ARTIKEL SAMLING

SINCLAIR QL

OPERATING ROOM

One of the QL's problems was that Sinclair couldn't get the OS and Basic into the ROM Ralph Bancroft looks at the GST's new OS.

When Sinclair was planning the QL, it followed a sound industry practice of not relying on one team for the design of its operating system.

A not particularly well known Cambridge software house, called GST Computer Systems, was commissioned to write an operating system (OS) to an exacting specification. Its solution was a powerful multi-tasking OS that had many of the features of Unix yet, was capable of being put on ROM.

Unfortunately, it took up more than 32K of ROM space — the amount that the QL designers had put aside for the OS *and* the Basic language. Sinclair went to a fall-back option of an OS designed by an in-house team to overcome the space problem.

The rest, as they say, is history. Even by cutting corners and leaving out a few facilities, Sinclair's own staff couldn't squeeze the OS and Basic into 32K of ROM. As a result, the first QLs that came out had an extra 16K ROM cartridge hanging off the back.

GST has now released its OS under the name 68K/OS for both end-users and companies using the QL board in their own products.

Features

GST's 68K/OS is a powerful multi-tasking operating system that owes its origins to Unix and other minicomputer operating systems.

The multi-tasking capability allows you to run several programs at once. How many depends on the size of the program and available memory.

It also has a 'pipe' facility to transfer data from one program to another. Pipes can be used with 'filter' programs that reprocess data.

An example of their use is in text processing. The output from a text editor can be written to a named pipe that transfers the data to a text formatter which in turn sends the final output to a printer.

The microdrive filing system uses a series of neat tricks to speed access times. With regards to the QL, 68K/OS supports screen windows and bit-mapped colour graphics.

Installation

The operating system provided was easy to install on a 'dongled' QL. The dongle was removed and the two 68K/OS ROMs subsituted for Sinclair's ROMs. For later versions of the QL, GST will be providing a circuit board that plugs into the internal expansion port.

Documentation

This is at times complex and confusing. It comes in the form of a fat ring binder and includes a substantial programmer's reference guide.

The detailed reference guide would certainly be an essential aid to anyone wanting to get to grips with the workings of the OS. But I would have welcomed a better presented introductory guide with illustrations and screen shots for those who merely want to use the OS to run programs.

In use

The difference between 68K/OS and QL SuperBasic becomes apparent as soon as you power up. Instead of the usual TV or monitor choice of the QL, 68K/OS gives a choice of five screen formats: four colour/85 columns for use with monitors; four colour/80 columns for use with monitors that tend to clip the edges of the display output; four colour/60 columns for use on TVs; eight colour/42 columns for RGB monitors; and eight colour/40 columns for use on TVs.

A little experimentation is advisable to find the best option for your particular set-up.

Selecting the screen format runs a program called Adam, which is a menu driven command program that splits the screen into several multi-coloured windows.

These windows display a command line, default program menu (programs on ROM or selected microdrive tape), default data menu and the log.

This last screen seemed an unnecessary luxury. It lists all the programs that have been run since you powered up the machine and whether the program runs have been suspended or killed.

At the bottom of the screen is a status line used to indicate the options that can be selected using the function keys.

To use a microdrive tape it first has to be 'mounted'. This is done by specifying the 'md:' followed by the drive number and then the directory name. Once mounted its directory appears in one of the screen windows.

A program can be run by either writing it on the command line or moving the cursor down the program menu and hitting return. As befits its origins 68K/OS files are referred to by a comprehensive path name that includes device, directory, name and type. So a full path name could be something like md:GST/date.prog.

Fortunately, some of thse components are optional and others are automatically provided by the selection of default values. And despite the comlexity of 68K/OS I soon found it easy to use and certainly more friendly than say CP/M or MSDOS.

Having loaded up more than one program it is a simple matter of switching between tasks. For each program a single line window appears at the top of the screen. At times I found this annoying, like when you wanted to use the full screen for text editing or using GST's Draw program. However, it did help in keeping track of which programs were still running.

Verdict

GST's 68K/OS is the first affordable operating system for personal computers that combines professionalism with functionality. It is also the operating system that Sinclair should have made its first choice for the QL.

Being in ROM it is instantly available — no booting of disks required. The limitation is that not all the features have been squeezed into ROM. Copy, Date, Format, Print and Rename are all commands that are annoyingly on tape and not in ROM.

I would have thought that with GST having to produce a plug-in card to implement the OS on the QL, it should have gone to the extra expense of adding one or two extra ROM chips to make these commands readily accessible.

Of course, the biggest drawback of 68K/OS is the complete lack of applications software. GST has released an assembler and is planning a word processing program. It is also bundling with the OS a text editor and terminal program.

However, the real test is whether independent software companies release versions of their QL software to run under 68K/OS. In the longer term the operating system's success depends on whether other manufacturers take up the system.

In the meantime, keen machine code programmers who want to turn their QL into a proper multi-tasking micro will find that 68K/OS is well worth the investment.

REPORT CARD: 1 TO 5

Features	●●●●●
Documentation	●●●
Performance	●●●●●
Overall value	●●●●

Name 68K/OS **Application** Operating system **Machine** Sinclair QL **Publisher** GST Computer Systems Ltd, 91 High Street, Longstanton, Cambridge 0954-81991 **Price** £99.95, Assembler £39.95 **Outlets** Mail order.

Key to the QL

John Cochrane kicks off a new page for QL owners with first impressions and a look at the SuperBasic Rom

My first impression of the QL was that it looked just like the photos. Not surprising perhaps but it left me a little at a loss for what to do next.

Without an up-to-date manual I felt a bit hamstrung, so one of the first things I did was to *Peek* through the Rom to find the keywords available. This proved to be very useful and I have listed most of them below. This list will continue next week.

SuperBasic is by no means complete yet, and may well change as time goes by — this is presumably why no manuals were sent out with the machines. However, it seems unlikely that major revisions will be made at this late stage so the list given below should not prove too unreliable. The program which I used to obtain the commands is as follows:

```
100 BAUD 9600

110 OPEN #5;"SER1"

120 FOR N=15050 TO 15352,17161 TO 17216,19980
    TO 20050,

26980 TO 27950,32210 TO 32240,34276 TO 34450

130  M=PEEK(N)

140  SELECT M=32 TO 128:PRINT #5;CHR$(M);

150 END FOR N

160 STOP
```

Notice that I am showing off by using the very useful extended versions of *For*, the short version of *Select*, and the SuperBasic terminator *End For N*. The program would work perfectly happily with conventional Basic statements, but I enjoy exercising new-found programming skills. The number ranges in Line 120 refer to blocks of Rom which I had previously noted as containing things of interest, such as error messages and commands, by listing through the whole Rom to screen. Lines 100 and 110 set up RS-232 Port 1 for output to the printer at a baud rate of 9600. Watch out for Channel 5, however; I used it almost continuously for output to the printer but I think that it is usually used for the sound channel. You can use another number. Line 140 restricts print-out to those characters which may be of interest, ignoring unprintable characters.

The end result of this is a somewhat jumbled listing of words and miscellaneous characters. The list below gives my interpretation of that listing. I have left out those "words" which appear most unlikely and have marked those which I am most unsure of. I have guessed the function of many of the keywords so watch out for changes.

