A series of articles from Sinclair QL World, August 1987, September 1987, June 1988, February 1989 and December 1990, © Simon N. Goodwin

Bugging The ROM

Compiler Simon Goodwin summarises the bugs, features and international variations which lurk inside each QL version and explains how you can upgrade your ROM.

Sinclair produced at least seven versions of the QL ROM the collection of built-in routines which look after Basic and machine code programs. Five of the ROM versions are still considered current, yet they vary enormously in functions and reliability. To check the version of your QL, turn it on without a tape in drive 1, press F1 or F2, and type PRINT VER$. A two-or three-letter code will appear at the top of the screen.

This article compares the features or, to be more honest, the bugs in current OL ROMs and explains how all the versions came into existence in the first place. Next month I will list the universal bugs in every QL ROM regardless of vintage and explain how to get around them.

This litany may make the QL seem rather a disaster but that is not really fair. The first QLs were unfinished and bug- ridden as a consequence but later versions are no worse than the average Amiga, ST or PC. All complex systems contain bugs but hardware manufacturers are curiously shy about admitting and correcting. In fact, bugs are rarely a problem if you know about them and can avoid them. This information is based on my experience, the Sinclair bug list and reports from QL users. If you have found any I have missed, please send details.

Team effort

In 1983 the QL ROM was planned as a team effort. The operating system was to be written by the Cambridge programming house GST, while Sinclair staff contributed code to handle devices and the new Basic interpreter. SuperBasic was designed originally in 1982 for the SuperSpectrum, one of the many Sinclair designs which never passed the drawing board stage.
On January 12, 1984, long before the QL hardware and software were finished, Sinclair launched the QL in London. The prototype machines at the launch used the GST 68K/OS operating system but Sinclair never shipped a QL in that form.
The original plan was to squeeze SuperBasic and the QL operating system into 32K of ROM. At first only a fraction of SuperBasic was to be built into the QL —just enough commands to load and run the Psion Packages. Most of the language was to be loaded as required from Microdrive; almost all the standard SuperBasic commands and functions are still implemented as extensions, although they are built-in to current ROMs.

Sinclair abandoned the GST operating system because it was slow and greedy for memory; it did not leave sufficient space for SuperBasic in ROM or the Psion packages in RAM. GST released 68K/OS independently as a 32K plug-in option at the end of 1984.

The first production QLs contained two-thirds of a stop-gap operating system, Qdos, written by Tony Tebby, a Sinclair engineer who had originally been hired to work on satellite TV hardware.

The other one-third of Qdos and SuperBasic was supplied in an ugly plug-in cartridge — ‘the kludge’. It had been impossible to squeeze the required code into 32K so Sinclair put an extra 16K outside the computer. All the software was in expensive, individually- programmed EPROM chips, rather than mass-produced ROMs.

The version number of Qdos jumped suddenly from 0.08 to 1 .00 when the first production QLs appeared but that was more brave than honest; the code was still being developed in April, 1984 as those machines slipped out. There were amazing bugs. You could not edit a 'bad line'. PRINT -2 - 2 gave zero. Basic programs could not exceed 32K. There were error-trapping keywords but no code to handle them.

Special tokens to allow array initialisation survive from that time but they have never worked; they are left-overs from the initial SuperBasic designs. READ, DATA, GOSUB and RESTORE had been added at a late stage, to make the original, elegant design more standard. The last-minute changes provoked a flood of problems.

Sinclair gave each ROM version a two-letter code. First was Qdos 1.00, the largely-untested "FB" version. The next major version, "PM", was faster and more tolerant of the Microdrives but it was still laced with faults. At that stage new versions, such as "EL" and "TB", were popping up inside Sinclair every week.

In all 13,000 kludged QLs were produced but in June, 1984 the so-called final QL ROM emerged, the "AH" version. By that time Sinclair had stopped naming ROMs after taxi-drivers and started picking on women in the office — "AH" stood for "Angela's Holiday".

The "AH" ROM was really three 16K EPROM chips. The plug-in kludge was avoided by soldering two chips piggyback in one socket. The chips contained Qdos 1.02, the first usable version of the QL built-in software. It was about 20 percent faster than "FB"; since then, code speed has changed very little. The 'AH" ROM and the mass-produced follow-up "JM" were supplied as a free upgrade to those with kludges.

Exceptional bugs

"AH" and "JM" were very similar. Only four bugs were exceptional to "AH". None was serious at the time, although two are worth bearing in mind if you have an expanded QL system.

If two tasks tried to read a file simultaneously, the second would miss the beginning and read the directory header instead. At the time that was fairly academic as there were no multi-tasking programs on the market. Floating point arrays were limited to 384K in size but memory expansion was not available in those days.

The other two bugs fixed between AH" and "JM" were trivial X="." Set X to zero under the "AH" ROM, instead of giving an error, and you could type-in integer FOR loops, although they would not work unless you changed the variable name to a floating point type. The correction for this bug was scarcely an improvement; rather than make integer loops work properly, 'JM" and later ROMs will not let you type them in at all.

The "JM" software was the first to fit into two chips, using the space allocated originally on the QL circuit board. The first socket was intended for a 16K chip but on "JM" systems it held a 32K component. The second socket held the remaining code in a partly-used 16K chip.

Upgrades from "AH" to "JM" or later ROMs are not a simple matter of swapping components. Unfortunately the control signals required by three EPROMs are not the same as those needed by two ROMs. I have converted my QL from EPROM to ROM; it is fiddly but not too difficult if you are used to messing around inside computers. If not, you should get a QL repair firm to do the job for you — it is easy to write off a QL by damaging the circuit board.

Turn off the power, then unplug the EPROMs, which are behind the CTRL connectors in sockets labelled lC33 and 1C34. Disconnect the trailing wire to the top EPROM; it is needed only to provide an extra signal for the third chip. Then plug the 32K ROM — marked '0000' after the version number — into the slot for 1C33; the other ROM, marked '8000', goes into the other socket.

There are six positions for wire links to the right of 1034, labelled JUl to 6. The first two are connected for EPROMs; to use ROMs you must cut link 1 and connect links 3 and 4. Finally, unplug the chip labelled SN74LSOON immediately to the right of the links; it is needed only byEPROMs

The upgrade from "AH" to "JM" is fiddly and does not fix many bugs. The next version of Qdos is a more popular upgrade, although it creates almost as many bugs as it cures. Early in 1985, Sinclair began shipping Qdos version 1.10, the "JS" ROM. At first it was claimed to be a development version, not intended for release. That may well have been the case but tens of thousands of ROMs were made. The "JS" ROM is the last version used in machines made for sale in Britain.

The "JS" ROM killed several annoying bugs of previous versions. It was the first to let you INPUT strings of more than 128 characters from Basic; it also handles CALL correctly in programs of more than 32K which usually crashed a machine running earlier versions.

The "JS" ROM can change the display mode without setting the ink and paper in SCR windows to black and lets you define new procedures and functions with names you have used previously in the same program.

Machine code programmers will be pleased to find that the bugs in the number-base conversion routines vectors 260 to 270 have been fixed. Task handling is more friendly; you do not have to type Control C to retrieve the SuperBasic cursor when a task stops.

The "JS" ROM is the first version which can link more than one plug-in device into the system when you turn it on, QL devices can use a 256K area divided into 16 slots for ROM and port addressing. Those slots are used by disc controllers, sound boards, modems, and so on. Previous ROMs linked only one device into the first slot, however many were connected.

This bug was not serious. Most users have only one such device, a disc; expansion RAM does not require a slot. Many peripheral designers avoid the problem by putting a routine to link other devices in their own start-up code, so that the one gadget which is called can look up the others. Such a routine appears in chapter 9 of Andy Pennell's Sinclair Qdos Companion.

The other changes in the "JS" ROM are less helpful. The revised Basic prevents you entering integer and string SELect statements, which did not work anyway unless you had a compiler. Even the ROM version function, VER$, was a problem. In the process of changing an "M" into an "S", Sinclair stopped VER$ allocating memory for the value it returns. The machine may crash if you try to test VER$ without copying it to a temporary variable first. T$=VER$: IF "JS"=T$ will work will but IF "JS"=VER$ stops the "JS" ROM in its tracks.

Parameters and SELect fell out in the "JS" ROM. It will not let you use a procedure parameter as a SELect variable unless it is the last one in the DEFinition. You must copy the value to another variable to avoid a 'bad name' report.

From the start, QL ROMs contained WHEN keywords to trap errors and monitor variable values. At first they did nothing at all; on ROMs from "JS" onwards they sometimes work and sometimes they just crash the machine.

Sinclair has been understandably reluctant to explain how WHEN trapping should work, as it never produced a QL ROM which can do it properly. Apparently it persuaded Jan Jones, author of the interpreter and The Definitive Super- Basic Handbook, to omit a chapter on WHEN handling from her otherwise- definitive tome.

The idea is to put a WHEN ERRor statement somewhere in your program, followed by program lines to be executed in the case of an error, and rounded off with an END WHEN statement. The computer keeps track of the most recent WHEN ERRor block and jumps into it if an error occurs, without printing the usual cryptic message or stopping the program.

You check the line and type of the error by reading the values of new functions ERLIN and ERNUM. ERNUM returns internal error codes between -1 and -21. Other functions let you check for a given error, without knowing the internal code; ERR_NC is true if the error was 'not complete', ERR_BN indicates bad name' and so forth. Unfortunately someone typed a BRA where they should have put a BSR, so any attempt to check ERR_DF, 'drive full', crashes the "JS" ROM.

The new REPORT procedure prints standard error messages. If you already use that name, or any of the other new ones, in your Basic, you will have to change it. REPORT on its own indicates the last error with a message to channel 0. Codes from -21 to -27 call up other text; REPORT -24 gives 'F1 ..Monitor F2..TV', for example.

No check

There is no check on the number you supply but only 27 messages are in the standard format, so beware. REPORT - 28 and its brethren print a very long string of gibberish. REPORT 1,-19 prints 'not implemented' to channel 1; -19 is the polite code a routine should use to indicate that it does not work yet.

The code for WHEN ERRor is not usually that kind. Errors in functions often crash the machine if WHEN trapping is in force; SQRT of a negative expression will do the trick, as will INKEY$ at the end of a file.

WHEN ERRor is extremely persistent. You can type LOAD or NEW and the computer will still try to trap your errors to a non-existent routine. Similar problems occur if you delete an active WHEN statement or type one as a direct command.

Tony Tebby's Supertoolkit clears WHEN after commands like NEW and LOAD and fixes the ERR_DF mistake but it cannot help with the other problems. The Digital Precision Turbo compiler gives you reliable WHEN ERROR trapping anywhere in a program, on all QL versions, but of course it does not fix the interpreter.

Another WHEN option in the "JS" ROM lets you monitor variable values. A block starting WHEN VAR>10 will be executed only when the condition becomes true; every time VAR is set Basic checks the new value and calls the WHEN routine if the value of VAR exceeds 10. Unfortunately this does not work reliably on any QL version either; sometimes it gives a 'bad name' report or calls the routine more than once.

The last new command for the "JS" ROM is TRA. This makes it easier to customise QL software for use in other countries. TRA normally has two parameters. The first points to a table to be used to TRAnslate characters sent through the serial ports and the second is the address of a new error-

message table. Both tables must start with the ‘nonsense’ word value 19195, which crops up all over the QL system as an indication that ‘data follows’.

The next two words in the serial table contain the offset to two translation lists, measured from the start of the table. The first list starts with a 256-byte list of substitute codes for each character code from 0 to 255. When a character is to be transmitted it is looked up in the table and the code from the appropriate place is transmitted instead; you can translate the QL end-of-line marker, CHR$(10), by POKEing 13, the usual code for carriage return, into the eleventh byte of this list.

Put the value zero into the appropriate slot in the first list if you want to translate one code into a sequence of several characters. They appear in the second list, which the QL uses only when transmitting serial data — there is no time to use it during input.

The second list starts with a byte value, the number of four-byte entries in that list. Each entry after that starts with the code to be translated, followed by three replacement codes. If you need only two replacement characters, the last code in a group should be zero.

The message table is simpler. After the 19195 there are 29 words, each holding the offset from the start of the table to a message. The messages are stored as normal QL strings — a word length, at an even address, followed by the appropriate text. Beware — the last two messages are sequences of three- character day and month names, with no length word. That is why REPORT - 28 goes haywire.

If either parameter of TRA is zero the corresponding table is left alone. TRA 1 sects the standard message table and allows characters to be transmitted through the serial ports unmolested.

Turbo Toolkit contains an example program which uses TRA to translate Sinclair error reports into plain English; other TRA demonstrations have been printed in the user-group magazines Quanta and Quaser.

A special version of the “JS” ROM was produced for American QLs, which must be compatible with the rather feeble National Television Standards Committee TV standard. The American ROM had a three-letter name, “JSU”; it contains all the “JS” ROM bugs and features, plus changes which you should bear in mind if you develop programs which may be used in the States.

An American television set can display only 192 lines of pixels. In TV mode the American QL hardware ignores the first and last 32 lines of screen memory; in monitor mode it works with the usual 256 lines. American QL owners can swap between U.K. and U.S. TV lineage by POKEing 1 or 2 to address 163890 and typing NEW.

