Debugging.

QuickCalc BASIC provides several tools to help you figure out why a program is crashing, looping, or otherwise not running as you expected.

Canceling a program which appears to be in a loop.

Press the **Kill Program** button to end the program. Do not exit QuickCalc with a BASIC program still running. In CONSOLE mode, typing **control-c** will exit the program and terminate QuickCalc.

Normally, the **Kill Program** button stops the program at the conclusion of the current BASIC statement. For LONGMATH operations with very long floating-point lengths, the program may appear to be in a loop but actually be doing a very long calculation. Operations at 1 million digits could take hours. These operations are interruptible with the **Kill Program** button, which will end the program after approximately 0.1 second without completing the current operation.

Stepping through the Program.

For a long-running program, you can **break into the program**, pausing it, by clicking on the **Step** button. (GUI mode only). This places the program into "stepping" mode. Subsequent clicks of the **Step** button will execute one statement at a time, showing the results of assignment statements. When you are done stepping through the program you may let it run by clicking the **Continue** button, or kill it by clicking the **Kill Program** button

To start a program in stepping mode, select the program in the Source Program window, then click **Run/Step**. The program will execute the first statement, then enter stepping mode. In CONSOLE mode, type **DEBUG STEP** or **STOP** before running the program.

You may also start stepping at a specified location in the program by using the statement **STOP** within the program. The program will begin stepping from that point until it ends or you click **Continue** or type **CONT**.

One way of identifying where a program is looping is to run it until it appears to be in a loop, and then interrupt it with the **Step** button. QuickCalc will tell you where your program stopped by showing you the hexadecimal "offset" within the program. If you type **LIST**, you can see where you are in the program by looking for that offset. You can also step through a few statements to identify the program location (statement numbers help here), and determine why the program is not exiting the loop.

Note that the **Step** button interrupts a program after the current BASIC statement. If you are in the process of executing a LONGMATH function (like **ATN**, **LONGPI**, etc.), with a very long floating-point length, it may still take some time before the program interrupts.

Often it is easier to debug a program using shorter length numbers, and then increase the length to get your final results.

Note: If you are in stepping mode and you encounter a **CHAIN** statement, the next statement executed will be the first statement in the chained-to program.

Entering commands while paused

The command line window is captioned "Stepping – Command" when the program is in stepping mode.

When the program is paused ("stepping" mode"), you can enter commands and statements from the command line window (end each line with <enter>).

Simply pressing <enter> is the same as clicking on **Step**.

Typing the **CONT** command is the same as clicking the **Continue** button.

Typing **END** is the same as clicking on the **Kill Program** button.

Typing **STOP** is the same as clicking on the **Step** button, except that you can't do it while the program is running. Do it before the program starts (CONSOLE mode) , or start the program with the **Run**/**Step** button.

The **STOP** statement is the same as **DEBUG STEP**.

Some commands (such as **RUN**) are disabled in this mode. In order to execute these commands, you must first run the current program to completion, or kill it.

You can change the flow of execution by setting variables to new values, or examine variables with the **PRINT** or **DEBUGPRINT** statement. You can run pretty much any BASIC command or statement, including compound statements, but be careful about branching into or out of loops.

You cannot change the actual program while it is running. You need to end it and edit the source file and then restart it.

Entering Data while Stepping

If you are in stepping mode and encounter an **INPUT** or **LINE INPUT** statement, the command line will turn pink and you will be prompted to enter a line of data. The caption on the command line will say "Stepping – Data:". Clicking **Step** has no effect at this time. You must type valid data and press ENTER in order to continue. After entering the data you will still be in stepping mode, ready for the next command.

If you are not stepping and the program asks for data, the caption will say "Data:". If you want to enter stepping mode at this time, you may click the **Step** button and the program will enter stepping mode and the caption will change to "Stepping – Data:". You must still enter data to continue. Stepping will begin as soon as the data is accepted.

Note: If you enter invalid data, the program will continue to ask for data until you type it correctly. Then it will continue stepping.

Clicking **Continue** while you are in data mode will exit stepping mode and resume running the program (after you have entered the data).

Debugging Tools.

It helps to have the Notepad window open with your BASIC program displayed as you are debugging.

While you are in stepping mode, you may print the value of any variable(s) using the **PRINT** (or **?**) statement, or the **DEBUGPRINT** (or **??**) statement.

The **PRINTALL VARIABLES** statement is a quick way to print all the currently-assigned variables.

