING "0##"; n; ELSE PRINT #1, USING "00#"; n; .. will output
n from 0 to 999 with leading zeros. Note the ';' after the n. This means 'don't start a new
line' and results in the next PRINT #1 adding data directly after the comma (',') Qbasic
automatically inserts instead of a line.
```

VAL()

```
name=VAL([variable$])
```

Converts the [variable string] contents into a numeric value so it can be used in calculations. If (name) is an INTEGER type, the VAL is rounded down. See also STR\$. A\$ = "2" B\$ = "3" X = VAL(A\$) + VAL(B\$) PRINT A\$; " + "; B\$; " ="; X

WHILE ... WEND

```
WHILE {NOT} [test condition is true]
[program code to execute]
```

WEND

The condition is tested and if true (or NOT true) the [program] code is executed until WEND is reached, at which point control passes back to the WHILE line.

```
WHILE NOT (EOF(1))  
LINE INPUT #1, A$  
PRINT #2, A$  
WEND
```

While the end of file #1 has not been reached, read each complete line and write it to file #2. Unlike FOR and DO, it is not possible to EXIT from a WHILE loop

20 Contributors

Edits	User
1	1997kB ¹
1	78.26 ²
1	Acagastya ³
1	Adrignola ⁴
1	Agtishan ⁵
1	Arthur Wright ⁶
2	Atcovi ⁷
5	BNUTTING77 ⁸
5	Bart Harner ⁹
1	BethNaught ¹⁰
1	Bllovely ¹¹
2	Chazz ¹²
4	Chuckhoffmann ¹³
4	Davidshq ¹⁴
44	Debanshu.Das ¹⁵
1	Defender ¹⁶
1	Derbeth ¹⁷
2	Doctormatt ¹⁸
2	Equinox ¹⁹
2	Eumolpo ²⁰

-
- <https://en.wikibooks.org/wiki/User:1997kB>
 - <https://en.wikibooks.org/wiki/User:78.26>
 - <https://en.wikibooks.org/wiki/User:Acagastya>
 - <https://en.wikibooks.org/wiki/User:Adrignola>
 - <https://en.wikibooks.org/w/index.php%3ftitle=User:Agtishan&action=edit&redlink=1>
 - https://en.wikibooks.org/w/index.php%3ftitle=User:Arthur_Wright&action=edit&redlink=1
 - <https://en.wikibooks.org/wiki/User:Atcovi>
 - <https://en.wikibooks.org/w/index.php%3ftitle=User:BNUTTING77&action=edit&redlink=1>
 - https://en.wikibooks.org/w/index.php%3ftitle=User:Bart_Harner&action=edit&redlink=1
 - <https://en.wikibooks.org/wiki/User:BethNaught>
 - <https://en.wikibooks.org/w/index.php%3ftitle=User:Bllovely&action=edit&redlink=1>
 - <https://en.wikibooks.org/wiki/User:Chazz>
 - <https://en.wikibooks.org/wiki/User:Chuckhoffmann>
 - <https://en.wikibooks.org/w/index.php%3ftitle=User:Davidshq&action=edit&redlink=1>
 - <https://en.wikibooks.org/wiki/User:Debanshu.Das>
 - <https://en.wikibooks.org/wiki/User:Defender>
 - <https://en.wikibooks.org/wiki/User:Derbeth>
 - <https://en.wikibooks.org/wiki/User:Doctormatt>
 - <https://en.wikibooks.org/wiki/User:Equinox>
 - <https://en.wikibooks.org/wiki/User:Eumolpo>

- 1 Ezeji Cynthia chinwendu²¹
- 1 Fernando28121²²
- 5 Frazzydee~enwikibooks²³
- 1 Frigotoni²⁴
- 2 Fusdoigbuiagdsghakis²⁵
- 5 Geocachernemesis²⁶
- 3 Glaisher²⁷
- 84 Grich²⁸
- 12 Habst²⁹
- 1 Holdoffhunger³⁰
- 2 ImeNtrav³¹
- 1 Indiachangers³²
- 1 JackBot³³
- 1 JackPotte³⁴
- 20 Jguk³⁵
- 2 Jmathishd436³⁶
- 3 Jomegat³⁷
- 3 Jorgenev³⁸
- 1 Keytotime³⁹
- 1 Leaderboard⁴⁰
- 1 Lincher⁴¹
- 1 Linuxlalala⁴²
- 1 Markhobley⁴³
- 1 Marshman~enwikibooks⁴⁴
- 2 Mild Bill Hiccup⁴⁵

-
- 21 https://en.wikibooks.org/w/index.php?ftitle=User:Ezeji_Cynthia_chinwendu&action=edit&redlink=1
 - 22 <https://en.wikibooks.org/wiki/User:Fernando28121>
