The Turtle System

Guide for Power Users

1. About this Guide

For users who are interested in learning more about machine code, and in how computers work "under the bonnet", the *Turtle System* provides a number of advanced features. This *Guide for Power Users* is intended for those who want to learn more about these features. For those who simply want to learn how to program, the accompanying *Guide for Normal Users* (and the associated documentation on the Turtle programming languages) is all you will need.

To gain access to these advanced features, you first need to switch to Power User mode, by clicking on "Display Power User Menu" under the "View" menu. You will notice that there are now many more items available in the menus. This guide explains these additional items.

2. Technical Overview: the *Turtle Machine* and its PCode

Before describing the particular features, it is necessary to give a brief overview of the *Turtle System* and its underlying *Turtle Machine*. As explained in the *Guide for Normal Users*, when you click on "RUN" your program code is first "compiled" (translated) into a special machine code, which is a set of instructions for a virtual machine (a software representation of some theoretically possible hardware). This machine code—called "PCode"—is a sequence of integers, some of which correspond to the integers used in your program, but most of which represent specific commands that the machine can execute. In an *actual* computer, compiled machine code is ultimately just a sequence of binary digits, but to make it easy for students to learn the basic principles of compilation and machine code execution, the *Turtle System* uses four-byte integers.

The *Turtle Machine* has three main "blocks" of memory. The first two, the memory stack and the memory heap, are in fact components of the same larger memory space, with the heap lying directly above the stack. By default—though as you will see below this can be customised—the memory stack runs from address 0 to address 9999, with the heap then starting at address 10000. In theory the memory heap has no upper limit, and certainly not one you are ever likely to reach. In practice, however, it is limited by the physical memory of your computer.

The memory stack and memory heap are for storing the variables of your program. Global variables are stored at the bottom of the stack, followed by local variables for any subroutines. Note that if you call a subroutine from within itself (a programming technique known as "recursion"), this creates a new instance of that subroutine, with its own set of local variables on the stack. For an example of recursion, see the "Recursive triangles" example under "Examples 2" in the "Help" menu. Global or local *string* variables, however, are stored as "pointers" to an address on the memory heap (the address where that string starts), with the string itself residing in the heap. This distinction is necessary because strings can be of any length, and so it is impossible to know in advance how much memory space to set aside for them. The memory heap also stores any *temporary* strings, i.e. strings that are used in your program without being assigned to any variable.

The third main "block" of memory is the evaluation stack (not to be confused with the *memory* stack discussed above). This is an area of memory of variable size that the *Machine* uses as a sort of notebook for doing any calculations. For example, to work out the value of 7 + 12, the *Machine* first puts the number 7 on the evaluation stack, then puts the number 12 "on top" of it, and then executes the "PLUS" PCode, which takes the top two values off the stack, adds them together, and puts the result back on the stack in their

place. The top of the stack now contains the answer (19), which can be stored in the main memory, or used in further immediate calculations.

This is just a very small flavour of the sorts of things to be learned about the *Turtle Machine* and the *Turtle System*'s compilers, but it should be enough to make sense of the rest of this guide. This guide is not intended to explain the workings of the *Machine* and the compilers in detail, but rather to explain the additional features of the *System* available in Power User Mode, most of which relate to them. The *Machine* and the compilers are explained in detail in Peter Millican's 2004 masters research thesis, available from the documentation page of the Turtle web site (www.turtle.ox.ac.uk/documentation), and in due course we will add more documentation and teaching resources designed for teachers and advanced students who want to learn more about the principles of Computer Science that these things illustrate.

3. The File Menu

When in Power User Mode, some further printing features are available in the "File" menu. In addition to printing the program text, you can also print the text from the console and the textual output area. There is also an option to "Show printing buttons within tables". With this option selected, a print button appears within all the tables, allowing you to print their contents. With the option to "Redirect printing of tables to CSV file" selected, furthermore, you can instead save the contents of tables to a CSV file (comma separated values).

There is a table of native commands under the "QuickHelp 2" tab. More tables are available in Power User Mode, under the extra tabs; see the section on the "Tabs" menu below.