ABS (*a*) — Returns absolute value of *a*.
ARC (*x,y,a,b,c*) — Draws a part of a circle. I couldn't deduce what the last three parameters did.
ARC_R — A version of *Arc* using relative co-ordinates?
AUTO *n,m* — Allows automatic generation of line numbers from Line *n* in steps of *m*.
BAUD *n* — Sets baud rate *n* for use with serial ports.
BECOMES *n* — Sets baud rate *n* for use with serial ports.
BECOMES — Looks interesting but I couldn't get it to do anything.
BEEP *a,b,c,d,e,f,g,h* — Sound. Can leave off most of the parameters for simple sounds.
BEEPING — Don't know. Tests to see if sound still being generated?
BLOCK *n,x,y,w,h* — Fill block in window *n* defined by width and height at *x,y*.
BOOT, MDV1_*BOOT* — Load and run program *BOOT* from Microdrive 1.
BORDER *n,w,s* — Sets up a border around window *n* of width *w*, colour determined by *s*.
CALL *n* — Calls a machine code subroutine at memory location *n*.
CHR$ (*n*) — Gives the strong with code *n*.
CIRCLE *x,y,r* — Draws a circle at centre *x,y* of radius *n*. May allow additional parameters *e,a* for eccentricity (0 to 1) at an angle *a*.
CIRCLE_*R* — Similar to CIRCLE but using relative co-ordinates?
CLEAR *n* — Clears the variable space. it accepts *n* but I don't know what it does.
CLOSE #*n* — closes channel *n*.
CLS *n* — Clears window *n*.
CODE (*a$*) — Gives the Ascii code of the first character of *a$*.
CONTINUE — Resumes program running after break.
COPY *a* TO *b* — Copies a named set of data from one channel to another.
COPY_*N* — Don't know.
COS (a) — Trig.
ACOS (n) — Trig.
COT (a) — Trig.
ACOT (*n*) — Trig.
CSIZE *w,h* — Sets display character height and width. I couldn't get this to work.
CURSOR *n,x,y* — Re-positions the print location in window *n*.
DATE — Gives stored date and time. Rumoured to be for the chop.
DATE$ — ?
ADATE — ?
SDATE *d,m,y,h,m,s* — Resets internal clock.
DATA — Used to store variables-data within a program (See *Read*).
DAY$ (*n*) — Returns a day of the week corresponding to *n*.
DEFine FuNction, DEFine PROCedure — Start of SuperBasic function or procedure.
DEG — Converts radians to degrees?
DELETE — Used for deleting files on a Microdrive cartridge.
DIM — For dimensioning arrays.
DIR "MDV*n*_" — Lists the files on Microdrive *n*.
DIV — Integer divide. (I'm suspicious that integers are not implemented fully, if at all. I didn't have time to check).
DLINE — Deletes program lines. Can delete single lines or blocks of lines.
EDIT *n* — Fetches line *n* for editing.
ELLIPSE *x,y,r* — May have more parameters, seems to work as *Circle*.
ELLIPSE_*R* — Relative co-ords?
ELSE — Used in long form of *If*.
END FOR, END DEFine, END SELECT — Used as a terminator.
EOF — Used to send end of file marker to a given channel?
ERRor — Probably not implemented yet, usually would expect this to be used with *On Error Goto*, etc.
EXEC — To load a sequence of programs and run them in parallel.
EXEC_W — As EXEC but waits for 1st program to finish before returning to the command level.
SEXECS — Used to save *Exec* to Microdrive?
EXIT — To exit a program construct such as a *Repeat* loop.
EXP (*n*) — Exponential.
FILL — Presumably fills a shape with colour. I couldn't get any response.
FILLS$ — ?
FLASH — Causes colour flashing in low-resolution mode.
FOR *n* = — Sets up start of loop with variable *n* taking values defined by the remainder of the line. These can be single values separated by commas and/or a range of values to be stepped through.
FORMAT — For setting up blank Microdrive cartridges for the storage of programs and data.
GOTO *n* — Transfer to line *n* of a program (frowned on by SuperBasic).
GOSUB *n* — Start subroutine at Line *n*.
HRESPR — If this is a command I don't know what it may do. I was hoping that it might allow a high-resolution copy of a window to be sent to a printer but I couldn't get it to do this.
INK *n* — Sets foreground colour.
INKEY$ — Returns the character last pressed at the keyboard (or from some other channel). Watch out for this one, the keyboard works through a buffer and INKEY$ gets data from the buffer. Thus if several keys are pressed between INKEY$ the value returned will be the first key pressed and so on until the buffer is empty, not the current key being pressed (See *Keyrow*).
INT (*n*) — Truncates *n* to leave an integer.
INSTREN — Not at all sure if this is a valid keyword.
INPUT — Allows input of data from a specified channel or the keyboard.
KEYROW (*n*) — The keyboard is set up as "rows" of eight keys, this command returns a number indicating the keys pressed inrow *n*. Does not work through the buffer and so is useful for real-time applications.
LBYTES — Load machine-code from microdrive?
LEN (a$) — Length of a string.
LET — Start of a Basic assignment (optional).
LINE *x,y* TO *x,y* — Draws a line from one point to another.
LINE_R — A relative version of *Line*?
LIST — Lists program lines.

continued next week

Key to the QL — part 2

***John Cochrane** continues his look at the SuperBasic Rom*

This week I'll continue the list of keywords available.