 - 23 <https://en.wikibooks.org/wiki/User:Frazzydee~enwikibooks>
 - 24 <https://en.wikibooks.org/wiki/User:Frigotoni>
 - 25 <https://en.wikibooks.org/w/index.php?ftitle=User:Fusdoigbuiagdsghakis&action=edit&redlink=1>
 - 26 <https://en.wikibooks.org/wiki/User:Geocachernemesis>
 - 27 <https://en.wikibooks.org/wiki/User:Glaisher>
 - 28 <https://en.wikibooks.org/wiki/User:Grich>
 - 29 <https://en.wikibooks.org/wiki/User:Habst>
 - 30 <https://en.wikibooks.org/wiki/User:Holdoffhunger>
 - 31 <https://en.wikibooks.org/w/index.php?ftitle=User:ImeNtrav&action=edit&redlink=1>
 - 32 <https://en.wikibooks.org/w/index.php?ftitle=User:Indiachangers&action=edit&redlink=1>
 - 33 <https://en.wikibooks.org/wiki/User:JackBot>
 - 34 <https://en.wikibooks.org/wiki/User:JackPotte>
 - 35 <https://en.wikibooks.org/wiki/User:Jguk>
 - 36 <https://en.wikibooks.org/w/index.php?ftitle=User:Jmathishd436&action=edit&redlink=1>
 - 37 <https://en.wikibooks.org/wiki/User:Jomegat>
 - 38 <https://en.wikibooks.org/wiki/User:Jorgenev>
 - 39 <https://en.wikibooks.org/wiki/User:Keytotime>
 - 40 <https://en.wikibooks.org/wiki/User:Leaderboard>
 - 41 <https://en.wikibooks.org/wiki/User:Lincher>
 - 42 <https://en.wikibooks.org/w/index.php?ftitle=User:Linuxlalala&action=edit&redlink=1>
 - 43 <https://en.wikibooks.org/wiki/User:Markhobley>
 - 44 <https://en.wikibooks.org/wiki/User:Marshman~enwikibooks>
 - 45 https://en.wikibooks.org/w/index.php?ftitle=User:Mild_Bill_Hiccup&action=edit&redlink=1

2 N!663R⁴⁶
 1 NicoScribe⁴⁷
 10 Octahedron80⁴⁸
 1 Panic2k4⁴⁹
 11 Pi zero⁵⁰
 1 Pnksbl~enwikibooks⁵¹
 1 Pontiac Grand Am⁵²
 1 Poponuro⁵³
 5 QuiteUnusual⁵⁴
 3 Recent Runes⁵⁵
 1 Rxy⁵⁶
 3 Scud43⁵⁷
 51 Sigma 7⁵⁸
 2 Someonehello⁵⁹
 2 Splat⁶⁰
 2 Stanglavine⁶¹
 5 Stryn⁶²
 3 Syum90⁶³
 1 Thenub314⁶⁴
 2 Tropicalkitty⁶⁵
 10 Van der Hoorn⁶⁶
 1 Vermont⁶⁷
 1 Wargo⁶⁸
 1 WereSpielChequers⁶⁹
 3 Xania⁷⁰

46 <https://en.wikibooks.org/w/index.php%3ftitle=User:N!663R&action=edit&redlink=1>
 47 <https://en.wikibooks.org/wiki/User:NicoScribe>
 48 <https://en.wikibooks.org/wiki/User:Octahedron80>
 49 <https://en.wikibooks.org/wiki/User:Panic2k4>
 50 https://en.wikibooks.org/wiki/User:Pi_zero
 51 <https://en.wikibooks.org/wiki/User:Pnksbl~enwikibooks>
 52 https://en.wikibooks.org/w/index.php%3ftitle=User:Pontiac_Grand_Am&action=edit&redlink=1
 53 <https://en.wikibooks.org/w/index.php%3ftitle=User:Poponuro&action=edit&redlink=1>
 54 <https://en.wikibooks.org/wiki/User:QuiteUnusual>
 55 https://en.wikibooks.org/wiki/User:Recent_Runes
 56 <https://en.wikibooks.org/wiki/User:Rxy>
 57 <https://en.wikibooks.org/wiki/User:Scud43>
 58 https://en.wikibooks.org/wiki/User:Sigma_7
 59 <https://en.wikibooks.org/w/index.php%3ftitle=User:Someonehello&action=edit&redlink=1>
 60 <https://en.wikibooks.org/wiki/User:Splat>
 61 <https://en.wikibooks.org/wiki/User:Stanglavine>
 62 <https://en.wikibooks.org/wiki/User:Stryn>
 63 <https://en.wikibooks.org/w/index.php%3ftitle=User:Syum90&action=edit&redlink=1>
 64 <https://en.wikibooks.org/wiki/User:Thenub314>
 65 <https://en.wikibooks.org/wiki/User:Tropicalkitty>
 66 https://en.wikibooks.org/wiki/User:Van_der_Hoorn
 67 <https://en.wikibooks.org/wiki/User:Vermont>