You still get 20 lines of text into a standard TV mode window, as the character set in the “JSU” ROM has been crushed vertically. Characters are drawn on an 8 by 5 dot matrix, rather than the 10 by 5 used on European systems. In monitor mode the crushed characters are still used but they are spaced by an extra two blank lines; rows of text are the normal height but look like a ransom note.

Another change compensates for the different shape of dots on an American display. The QL graphics co-ordinate routines compensate for the shape of each dot so that circles do not look elliptical and squares do not appear as rectangles. Routines which use pixel coordinates, such as WINDOW and BLOCK, do not perform any compensation, which is why vertical and horizontal units are different.

Atlantic ROMs

European and American ROMs use different compensation factors, so that graphics shapes look the same on either side of the Atlantic. Unfortunately there is no way to compensate for the difference in BLOCK and WINDOW shapes. Many programs use a mixture of graphics and pixel co-ordinates; they may look satisfactory on one side of the Atlantic but they will not line-up properly across the water.

Things are still tricky, even if you stick to one co-ordinate system. If you work entirely in graphics co-ordinates your shapes will not be distorted, although they may escape off the edge of the screen. If you use pixel co-ordinates everything will fit on the display but the vertical and horizontal proportions will be different on an American screen.

Sinclair's last fling was Qdos version 1.13, which usually crops up in "MG" ROMs. They have never been supplied in the U.K. although the chips work well in a British machine. The "MG" ROM has only one new bug and kills several important faults in previous versions.

First, the new bug. The "MG" ROM line-drawing routine does not always plot the point at the end of a line or arc, so that one-pixel gaps may appear at the corners of graphic drawings. If that disturbs you, a 'patch' program to correct the bug is available free from Qsoft, PC Box 56, DK 4000, Roskilde, Denmark. Send a disc or cartridge for the program and an international reply coupon for return postage.

Serious bug

The most serious QL filing bug has been fixed in the "MG" ROM, The Micro- drive system does not hang up, stalling the computer, if file access is performed when the system is very short of memory. The QL file system uses spare RAM to buffer information en route between Microdrives and your program. The "AH" and "JM" versions could get stuck in a loop if the free memory fell to 1K, because the multi-tasking Microdrive handler would over-write the current block with new information before the application program had time to digest the original data. That is a common cause of failure when RAM-hungry Psion packages are used.

Sinclair tried to cure the problem in the "JS" ROM but managed only to fudge things so that all was well until there were just 512 bytes of free RAM — not much of an improvement.

Unless you have an "MG" ROM, it is worth performing a check for free memory before Microdnve access. This function returns the amount of space free for Microdrive buffering:

```
DEFine FuNction BUFFER_SPACE
  RETurn PEEK_L(163856) - PEEK_L(163852)
END DEFine BUFFER_SPACE
```

The "MG" ROM is the first to be able to close the second serial port, SER2. Earlier ROMs used to close SER1 instead whenever you tried to close SER2.

Apparently the "MG" ROM is the first to work correctly on a QL with eight Microdrives. I suspect such systems are rare, even though Spectrum Micro-drives will work if plugged into the QL extension drive socket. It is claimed that earlier ROMs used to forget about MDV2 after you had used MDV8.

"MG" SuperBasic has been thoroughly spring-cleaned. You can use any number of parameters and LOCALs in a procedure or function. Previous versions of the Basic allowed only enough space for nine such names. If you used more, the program could lock up or be corrupted by the appearance of

spurious PRINT keywords in place of names towards the end of the program. That knowledge may help you to spot program listings which were improved by their authors, untested, before publication.

The “MG” system is less prone than its predecessors to use up RAM as a program runs. Earlier ROMs lost track of some memory every time a slice of a dimensioned string array was passed as a parameter — PRINTed, for instance. If that happened in a loop, as usual, the program ran slowly, constantly grabbing more and more memory, until it failed ‘out of memory’ discarding all variable values. “MG” does not get into this state.

The only way to recover memory lost in this way was to enter CLEAR or NEW, both difficult to do in a running program. You could avoid the problem by copying slices to temporary, undimensioned strings but that is a slow and messy solution.

The RENUMber routine in the “MG” ROM can cope with RESTORE statments at the start of lines containing DATA. Earlier ROMs used to RENUMber DATA elements as if they were line numbers. This bug was a hangover from the days of the kludge — the original RENUM totally ignored RESTORE.

Any channel

The “MG” CURSOR command lets you use graphics co-ordinates on any channel. Other ROMs let you use only four co-ordinates with the default channel, channel 1. DATA values in brackets no longer cause the other items on the line to be ignored. CLS and PAN can cope with windows narrower than the cursor.

String comparison works properly on characters with ASCII codes greater than 127. Previously you had to check the CODE of the character, rather than compare the string correctly, to check reliably for cursor, function and other special keys.

The WHEN routines are still unfinished in the “MG” ROM, although the trivial ERR_DF bug has been fixed. Daft parameters no longer upset READ and INPUT and you can OPEN and CLOSE channels ad nauseam without the system complaining; older ROMs limited you to 32,767 OPENs in a session, which used to upset dreadful programmers.

The “MG” ROM was designed for use in continental Europe and is in several versions, with key layout, characters and messages customised for different nations. The 32K ROM is the same for each country and the 16K ROM holds all the information, which varies between versions. VER$ has an extra letter at the end, to show the country — “MGE” for Spain, or “MGS” for Sweden, for example. The dot in the Qdos version is replaced by the country-code letter.

If you want to use a foreign-language “MG” ROM it is easy to create a new set of error messages using TRA but that may not be sufficient. The French “MGF” ROM expects a particularly odd key layout. “A” and “Z” swap places with “Q” and “W”, so the layout, like that of French typewriters, is known as AZERTY rather than QWERTY. The ‘M” key is moved to where “,“ is normally found, and the “c” key works as an extra shift, putting a circumflex accent or an umlaut over the next vowel typed. Several other characters are shuffled to make space for accented letters.

The last Sinclair effort was the Greek ROM. It works well in the U.K. as soon as the error messages are changed from their heiroglyphic Greek form which, to be fair, is about as clear as the original English. These ROMs use Qdos 1.13 but they are further developments of the “MG” ROM and cannot

be mixed with other Qdos 1.13 chips. Greek ROMs display their version as “Sigma FP”,although the chips are marked “EFP”;presumably the Sinclair Mexican chip makers could not find a sigma stamp.

The “MG” ROM is a great improvement on its predecessors but even so it contains 30-odd unfixed bugs. Next month I will list them and explain how to circumvent them.

All the QL ROMs from JM onwards, including JSU, are plug-compatible. All you need to do the change versions to disconnect the power and replace the components in sockets 1C33 and 1C34 with a different version.

“AH”, “JM”, “JS” and “MG” ROMs are available in Europe from Adman Services, 53 Gilpin Road, Admaston, Telford TF5 OBG. Remeberthat the wiring of the ACM sockets must be changed if you switch to or from version “AH”. The American “JSU” ACM is supplied by Curry Computer, PD Box 5607, Glendale, AZ 8531 2-5607, U.S.A.

Beating The Bugs

Simon Goodwin concludes his survey of QL system bugs with a summary of the faults shared by every QL, regardless of vintage or nationality.

Forty-six errors in early versions of the QL operating system were described last month and I explained how they could be fixed by ROM upgrades. Now I list another 31 bugs which crop up in every version of the QL and tell you how to circumvent them. If you have found others, please let QL World know.

There is no easy way to define a bug. It is well-known in the computer industry that one person's bug is usually someone else's feature. It is almost as well; known that the person with the bug will be a customer and the person with the feature will be a marketeer.

In the absence of an industry-standard definition of a bug, I have set out to list all the quirks of the QL ROM which cause apparently correct programs to give unexpected results, or no results at all. I have also counted a few undocumented restrictions. Any nonviolent action which prevents the entry of further commands is automatically considered a bug, unless Sinclair specifically warns against it in the big black QL User Guide.

That raises the question of how should a bug be fixed? In some cases, it is sufficient to detail the problem so that people can avoid it. If you document a bug, it is vital to tell users how to get the result they want without getting into difficulty

Another approach is to make it impossible to reach the circumstances which cause the problem. Integer FOR loops did not work on the AH version of the QL, so Sinclair changed later versions so that you could not even type them in. That approach may be justified on grounds of expediency or efficiency but often it is just an excuse for leaving a real flaw uncorrected.

The last kind of bug-fix is a rare and wonderful thing. It lets you do what you originally wanted, in the way you intended. That is the best for customers, unless their original idea was a daft one, but it is the most expensive for all concerned. It often leads to the introduction of new problems, because a technique which used to work is clobbered by the fix, or because the correction passes you to more faulty code.

This list deals only with idiosyncracies of the QL ROM, the SuperBasic language and the underlying collection of operating routines called Qdos. Some of these bugs can cause other programs to fail. Software developers should guard against the most common problems by defensive programming in their own code, which is why I have included plenty of detailed technical information.

The problems are collected undertwo headings. Input/Output bugs affect the flow of information between the QL and peripherals such as drives or the display. SuperBasic bugs affect the executive of QL Basic programs; many are corrected if you compile your programs.

SuperBasic bugs

PROBLEM 1: When a REPeat or FOR statement is encountered, the value of the corresponding identifier is set to zero. For instance, this line prints five values 0 to 4, rather than 3 and 4, as you would expect in any other Basic:

```
X=3 : FOR X=X TO 4:PRINT X
```

The same problem occurs if the value of X is used to compute the end-point of the loop, or the step.

REASON: The SuperBasic interpreter does not use the same format to store loop details and simple variables. Whenever a loop starts the old value is thrown away and a new, zerod storage area is allocated.

CURE: The best solution is to change the variable name — it is poor programming practice to use the same name for two logically-distinct purposes, in this case as a terminal value and an instance count. Nonetheless, there are times when that may be convenient. Super-Basic compilers do not have this bug, as they analyse the program during compilation and allocate space for the value and loop details from the start.

PROBLEM 2: The QL trigonometric package gives silly results for COS between 16384*PI radians – about three million degrees – and 65535 radians. Greater values give an overflow error. For a weird result, type:

```
PRINT COS(60000)
```

CURE: use SIN(X+P1/2) instead of COS(X), or do not do it. It is extremely unlikely that your program will fall foul of the bug unless it has gone haywire, in which case it will probably run into the overflow error. Three million degrees should be enough for anyone.

PROBLEM 3: RESPR, the function to reserve memory, does not work while a task is running. It gives a 'not complete' error.

REASON: The memory map in the Concepts section of the QL User Guide shows that the area of memory used by RESPR fits between the top of RAM and the first task loaded. Tasks may not be moved, so you cannot expand the RESPR area while any task is loaded.

SOLUTION: Use a toolkit function like ALLOCATION or ALCHP to obtain memory from the common heap at the other end of the memory map. Compilers re-direct RESPR calls to the heap automatically, since all compiled programs run as tasks.

PROBLEM 4: Every time you type RUN after a SuperBasic error – or BREAK – the system loses track of another 16 bytes of memory. You do not get this memory back until you type NEW.

REASON: The system does not tidy data allocated on the user A7 stack when an error occurs after RUN.

CURE: Start your program with GO TO instead of RUN. CLEAR and the 'Out of Memory' condition do not release the space — you must type NEW. Remember to SAVE your program first. Luckily the creepage is only small and you have to type RUN many times before you exhaust the capacious QL memory.

PROBLEM 5: You cannot de-allocate space reserved from SuperBasic with RESPR. There is no command to do this in BASIC. The required machine code routine exists in the ROM – TRAP 1 with d0=15 – but it does not work; it may do nothing, allocate more space, or crash the system.

REASON: Resident procedure space is intended for device drivers and commands which are linked permanently into SuperBasic. If such code was overwritten the machine would crash as soon as the system tried to use it. That does not explain why the ROM routine was partly-written in the first place

and it does not help people who use RESPR memory to store data such as character sets, or machine code to be accessed with CALL.

CURE: Use toolkit functions to work with space on the common heap. ALLOCATION orALCHP will grab space and DEALLOCATE or RECHP will release it. Memory allocated by a compiled task is usually released automatically when the task stops but you can prevent this, if need be, with Turbo Toolkit. Extension commands and devices should always be linked from Super- Basic with RESPR.

PROBLEM 6: A GO SUB in a single line FOR loop acts like an END FOR. This prints all the values of X from zero to ten, but only displays 'Counting' after the last value:

```
5 FOR x=1 TO 10: PRINT x:GO SUB 20
10 PRINT "Finished":STOP
20 PRINT "Counting":RETurn
```

CURE: Compile the program or use a multi-line FOR loop, with an explicit END FOR, instead of the short form. The problem crops up only if the GO SUB is on the same line as the FOR. Alternatively, replace the GO SUB with a procedure or function call.

PROBLEM 7: It is impossible to use an integer or string variable as the identifier of a FOR loop. Early QLs let you enter these statements but could not run them; versions from JM onwards reject the loop as a 'bad line'.