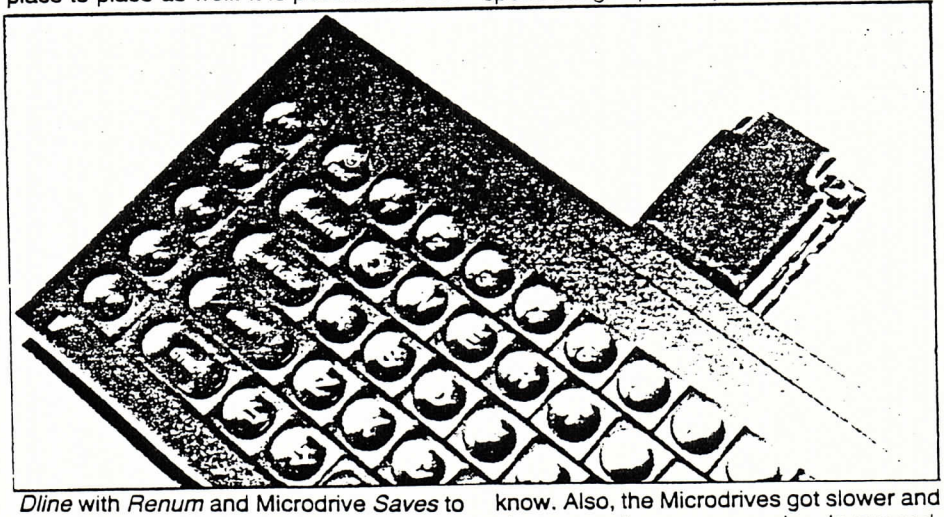
LOAD — Loads a program from microdrive.
LOCal — Specifies a set of variables to be used within a *Define Procedure* or *Function* which are separate from any globally based variables.
LOG10 *(n)* — Log to base 10.
LRUN — Load and run a program from Microdrive.
MERGE —— Merge a program from Microdrive.
MISTake — Another intrigue with no answer from me.
MOD *(n)* — Modulus?
MODE *n* — Modulus?
MODE *n* — Sets display mode to high or low resolution. *n*=256 for low resolution, *n*=512 for high resolution.
MOVE *n* — Turtle graphics. Move forward *n* units.
MRUN — Merge and run?
NET *n* — Used to define source/destination when using Sinclair Net?
NEW — Clears program from memory.
NEXT — Used as loop end in a *For* construct, can be followed by additional statements and an *End For* for a more complex structure than is available with other Basics.
OPEN #*n* — Attaches a device to Channel *n*.
OPEN_IN — To input data as a pseudo-random file from Microdrive.
OPEN_NEW — Sets up a pseudo-random (or possibly true-random?) file on a Microdrive cartridge for the first-time storage of data. Subsequent data saves use *Open*.
OVER *n* — Not the same as Spectrum *over*.
PAN *n* — Moves screen *n* pixels to the right.
PAPER *n* — Sets background colour.
PAUSE *n* — Waits *n* times twenty milliseconds.
PEEK *n* — Value of byte at memory location *n*.
PEEK_L *n* — Ditto for long-word (4 bytes).
PEEK_W *n* — Ditto for word (2 bytes).
PENDOWN — Turtle graphics. Commences drawing sequence as turtle moves.
PENUP — Turtle graphics. Halts drawing sequence as turtle moves.
PI — 3.142 . .
POINT *x,y* — Plots a pixel at co-ordinates *x,y*.
POINT_R — A relative co-ordinate version of *Point?*
POKE *n,m* — Sets byte at memory location *n* to *m*.
POKE_L — Ditto for long-word (4 bytes).
POKE_W — Ditto for word (2 bytes).
PRINT — Send character data to screen or other specified channel.
RAD — Degrees to radians conversion?
RAND — Not implemented but exists as word in Rom.
RANDOMISE — Sets seed for random number function.
READ — Should read data from *Data* statements but instead gives "not implemented" message.
RECOL — I don't know what this is but I see Andy Pennell said something about colour palettes.
REMAINDER — Catch all at end of *Select* structure.
REMark — Starts a comment line.
RENUM *n,m* — Renumbers the program from the first line to start from *n* and increment in steps of *m*. Default 100,10.
REPeat — Starts a general program loop, terminated by *End Repeat*, jumped out of by *Exit*.
RESTORE *n* — Sets line for the reading of data from *Data* statements (See *Read*).
RETRY — Don't know.
RETurn — Jumps out of *Procedure* or *FuNction*.
RND *(n,m)* — Gives random number (0-1) if no parameters or random integer between *n* and *m*.
RUN *n* — Runs program from line *n*.
SAVE — Save program to Microdrive.
SBYTES — Save machine-code to Microdrive.
SCALE *n* — Sets scaling factor for plot commands.
SCROLL *n* — Moves contents of window up by *n* pixels. Can define whole or part of a window for scrolling.
SELect — Structure for multiple-choice programming.
SIN*(n)* — Trig.
ASIN*(a)* — Trig.
SQRT *(n)* — Square root of *n*.
STEP — Defines step interval in *For* range.
STOP — Terminates program execution.
STRIP — The provisional manual indicates that a stripe effect is available (fine matrix of contrasting colours) but I did not test this out.
TAN *(a)* — Trig.
ATAN*(n)* — Trig.
THEN — Used with *If*. The provisional manual implied that this need not always be used but I never found an occasion when it was not required.
TURN *a* — Turtle graphics. Turn clockwise through angle *a*.
TURNTO *a* — Turtle graphics. Turn to absolute angle *a*.
UNDER *n* — If set to one then display all printed text with underline until reset to 0.
VER$ — ?
WHEN — Still to be implemented.
WIDTH — ?
WINDOW — Defines size and position of a screen-display window.
XOR — Logical exclusive Or.

I am very glad to see Renum but would like to see it extended to allow blocks of program lines to be copied or moved from place to place as well. It is possible to use *Dline* with *Renum* and Microdrive *Saves* to move blocks of program around but this is a little long-winded. The *Trace* command mentioned in the provisional manual has not been implemented (along with *Step*) which is a great pity, especially as the programs which I write tend to attract bugs like nobody's business. Turtle graphics are a bit of fun, it will be interesting to see if Sinclair provides the hardware add-ons to drive real turtles.

There are one or two bugs present in the machine, which unfortunately make it very difficult to do much more so far than play around with SuperBasic. For example, using the string-slicing features so beloved of previous Sinclair Basics can lead to program crashes, as can trying to enter a program of more than about 300 lines, as can trying to list a line of more than about 90 characters, as can letting the infuriating automatic-listing-on-line-edit run to the end of a listing! The *Alt* key is supposed to switch between an insert and a replace mode dueing line editing but does not. There seems to be some obscure interaction between some operations which can cause problems, for example I had problems with *On n=Remainder* and *Print* when used together.

Without a reliable manual it is difficult to be sure that any problems that come up are not the misuse of perfectly good functions. Some things are obvious, others are not. Along with the keywords given above I also listed the error messages ("not implemented" is a bit of a give away) and a list of days of the week and months of the year. Typing *Print* DAY$(3) gives a printout of "wed". Typing *Print* MONTH$(4) gives an immediate crash.

The QL is not a spectacularly fast machine, at least not as fast as I had been hoping. It compares favourably with the Apricot (as an example of a contemporary 16-bit microcomputer) when running calculations but the display is slow — very comparable to the Spectrum. This is no doubt partly due to the windowing and scaling facilities but is a bit of a let down. I am told that Sinclair Research are trying to speed things up but by how much I do not know. Also, the Microdrives got slower and slower as the program size increased. They work through some form of buffer, which makes some operations very fast – much faster than floppy disc operations – but I suspect tht the buffer size is not correctly set which leads to additional delays as either data is read into and out of memory or the buffer is expanded.



This brings me to a rather important observation. I was expecting so much from the QL that I was inevitably disappointed with the machine when I at last got my hands on one. This was not rational and it is only now that I have had time to sit back and think about it that I appreciate the value that is offered by the beast.

To pull my thoughts together. The lack of programming speed and the number of bugs still remaining is disappointing, but the facilities offered by the SuperBasic language are comprehensive.

An obvious answer

SuperBasic has many of the powerful features of Pascal including recursive techniques. ***Alan Turnbull*** *takes a look*

One of the most interesting and useful features of the QL for me is its structured programming.

I have been used to programming in Pascal for several years, but now I can get most same features in QL SuperBasic — although the data structuring facilities of Pascal are not available on the QL.

As well as structured loops, the programmer can construct procedures and functions, totally transparent in purpose to the user — just as in Pascal, and use them as if they were part of the SuperBasic language.

A special class of procedures and functions — referred to as 'recursive' — are of particular interest to programmers. A recursive object is one which is defined partially in terms of itself. Recursion is *not* the same as one of those 'circular' arguments you start in the pub when you have had one too many! In describing an object recursively, we begin by describing a simple case directly. Solutions to others more complicated are then found in terms of the solution to the simple case.

Unfortunately, early programmers, especially Cobol programmers in the 1960's, looked upon recursion as an ivory tower plaything and ignored it. But recursion, if used wisely, can give the most obvious solution to a complex problem.