4. The View Menu

In addition to the standard options for attaching or detaching the canvas, when in Power User Mode you can also attach the canvas and place the *Turtle System* window in the middle of the screen (i.e. return to the initial setup), and have a large detached canvas. You can also set custom fonts for the console and the textual output box.

There are two further "View" menu options in Power User Mode, one for hiding the language menu, and one for hiding *and disabling* the power user options (to merely hide the options and return to Normal User Mode, deselect the "Display Power User Menu" option). The facility to disable power user options altogether is intended for teachers, and is explained fully in the *Guide for Teachers*. This action is interactively irreversible (except by exiting the *System* and starting it up again), and is not recommended unless you know what you are doing!

5. The Tabs Menu

In Power User Mode there is a new "Tabs" menu, which allows you to show or hide any of the tabs on the right hand side of the *System* window. More importantly, there are also several additional tabs available, which are all made visible by default when you enter Power User Mode for the first time: "Usage", "Comments", "Syntax", "Vars/Subs", "PCode", and "Memory". The first two are intended for teachers, and are also discussed in the *Guide for Teachers*. The "Comments" tab strips out and displays any comments in the program, while the "Usage" tab shows a table of all the commands and command structures used in the program, helpfully categorised with totals and line numbers indicating where they appear.

The "Syntax" and "Vars/Subs" tabs show data generated during program compilation (the syntax tree created by the parser, and tables of all the variables and subroutines respectively). The "PCode" tab shows the result of the compilation (the compiled PCode that is executed by the *Turtle Machine*). By default this is displayed in assembly format, but pure machine code can optionally be shown instead, either in decimal or

hexadecimal. Furthermore, by selecting "Trace Display" to make the trace display visible, and then selecting "Trace on Run", you can see here a record of every machine command executed by your program when it runs, and the state of machine at each cycle. The trace facility can also be turned on and off within your program itself, using the `trace` command.

The last three columns of the trace display show, by default, the top three values on the evaluation stack. In place of the third value, however, you can optionally display instead the height of the stack, or the value of a specfic address location in the main memory. To choose between these options, click on "Run Options (screen updating / trace display / memory)" in the "Run" menu (some further settings are available here that are described below). To set the address location of the main memory that is shown, use the `watch` command within your program itself. This can be very useful for debugging.

The "Memory" tab, finally, shows the main memory of the *Turtle Machine* (the memory stack and heap), alongside the table of variables also shown in the "Vars/Subs" tab. If you pause your program at any point during execution, switch to this tab and click the "Show Current State" button, you can inspect the memory at that point, checking the values of all of the variables used in your program. The `dump` command can also be used in your program itself, which will dump the current memory state into this table, for inspection at that particular point.

6. The Compile Menu

Also new in Power User Mode is the "Compile" menu, which allows you to compile your program without running it, so that you can check for errors in your code, or inspect the compiled PCode (in the "PCode" tab). (When you click "RUN", your program is automatically compiled before being executed.) There are also three compilation options that you can turn on or off. The first is whether to include some PCode at the start of your program to set up the default key buffer of size 32. This is equalivalent to the explicit command `keybuffer(32)`. This option is enabled by default.

The second option is whether to treat the Turtle attributes as global variables. The Turtle is a global array of five integers (one for each of the attributes, `turtx`, `turty`, `turtd`, `turtt`, and `turtc`), and with the command `newturtle` it is possible to assign your own array to this variable. (See the "Five turtles moving to the mouse" example under "Examples 3" in the "Help" menu for an illustration of this. Note that this feature is currently only available in *Turtle Pascal*, since the other languages do not yet have support for arrays.) For simplicity, however, and by default, the compiler ignores the array pointer, and treats the five attributes as individual global integer variables. The option is automatically disabled if you use the `newturtle` command in your program (otherwise that command would not work), but you can also forcibly disable it here.

Finally, there is an option to initialize local variables to zero/false. Enabling this option adds code to your compiled program that sets any local variables in your procedures or functions to zero/false at the start of each call of that subroutine. With the option disabled (as it is by default), it is wise to include explicit initial assignments for local variables in your program code. Otherwise you may find your program has unexpected results, since previous subroutine calls may have left values in the same memory space.