CURE: All QL compilers support integer FOR loops, although you may have difficulty entering them. Turbo and Q-L/berator have implicit type directions so that you can tell the compiler to make a variable an integer without upsetting the interpreter by putting a percent sign after the name. No-one has yet implemented string FOR loops.

PROBLEM 8: The integer division operator DIV does not check for all cases of binary overflow; it gives spurious results instead. Try:

```
PRINT -32768 DIV -1
```

REASON: The QL uses the common "twos complement" format to encode signed integers into 16 bits. In this scheme, the representation of 32768 and —32768 is identical. To avoid ambiguity, the maximum valid integer is 32767 and the minimum is —32768. DIV does not check for the case when the minimum is negated; this should cause an overflow.

CURE: Avoid extreme values with MOD and DIV, or use a compiler to check your code. Turbo and Supercharge diagnose the overflow correctly. Q-Liberator repeats the division using floating point maths, postponing diagnosis until the value is assigned to an integer variable.

PROBLEM 9: You cannot use an integer or string variableas the identifier of a SELect statement. Early QLs let you enter these statements but could not run them; versions from JS onwards reject the statement as a'bad line'.

CURE: Compile your program with Turbo on any QLversion, orwith Supercharge on AH or JM versions. Integer SELect is extremely fast.

PROBLEM 10: The INT function gives an overflow error if its argument exceeds 2 to the power of 31 minus 2, about 2.14 billion (U.S.)

REASON: The QL floating point maths scheme uses 31 bits to store the digits of a value. That allows any number from about minus 2.14 to plus 2.14 billion to be stored exactly. If a value is outside this range the computer can store it only approximately, as the first nine digits with an exponent to indicate the position of the decimal point. The machine cannot be sure of the value of the last digits. INT fails rather than return an approximate result.

CURE: Avoid such values, or use a range check to filter out values which INT cannot truncate. There is no way to INT larger values accurately without increasing the precision of the floating point format. That would require a major ROM re-write and make floating point maths much slower and more cumbersome.

PROBLEM 11: You cannot BREAK into a single-line recursive procedure.

REASON: Sinclair did not put a check in the procedure-call code. This oversight saves time and is unlikely to cause problems.

CURE: Split the procedure over several program lines.

PROBLEM 12: You cannot interpret more than one SuperBasic program at a time.

REASON: The SuperBasic task is handled specifically by the ROM – it is the only task which can grow and shrink dynamically as it runs. Most of the interpreter code would multi-task satisfactorily but interpreted programs would be very slow, because of the need to move programs around memory as they collided with one another. Much information, such as resident procedure details, would have to be duplicated between copies. The BREAK mechanism would also have to be altered, as at present it will interrupt only task 0.

CURE: Several firms have advertised routines to make the interpreter multi-task; none has reached the market. SuperBasic compilers will translate programs into standard tasks which can run within fixed bounds.

PROBLEM 13: Contrary to Sinclair claims, the speed of SuperBasic interpretation declines steadily as program size increases. A jump to the end of a large program on a standard QL can take 100 times as long as a jump to a routine at the start.

REASON: When the interpreter jumps to a line in a program it has to read and skip over the start of all lines inbetween. Loops and calls to procedures and functions work the same way, so the position of a DEFinition in a program can make a big difference to the speed with which it is found. Other BASIC interpreters look up variable names in a similar way but the QL stores them by index, so it does not slow as the number of names in your program increases.

CURE: Compile the program. That fixes the address of each line or routine so that it can be found instantly wherever it may be in the program.

PROBLEM 14: If you type EDIT after breaking into a procedure or function, SuperBasic can present you with a ‘not implemented’ error and the wrong line.

CURE: Press BREAK and type the EDIT command again. Do not be tempted to edit the line you are given, or you could corrupt the whole program.

PROBLEM 15: SuperBasic lucks-up if you type CLEAR or edit a line after trying to call a procedure or function which was defined at the end of the program but deleted later. This happens only if there is no line left with a number beyond that of the DEF of the deleted routine.

REASON: The interpreter gets stuck in a loop if it is asked to clear details of a PROC/FN call which does not make sense.

CURE: Do not do it, or keep a STOP or REM on line 32767 at the end of your program.

PROBLEM 16: If you try to READ or INPUT a value into a slice of a string which has not been dimensioned, the value will nto be stored and Basic may stop without a message. For example, this READ will not change the value of A$:

```
20 A$="QL USER!"
30 READ A$(4 TO)
50 DATA "WORLD"
```

CURE: Either dimension the string:

```
10 DIM A$(8)
```

Or read the value and assign it to the slice in two steps:

```
30 READ T$
40 A$(4 TO) = T$
```

PROBLEM 17: If an END is missing from a program, interpretation may stop with no indication of the problem or the location. Spurious ENDs are ignored with no message.

REASON: When the interpreter needs to find an END – after a conditional clause, or an EXIT, for instance – it searches forward through the program until it finds the required statement, advancing the "continuation line" indicator as it does so. If the END is never found, the program stops, with no record of where it 'came from'.

CURE: Use a compiler or style checker, such as Better Basic, to make sure that starts and ends are matched properly. Under such circumstances the Turbo Toolkit routines TRACE and HOW COME help you to determine where a program failed.

PROBLEM 18: Memory remains allocated every time you jump out of a procedure or function without performing a proper RETurn or reaching END DEFine. The space is recovered by CLEAR.

CURE: Do not do it. In a well-designed program, every routine should have a single entry and exit point. Avoid GO TOs and arbitrary use of REPeat and END REPeat to perform jumps, e.g.:

```
REMark Very bad style
REPeat loop
  FRED
  DEFine PROCedure FRED
END REPeat loop
END DEFine
```

This will work but it is horrendous style and will consume memory at a rate of about 1K per second.

PROBLEM 19: Very large numbers take a long time to be converted for PRINTing. Try this:

```
FOR I=1 TO 10:PRINT 123456E610
```

REASON: The binary to text value conversion routine works by multiplying or dividing by 10 repeatedly until it has a nine-digit integer to deal with. This is fast for common values but very slow for extreme ones.

CURE: Wait.

PROBLEM 20: The QL internal arithmetic routines are accurate to more than 9 digits but only a maximum of seven digits are displayed.

CURE: Compile the program with Turbo or Supercharge, both of which show all nine digits and correct small errors in ROM floating point routines.

PROBLEM 21: You can only use one short-form FOR or REPeat statement on a SuperBasic line. If you put more than one, the last one will not work.

CURE:Add END FORs and END REPeats to convert the loops into long forms.

INPUT/OUTPUT BUGS

PROBLEM 22: FILL sometimes colours the same line twice. This causes problems only if you are using OVER -1, as the effect is to reverse the effect of the FILL on that line.

CURE: Start drawing the pattern from a point at the top or the bottom.

PROBLEM 23: If you MERGE a file of direct commands, only the first line will be read and the file will not be closed. That makes it impossible to use another tape or disc in that drive later.

CURE: Put the Supercharge/Turbo Toolkit command END_CMD at the end of the command-file line. That closes the file.

PROBLEM 24: The machine may crash if a syntax error – 'bad line' – on a line without a number is read from a command line.

CURE: Do not do it. Check your command file by putting a number at the start of each line and LOADing it; potential 'bad lines' will be marked with 'MISTake'.

PROBLEM 25: Reports may be lost or the SuperBasic interpreter may be locked out if the standard SuperBasic channels 0, 1 and 2 are closed.

CURE: Do not do it — Psion take note.

PROBLEM 26: If you load a file with LBYTES after editing and saving it there is a risk that the file loaded will not include the changes you made.

REASON: When you CLOSE a file, Qdos prepares to replace all the parts which have been changed. They are copied from temporary storage in memory 'slave blocks' into the correct place in the file. In the interests of speed, LBYTES suspends this copying and loads the file directly from the drive, without checking to see whether or not some blocks have changed recently but have not yet been written back to the medium.

CURE: Wait for the drive to stop after CLOSE before using LBYTES, or use the Super Toolkit FLUSH command.

PROBLEM 27: The microdrive handler gives a delayed and misleading 'bad or changed medium' message if you try to store something on a tape which is write-protected.

REASON: The handler does not check whether or not a drive is write- protected when a file is opened or data is written. Luckily for the existing data, the low-level routines detect that the tape is protected and do not allow writing. The control software assumes a 'bad medium' when several attempts to write have failed.

CURE: Beware, and do not assume disaster if you get a 'bad medium' report on one of your master tapes. Machine code programmmers can tell whether or not the currently-turning tape is write-protected with IPC call 1 but this does not tell you which drive is turning. To find that, enter supervisor mode to prevent asynchronous changes and read the drive number byte at 164078 before interrogating the IPC. A proper fix should be buried deep in the code of the device driver.

PROBLEM 28: If you set a position for binary random access far beyond the capacity of a cartridge you may get a misleading 'bad or changed medium' message instead of 'out of range'.

REASON: The Microdrive handler uses 24-bit addresses internally, limiting the length of a file to about 16.7 million bytes. Larger values corrupt other information packed into a 32-bit register with the address.

CURE: Do not be misled by the message — your tape is intact. Then lower your sights; 16 MB Microdrives are some way off yet.

PROBLEM 29: You cannot draw a block width of 512. Nothing happens if you try it.

CURE: Use CLS or two horizontally- adjacent BLOCKs.

PROBLEM 30: The priority of SuperBasic, task 0, may be set to zero, preventing further command entry.

CURE: Use Turbo Toolkit or Supercharge extensions which check for this case and exclude it.

PROBLEM 31: Pressing the combination of keys CTRL, ALT and 7 – or 2 or 5 on some machines – usually causes the QL to crash at once.

REASON: These keys trigger a level 7 interrupt, which is intended to call-up a hardware debugger which Sinclair staff used while testing development systems. The facility is still there in production machines, even though the external debugger is not. The interrupt re-sets the 8049 second processor but not the main 68008. If the interrupt occurs while these two chips are communicating, as is highly likely, the 8049 'loses its place' and crashes, preventing keyboard input.

CURE: In general, do not do it. The keys have been chosen to make accidental entry very unlikely. The system call MT.TRAPV (TRAP 1, D0=7) lets you specify a routine to be executed when hardware errors d interrupts occur. Unfortunately this is of limited use, as the keyboard and RS232 die when the 8049 re-sets.

You should not treat this list as an indictment of the QL. Every computer system, of whatever vintage, contains bugs and the QL is no worse in this respect than other ambitious designs. At least, and at last,

you are forewarned by this list. Each version of the QL ROM has its own bugs, besides those listed. Read the article entitled Version Therapy, in last month's QL World, to find more about specific ROM faults.

Return of the ROMs

Simon Goodwin follows up last year's look at the QL built-in ROM software with 11 new bugs and more about the QL.

When I revealed the results of three years' research into the idiosyncracies of the QL built-in software — the operating system Qdos and the SuperBasic interpreter — I found and explained 77 bugs in the QL ROM.

Since then, with the help of QL World readers, I have identified another 11 problems, so now is the time for an update.

This is more than just a list of faults; it explains how to circumvent them. All complex systems contain bugs, though hardware manufacturers curiously are shy about admitting them and sorting them out. Bugs are rarely a problem if you know about them and how to avoid them. All most users want to know is how to get the result they need without getting into difficulty.

There is no sure definition of a bug. One person's bug is usually someone else's feature. I have included quirks of the QL ROM which cause apparently correct programs to give unexpected results, or no results at all, plus a few undocumented features. The list deals only with idiosyncracies inside the QL ROM — the SuperBasic language and the associated collection of operating routines called Qdos.

Some of the bugs may cause other programs to fail, so I have included technical information to help software developers guard against the most common problems by defensive programming in their own code. The bugs are in two groups — problems which afflict all QLs, followed by a list of faults specific to certain ROM versions.

'New' bugs in QLs everywhere

Integer input

Dilwyn Jones reports a sometimes annoying bug in all QL ROMs. Integers — whole number values stored in variables with a percent sign at the end of their names — can have values between -32768 and 32767. The statement X% = -32768 works satisfactorily but X% = -32769 gives an error as you might expect.

You cannot INPUT a value of -32768. If you try to do so you get an 'error in expression' report because the QL works out the value of the digits before its sorts out the sign, plus or minus, and + 32768 is not a valid integer. Qdos uses the same code to convert values from all devices, so the bug is present whether your INPUT is from the keyboard or a file.

It is really just sloppy coding on the part of the ROM authors who seem to have difficulty with the value -32768. I pointed out previously the weird results you can get using that value with the integer DIV and MOD operators.

Window Rules

You can define the position of any window on the screen in terms of coordinates in picture elements or pixels. The co-ordinate scheme assumes that there are 512 pixels across the screen and 256 downwards. Window widths and horizontal co-ordinates are always rounded to an even value. This

means you cannot put a one-pixel gap between two windows in MODE 4, the highest resolution QL display mode. The minimum gap is two pixels.