Consider the mathematical definition of a factorial. The factorial of *n* written as *n*!, is defined as the product of all integers from *n* down to 1. For example 6! = 6×5×4×3×2×1. There is also a special, so-called 'base case', 0! = 1, which is defined to be true.

From this a recursive definition of factorial can be formed. Factorial(0) = 1 and factorial(n) = n×factoirial(*n*-1). A SuperBasic function can be written directly from this definition — it is shown in Fig.1. Note that the SuperBasic definition looks just like the mathematical definition.

```
100 REMark Factorial function for
    the Sinclair QL
110 REMark  July 1984, Alan
                       Turnbull
120 :
130 DEFine FuNction FACTORIAL(n)
140   IF n=0 THEN
150     RETurn 1
160   ELSE
170     RETurn n*FACTORIAL(n-1)
180 END DEFine FACTORIAL
```

Fig 1

From this start with an easy and well-known example, one can progress quite a long way. Strictly for the mathematicians among you a function exists called Ackermann's Function which is used mostly to illustrate recursion to students! It may be summarised as follows:

ack(0,n) = n+1
ack(m,0) = ack(m−1,1)
ack(m,n) = ack(m−1,ack(m,n−1))
for m,n >= 0.

Fig. 2 shows a SuperBasic listing to implement this function. Just try to implement it in Basic on the ZX Spectrum! Indeed, the only way to do it would be to explicitly use a stack and even then, problems of variable clashes would arise.

```
100 REMark Ackermann's Function
          for the Sinclair QL
110 REMark        1984, Alan Turnbull
120 :
130 DEFine FuNction ACK(m,n)
140   IF m=0 THEN
150     RETurn n+1
160   ELSE
170   IF m<>0 AND n=0 THEN
180     RETurn ACK(m-1,1)
190   ELSE
200     RETurn ACK(m-1,ACK(m,n-1))
210   END IF
220 END DEFine ACK
```

Fig 2

Onto procedures and Fig. 3 shows a SuperBasic procedure to sort a numerical array. So what?, you may say. Well, this sorting algorithm is different from the ones you will be used to. For a start, it is not the good old 'bubble sort', which is thrashed out in many articles and secondary school computer studies courses.

```
100 REMark QUICKSORT algorithm for
    the Sinclair QL
110 REMark  July 1984, Alan Turnbull
120 :
130 DEFine PROCedure QUICKSORT
    (array,left,right)
140 LOCal i,j,median,temp
150 IF left<right THEN
160   LET i=left:LET j=right
170   LET median=array((left+right)
      DIV 2)
180   REPeat main_loop
190     REPeat expand_s1
200       IF array(i)>=median THEN
          EXIT expand_s1
210       LET i=i+1
220     END REPeat expand_s1
230     REPeat expand_s3
240       IF array(j)<=median THEN
          EXIT expand_s3
250       LET j=j-1
260     END REPeat expand_s3
270     IF i<=j THEN
280       LET temp=array(i)
290       LET array(i)=array(j)
300       LET array(j)=temp
310       LET i=i+1
320       LET j=j-1
330     END IF
340     IF i>j THEN EXIT main_loop
350   END REPeat main_loop
360   IF left<j THEN QUICKSORT
      array,left,j
370   IF i<right THEN QUICKSORT
      array,i,right
380 END IF
390 END DEFine QUICKSORT
```

Fig 3

It is 'Quicksort', developed way back in 1962 by Professor Tony Hoare at Oxford University. Why have you not heard much about it, then? The answer lies in the fact that a good many Basic dialects cannot support the concept of recursive procedures and functions, and local variables. BBC Basic can, and so too can Sinclair QL SuperBasic. So the six-month wait was worth it, after all!

The algorithm may be summarised in English as follows:

PROCEDURE quicksort(s)
BEGIN
IF s contains more than one element THEN
BEGIN
choose the median element *x* from *s*,
partition *s* into sequences *s1*, *s2* and *s3*
with elements
less than or equal to *x*, equal to *x* and
greater than or equal to *x*;
quicksort(s1);
quicksort(s3)
END
END

To call the procedure, you should state its name in a SuperBasic statement, with three parameters: the array you wish to sort, the lower bound of the array and the upper bound. A typical call to sort an array dimensioned as DIMension array_1(100) would be quicksort array_1,1,100

In Fig. 3, note that *LET* statements are used throughout. SuperBasic defines the keyword *LET* to be optional but I have left it in to aid clarity. The Pascal *Repeat/Until* and *While/*DO loops are both replaced in SuperBasic by the *REPeat/Exit/END REPeat* construct.

It is easy to see that the *While* and *Repeat* constructs in Pascal, and the *REPeat* construct in SuperBasic compare in the following manner:

WHILE(condition)DO	REPeat (loopname)
BEGIN	IF NOT (condition)
	THEN EXIT (loopname)
statements	statements
END;	END REPeat (loopname)
REPEAT	REPeat (loopname)
statements	statements
UNTIL (condition);	IF (condition) THEN
	EXIT (loopname)
	END REPeat (loopname)

Note that the *Quicksort* algorithm is very fast (subject, of course, to the speed of the system it runs on) and that it may be used as a direct command or as a program statement. This could perhaps, be a procedure you may wish to place in a file called *Boot*, on MDV_1, so that upon the QL bootstrapping, it loads the procedure into memory ready to be used. This way, one could develop a QL turn-key system!

I hope this article proves to be a good introduction for readers to the concept of recursion in programming.

Tal i () = version JM

The Rom approach

Alan Turnbull reveals the dark secrets of the QL ROM

One of the first things the new QL owner must surely do is examine the read-only memory (Rom) of his or her machine and find out how the whole thing works.

With the Sinclair QL, this may prove difficult as there are at least four versions of the machine in existence: code-named 'FB', 'PM', 'AH' and the latest, 'JM'.

Version 'FB' was in a sorry state with Rom bugs too numerous to mention. Version 'PM' was a vast improvement but, Sinclair said, 'AH' was to be the final Rom.

As ever true to their word, Sinclair brought out a new Rom called 'JM', which "puts right all major Rom bugs, implements multi-tasking and makes SuperBasic much faster". Apparently, all customers will be offered an up-grade to this version by a strict 'recall operation' which involves you posting your precious QL off to Camberley, and Sinclair engineers plugging in the new Rom chips.

Meanwhile, if you are lucky enough to own the quite respectable version 'AH' (and you can find out by typing *PRINT VER$* at your console), this article may prove very useful if you want to reveal the dark secrets of the QL Rom.

The program in Figure 1 gives a tabulated 'dump' of two special tables in the QL Rom. The first table, residing at address 26652 in version 'AH', lists all command keywords and their run-time module address. The second table at address 27328 lists all function keywords and their run-time module address.

Each table is held identically in the following format:

›number of entries‹

›first module address offset‹
›number of characters in first keyword‹
›first keyword‹

›second module address offset‹
›number of characters in second keyword‹
›second keyword‹

and so on.

The SuperBasic procedure *Tabulate* in Figure 1 automatically tabulates on the QL screen any table held in this format, given its start address. Suitable alteration will allow output through the serial ports to a printer.

The output from the program is shown in Figure 2 and consulting this list and looking through the Rom routines at the addresses given may reveal many secrets.

For instance, any of the commands which take zero or optional parameters, such as *Run*, *List*, *Renum*, *Pause*, etc, may be called directly from SuperBasic using the Call command! For example, to list all of the current program in memory type *Call 28036*.