 68 <https://en.wikibooks.org/wiki/User:Wargo>
 69 <https://en.wikibooks.org/wiki/User:WereSpielChequers>
 70 <https://en.wikibooks.org/wiki/User:Xania>

4 Xerol⁷¹
1 Zzo38⁷²

⁷¹ <https://en.wikibooks.org/wiki/User:Xerol>

⁷² <https://en.wikibooks.org/w/index.php%3ftitle=User:Zzo38&action=edit&redlink=1>

List of Figures

- GFDL: Gnu Free Documentation License. <http://www.gnu.org/licenses/fdl.html>
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. <http://creativecommons.org/licenses/by-sa/3.0/>
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. <http://creativecommons.org/licenses/by-sa/2.5/>
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. <http://creativecommons.org/licenses/by-sa/2.0/>
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. <http://creativecommons.org/licenses/by-sa/1.0/>
- cc-by-2.0: Creative Commons Attribution 2.0 License. <http://creativecommons.org/licenses/by/2.0/>
- cc-by-2.0: Creative Commons Attribution 2.0 License. <http://creativecommons.org/licenses/by/2.0/deed.en>
- cc-by-2.5: Creative Commons Attribution 2.5 License. <http://creativecommons.org/licenses/by/2.5/deed.en>
- cc-by-3.0: Creative Commons Attribution 3.0 License. <http://creativecommons.org/licenses/by/3.0/deed.en>
- GPL: GNU General Public License. <http://www.gnu.org/licenses/gpl-2.0.txt>
- LGPL: GNU Lesser General Public License. <http://www.gnu.org/licenses/lgpl.html>
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. <http://artlibre.org/licence/lal/de>
- CFR: Copyright free use.

- EPL: Eclipse Public License. <http://www.eclipse.org/org/documents/epl-v10.php>

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses⁷³. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrowser.

⁷³ Chapter 21 on page 97

1	The people from the Tango! project ⁷⁴	
2	Debanshu.Das ⁷⁵ , Debanshu.Das ⁷⁶	
3	Debanshu.Das ⁷⁷ , Debanshu.Das ⁷⁸	

74 http://tango.freedesktop.org/The_People

75 [http://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&redlink=](http://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&redlink=1)

76 [https://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&](https://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&redlink=1)
[redlink=1](https://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&redlink=1)

77 [http://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&redlink=](http://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&redlink=1)

78 [https://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&](https://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&redlink=1)
[redlink=1](https://commons.wikimedia.org/w/index.php?title=User:Debanshu.Das&action=edit&redlink=1)

21.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

* b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

* b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

* b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

* c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

* d) Do one of the following:

- o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
- o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

* b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.