You cannot deal with this by setting a BORDER width of one in the window, as horizontal border widths are also rounded up, so that BORDER 1,7 gives a white border one pixel wide in the horizontal lines but two pixels wide vertically. You can easily see this if you use a stippled border pattern:

```
MODE 4: BORDER 1,7,0
```

This bug is not properly-documented but understandable when you think about the QL display design. The restriction exists because QL windows are designed to be able to cope with a change of MODE at any time. One mode allows four colours, with 512 dots across the screen, while the other allows eight colours with 256 dots on each line. A gap of one pixel in MODE 4 would become a problematic gap of half a pixel as soon as MODE 8 was selected.

Merge bugs

The MERGE and MRUN commands become confused if you use them inside a SuperBasic procedure or function because the act of merging new program lines invalidates stored information about where in the program execution should continue.

SuperToolkit 2 re-defines those commands to detect attempts to use MERGE inside a DEFinition. It stops the program with a 'not implemented' report if it runs into trouble.

Cotangent error

Dr. Helmut Aigner of Austria discovered that the Co-Tangent function, COT, gives a result of 1 when asked to fmd the co-tangent of zero, whereas COT(0) is undefmed and should really give an 'overflow error.'

The error is in the Qdos maths package, rather than in SuperBasic, so it affects other languages. In general, if a language uses the Basic 7-9 digit precision, it is likely it will inherit this bug. It is easy enough to check for the special case of zero explicitly in programs which use co-tangents.

Startup keys

According to published documentation about Qdos it should be possible to tell whether the user started the QL by pressing F1 or F2 by reading the value in address 163890, known as SV.TVMOD. This information would be very useful when programs start as they could work out whether or not the user had a monitor and set windows to suit automatically.

When the QL starts PEEK(163890) is 0 for a monitor display (F1) and 1 or 2, for a TV display (F2); 1 indicates a European TV, capable of displaying 256 horizontal lines of pixels, and 2 means that a 525-line American display was selected, with 192 lines of pixels and characters eight rather than 10 pixels high.

Unfortunately the MODE command, used to switch between four- and eight- colour displays, has a bug which means that the value of SV.TVMOD, the F1/F2 flag, is affected as soon as you issue your first MODE command. The result is that programs have to deduce whether you are using a TV or a monitor indirectly by checking the screen mode — four or eight colours — rather than the initial selection you made after turning on the machine. This is a fault because it does not necessarily follow

that you are using a monitor because you are in MODE 4 before you start using Quill. Nor does it follow that you have a TV because you load a Psion program from MODE 8.

Current versions of the Psion package no longer check SV.TVMOD because of the bug. You can circumvent the fault when using programs which test SV.TVMOD by POKEing the required value back into 163890 but this will not help if your program loader issues a MODE command before it tests for TV or monitor selection.

This bug can be cured by re-writing the MODE command to set register D2 to -1 meaning no change — before calling the operating system. Anyone who owns a copy of Speedscreen will find that it fixes this bug automatically by replacing the standard MODE routine with an enhanced and corrected version. If you want to use it to keep the original F1/F2 value you should load Speedscreen at the start of a session before the first MODE command.

CLS

By far the most interesting QL bug occurs in the CLS command. All known ROM versions accept undocumented CLS parameters and do unexpected things as a result. The CLS command allows a single optional numeric parameter. Officially it is a value between 0 and 4, referring to different sections of the display as documented in the QL User Guide.

Non-standard values cause calls to other display device routines, using whatever parameters happen to be in registers when the call takes place. Some of those routines are not normally accessible from standard SuperBasic. The property appears to be an accident, although it can be useful in practice.

The internal routines to clear different areas of the screen form a sequence of distinct system-calls — SD.CLEAR, SD.CLRTP, SD.CLRBT, SD.CLRLN and SD.CLRRT, using system call numbers 32 to 36 inclusive. CLS converts parameters between 0 and 4 into a call number of 32 to 36. So choose the ppropriate ROM routine.

Other system call numbers correspond to different display operations and the code for CLS allows parameters outside the documented range of 0 to 4.

Parameters between 5 and 7 give a ‘bad parameter’ report but CLS changes the current STRIP colour, the background colour used when printing characters. CLS 8 works like STRIP 0! You can put a channel number before the parameter to select the window affected by the command CLS #0,8.

CLS 9 works like INK 0, which is particularly interesting when you realise that the system call to set the strip colour is number 40 and the call to set the ink is number 41. The sequence continues through the TRAP #3 display routine, so CLS 10 sets FLASH 1, CLS 11 sets UNDER 1 and CLS 12 selects OVER 0.

Values between 13 and 15 give a ‘bad parameter’ again, as do all parameters which give results between 5 and 7 if you make them MOD 8 but then things become really interesting. CLS 16 plots a point at graphics co-ordinate 0,0. The next three have no obvious effect but CLS 17 draws a zero-length line, while 18 and 19 draw zero length arcs and ellipses.

CLS 20 calls SD.SCALE, system call 52, and has the rather annoying effect of setting an enormous graphics scale so that lines, arcs and ellipses all appear in the bottom left pixel of the window. Use SCALE 100,0,0 to set things to rights.

Parameters from 21 to 95 give ‘bad parameter’. CLS 96 appears to do nothing but in fact it checks the channel for pending input, using system call zero. The parameter values have ‘wrapped around’ internally to start again at the lowest call numbers. CLS 97 waits for one character before returning. CLS 98 uses the INPUT routine IO.FLINE to read the character codes between 32 and 191. Characters are displayed but not returned and the buffer size is just 3.

CLS 99 calls IO.FSTRG; it fetches a line of up to three characters of any code without displaying them. ALT keystrokes count as two characters. CLS 100 calls IO.EDLIN, the Basic line editor. A long strip of gibberish appears; you can edit the text but any attempt to insert characters after the second gives a ‘buffer full’ error.

CLS 101 to 104 give ‘bad parameter’ again. You should not enter CLS 105 as it locks up the machine; it corresponds to SD.EXTOP, the extended operation call, and in this case it hands over control to a non-existent routine.

CLS 106 and 107 have no obvious effect but they read the window size in pixels and characters. You get a three- pixel-wide, horizontally-striped border with CLS 108.

CLS 109 to 111 give ‘bad parameter’, unfortunately, so you cannot access the routines to turn the cursor on and off. CLS 112 and 113 call the POS and TAB routines, in both cases causing an

‘out of range’ report. CLS 114 moves to the next line unless the window could need to scroll in which case it gives an ‘out of range’ error. This call, to SD.NL, could be useful in screen- handling programs if you have some way to trap the error — it prevents having to keep checking the current line when moving. CLS 115 moves the cursor left, giving ‘out of range’ after the left most column of the window, and CLS 116 moves the cursor right.

CLS 117, 118 and 119 give ‘bad parameter’ but CLS 120 scrolls down the window by 10 lines; 121 and -122 scroll each side of the window, while CLS 123 pans the window right. Parameters from 124 to 127 are rejected and at CLS 128 we are back to the same effect as CLS 0. The parameter values cycle round in a 128-step sequence.

Special bugs

The remaining bugs affect only the specific versions of the QL noted. The August 1987 edition of QL World explained how you can upgrade the ROMs in your QL. You can obtain most QL ROM versions from Adman Services at 53 Gilpin Road, Admaston, Telford TF 5 OBG.

I said initially that JM and later versions were made using mass-produced ‘mask-programmed’ ROM chips, whereas the AH and earlier versions used individually-programmed EPROMs. The upgrade procedure from EPROM to ROM is significantly more complicated than from one ROM to another, when you can just swap the chips in their sockets.

Since then I have heard from D. A. Masters, who bought a JM QL with EPROMs in it. It appears that the first 100 or so JM QLs were made with EPROMs rather than ROMs because the JM software was ready but had not arrived from the manufacturing subcontractor. The upgrade procedure from JM EPROMs is the same as that for AH chips.

Second processor

I have found a cure for the CTRLALT-7 bug, documented last year. Most QLs lock up if you type those characters because the software in the second processor, separate from the main ROM, treats that keypress as a request to call up external hardware which only Sinclair owned.

Add-on keyboard manufacturer Schoen recently produced a replacement second processor to cure key- bounce problems for people using its keyboard and this upgrade also prevents CTRL ALT-7 interrupting the machine. Unfortunately the Sinclair key-bounce fix, the version 1.2 chip from Applied Technology, does not correct the CTRL-ALT7 bug.

Editing cursor

The first two workable QL versions, AH and JM, have a bug in IO.FLINE and IO.EDLIN, routines used by INPUT and EDIT. If the data entered becomes too large for the available storage buffer the routine gives an error message but leaves the cursor turned on in the input window.

This does not cause problems in the JM ROM versions of those commands because the ROM code turns off the cursor after an error to be on the safe side. It can cause problems if you write your own machine code programs and call IO.EDLIN or IO.FLINE.

You can prove that the error exists by typing CLS 98 to call 1O.FLINE directly, then typing three characters to fill the buffer. An error is reported and the command cursor appears at the bottom of the screen but the cursor at the top of the screen is still flashing. Type CTRL C to get back to the command line, then enter INPUT X$. Finally, press ENTER and let INPUT turn off the stray cursor.

The DIY Toolkit function EDLINE$ and the Turbo Toolkit EDIT%, EDIT$, EDITF functions all contain code to turn off the cursor explicity so they are not affected by the bug.

Bad names

The AH and JM versions of QL SuperBasic have the annoying bug that they will not let you re-define names which have caused the computer to give a BAD NAME report. You might become irritated by the standard QL display speed and type: _SPEED 2 to turn on Speedscreen, only to find that it was not loaded. The system reports a BAD NAME error unless you have the ROM version because it does not recognise the command. After that normally you would use RESPR to reserve some space for the code, load it with LBYTES and call the start address.

In this case you are usually safe, because Speedscreen turns itself on automatically when you load it but the _SPEED command is still considered a ‘bad name’ by an old QL system. So you cannot check the Speedscreen version with _SPEED 1, because _SPEED is still defined as a Basic ‘bad name’ rather than the name of an extension command.

The same confusion occurs ifyou try to use any other Toolkit commands before loading them. If you try to use them before they are loaded you confuse the system. Such commands work properly after you type NEW, because that clears out all prior SuperBasic definitions, leaving only the resident commands and functions. Unfortunately this also gets rid of your program and all the variable values.

If you try to run a program compiled with version 2 of Turbo on a system where commands are multiply-defined the compiled code produces a message and a list of re-defined names it needs to use. Type NEW and try again.

CLEAR is not sufficient to persuade SuperBasic to release unset names. The interpreter tends to grab memory whenever possible and release it only under extreme circumstances. Resident command definitions over-rule SuperBasic ones in JS and MG versions of the QL, so this bug does not affect later ROMs.

No Cursor

Another annoying quirk of the AH and JM ROMs is the way the cursor vanishes after you have finished using a task. Nothing appears on the screen until you type CTRL C to switch to another window.

Later QL ROMs turn on the cursor automatically in the 'next' task window usually the SuperBasic window zero at the bottom of the screen when the task which was previous accepting input terminates.

Translation

The JS version of the QL introduced a new command, TRA, which translates or exchanges the codes of characters transmitted through the serial ports automatically. The bad news, according to top Danish software house Dansoft, is that TRA translates values only after it has adjusted the parity of characters, so that character codes greater than 127 may not be translated.

JS ROM key

The JS ROM has another, exceptional bug in its handling of the CAPS LOCK key. If you press CTRL and ESC at the same time on most QLs you get character code 128. ESC is not a letter of the alphabet so you would not expect pressing CAPS LOCK to have any effect on the code you get.

A sloppy comparison statement in the JS ROM means that CTRL ESC gives a code 160 if CAP LOCKS is in effect and code 128, as expected, otherwise. This is a very esoteric bug but it is worth noting if you are writing a program and planned to use CTRL ESC as a control keystroke. Code 160 normally is obtained by pressing CTRL SHIFT "2"

Simon Goodwin revealed 46 other bugs specific to particular QL ROM versions in the August, 1987 issue of QL World and listed 31 bugs in all QL ROM versions in the September issue. This list brings to 88 the number of published ROM bugs. Doubtless there are more, although we must have tracked most of the important ones by now. Ifyou have found others, please let us know.

BUGS AT LARGE

Tame hacker Simon Goodwin tracks another batch of bugs which afflict all QL systems, plus secret priority levels, and undocumented commands which let you use cursor-control and random access files on any unexpanded QL .

In three previous articles I have revealed the results of four years' research into the idiosyncracies of the QL built-in software, the Qdos operating system and the SuperBasic interpreter. To date I have found and explained 88 bugs in the OL system.

This is an update to my original articles, which were published in the August and September, 1987 and June, 1988 issues. Since then I have found 14 new bugs, bringing the total to 102. I have also found more about the Microdrive write- protection fault mentioned in September, 1987.

There is no sure definition of a bug. I have concentrated on quirks of the QL system software which cause apparently correct programs to give unexpected results, or no results at all. In some cases programs which should not work are prone to do bizarre things. I count these as bugs, although arguably they are undocumented features. Some of them are even useful.