In fact, if you wish to call your own machine-code routine in Ram from SuperBasic, you should make sure the MC68008 data register D0 holds zero before doing a RTS. Also, A6 should not be altered: it is used by SuperBasic and QDOS as a pointer, similar in function to the IY index register on the ZX Spectrum.

If D0 holds a number between 235 and 255 inclusive, the QL will use this as an error number. D0 = 235 gives *"Bad line"*, 236 gives *"Read only"* and so on up to 255, which gives *"Not complete"*.

Calling routines like *List* is of no direct benefit — just illustration. But given these Rom routine addresses, the adventurous programmer could find out how to *Load* and *Save* Microdrive files or draw ellipses from machine code.

It must be noted, however, that whilst some of the command routines may be called directly, the function routines cannot. This is because the result of each function is placed in an area of Ram analogous to the ZX Spectrum's 'calculator stack', ready for picking up by the expression evaluator, and hence no return is made to the SuperBasic user.

It is hoped, nonetheless, that readers will find the routine and output presented in this article useful and that they, too, will delve into the secrets of the QL Rom.

```
100 REMark Program to tabulate routine addresses in QL ROM.
110 REMark     (c) COPYRIGHT August 1984, Alan Turnbull.
120 :
130 MODE 512
140 CLS
150 CSIZE #0,1,1
160 PRINT #0;"Use CTRL & F5 keys together as 'toggle' to pause output."
170 UNDER 1:CSIZE 2,1:PRINT "COMMAND ROUTINE ADDRESSES":CSIZE 0,0:UNDER 0
180 PRINT
190 TABULATE 26652      (26274)  (JM)
200 PRINT
210 UNDER 1:CSIZE 2,1:PRINT "FUNCTION ROUTINE ADDRESSES":CSIZE 0,0:UNDER 0
220 PRINT
230 TABULATE 27328      (27400)  (JM)
240 CLS #0
250 PRINT #0:"Program finished o.k."
260 CSIZE #0,0,0
270 STOP
280 :
290 REMark Procedure to tabulate ROM table
300 :
310 DEFine PROCedure TABULATE(table_address)
320 LOCal padding$,number_of_entries,ROM_address,entry_number,offset,routine_add
ress,number_of_characters,keyword_character
330 LET padding$=FILL$(" ",12)
340 LET number_of_entries=PEEK_W(table_address)
350 LET ROM_address=table_address+2
360 FOR entry_number=1 TO number_of_entries
370   LET offset=PEEK_W(ROM_address)
380   LET routine_address=ROM_address+offset
390   LET ROM_address=ROM_address+2
400   LET number_of_characters=PEEK(ROM_address)
410   LET ROM_address=ROM_address+1
420   FOR keyword_character=1 TO number_of_characters
430     PRINT CHR$(PEEK(ROM_address));
440     LET ROM_address=ROM_address+1
450   END FOR keyword_character
460   PRINT padding$(1 TO 12-number_of_characters);
470   IF PEEK(ROM_address)=0 THEN LET ROM_address=ROM_address+1
480   PRINT routine_address,
490 END FOR entry_number
500 PRINT
510 END DEFine TABULATE
```

Figure 1

```
COMMAND ROUTINE ADDRESSES

PRINT     28586 RUN       30232 STOP      30334 INPUT     28584 WINDOW    30646
BORDER    30684 INK       28364 STRIP     28368 PAPER     28372 BLOCK     30660
PAN       28406 SCROLL    28410 CSIZE     24756 FLASH     26026 UNDER     26020
OVER      26048 CURSOR    24792 AT        24806 SCALE     26100 POINT     26118
LINE      26136 ELLIPSE   26160 CIRCLE    26160 ARC       26240 POINT_R   26122
TURN      30416 TURNTO    30408 PENUP     30474 PENDOWN   30478 MOVE      30492
LIST      28036 OPEN      25926 CLOSE     25892 FORMAT    25714 COPY      25740
COPY_N    25744 DELETE    25570 DIR       25576 EXEC      25246 EXEC_W    25250
LBYTES    25360 SEXEC     25414 SBYTES    25418 SAVE      25964 MERGE     30270
MRUN      30230 LOAD      30312 LRUN      30318 NEW       30330 CLEAR     30220
OPEN_IN   25930 OPEN_NEW  25934 CLS       29402 CALL      24540 RECOL     29536
RANDOMISE 29318 PAUSE     28490 POKE      28526 POKE_W    28534 POKE_L    28540
BAUD      24308 BEEP      24368 CONTINUE  30404 RETRY     30394 READ      25200
NET       28336 MODE      28308 RENUM     29628 DLINE     28006 SDATE     25006
ADATE     24936 LINE_R    26140 ELLIPSE_R 26164 CIRCLE_R  26164 ARC_R     26244
AUTO      29582 EDIT      29578 FILL      25990 WIDTH     30624

FUNCTION ROUTINE ADDRESSES

ACOS      30860 ACOT      30866 ASIN      30872 ATAN      30878 COS       30884
COT       30890 EXP       30896 LN        30902 LOG10     30908 SIN       30914
SQRT      30920 TAN       30926 DEG       30932 RAD       30938 RND       31010
INT       31110 ABS       30970 PI        31096 PEEK      31134 PEEK_W    31142
PEEK_L    31152 RESPR     31186 EOF       31220 INKEY$    31274 CHR$      31360
CODE      31476 KEYROW    31614 BEEPING   31208 LEN       31456 DIMN      31516
DAY$      31690 DATE      31596 DATE$     31684 FILL$     31378 VER$      31258
```

Figure 2

Pirate copies

Since reading your article in the August 30 issue concerning pirated software in Portugal I have realised that most — if not all — of the software titles are pirate copies.

The problem with software in Portugal is price. If the software companies were legitimate the programs would reach Portugal at astronomic prices. To put things in perspective, for £6 here we can go to the cinema six times or have two substantial dinners at a good restaurant.

At those sorts of prices, I doubt if the software companies would sell much software.

If the software companies reckon they are losing so much money in my country because of piracy, why don't they join together and compete with the pirates on price.

Another reason is that, for anyone making a home tape-to-tape copy, there is no established mechanism whereby the copier can send a donation to the relevant software house. Many of the programs out here are not easily obtainable as a legitimate title.

Since your August issue I haven't purchased any more pirated copies or made home copies without sending a letter to the company concerned. The one program I did copy, I sent a letter to the software house concerned, but I am still waiting for reply.

Fernando Hugo Dias De Oliveira
P O Box f35
2700 Amadora
Portugal

QL versus BBC

So...According to Phil Rogers in *Peek & Poke*, October 12 issue, the QL is more powerful than the BBC.

The BBC is old-fashioned because it has too many chips? I suppose Phil Rogers will soon be going around telling youngsters that main-frame computers are old-fashioned because they have too many chips as well. Sinclair has the idea that everything he can't do in hardware can be got round using software. I don't mind. If he wants to go ahead and produce a 16-bit computer which operates at the speed of a 7-bit then he can go right ahead. It's fine by me.

The BBC micro is faster than the QL mainly because it has more chips inside it. The QL, on the other hand, is the slowest 16-bit computer I have ever come across. And the QL is a lot more powerful? Ever heard of expansion — little things like Z80, 6502 and 16-bit second processors and Unicorn? And, what about the QL's windowing facility and multi-tasking? I'd be interested to see *one* QL program which uses windows and multi-tasking.