You may change the value of any variable with an assignment statement. This can force the program to exit a loop. Do **not** **GOTO** a statement outside the loop.

Placing a **STOP** statement in a program will put you into "stepping" mode when the **STOP** statement is encountered. You can type **CONT** or click the **Continue** button to resume normal running.

You can place **PRINT** or **DEBUGPRINT** statements within your program to help identify which routine you are in, or watch the value of certain variables. Examples:

```
PRINT "Now entering subroutine at 10000"
IF i > 50000 THEN PRINT "i > 50000"
IF a < 0 THEN PRINT "Warning: a is negative"
```

Tracing.

You may run the program in "Tracing mode", which prints out the statements as they are executed along with the results of assignment statements and conditional tests, such as **WHILE, FOR**, and **IF** statements. The same information is printed as when you are running in "Stepping" mode. The difference is that the program "free-runs" – you don't have to click **Step** or type <enter> after each statement.

In GUI mode, you can still pause the program while you are tracing, using the **Step** button, and resume it using the **Continue** button. In CONSOLE mode, you don't have the

STEP button, however you can click on the scroll bar and the program will pause until you release the mouse button.

Tracing mode is turned on using the **DEBUG TRACE** command, and turned off using the **DEBUG TRACE, OFF** command. These commands may be placed in your program to trace only the suspect part of the program, or typed into the command line while the program is paused (in "Stepping mode"). If you want to trace from the beginning of the program, start the program in stepping mode, enter the **DEBUG TRACE** command, and then **Continue** (or **CONT**). You could also place the **DEBUG TRACE** command as the first statement in your program.

Note: Tracing generates a lot of console output. In GUI mode, everything is written to the log file, however, in CONSOLE mode, the output scrolls off the screen buffer after a while, and for that reason, you may have to take steps to limit the amount of tracing so that you don't lose important results.

Detail Debugging.

Timing

QuickCalc BASIC provides a timer for your use. You start it with the **STARTTIMER** function, and end it with the **ENDTIMER** function, which returns the interval in microseconds.

Timing includes the time for interpreting the BASIC statements and all system overhead that occurs between the **STARTTIMER** and **ENDTIMER** function calls. For that reason, it is not an accurate indication of the time required to do individual mathematical functions (see below for **DEBUG TIMER**). It can, however, point up the differences in speed between two different coding techniques, and help to show which portions of your program are consuming the most time.

If you place a **STARTTIMER** function call inside a section of code which is being timed, the second **STARTTIMER** will override and reset the timer, thus invalidating the outer timing measurement. You can, however, use **ENDTIMER** more than once to measure the time from a common start point to several different end points. Each one will reference back to the **STARTTIMER** function call. Keep in mind that each call to **ENDTIMER** uses up some time itself, which can throw off the total.

Example:

```
STARTTIMER
…. (lines of code to be timed)
microsecs = ENDTIMER
PRINT "The code took " ; microsecs ; " micro-seconds to process."
```

DEBUG TIMER:

This statement allows you to get timing for various LONGMATH functions. It times the actual mathematical function, not counting overhead for BASIC statement interpretation.

The format is: **DEBUG TIMER,** *function* [**,OFF**]
where *function* is one of the following:

ATN (also includes **ACOT**)
SQR
LONGPI
LONGE
SIN (also includes **COS, TAN** and **COT**)
EXP (also includes **EXP10**)
LOG (also includes **LOG10**)
ROOT
POWER (i.e., *x* **^** *y*)
MULT
DIV
FACTORIAL
MOD (also includes '**%**')
HALF

Example: **DEBUG TIMER, ATN** turns on timing for **ATN** and **ACOT** functions.

The optional [**,OFF**] turns off timing for that function.

When a debug timer function is turned on, the message
"*function* calc. time: nnnn.nnn ms."
appears following each execution of the specified function(s).

Note: Some functions call other functions, so you may get more than one message if more than one timer function is turned on. For example, **ROOT** calls **LOG** and **EXP**. This can give misleading results, since the outer function time will include the time to format and print the timer message for the inner function. It is recommended that you only turn on one timer function at a time.

Note: The times for successive calls to a function may differ due to the fact that some functions require calculating π (*e.g.*, to convert from degrees to radians) or **LOG**(10) (to convert between common and natural logarithms). The first time these constants are required, they are calculated and saved. Subsequent requests for the same or fewer digits will not require recalculation and therefore the calling functions will execute faster.