All complex systems contain bugs. Hardware manufacturers are curiously shy about admitting them, apparently on the basis that what you do not know will not hurt you. This encourages naive users to place unreasonable reliance on inherently fallible systems. In fact, bugs are rarely a problem if you are forewarned. All most users want to know is how to get the result they need without getting into trouble.

Some system bugs may cause other programs to fail, so I have included technical information to help software developers guard against the most common problems by defensive programming in their own code. Where relevant, I have discussed the implications of the bugs for people intending to compile their programs.

This list deals with idiosyncracies inside the QL. Most of these concern the SuperBasic language or the associated collection of routines called Qdos. Some of the most obscure bugs occur in the second processor software which is programmed on to the 8049 IPC chip rather than the main system ROMs.

All QL programmers write new and original bugs as a matter of course but they are outside the scope of this article. So far as I can tell, all the new bugs occur on every version of the QL. The last five are concerned with the pair of linked QL processors. All the others stem from mistakes in the 48K QL system ROM.

(* Bug 89— RENUM *)

When you RENUMber a program the OL automatically adjusts references to line-numbers after RESTORE, GO TO and GO SUB commands. Unfortunately it does not alter the parameters of resident procedures which refer to line numbers; they include commands like RUN, SAVE, LIST and EDIT.

If you need to re-number a program with RUN in it, use GO TO [line] instead of RUN [line]. If you write programs which LIST or SAVE parts of themselves, you can not re-number them automatically. You must fix the LIST or SAVE statements. Unfortunately that is just the kind of program it is useful to be able to re-number.

(* Bug 90— NETWORKING *)

The QL is incapable of writing an empty ‘packet’ of data over the network. If you try to make it do this, the whole machine locks up. It is easier to run into this bug than you might expect. Imagine you have opened a network file in the usual way, with:

```
OPEN #3, NETO_1
```

Then you realise you do not want to write anything to the file, or have nothing to write yet. Being tidy-minded, you type:

```
CLOSE #3
```

At that point the QL freezes, trying to write a zero-length block. The same thing can happen if you supply the device name NETO_1 to a program which tries to write an empty file.

This bug was found by David Oliver of CST. His fix is not very elegant; the Thor XVI locks up for 30 seconds if you try to write a zero-length block, then normal service resumes, accompanied by an ‘Xmit Error’ report.

(*Bug 91 — TOP PRIORITIES *)

According to all the published documentation, QL task priorities are numbers between 0 and 127. The higher the priority of a task, compared to that of other tasks running at the time, the larger the share of available processing power used to run that particular task. It transpires that priorities greater than 127 are allowed. The system call MT.PRIOR (TRAP #1, DO11) accepts higher priority values, in the range 128-255.

Most people set priorities with toolkit commands like the Turbo Toolkit SET_PRIORITY or Tony Tebby’s SPJOB. SET_PRIORITY follows the book and rejects priorities greater than 127 but SPJOB passes the low byte of any integer value to MT.PRIOR. Thus you can use values above 127.

Negative values work, too. SPJOB 0,0,-1 sets the priority of SuperBasic (task 0,0) to 255 in the JOBS list. Other negative values —2 to —256 correspond to priorities from 254 down to zero.

LIST_TASKS in Turbo Toolkit shows non-standard priorities as negative values, whereas the SuperToolkit JOBS command shows them as positive. I tested the new priorities by running three compiled SuperBasic integer print loops at the same time. One ran at a priority of 255 and the others at the default priority, 32. The high-priority task received about five times as much processing time as each of the other two. This matches the results for similar ratios of conventional priorities, like 8, 8 and 63, so it seems that the non-standard values work as an extension of the normal range.

(Bug 92 — MICRODRIVE FORMATTING *)

Microdrive formatting fails at once with an ‘in use’ error if any Microdrive is running when you issue the FORMAT command. This can stop SuperBasic unexpectedly if you write to one drive and FORMAT another soon after, while the computer is still verifying files which have just been written.

You also run into trouble if you try to add files to a cartridge and fill it. You cannot re-format the tape to get some more space until the system has finished checking the data it managed to write, even though you have indicated that you want to wipe the cartridge.

To avoid the ‘in use’ report, check the system variable SV.MDRUN before issuing a format command. PEEK (164078) gives you the number of the currently-running Microdrive, or zero if no drive is in use. The procedure SAFE_ FORMAT, in listing one, will format any device, checking SV.MDRUN if necessary.

Listing 1 — Safe microdrive formatting.

```
DEFine PROCedure SAFE_FORMAT(device$)
  IF LEN(device$) > 4
    IF device$(1 TO 3)==”mdv “
      REPeat poll: IF PEEK(164078)=0 THEN EXIT poll
    END IF
  END IF
  FORMAT device$
END DEFine SAFE-FORMAT
```

(* Bug 93 — SCROLL quirks *)

Explained how the CLS command could recognise undocumented parameter values. Non-standard parameters are accepted and give results which seem bizarre at first but correspond to internal OL system calls.

The SCROLL keyword has a similar but superior feature. SCROLL does much the same thing as CLS but expects a second parameter eight greater than the CLS equivalent. For instance, SCROLL 0,24 has the same effect as CLS 16 which, in turn, works like the documented command POINT 0,0. All three call the system routine SD.POINT.

SCROLL is more useful than CLS because it accepts an extra parameter — the number of pixels to be scrolled. This is passed to Qdos in register Dl. Several other calls expect parameters in Dl, so we can use SCROLL as an alternative way to pass values to the system. For instance, try:

```
FOR X=0 TO 255,7: SCROLL X,17: PRINT “Hello”;
```

In this case the SCROLL command passes the value of X to SD.SETIN, with results which are interesting but not particularly useful. We can do better.

(* SECRET RANDOM ACCESS *)

Microdrives and discs allow random access to file data. You can wind back and forth through a file, re-reading or rewriting information, with no need to CLOSE and RE-OPEN the file every time you want to move backwards and no need to read intervening information as you move round a file.

This is very useful if you are writing a data-handling program, as it means you can extract data from anywhere in a file without the system having to fetch irrelevant data.

The TRAP #3 keys FS.POSAB and FS.POSRE are recognised by every QL and let machine-coders move the file pointer to any ABsolute or RElative position. There are apparently no SuperBasic commands to move the file pointer; you appear to need a Toolkit command like SET_POSITION.

This is a problem if you want other people to use your SuperBasic. You cannot rely on other users owning a particular toolkit. All toolkits soak some RAM and that is particularly precious on an unexpanded QL.

Packing

Hackers may be amazed to learn that many QL users are still struggling in 128K and they are in particular need of improved file-handling. Would it not be pleasant if we could find a way to use random access, on any QL, without a Toolkit?

It transpires that SCROLL can do the job. SCROLL #3,N% ,42 allows random access to a file open on channel 3 — on disc, RAM-disc, Winchester or Microdrive. This calls FS.POSAB, setting the file pointer to the value of N%. SCROLL #3,0,42 re-winds to the start of the file, SCROLL #3,1,42 points after the first character, and so on. Listing two illustrates a simple random access program. It was tested on a JS QL but should work on other models. The first line opens a file and the second fills it with 10 sample lines of data, each seven characters long including the 'enter' code at the end of each line. Then a SCROLL command re-winds the pointer to the start of the file without closing it.

Listing 2 – Random access without a Toolkit

```
OPEN_NEW #3,MDV1_TEST
FOR L=0 TO 9 : PRINT #3;"Line:";L
SCROLL #3,0,42 : REMark Rewind
REPeat show
  INPUT #3,a$
  PRINT #3,A$
  IF EOF(#3) : EXIT show
END REPeat show
REPeat scan
  INPUT "Enter record No. 0-9:";R
  IF R<>INT® OR R<0 OR R>9 : EXIT scan
  SCROLL #3,R*7,42
  INPUT #3,A$
  PRINT #3,A$
END REPeat scan
CLOSE #3
```

Loops

A REPeat loop is used to read and display each line until the end of the file is reached. The EOF function works well with random access files; you get an 'end of file' error, as you might expect, if you try to set the file pointer to a number greater than the total length of the file.

The final loop lets you select any record by number, using random access to find, read and print the appropriate line from the file. Type 10 to stop the program.

It is feasible to use SCROLL to position the pointer and PRINT new data into the middle of a file. The characters printed over-write the old ones at that position, so it is a good idea to use fixed-length records. The file is extended if you print at the end but you cannot insert characters in the middle of a file without over-writing what was there previously.

Unfortunately, SCROLL expects integer parameters, so you cannot use SCROLL 42 to move more than 32 down a file. SCROLL #3,n,43 might cure this by allowing relative moves with FS.POSRE. In practice it is rejected by the poor checking in the SCROLL keyword code and gives a 'bad parameter' error.

(* Bug 94 — PAN possibilities *)

Once I had investigated the SCROLL and CLS bugs it seemed worth checking PAN in case it allowed access to other system calls. PAN uses the same technique to convert its parameter into a Qdos trapkey but it adds 27 to the parameter value. PAN #c% ,0, 124 has the same effect as AT #c%,0,0.

All these keywords are meant to handle parameter values between 0 and 4, so they use shared code which checks the value and rejects it if it gives more than 4 when taken modulo 8. SCROLL and CLS add 32 and 40, so they both reject the same values, but PAN adds 27, allowing access to different system routines.

PAN lets you turn cursors on and off without a Toolkit. PAN 0,115 turns the cursor on in the default channel, while PAN 0,116 turns it off again. These instructions are vital when writing compiled multi-tasking programs which use INKEY$ or PAUSE; if a task does not display a cursor it cannot be selected for input with Control C.

To make PAUSE and INKEY$ work correctly, put PAN #0,0,115 at the start of a task. Note that the default channel for PAUSE and INKEY$ is #0; Sinclair did not reveal this.

Like SCROLL, PAN passes an extra parameter to Qdos in register Dl. We can use this to call FS.POSRE, passing a relative offset for the file pointer. You can access any part of a long file by using SCROLL 42 to get to a known place, then PAN 40 to move fowards or backwards from there. PAN #,N%,40 passes the integer N% to FS.POSRE. N% is the offset from the current position in the file, so:

SCROLL #3,30000,42: PAN #3, 20000,40

positions the file pointer after the 50,000th byte in a file. Use PAN 40 repeatedly if you need to wind more than 64K down the file.

It would be pleasant to be able to 'truncate' a file, discarding characters after a certain point so that space could be re-used. You may find that PAN #3,0,48 will truncate the file on channel #3 after the current position but this relies on the undocumented system — call FS.TRUNC, TRAP #3, D0=75.

Unfortunately, FS.TRUNC was a real afterthought rather than something which was not documented. Standard QLs do not recognise FS.TRUNC; you need the Sinclair QL Toolkit, SuperToolkit 2 or a disc expansion to make this call work. They include an extra command TRUNCATE, so there is not much point using the PAN version unless you want to be deliberately obscure.

(* Bug 95 — WINDOW parameters *)

WINDOW does not check the number of parameters you pass to it. Quanta members who discovered this bug hoped that the 'undocumented' parameters would allow extra control over windows but this is not the case. The WINDOW code uses only its last four parameters, plus the first one — the optional channel number — if the parameter list starts with a hash. The other values are ignored.

If you put extra parameters accidentally in a WINDOW command it can be difficult to determine what has gone wrong, unless you know about this minor bug. The keyword code could be fixed by adding a check on the value in D3 after calling the parameter-fetching subroutine.

(* Bug 96 — DLINE channel *)

DLINE allows an optional, undocumented channel parameter. If you put a hash and a channel number between the command and line details you can redirect the 'automatic listing' which normally appears in channel 2 when the program is changed. It is difficult to imagine how this could be useful.

(* Bug 97 CHARACTER CODES *)

The CHR$ function is meant to convert a number between 0 and 255 into a character with the corresponding code. In fact, it accepts any integer value from —32768 to 32767 as a valid parameter. The, resultant character depends on the value of the bottom eight bits of the integer.

This bug is unlikely to cause problems as it does not affect correct programs; it means that some technically-incorrect programs produce useful results. For instance, consider listing three, a useful snippet of code which compresses an integer value, X%, into a two-character string.

Listing 3 – Integer packing with CHR$

```
IF X%<0
  RETurn CHR$(256+X% DIV 256) & CHR$(X% MOD 256)
ELSE
  RETurn CHR$(X% DIV 256) & CHR$(X% MOD 256)
END IF
```

This is useful when packing numbers into fixed-length records in a file. If you did not compress the value and PRINTed it normally, it would occupy between two and seven characters, depending on the value. The packed representation uses a fixed length of two bytes for every value, saving space in most cases and making it easy to skip over values.

Note that the code must handle negative values of X% separately and the second character code must be reduced modulo 256, to ensure that the parameter value never strays from the range 0 to 255. You can manage with faster and simpler code, like this:

```
RETurn CHR$ (X% DIV 256) & CHR$(X%)
```

The first CHR$ expression takes negative values in its stride, because of the bug. The second part does not need the MOD because CHR$ ignores the top eight bits of its parameter.

The TURBO SuperBasic compiler has its own fast code for CHR$, to avoid the need to call the Sinclair slow resident function. This bug is duplicated deliberately in compiled code to preserve compatibility. The parameter of CHR$ is always an integer, so the code is not slowed by the need to handle the bug correctly.