One day in the future (Sinclair's favourite word), Sinclair Research will produce a computer that will actually be expandable.

Jagdeep Sandu
2 Bulls Bridge Road
Southall
Middx

A proper keyboard

What is this I hear? A spokesman for Uncle Clive saying 'Sinclair Research are listening to all the people who say that the Spectrum needs a good keyboard? Does it really take two years for them to hear?

If your needs dictate that you should have a proper keyboard then there is already a good choice of add-on 'professional' keyboards available for the Spectrum at prices ranging from £30 to £80.

As a Spectrum owner I would like to see the Spectrum+ do well — it's a good machine with lots of excellent software. But somehow I fear that Uncle Clive may have opened his ears too late.

J Jago
Flat 4
238 Royal College Street
London NW1

"They're playing our tune!"

Forgotten QL owner

Have I been forgotten? I have a QL (I suppose I should be grateful for that) but it is still with the monstrosity stuck in the back.

No recall letter has yet been received from Sinclair and as they were due to have recalled all the 'dongled' QLs by the end of August I am beginning to wonder if they have lost my address.

Is there anyone else in the same boat? I was patient enough waiting for the machine with all its faults — how much longer must I wait for the corrected version?

Richard Chambers
21 Chadwell Springs
Waltham
nr Grimsby
S Humberside

According to Sinclair, all the Rom refit vouchers should have been sent out weeks ago. If for some reason yours has gone astray you should get in touch with Sinclair's Customer Services Department and ask to be sent another voucher straight away.

Spectrum versions

I am writing in response to M Payne's letter printed in the August 2 issue, concerning Phil Rogers' article about determining the different versions of Spectrums.

By using Print In 16602, 255 is returned if the Spectrum is an Issue 2, as my micro is. If, however, Interface 1 is fitted then 63 is returned. Issue 3 Spectrums return 1891 when Interface 1 is connected. Finally, Issue 2 Spectrums give a figure of 0 if the Protek joystick interface is connected.

C E Baker
Wordsley
Stourbridge
W Midlands

Dark secrets of the QL

Alan Turnbull's article revealing the dark secrets of the QL Rom was written for an AH version QL.

For the JM version — *Print Ver$* gives JM when typed in — then change Alan's program so that, at Line 190 put *Tabulate* 26724 and at Line 230 put *Tabulate* 27400. The resulting print-out for the JM QL is given in the table.

B J White
Wirral
Merseyside

COMMAND ROUTINE ADDRESSES					
PRINT	28662	RUN	30322	STOP	30424
INPUT	28660	WINDOW	30736	BORDER	30774
INK	28440	STRIP	28444	PAPER	28448
BLOCK	30750	PAN	29482	SCROLL	28486
CSIZE	24828	FLASH	26098	UNDER	26092
OVER	26120	CURSOR	24864	AT	24878
SCALE	26172	POINT	26190	LINE	26208
ELLIPSE	26232	CIRCLE	26232	ARC	26312
POINT_R	26194	TURN	30506	TURNTO	30498
PENUP	30564	PENDOWN	30568	MOVE	30582
LIST	28112	OPEN	25998	CLOSE	25964
FORMAT	25786	COPY	25812	COPY_H	25816
DELETE	25642	DIR	25648	EXEC	25318
EXEC_W	25322	LBYTES	25432	SEXEC	25486
SBYTES	25490	SAVE	26036	MERGE	30360
MRUN	30370	LOAD	30402	LRUN	30408
NEW	30420	CLEAR	30310	OPEN_IN	26002
OPEN_NEW	26006	CLS	28478	CALL	24612
RECOL	29626	RAMDOMISE	29408	PAUSE	28566
POKE	23602	POKE_W	28610	POKE_L	28616
BAUD	24380	BEEP	24440	CONTINUE	30494
RETRY	30484	READ	25272	NET	28412
MODE	28384	RENUM	29714	DLINE	28082
SDATE	25078	ADATE	25058	LINE_R	26212
ELLIPSE_R	26236	CIRCLE_R	26236	ARC_R	26316
AUTO	29672	EDIT	29668	FILL	26062
WIDTH	30714				
FUNCTION ROUTINE ADDRESSES					
ACOS	30950	ACOT	30956	ASIN	30962
ATAN	30968	COS	30974	COT	30980
EXP	30986	LN	30992	LOG10	30998
SIN	31004	SQRT	31010	TAN	31016
DEG	31022	RAD	31028	RND	31100
INT	31200	ABS	31060	PI	31186
PEEK	31224	PEEK_W	31232	PEEK_L	31242
RESPR	31276	EOF	31310	INKEY$	31364
CHR$	31450	CODE	31570	KEYROW	31708
BEEPING	31298	LEN	31548	DIMN	31610
DAY$	31784	DATE	31690	DATE$	31778
FILL$	31468	VER$	31348		

Calling long distance . . .

Your Spectrum & QL should be talking to each other . . . Alan Turnbull shows how

It is a fair assumption to make that most of the owners of the QL, like myself, already own a ZX Spectrum and have progressed from that machine rather than purchasing the QL as their first computer.

People in this situation will, most probably, want to take full advantage of the QL and ZX Spectrum networking facilities — *Qlan* and *ZX Net*, respectively.

This article, then, introduces a routine for use with the following minimum equipment: a Sinclair ZX Spectrum, a Sinclair ZX Interface 1, a Sinclair QL, and a ZX Net/Qlan networking lead.

The routine, when loaded into the QL, will enable the sending of ZX Spectrum programs over the network and the saving of them on QL Microdrive, with simple conversion done as well.

The program at the heart of it all can be seen in Figure 1. It receives listings of programs (ie, files generated by the Basic command *List*) from the ZX Spectrum, converts their format and sends them to QL Microdrive *mdv1*, ready for subsequent loading and editing with the QL commands: *Load mdv1_‹file name›* and *Edit ›start‹, ›increment‹*.

Upon running the program in Figure 1 on the QL, you will be presented with a title screen and prompt: 'Enter the name of ZX Spectrum Program:', to which you should reply with the name of a file to be generated on QL Microdrive *mdv1*, and by which you wish the program to be called.

A file with this name will be generated and you will be prompted with the message: 'Receiving and accessing Microdrives'. At this point, you should have the program you wish to send loaded into the ZX Spectrum and execute the direct commands listed in Figure 2.

A delay will follow, the duration of which will depend on the length of the program to be received, and the QL Microdrive will whizz around quite a lot. The whole process is completed when the QL display reads 'Reception completed', the flashing cursor re-appears and the QL Microdrive *mdv1* stops.

The process can be checked by executing on the QL the command: *Copy mvd1_‹file name› TO scr_*, whereupon the listing will be displayed on the QL screen. The ultimate test is the *Loading* of the file as a program from *mdv1*. You will find in practise that a lot of program lines will have the keyword *Mistake* inserted in them. You will have to go through the program with the multiple edit command, changing obvious syntax violations. The most common will be that of having no brackets around arguments for functions.

The program in Figure 1 has a set-up procedure that places expansions of the keywords in an array. Certain keywords on the ZX Spectrum fo not work on the QL. these have been prefixed by *ZX_*, so that you may, if you wish, give definitions for them in *Def Fn* and *DefProc* constructs. where appropriate, keywords have been changed to their new name.