7. The Run Menu

Several additional runtime options are available in Power User Mode, under the "Run" menu. Clicking on "Run options (screen updating / trace display / memory)" brings up a dialogue box with some of these options. The first concerns when (if ever) to force the screen to update during your program execution. By default, a forced update will occur every 3,000,000 cycles. The reason for this is that a program that runs through a large number of PCodes without visibly doing anything on the canvas can cause the *System* to

"hang", as it eats up more and more resources on your computer. The default is sufficiently high that this is very unlikely ever to be an issue, as most ordinary programs will update the canvas anyway long before the limit is reached.

The second option concerns the trace display, and was already explained in section 5 above. The third option was also hinted at above, in section 2, and concerns the size of the memory stack. By default, the memory heap (on top of the stack) starts at address 10,000. Here you can make the stack smaller or larger. A larger stack will allow more room for global variables and local variables for subroutines, which may be necessary if you use a lot of recursion, especially with a subroutine that requires a lot of memory space.

There are three more runtime options available directly within the "Run" menu itself (not in the "Run options" dialogue box). The are all on/off options, enabled by default, but which can be disabled if you wish. The first concerns the automatic deletion of temporary strings on the heap (i.e. strings that are not assigned to any variables). If your program uses a lot of temporary strings, this memory can quickly build up if it is not kept in check, often storing strings when are no longer needed by the program and therefore could harmlessly be deleted. To combat this, the compiler adds the HCLR ("heap clear") PCode following some string commands, which instructs the *Turtle Machine* to reclaim the memory space containing those strings. With the "Auto-delete temporary Heap strings (& HCLR)" option disabled, the *Machine* will ignore this instruction. This may help in the correct running of some programs that require a lot of complicated operations on temporary strings. It can however cause the memory heap to fill up very fast. Note that this is not the same as—though it is related to—the `heapreset` command, which resets the memory heap completely to reclaim memory used by all local string variables and temporary strings (retaining only the values of any global strings, including the key buffer).

The second option concerns the collision of the memory stack with the memory heap. As explained in section 2, the memory heap lies on top of the stack, by default starting at address 10,000. If your subroutine calls eat up too much memory, they could potentially "overflow" into the heap. Generally this is not something you want, since this will overwrite your heap data. Also it typically arises because of an error in your program, most likely an unterminated recursion. By default, therefore, the *Machine* will force your program to halt, and display an error message, if this overflow occurs. You can however turn off this behaviour by deselecting the "Prevent memory stack collision with Heap" option. Advanced students might want to do this in order to play around with "hacking" the *System*: by *deliberately* overwriting heap data that is not meant to be overwritten that way, you can create some interesting results!

The third and final option is similarly intended for advanced students who want to explore how to "hack" the *System*. (Teachers should note that, because this is all done within a sandboxed virtual machine, it is impossible in this way to do any damage to the actual computer the *Turtle System* is running on.) An array of five integers, for example, is stored in six adjacent memory slots, the first of which contains the length of the array (in this case the value 5), and the other five of which contain the array values themselves. (In fact an array of five integers takes up *seven* slots in memory, the other one being a pointer to the length value.) If, in your program code, you ask for the *sixth* element in this array, which of course does not exist, then the *System* by default will stop the program execution at this point, and show an error message. By deselecting the "Range-check all references to array elements" option, however, you can have the *Machine* ignore this check, and instead return what would be the sixth element of the array, namely the value of the memory slot immediately following the array. In this way, you can gain access to parts of the memory that you would not normally be allowed to access in this way.

8. The Options Menu

Under the "Options" menu, three additional options are available to Auto-Compile, Auto-Run, and Auto-Format on opening a new file. These are primarily intended for teachers, and so are discussed in the *Guide for Teachers*.

There is also now an option to "Auto-save PCode on compiling", which will save the compiled PCode every time you compile your program (opening a file dialogue prompt the first time, allowing you to choose where to save the compiled code).

Finally, there is an option to "Delete default file (i.e. start with built-in defaults". If you have previously created a Default.tgo file in the same directory as the *Turtle System* executable, this options simply deletes that file. This has exactly the same effect as deleting it yourself in the ordinary way.