Q-Liberator does not generate its own code if it can use an existing resident routine. In this case it is exactly compatible with the interpreter, because it calls the interpreter routines for every resident command or function it executes.

(* Bug 98 POKE PARAMETERS *)

The POKE and POKE_W commands have a similar bug to CHR$; again this quirk can sometimes be useful. To be compatible with the ZX Spectrum, the POKE and POKE_W commands let you store signed values as well as unsigned ones. For instance, you can POKE X,-1 or POKE-2 X,65530 even though strictly the parameters of POKE should be in the range 0 to 255, or —32878 to 32767 for POKE_W.

Ignores

All POKE commands accept any 32-bit long integer value as a second parameter without complaint. POKE ignores the top 24 bits of the value and POKE_W ignores the top half. The effect is that, like POKE_L, POKE and POKE_W allow any parameter values in the range plus or minus about two billion. For instance:

```
POKE 131072, 131074
```

stores the value 2 in the first byte of QL display memory. The low byte of the value 131074 is 2 and POKE ignores other bytes.

Once again, Turbo duplicates this bug for compatibility in compiled programs. Turbo code converts both parameters from floating point values, as SuperBasic has no ‘long integer’ data-type. This makes it slower than it would be if POKE and POKE_W worked only with integers but the Turbo code is still much faster than the Sinclair resident POKE routine.

(* Bug 99— BEEP INTERACTION *)

Very high-pitched notes produced with BEEP interfere with keyboard polling. If you use BEEP 0,0 to generate a continuous tone you will find it difficult to type-in anything else while the tone sounds. BEEP 0,1 is not bad but keystrokes are still lost while the beeping is active. The problem occurs because the keyboard is read by a program in the 8049 second processor.

The same program also generates sounds by sending pulses to the QL squeaker. When the 8049 is making a high-pitched note it does not have sufficient time between clicks to scan the keyboard.

There is no easy fix for this, as the faulty code is buried in the IPC, which includes ROM, RAM and processor all in one chip. NEC makes a user-programmable version of this chip, the 8749HC, but you will need a few specialised tools to disassemble, patch and re-program the IPC. In practice it is much easier to avoid sustained use of pitches 0 and 1.

(* Bug 100 — KEYROW *)

Another IPC bug is concerned with keyboard polling. The QL manual says it is tricky to detect three or more - key depressions with the KEYROW function but reassures the reader that SHIFT’, ALT and CTRL do not interact misleadingly with other keys. Unfortunately this is not true. Compware programmer Francesco Balena has discovered that the arrow keys can interfere with CTRL and ALT.

If the UP and LEFT arrow keys are pressed at the same time, a common event in games and joystick programs, the CTRL and ALT keys are indistinguishable. Normally KEYROW (7) returns a set bit for each of the keys but if you press UP, LEFT and ALT you get the same KEYROW pattern as for UP, LEFT and CTRL; in either case, both the bits for CTRL and ALT are set, even though only one of them is pressed.

The only way to avoid this problem is not to use diagonal cursor movements in conjunction with CTRL or ALT. This bug is probably a documentation error rather than a real coding mistake, as it would be rather inconsistent if the QL could cope with those key combinations without trouble.

(* Bug 101 — BEEP/KEYROW CRASHES *)

If a task if loaded or unloaded while the SuperBasic interpreter executes a BEEP or a KEYROW instruction, the whole computer is likely to crash. BEEP and KEYROW call MT.IPCOM, passing the address of a parameter table stored on the user A7 stack. This is bad programming because SuperBasic may re-locate the stack in memory at any time if memory is needed for other tasks.

If Basic moves during BEEP (IPC 10/ II) or KEYROW (IPC 9) the table address is invalidated and gibberish may be passed to the IPC. In the original Qdos documentation, designer Tony Tebby warned: "IPC communication is completely unprotected. The command must not contain any errors or the entire machine will hang up.

Interpreter

To fix the interpreter you must redefine the BEEP and KEYROW keywords. The machine code could be the same apart from a switch into supervisor mode — freezing multi-tasking — during the IPCOM call. The easiest alternative is to compile the program. This cures the problem because the stack of a compiled program never moves while a task runs.

(* Bug 102 — SERIAL OVER-RUN *)

Chas Dillon and Tony Price report a problem in the handling of serial queues. The second processor can get its pointers in a muddle, when running fast communications programs, if it is asked to do something else at the same time, like generating a sound or recognising a keypress.

Characters are delayed and jumbled inside the IPC, so that they reach the buffers in QL main memory out of order. It is not an easy bug to demonstrate as it depends on precise external timings but it is consistent enough to make fast serial input irritatingly unreliable. The bug occurs even if you use 'handshaking' hardware to regulate the flow of data.

If the IPC is disturbed while reading from a serial port it may lose several characters or introduce a 'lag' so that each new character received causes an earlier character to be passed from the IPC to the main processor.

A proper cure for this bug would involve re-programming the 8049. It might be wiser to circumvent this by building a dedicated QL serial port but that would still take a great deal of hardware and software effort. Alternatively, use a slow data rate like 300 baud or buy a Thor XVI.

(* UPDATE — MICRODRIVE WRITE PROTECTION *)

In my September, 1987 bug list I mentioned that write-protecting a Microdrive does not stop the system trying to write to it.

In fact, the OL tries to write the data eight times but each time the low- level code aborts because the tape is write- protected. This means the drive runs for a little more than a minute, then a 'bad or changed medium' message appears.

At the time I suggested that you might cure this problem by using IPC call 1 to test the write-protect status of the currently-turning drive.

I have since tried this and it does not work. IPC call 1 has a bug in it which means it always indicates that the tape can be written-to, even if it is write-protected. This is a really annoying bug, because you cannot re-program Qdos round it.

The only way to avoid spurious bad or changed medium errors is to ensure that you never try to write to a cartridge which has the plastic ‘write-protect’ tag removed. It is for you to check this, because the computer cannot check for you.

(* FUTURE BUGS *)

This is certainly not a definitive list of QL bugs, although it covers all the problems I have been able to analyse in detail. If you have extra information about these or other QL bugs, please share your discoveries by writing to me, care of *QL World*.

RIGHTING ROMS

The glut of Minerva versions has triggered further confusion about QL ROM differences. Simon Goodwin summarises.

There are at least 50 different QL-compatible operating systems in use these days: each version has its own bugs and features. This short series explores the changes in the latest roms, testing Thor XVI and Minerva updates, and uncovering more fiendish faults in old Sinclair roms.

Background

QDOS is the OL Operating System — the program that's there when you turn on the computer. Argos is the Thor equivalent. Both have a built-in programming language, SuperBasic.

QDOS and SuperBasic are supplied in preset read only memory chips (roms) or electrically programmable eproms. Internal codes identify each version. The 'SuperBasic version' is identified by initials revealed by the SuperBasic command PRINT VER$. The 'OS version' is found with the machine- code system-call MT.INF; it's usually a decimal value packed into four Ascii bytes.

QDOS was near 0.08 within Sinclairwhen MD Nigel Searle decided to launch the unfinished machine, boosting Sinclair's stock price and scooping Apple's long-previewed Macintosh. Three months later, the first QDOS systems were 1.00 and 1.01, corresponding to VER$ FB and PM. They were packed with bugs and needed a 'kludge board' to carry extra chips outside the computer.

Soon they were replaced by AH and JM, inside the box, with fewer bugs. These were QDOS 1.02 and 1.03, and some people still use them happily today. AH and JM roms have theirfaults but the core works well, and most users have had five or six years to get used to the bugs. I designed and wrote Supercharge on an AH QL without serious rom problems.

The next Sinclair release was the differently-bugged JS (1.10), followed by JSU, for the US market. Then came MG (1.13), a major improvement, supplied in chips marked Copyright 1985, rather than 1983.

MG roms come in national variants; a third letter indicates the nationality of the messages, keys and character-set in the smaller (16K) rom. MGF roms expect a French Azerty keyboard, MGI suits Italians, and so on. The extra letter replaces the decimal point in the QDOS number, giving 1 Fl 3 or whatever.

Sinclair released relatively few roms compared with the prolific output of QView and Thor International. These use reprogrammable eproms, so they have tended to release code as soon as it seemed to work. They're more cautious these days. and with good reason.

System rom changes are a potential minefield. QDOS was hurriedly written, and has since been hacked by a succession of Sinclair programmers. Many routines that could have been done better remain buried in official roms; enthusiasts with the time and the know-how are tempted to re-code them and release the result. That's fine, as long as the modified system does not fail where it used to do something useful.

QView's Minerva code is a 'fixed' system which improves upon Sinclair's JS, MG and the un-released 64K QDOS, code-named Tyche (fate, appropriately enough) shortly before the 1986 sell-out.

All Minerva eproms show VER$ as JSL1, but the current family have OS numbers in percentile steps from 1.61 to 1.82. PRINT VER$ (-2) shows Minerva's OS version. There have been 19 releases of Minerva since my original article a year ago.

Obsolete Thor 1 and Thor 20 systems purport to use QDOS 4.xx and 5.xx, but they're really based on Sinclair's JS. The 16 bit Thor XVI has gone through ten rom revisions since its launch in 1988. Argos has stabilised at 6.41, which has given me a chance to identify and isolate some of its peculiarities.

Minerva 1.82

From the start, Minerva has offered many advantages over early Sinclair roms; there is no room here to list these again. See my article in November1989, or Mike Lloyd's update in September 1990.

The price of the eprom has risen by £10 since the launch, but QView now supply printed documentation for Minerva, in the form of a 50 page typeset manual spiral bound with card cover. Early versions had much of the same text on disk, but the information in the Technical Guide is easier to find and more complete.

Since my last report QView have documented vectors fordata conversion, NEW, INSTR, arithmetic, parameter handling, serial port and microdrive control. The keyboard routines are now vectored, and there's a fast memory-mover, accessible with CALL. Even RANDOMISE takes an optional parameter, to make random numbers extra-hard to predict.

The accompanying disk includes several 1.5K files which replace messages, key assignments and with Sinclair national variations. Versions mimic MGF, MGG, MGD and MGY (Finland); there's also a driver for ABC's PC-style keyboard, with source code; QView offer to put these in eprom at nominal charge to anyone with a spare socket.

Reports now show the statement number as well as the line where errors occur. The manual says that the cursor is automatically positioned at the error location when a 'bad line' is re-presented, but I'm glad to say that feature has been removed from 1.82, as it stops ALT-ENTER from working as expected.

Minerva has gone through 22 incarnations at the time of writing. I bought the original 1.61, upgraded to 1.63, and received 1 .82 for review. I shall return my 1.63, with some trepidation. We have received anguished reports of Minerva bugs from readers and users. Some of these are genuine problems, but many people seem just confused. It is almost impossible to investigate bugs unless you tell me your rom version and provide a short example to illustrate the problem.

QL World has received a postcard from Hans-Peter Recktenwald of Berlin, who says: "There are some really annoying bugs in Minerva", but the 'serious' bug he cites is not a bug at all. He complains that the file pointer variable P is updated by Toolkit2 PUT command, so PUT#f\P,x% changes the value of P.

That is exactly what PUT is meant to do, according to Section 12.2 of the Toolkit manual. Minerva makes no difference. The variable is unchanged if you put it in an expression, like: PUT #f\(P),x%.

Basic tutor Mike Lloyd tells me he has reverted to the JS rom after serial port problems were exacerbated by Minerva 1.82. He found that occasional characters were corrupted en route to his printer, even under JS, but the problem became much worse when he switched to Minerva. It is

possible that the problem is a hardware/software interaction, as Mike has not tried the same set-up with a different computer.

In their documentation of Serial Drivers, QView admit “the current version still suffers from some problems”, but they have made progress. Sinclair’s serial driver used to put the wrong parity on the Control Z at the end of a file, and ignore it on input, but QView have fixed that.

QL task priorities range from 0 to 127. System designers Tony Tebby and David Oliver have told me that the higher priorities up to 255 should work, and some people use them deliberately: for instance Digital Precision recommended the use of priority 255 as a way of squeezing extra performance from Solution.

The common Toolkit extension SPJOB passes a parameter byte to the system without any checking. The parameter is read as an integer from -32768 to 32767, but only the least significant byte is transferred, reducing the value MOD 256. SPJOB 0,0,179 gives Basic priority 179, SPJOB 0,0,258 sets priority 2 and SPJOB 0,0-6 sets it to 250. This whacky rule works consistently on Argos and all Sinclair roms, but Minerva has other ideas.

Bytes can be signed or unsigned. LIST_TASKS shows signed values -128 to 127, while the similar JOBS command shows them as unsigned 0-255. Argos and Sinclair roms treat the undocumented priorities as a range upwards from 127, so a task at priority 255 runs faster than one at 127. QView think differently; they express the same range as -128 to -1 and assign it to background tasks. These only run in the absence of active tasks with ‘positive’ priorities, 1-127.

This change was not as helpful as it first may have seemed, as some users already use those priorities in a consistent way, but I can see the appeal of background tasks to anyone who does not find their most urgent tasks unexpectedly giving way to all others. Maybe QView will make this configurable, and default to compatibility.