Obviously, only limited conversion can be done because of the great difference between the computers. You will be amazed, however, at just how many of your simple ZX Spectrum programs will travel across to the QL.

```
100 REMark ZX Spectrum/QL Program converter
110 REMark   (c) July 1984, Alan Turnbull
120 MODE 256
130 PAPER 1
140 INK 7
150 FOR channel=0 TO 2
160   CLS #channel
170 END FOR channel
180 SET_UP
190 CSIZE 0,1
200 AT 0,2:PRINT "ZX Spectrum ";CHR$(189);" QL Program Converter"
210 AT 2,7:PRINT "(c) 1984, Alan Turnbull"
220 CSIZE 2,0
230 FOR e=2.5 TO 0 STEP -5E-2
240   FILL 1
250   INK RND(0 TO 7)
260   CIRCLE 80,35,30,e,0
270   BEEP 32767,0,2,4,8,16
280   FILL 0
290 END FOR e
300 INK 7
310 INPUT #0;"Enter name of ZX Spectrum program:"&CHR$(10);file$
320 NET 2
330 OPEN_IN #4,neti_1
340 OPEN_NEW #5,"mdv1_"&file$
350 CLS #0
360 PRINT #0,"Receiving and accessing Microdrives"
370 REPeat receive
380   IF EOF(#4) THEN EXIT receive
390   LET received_byte$=INKEY$(#4)
400   LET received_code=CODE(received_byte$)
410   SELect ON received_code
420     = 32 TO 127
430         PRINT #5,received_byte$;
440     = 13
450         PRINT #5,CHR$(10);
460     = 165 TO 255
470         PRINT #5," ";
480         LET char=1
490         REPeat send_ch
500           LET ch$=token$(received_code-165+1,char)
510           IF ch$="." THEN EXIT send_ch
520           PRINT #5,ch$;
530           LET char=char+1
540         END REPeat send_ch
550         PRINT #5," ";
```

Continued over the page

```
560   END SELect
570 END REPeat receive
580 CLOSE #4
590 CLOSE #5
600 CLS #0
610 PRINT #0,"Reception completed"
620 STOP
630 :
640 DEFine PROCedure SET_UP
650 LOCal codes
660   DIM token$(91,15)
670   RESTORE
680   FOR codes=1 TO 91
690     READ token$(codes)
700   END FOR codes
710 END DEFine SET_UP
720 :
730 DATA "RND.","INKEY$.","PI.","ZX_FN.","ZX_POINT.","ZX_SCREEN$.","ZX_ATTR.","A
T.","ZX_TAB.","ZX_VAL$.","CODE."
740 DATA "ZX_VAL.","LEN.","SIN.","COS.","TAN.","ASN.","ACS.","ATN.","LN.","EXP."
750 DATA "INT.","SQRT.","ZX_SGN.","ABS.","PEEK.","ZX_IN.","ZX_USR.","ZX_STR$.","
CHR$.","NOT."
760 DATA "ZX_BIN.","OR.","AND.","<=.",">=.","<>.","ZX_LINE.","THEN.","TO.","STEP
."
770 DATA "DEFine FuNction.","DIR.","FORMAT.","COPY.","DELETE.","OPEN #.","CLOSE
#.","MERGE.","ZX_VERIFY.","BEEP."
780 DATA "CIRCLE.","INK.","PAPER.","FLASH.","ZX_BRIGHT.","ZX_INVERSE.","OVER.","
ZX_OUT.","ZX_LPRINT.","ZX_LLIST."
790 DATA "STOP.","READ.","DATA.","RESTORE.","NEW.","BORDER.","CONTINUE.","DIM.",
"REMark.","FOR."
800 DATA "GO TO.","GO SUB.","INPUT.","LOAD.","LIST.","LET.","PAUSE.","NEXT.","PO
KE.","PRINT."
810 DATA "POINT.","RUN.","SAVE.","RANDOMISE.","IF.","CLS.","LINE.","CLEAR.","RET
urn.","ZX_COPY."
FIGURE 1: The ZX Spectrum → QL Program Converter.
FORMAT "n";1
OPEN #4;"n";2
LIST #4: CLOSE #4
FIGURE 2: The direct commands for the ZX Spectrum.
```

Open Forum

Open Forum **is for you to publish your programs and ideas. Take care that the listings you send in are all bug-free. Your documentation should start with a general description of the program and what it does and then give some detail of how the program is constructed.**

Windows

on QL

This program is designed to illustrate the *Window* facility on the QL. Four new windows will be opened, one in each corner of the screen, covering the whole of the display area. In each window a different type of graphical pattern is drawn; these patterns are described below.

In the top left hand corner, filled circles are drawn at random positions in random colours. In the top right hand corner filled squares are drawn at random positions in random colours. In the bottom left hand corner filled ellipses are drawn and in the bottom right hand window filled triangles are drawn at random positions in random colours. A border is also set up around each window.

The program will *Run* indefinitely, and should be stopped by pressing *Ctrl* and *Space* together.

```
100 REMark WINDOWS BY ANDREW FILBY
110 OPEN#5,scr_256x128a0x0
120 PAPER#5,0:CLS#5
130 INK#5,7
140 BORDER#5,15,1
150 SCALE#5,100,0,0
160 OPEN#6,scr_256x128a256x128
170 INK#6,0
180 PAPER#6,2:CLS#6
190 BORDER#6,15,3
200 SCALE#6,100,0,0
210 OPEN#7,scr_256x128a256x0
220 INK#7,0
230 PAPER#7,7:CLS#7
240 BORDER#7,15,6
250 SCALE#7,100,0,0
260 OPEN#8,scr_256x128a0x128
270 INK#8,0
```

Arcade Avenue

```
280 PAPER#8,4:CLS#8
290 BORDER#8,15,5
300 SCALE#8,100,0,0
310 INK 7:CURSOR 40,103:FLASH 1:CSIZE 3,1:PRINT"WINDOWS  BY ANDREW FILBY":F
LASH 0
320 REPeat loop
330 window_1
340 window_2
350 window_3
360 window_4
370 END REPeat loop
380 DEFine PROCedure window_1
390 INK#5,RND(0 TO 7):FILL#5,1:CIRCLE#5,RND(0 TO 149),RND(0 TO 150),RND(0 T
O 50):FILL#5,0
400 END DEFine
410 DEFine PROCedure window_2
420 FILL#6,1:INK#6,RND(0 TO 7):a=RND(0 TO 100):l=RND(0 TO 149):LINE#6,l,a T
O RND(0 TO 148),RND(0 TO 100) TO RND(0 TO 148),RND(0 TO 100) TO l,a:FILL#6,
0
430 END DEFine
440 DEFine PROCedure window_3
450 a=RND(0 TO 100):w=RND(0 TO 149):e=RND(0 TO 70)
460 INK#7,RND(0 TO 7):FILL#7,1:LINE#7,l,a TO l,a+e TO l+e,a+e TO l+e,a TO l
,a:FILL#7,0
470 END DEFine
480 DEFine PROCedure window_4
490 FILL#8,1:INK#8,RND(0 TO 7):CIRCLE#8,RND(0 TO 149),RND(0 TO 100),RND(0 T
O 50),RND,RND(0 TO (2*PI)):FILL#8,0
500 END DEFine
```

Windows
by A Filby

By definition

Ian Logan shows you how to produce user-defined graphics characters on your QL

The QL and the Spectrum are meant to be complementary machines and are not intended to compete with each other.