The QL has its own way of sorting values. Sadly the rule used to put things in order has changed several times, and it is still not always right. Ordered lists can change sequence if you run your programs on a different rom. The correct sequence has never before been published; the ‘string comparison’ section in the Concepts part of the QL User Guide is wrong, and Jan Jones’ book skates over this topic in just seven lines.

I have spent long hours investigating this problem, and find that AH roms are subtly different from JM and JS, which are markedly different from MG and Minerva. Before I catalogue the differences I should explain further.

Most micros use the American Ascii sequence for ordering characters. This is fast, but not the same sequence as a librarian would use. Often Ascii lists end up in unexpected order after sorting. Ascii puts all the capital letters before lower case, so “IBM” comes before “lain”. Digits are compared one by one, from the left, so PCs think “$270” is less than “$28”.

QLs get the right answer in both cases, on all roms, because they follow a complicated rule to recognise imbedded numbers and treat small and CAPITAL letters equally. But bugs in roms up to JS mean that comparisons and INSTR searches may go wild if character codes are greater than 127. This affects accents, special characters, and comparisons meant to screen out control key-codes in the range 192-255.

QDOS users are understandably confused about the sorting order of the full stop, which often crops up in abbreviations, initials, and decimals. The AH rom treats "." as if it were a minimalist form of "0.0". JM and JS roms are similar, but consider that "." is slightly less than zero.

Argos, Minerva and MG roms put "." after the digits, letters, accents and control codes.

The MG collating sequence starts with digits from 0 to 9, followed by letters of the alphabet:A, a, B, band so on uptoz. Then cornesthe underscore,followedbytwelve accented pairs (capitals first), 16 accented vowels, Beta, and all remaining codes, including '.", in Ascii order from CHR$(0) to CHR$(255).

UK roms cope fairly well with standard codes from 32 to 127, but higher codes are treated increasingly oddly. JS quicksorts accented codes 128-191 first, followed by ".", digits 0 to 9 and letters from Aa to Zz, underscore, CHR$(0) to CHR$(45), /, then remaining Ascii punctuation from colon, CHR$(58), to © (copyright), CHR$(127).

Codes 192 to 255 are scattered among this lot in a bizarre way. CHR$(192) to CHR$(205) come as expected after CHR$(191), but CHR$(206) matches "." followed by 207, then digits with codes 208 to 217 interspersed. Next come 218- 224, then letters, with 225 to 250 interspersed. The remaining codes 251 to 255 fit in just before underscore, followed by the rest in Ascii order.

The Thor XVI follows the same rule as MG and Minerva, but it goes wrong on character-sets that have more than the usual number of small and capital equivalents. Argos can swap founts with alacrity, but it carries on ordering Greek and Russian alphabets as if they were standard QL characters, and fails to equivalence many characters in the extended second font.

It is good that the QL knows that "$4" is less than "$20", but the scheme can't cope with negative numbers in text. ODOS thinks "Balance -20" is less than "Balance -22". If you need to sequence signed values correctly you must extract the value from the string and compare numbers, not text.

JS and earlier roms go wrong if you write: IF KEY$ > CHR$(191), trying to select control key codes from 192 to 255. That tests succeeds for all codes below 128 as well as codes over 191. QDOS works as well as expected if you use an Ascii numeric comparison: IF CODE(keys$)>1 91.

QView ask for ideas that "add to the extendability of the system". I'd like to see a vectored comparison table: a pointer to 256 bytes giving the sequence number for codes 0-255. Thus anyone could re-order the set, bringing accents and un-accented characters together, or making codes equivalent by using the same sequence number for both. This would make QDOS more configurable, and could hardly make comparisons more confusing.

Minerva 1.82 has an improved ram test which displays the address of faulty ram in big letters on-screen ifaproblem is found. This test is faster yet more thorough than Sinclair's, so it may show faults that earlier roms let through. I think this is a good thing, but no ram test can be exhaustive. Faults may depend on the exact pattern of values in memory, and a mere128 bytes can hold 1 .797693E+308 distinct patterns — that's a 309 digit number!

Version 1 .82 reduces the delay before the system auto-starts to ten seconds, while early Minervae waited half a minute for a function key before starting anyway. I think the delay should be either nothing or infinite, depending on whether a BOOT file is found.

It is annoying to have to race to get a disk in the drive before Minerva starts without me, but I'd like the machine to re-start as soon as it's ready, if it can find a BOOT file. Ideally Minerva should check the drive twice — once during start-up and, if necessary later, when the user is ready.

If it finds a BOOT file during the rom start-up sequence it should load it without waiting for F1 or F2, like the Thor. Otherwise it should wait for a key, then consult the drive again to see if a BOOT file has appeared. At the moment we have to type LRUN FLP1_BOOT to get things going again if our disk misses the ten-second curfew.

Minerva programmer Laurence Reeves says he has fixed an obscure but nasty bug in the ATAN function on Argos and Sinclair roms. If the SuperBasic task is more than 32K long four bytes are stored in memory almost at random, every time ATAN is called.

The problem is similar to the CALL fault on early roms; ATAN includes some word addressing that misses its target if a large program is loaded. ATAN gets the right answer, but randomly corrupts your memory in the process. Minerva is the only known cure for this bug, buried deep in the QDOS maths package.

Benign Bugs

I'm pleased to see readers using undocumented CLS, PAN and SCROLL parameters uncovered in past rom reviews, controlling cursors or file pointers without need of any Toolkit. These work on all Sinclair roms, Minerva 1.61 and 1 .82, but QView 'corrected' the procedure code on Minerva 1 .63, so that the undocumented parameters were rejected. These 'benign bugs' are restored in later eproms, after howls of protest. Thor International have also rewritten the commands, so the tricks worked on early versions of Argos, but are rejected by Argos 6.41.

A mistake in Minerva 1 .63 meant that ADATE has the same effect as SDATE; any attempt to advance the clock by one hour took you back to lam on lst January 1961 . The cure is to add the current DATE to the ADATE parameter, or upgrade to a later eprom.

Some programs are hesitant on Minerva, Thor XVI and MG roms because they use repeated calls to ADATE 1 to wind the clock forward by a second under keyboard control. This is quite common in 'clock setting' utilities, though I prefer to type the digits.

Early Sinclair roms just bumped on the time by the required amount, regardless of when the clock was next due to 'tick', but MG and its followers wait up to a second for the next tick before advancing the time. This shows the difference:

```
10 REPeat 1: ADATE 1: PRINT DATE$
```

Later roms print just one line per second, with successive dates two seconds apart. Early roms produce a rapid stream of dates one second apart. Which is 'best' depends on your existing software, but it is good to be aware of the inconsistency.

A loop/parameter bug mysteriously stops some programs working on Thors and later Sinclair roms, even if they were fine on vintage AH and JM systems. This problem is well worth noting if you develop structured programs that may have to run on JS or MG roms. For once, spaghetti programmers can relax. You will never see the bug unless you use SuperBasic procedures or functions with two or more parameters.

It transpires that while any name has been used as the first of several parameters in a DEF PROC or DEF FN, it may not be used as a FOR loop name. If you try, JS and MG roms report a ‘bad name’ error.

This would not be too bad if LOCAL declarations made a difference, but they don’t. Listing 1 shows the problem as it usually appears. The bug just affects the first parameter name, and then only if there is more than one.

QL World December 1990 – ROMS RIGHTED

Listing 1:

```
100 REMark “MG / “JS”  parameter bug
110 REMark OL World December 1990
120 :
130 TEST i,j
140 :
150 DEFine PROCedure TEST(i,j)
160 LOOP
170 END DEFine
180 :
190 DEFine PROCedure LOOP
200 LOCal i
210 FOR i=1 TO 2:REMark Bad?
220 END DEFine
```

Listing 2:

```
100 REMark OUT & FOR parameter bug
110 REMark OL World December 1990
120 :
130 LET count=10 : limit=0
140 DIM values(count)
150 TEST total
160 PRINT”Totalled at “;total
170 :
180 DEFine PROCedure TEST(sum)
190 REMark Scan COUNT values for TOTAL
200 REMark Some imagination required....
210 FOR sum=l TO count
220   IF value(sum)=limit : EXIT sum
230 END FOR sum
240 END DEFine TEST
```

I found this bug while exploring the Quanta library. ChessSet_Bas on the MISC/DEMO disk stops with a spurious ‘bad name’ on all Thor XVI roms, QL MG and JS systems, but works perfectly on the AH and JM roms available when it was written.

The program is elegantly structured as lots of small, independent procedures. In deference to a common convention in mathematics (and Fortran) the author uses the identifiers I and J for temporary counts, often declared as LOCAL. The problem stems from DEF PROC DRAWTO(I,J), and vanishes if you rename I to XX between lines 900 and 1180. I can imagine X or A causing similar problems in other programs. This could be seen as a punishment for un-imaginative variable naming, but that's unfair. We should be able to use anything we like as a LOCAL identifier. If we can't, it's a bug. Ten years ago I worked with a programmer who used arbitrary animal names in rough alphabetical order to label all the routines in a big minicomputer assembler program. Imagination can take over!

I should define this problem in precise jargon. On MG and JS roms, the first identifier of more than one in an argument list may not be used as a FOR identifier, in that definition or any procedure or function it calls, even if that identifier is later declared LOCAL. This describes the problem precisely to those who understand the terms or can look them up in Jan Jones' Definitive SuperBasic book.

Minerva 1.82 does not have that bug, but it does show another peculiarity in parameter handling that may cause confusion on any QDOS or Argos to date. In theory, parameter values can be passed in, out or both ways when a procedure or function is called. In practice you can't pass a parameter OUT if it is used internally as a FOR identifier, unless it had a value before the call.

In other words, SuperBasic supports IN and IN/OUT parameters, but not OUT parameters that start life as FOR identifiers. Listing 2 shows how this can cause problems. The cure is to add a line LET total=0 (or any other value) at the start.

This is not a serious problem,and should not be confused with the other FOR parameter bug; it can be a bit disconcerting to find that a FOR loop prompts a 'bad name' report. Fix it by assigning a dummy value to the parameter.

QView have made one major change to the SuperBasic interpreter, with repercussions for many programming tools; thankfully they give a POKE to restore the old state of affairs; I wish they had set the opposite default. Like everything else in a SuperBasic program, numbers are 'tokenised'. That means they are stored in a standard internal form. Regardless of the number of digits, every number in a loaded program is held as a six byte floating-point value.

That's convenient, as the interpreter uses floating point numbers almost exclusively, and spends so much time in floating-point conversions that X%=X%+1 takes longer than X=X+1. Compilers prove that the QL can perform integer arithmetic much faster than floating-point, but the reverse is true for Sinclair's interpreter.

Lawrence took pains to ensure that Minerva used integer arithmetic where possible, and it irked him to think that X%=X%+1 should involve reading the six byte floating-point value '1' and telescoping it into a simple integer before the fast adder came into play. What's more, it slowed things down again.

Minerva 1.76 changed the rules, introducing integer tokens that Sinclair planned but never used. These save a little memory, taking four bytes for standard integers +/- 32K, and two for numbers in the range -128 to 127.

QView say that "programs using integer tokens run about 10% faster and take about 15% less space." Of course, that depends on the program. Big lists of byte DATA may shrink even more, though they will probably still be larger than corresponding Ascii SAVE files.

Integer tokens can slow some programs down: if you write X=X+1 the integer 1 must be converted to floating-point before it can be added to X. This may happen a lot unless you are happy to type per cent signs after integer names. Turbo and Q-Liberator have their own IMPLICIT directives so any name can be treated as an integer; many programmers prefer that approach. Turbo keeps track of the context of an expression, and uses the appropriate number formats to suit the program and minimise conversion. For instance X=X+1 uses floating-point 1,albeit in a fast packed 4-byte form. X%=X%÷1 generates an integer 1, for speed and conciseness; X=(X%+1)+X÷1 uses an integer inside the brackets, and a float outside.

Integer tokens cause problems for current Basic compilers. They also upset DIY Toolkit's REPLACE (Volume R/June 1988) and other program-scanners, like Supercharge, XREF and SuperBasic-C-Port.

QView warn that a technique shown in the Toolkit 2 manual is invalidated by integer tokenisation. Sinclair roms convert integers to floating point form if you add zero to them in BPUT. Try that on Minerva and you wind up four bytes short; the value remains an integer unless you add a floating-point variable or bias like 1E-555.

QView have themselves fallen foul of the new tokens. The built-in RENUM command ignores references to lines with numbers from 1 to 127; other numbers are correctly updated, but small integers stay the same. This could cause a lot of confusion. The cure is simple, and suits the BOOT file:

```
POKE \\212,128
```

This restores the Sinclair scheme, with all values in bloating point form, and compilers, scanners and REPLACE working perfectly. Phew!

Minerva 1.82 has improved PEEK and POKE routines, which use punctuation characters to specify address offsets inside SuperBasic or among the System Variables. Whole lists of values of the same size can be POKEd in one go. Turbo can't compile these variants, It does not call the rom to POKE or PEEK when it could use a single machine instruction to do the same, and avoid the great overhead of passing parameters to rom.

Turbo and Supercharge deliberately check the parameters of standard' procedures and functions against the documented syntax, and report typing mistakes like WINDOW 512,256,0 or SCALE 100,0,0,0. I think good compilers report every error they can find at compile time, rather than leave easy checks until execution. It's easier to do otherwise, but the compilers would make slower and less reliable tasks as a result.

Unfortunately this checking means that the new flexible parameters of the latest Thor and Minerva roms are not accepted. I have sent relevant details to Chas Dillon, who has the Turbo parser source; I understand that he has finished Turbo 3, with such features and hopes that DP will pay for it. The same goes for PDQL and SuperBASIC C-Port.

Early Minervae clash with common programs which POKE system bytes that Sinclair left free, but QView used. By the time Minerva arrived Eidersoft, DP, Sector and QJump had found and claimed these bytes for various purposes, so they were already causing contention.

The latest Minerva roms cure this with a new system area in space formerly occupied by QDOS channel table entries. Now you can only open 304 channels at once, on an expanded machine; Sinclair allowed 360. 128K usersget 'just' 112,which should be enough. Laurence puts the extra 224 bytes to

good use, with device linkages and other ram vectors, plus bytes to configure the cursor, fonts, messages, tokens, keys and VER$!

An apparent bug in Minerva's BLOCK routine causes spurious crashes on Lear Data Systems' new PCB Designer and the Flightdeck simulator. Both work fine on Sinclair roms, but fall foul of the latest Minerva 1.82; tasks abort, reporting an unexpected 'out of range' error. I have tracked thisdown to calls to SD.FILL (TRAP #3) and presume that Laurence has 'corrected' an internal check so that programs that used to work now report an error.

Unfortunately that's enough to clobber all the programs that used such parameters with impunity, and you can't go into a shop and ask suppliers if a programmer or compiler has used SD.FILL near its limits — you just have to trust to luck. I'm sure these are not the only cases that fail on Minerva 1.82, but several other programs that use BLOCK work fine.

The first versions of Minerva could get the 'owner' wrong when creating tasks, preventing tasks from chaining correctly. This bug seems to be fixed in 1.82.

I understand that earlier Minerva roms had a problem in the BORDER statement; Sinclair roms let all its parameters default, so BORDER alone resets the width of #1 to zero. Some Minervae need BORDER 0.

This must be an easy bug to introduce, because Argos 6.41 has just the same problem. The cure is to add an explicit width of zero, in interpreted Basic — or swap to your old roms if the error is in a task that you must use and cannot re-compile. Thor International are proud to tell me that Argos 6.41 computes COS (P1/2) to be exactly zero, rather than very nearly zero. Minerva returns 6.076917E-11 ,which is good enough for me, and NASA.

Eidersoft's long-gone ICE front end is incompatible with both new roms — it has never run on the Thor XVI, and the pointer is frozen on Minerva 1.82 after the pretty icons appear. I don't think this is a great loss, although a few people still use ICE to manage disks and copy files.

I understand that V1 of the Belgian art program Painter suits all QLs and Minerva roms, but the latest twelve screen version crashes as soon as you try to use the third screen. I have not been able to investigate this.

Minerva 1 .67 interacts badly with QJump's Toolkit 2. Normally the micro- drives are controlled by a 'linkage block' in rom, but Toolkit 2 replaces that with a ram linkage that allows microdrive files to be renamed, truncated, overwritten or flushed, as well as the standard operations. Unfortunately 1 .67 goes wrong because the Toolkit sees that Minerva's linkage block is already in ram, and thinks it has linked the new code already.

Later versions of Minerva include the new code, so it doesn't matter what Toolkit 2 does. It's possible to tell who has won by checking the date with WSTAT, after renaming a file. Toolkit 2 changes the update date, Minerva does not.

Minerva has character shapes for every possible character code from 0 to 255, but QView warn "don't rely on them" — they may be supplanted by new code. Some versions work by a trick that disturbs Speedscreen and several font editors. They store patterns for character codes 0 to 30 in the second font, immediately after the new patterns for codes 191 to 255.

Speedscreen presumes that a font is a single contiguous group of characters, and works faster accordingly, but this means character codes 0 to 30 appear as splodges unless Speedscreen is turned off. It makes no difference to Psion programs, Qmon, or other utilities which suppress control codes or show them in their own way, like The Editor.

Speedscreen can handle all 256 characters, at top speed, if you merge the entire character set into a new font and use that. QView has helped in this regard by putting the system font addresses in an easilyPOKEd place, so you can assign an alternative font to all new windows. Version "p' suits Minerva's twin-screen mode.

QView let RESPR work while tasks are running, allocating space at the other end of the system memory. This is usually convenient, but can cause problems, as RESPR has two uses. RESPR(0) returns the address of the start of Sinclair's RESPR area, which moves down as the area expands, while RESPR(n) reserves n bytes (rounding up to the next 512 bytes) from the RESPR area—or the heap, on Minerva.

Some programs in the Quanta library eliminate a variable by loading code like this:

```
LBYTES "FLP1_CODE",RESPR(512) : CALL RESPRO(0)
```

Don't try this on Minerva with tasks running; the code will end up on the heap, but RESPR(0) will not find it. Use

```
x=RESPR(512) : LBYTES "FLP1 _CODE",x : CALL x
```

or LINKUP from DIY Toolkit Volume H.

Minerva allows long strings of prefix operators like —3, or -~~3 (aka 4). Sinclair and Thor roms prohibit two prefix operators in succession, although they do allow

—3, and -(~~3), so this 'improvement' is minor.

Minerva 1.82 fails to link the functions in Turbo Toolkit when they are loaded. The manual says this is because the functions PROCEDURE and FUNCTION are illegal as they clash with rom keywords — but they work perfectly on earlier Minervae, all Sinclair roms, Thors and emulators. Turbo uses them in directives like GLOBAL FUNCTION and EXTERNAL PROCEDURE which declare routines shared between concurrent tasks.

Turbo Toolkit users who want to accommodate Minerva must change the names in every copy of the toolkit file. I replaced the Cs with Ks, for PROKEDURE and FUNKTION, but I wince every time I see it. I designed Turbo Toolkit, so I'm biased, but I find this change in Minerva rather silly.

If Sinclair had applied QView's 'law' the AH rom would have been unusable. It uses "INPUT" and "EOF" as keyword tokens 15 and 16, and also defines the same names in the Name Table, as a procedure and a function. Yet the resident extensions INPUT and EOF work as well on AH as on any other Sinclair rom, although they also appear as fixed keywords, listed on page 118 of Andrew Pennell's QDOS Companion.

SuperBasic was planned to have WHEN INPUT and WHEN EOF keyword clauses (like early Tandy laptops), but these were abandoned as Sinclair struggled with WHEN ERRor and WHEN Variable. Jan Jones' tokenisation routines can tell the difference by spotting the prior WHEN keyword, just as they tell GLOBAL PROCEDURE from DEFine PROCedure by context, and tokenise commas

differently depending whether they delimit subscripts or parameters. QView should restore Sinclair's code.

Minerva also prohibits 'illegal' characters in procedure and function names which Sinclair accepted. For a while this stopped Oliver Neef's epic Return to Eden running on Minerva, as it used resident names containing hairpins and dashes "<->"; CGH Services can supply an alternative version for Minerva 1 .81.

Six years' experience confirms that many corrections cure one program only to undermine another. You're no better off with the latest rom if it stops you using your old software. QView has a sensible policy of providing one free upgrade to users beset by bugs, but it's hard to know which version is 'best'. My favourites are QDOS 1.13, Minerva 1.64 or 1.82, Argos 6.39 or 6.41. By the time you read this, there may be new champions, but those five are pretty good.

As I explained last year, you need to take the computer apart to fit new roms or eproms. This is a fiddle, but it means you can put the old chips back if new problems

develop. Users with early UK roms AH, JM and JS will probably find their machine more reliable if they switch to Minerv, but they should bear in mind that not all the subtle changes may be improvemenls.

MG and JSU users must base their judgement on a personal assessment of Minerva's 'extras'. Some Sinclair variants are arguably better-tailored to users outside the UK. The price has risen but the extra interpreters, documentation and command improvements make Minerva very desirable to avid QL hackers.

Minerva 1.82 costs £40, including one free upgrade on request. Quanta members get £5 discount; overseas delivery £2.50 extra.

Recent Minervae can run extra interpreters, keeping SuperBasic tokens in other tasks besides the initial task (0,0). This feature was announced in the July 1990 issue of Quanta, duplicating the name of my MultiBasic feature in the March 1990 QL World, which was, incidentally, held up by the sale of Focus Magazines.

The confusion of names is misleading, as there are many differences between QL World MultiBasic and QView's extra interpreters; both take about the same amount of memory for extra tasks, but they have unique advantages and disadvantages. Luckily you can get the best of both worlds by using both at once.

DIYT MultiBasic tasks are statically allocated, like compiled tasks. They are created with SuperBasic commands UNLOAD, RESAVE and RELOAD. New MultiBasic's inherit the program, extensions, variables and screen of task 0.

DIY Toolkit MultiBasic means compiler users can avoid repeated slow re-loading, even if they want to use several programs in the course of a session. It also helps in software development, because you can save old versions and revert to them instantly if experimental changes make things worse. MultiBasic tasks can have long, user-chosen names so it is easy to indicate their development hierarchy.

MultiBasic works on all QDOS and Thor roms, without interpreter changes, so it can only interpret one token file at a time, but it lets them share control of the machine, offering 'co-operative multi-tasking' like Apple's Multi-Finder. Minerva rom tweaks mean it can swap between several token files preemptively. Multiple interpreters run as quite separate tasks, and tokens and values cannot be passed between them except through channels, POKE or PEEK.

Extra interpreters are dynamic, so they move around memory as programs load or run, expanding and contracting internal memory areas and other interpreters as well. To avoid constantly disturbing one another, they seem to grab spare memory in big steps when they need to expand. I wrote a minimal two-line recursive program to test ram allocation, and monitored it from another task. After a few seconds free ram fellfrom 382,464 to 263,168 bytes. The extra interpreter grabbed 119,296 bytes in one swoop, but I got it all back when I killed the task.

Extra interpreters can share or inherit resident procedures and functions, but each one starts empty, apart from a single string variable passed as a parameter to the task. Extra interpreters are created by running a small task:

```
EX RAM1_MULTIB_EXE;"flpl_boot>param"
```

This command creates an empty Super- Basic task, with the name "SB.<jobnum>", copies 'param' to the variable CMD$, and starts loading from "FLP1_BOOT". The task starts with only one window, accessible with #0 or #1, but you can pass channels as parameters, or open more as long as you're careful to do it in the prescribed order. CLOSE #0 kills the task.

CTRL-ALT-SPACE breaks into all SuperBasic interpreters except the normal one, task 0, which remains under the control of CTRL SPACE. This makes it hard to leave a program running continuously in extra interpreter while you adjust another, as they all stop whenever you need to break into any one.

The point is that Minerva gets rather unwieldy when you use more than one extra interpreter; when you break into one they all report not complete at once, and you must CONTINUE any that should not have stopped. Earlier Minerva versions used CTRL-ALT SPACE as a standard way to switch between tasks. This was a good idea as CTRL-C is redefinable and does not always work. Unfortunately this idea was dropped to make way for extra interpreters.

You can do most of the things in an extra interpreter that you can do in task 0, with a few exceptions. Protected programs like Fleet Tactical Command and Psion's QDraw only load into task 0. Extra interpreters simply vanish after loading the boot program.

Current Basic compilers only read tokens from task 0, so you can't use CHARGE or LIBERATE from an extra interpreter. Q-Lib3.3 is compatible with earlier Minervae but reports 'not implemented yet' if you try to QLOAD a file QSAVEd from an extra interpreter. QView provide a patched version of QLOAD that suits Minerva's second screen mode, but even that will not fast-load Basic into extra interpreters, which make LOAD even slower than usual.

SET_PRIORITY gives 'not implemented yet' if used from an extra interpreter, but SPJOB is OK. Turbo Toolkit BASIC_PEEKs seem happy, but always access task 0; DIY Toolkit's BPOKE and BPEEK give expression errors or 'overflow' in extra interpreters, but work as usual in Minerva's Task 0.

Multiple interpreters are great fun for hackers and code explorers; QDOS is much more resilient if you can Control C out of a smashed interpreter and into a new one, ready to load another if need be. There's a lot of scope for linked programs, and filter testing is made relatively easy, but production systems will continue to be built with SuperBasic compilers, to avoid the compatibility, speed and size limitations of multiple interpretation.

QView's achievement is impressive, but there is much work to be done. There should be a way to break into one particular interpreter, without upsetting all the other extra ones. Assembler programmers should have little trouble changing MULTIB_EXE to use meaningful task names. It would be nice to be able to move tokenised lines between interpreters or inherit programs that are already LOADed, to avoid slow re-tokenising. Of course, you can still do thatwith MultiBasic, but RELOAD only works into the initial SuperBasic task, even on Minerva.