Hence, the ability to define one's own graphics characteristics, which is so much a feature of the Spectrum, was purposely left out of the QL's SuperBasic.

However, within certain limits, it is still relatively easy to create user-defined graphics (UDG's) on the QL. All that is required is an understanding of how the standard characters are produced.

The character set for the characters from Space to copyright symbol (addresses 20H to 7FH, 32 to 127 dec) is to be found in the QL's read-only memory (Rom). However, its base address varies from the Rom version to another and it is perhaps best to find this address by looking into a channel header block. For example, if the standard channels 0,1 and 2 have not been distributed, the base address of the first character sets is given by

PRINT PEEK_L(167722)

Note, there is normally a separate character set for the characters 80H to BFH (128 to 191 dec); and this set's base address is found by using

PRINT PEEK_L(167726)

Each character set has eleven header bytes. These are: 1) The character code below the starting character, ie, if the first character is to be character 20H (32 dec, then this byte is 1FH (31 dec)

2) The number of character forms held in the current set, ie, for characters 20H to 7FH (32 to 127 dec) this byte is 60H (96 dec).

3-11) Nine bytes to hold the form of a 'default character', ie, normally the QL uses "5428542854285428542854H" which gives a cross-hatch character.

The main body of the character set holds the character forms of all the characters. Each character has its form held in nine consecutive bytes, but, in each byte, only bits 6,5,4,3 and 2 are used.

As an example consider the form for the character '7'. The bytes are "007C04081020404000H", which can be represented as:

. . . .	00H, 0 dec
. 1 1 1 1 1 . .	7CH, 124 dec
. . . . . 1 . .	04H, 4 dec
. . . . 1 . . .	08H, 8 dec
. . . 1	10H, 16 dec
. . 1	20H, 32 dec
. 1	40H, 64 dec
. 1	40H, 128 dec
. . . .	00H, 0 dec

Now try the accompanying QL UDG program made up of the following four procedures.

Procedure udg. This is called just once. The procedure identifies the 'old' character set base address and copies over the whole of the set into the resident procedure area. Then, procedures *'newset'* and *'defchars'* are called.

Procedure defchars. This is a simple procedure that allows you to define your own characters. The new character is displayed as it is created. This procedure can be called independently, as required.

Procedures newset and oldset. These procedures allow you to 'toggle', if wished, from the oldset to the newset, or vice-versa.

```
100 DEFine PROCedure udg
110 chan1font=167722
120 oldbase=PEEK_L(chan1font)
130 newbase=RESPR(875)
140 FOR d=0 TO 875 STEP 4
150 POKE_L newbase+d,PEEK_L(oldbase+d)
160 END FOR d
170 newset
180 defchars
190 END DEFine udg
200 REMark ------------------
210 DEFine PROCedure defchars
220 CLS
230 REPeat looo
240 PRINT "Select the character to be re-defined"
    \"by entering its code (32-127)
."\"Use anything else to quit."
250 INPUT \"Character code? ";a
260 PRINT "     Character <";CHR$(a);">"
270 IF a<32 OR a>127 THEN EXIT loop
280 PRINT\"Now enter the 9 values (0-255) for"\
    "this character"\
290 charbase=newbase+10+(a-32)*9
300 PRINT "              Old   New   Character"
310 FOR d=1 TO 9
320 PRINT "Line ";d;"      ";PEEK(charbase+d),
330 INPUT b;
340 IF b<0 OR b>255 THEN PRINT \: EXIT d
350 POKE charbase+d,b
360 PRINT "          <";CHR$(a);">"
370 END FOR d
380 PRINT "Another character? (y/n) ";
390 INPUT a$
400 IF a$=="y" THEN CLS: NEXT loop
410 EXIT loop
420 END REPeat loop
430 END DEFine defchars
440 REMark ------------------
450 DEFine PROCedure newset
460 POKE_L chan1font,newbase
470 END DEFine newset
480 REMark ------------------
490 DEFine PROCedure oldset
500 POKE_L chan1font,oldbase
510 END DEFine oldset
520 REMark ------------------
```

Printing prettier

Take the strain out of listing with Pretty Prints by **B G Merrick**

SuperBasic is a powerful language designed for ease of programming. Hence it is perhaps surprising that no facility exists for indenting sections of a program when listed to show its structure, especially considering that this function is provided automatically in the database programming languague provided in Archive.

This routine overcomes this deficiency by reading a program file from microdrive (or any other source) and copying it to another device (or back to the same file), formatting it by indenting all multi-line *RE*Peat loops, *FOR* loops, *SEL*ect switches, *PROC*edures, *FUN*ctions and *IF* statements as it goes.

In order to use this program, first save the code to be formatted. Then load and run this program. The program prompts for a source and a destination file. The full name including the device should be entered, for example, *'mdv1-pretty'* as this provides consistency with SuperBasic commands and allows a file to be formatted to a printer. However, if just *Enter* is pressed in response to the destination file prompt, and the file is on microdrive, this program will just copy the file to a temporary file as the source, delete the original file and use this as the destination file. At the end of the program the temporary file is deleted.

Main Routine

This initially prompts the user to determine the file names to use and if required copies the original file *(Source$)* to a temporary file and deletes the old version.

After opening the files (Lines 240 and 250) the routine enters a loop which terminates when the end of the source file is reached. Within the loop the program reads a line at a time, calling *Num* to strip any leading spaces from the line, and print the line number to the destination file. The variable *Spc* holds the number of spaces to be placed between the line number and the first command of a line. The amount this is to be altered before and after printing the line is held in Before and After respectively. The procedure *Spaces* determines the values of these variables for the line. Line 340 ensures that an error will not occur if a program with incorrect syntax is passed through this program. Line 350 actually prints the line using *Fill$* to provide a string of spaces.

On exit from the loop, the channels are closed and if used the temporary file is deleted.

Procedure Num

This routine consists of three parts. The first, Lines 460 to 490 simply removes any leading spaces from the line. The second, Lines 500 to 550 gets the line number and prints this right justified in a space of 5 characters (Line 550).

The final part (Lines 560 to 600) is identical to the first part except that it removes spaces from after the line number.

Procedure Spaces

This section of the program compares the start of a program line with the statements listed in Lines 930 and 940 as data. If a match is found the *For* loop terminates with a value of 1 to 9 in *N*, representing the commands in the *Data* statements. If no match is found *N* contains the value 10, hence the dummy command '*' at the end of Line 940 preventing an error on the tenth *Read*.

Colon is then set to a truth value representing whether the line contains more than one statement by first testing for the presence of a colon and ensuring that this is not the end of the line.

This can then be used to determine whether or not a structure exists in its single or multi-line form.

The routine then switches on the value of *N* to determine the change in spacing for different statements. The only points to note here are that as a multi-line structure can start on the same line as an ON= or =switch. *Spaces* is called recursively, with the remainder of the line being passed as its parameter. A test for *Then* statement is made in the part that deals with *If*.

Procedure Diff

This procedure simply updates the values of *Before* and *After* by adding its parameters to them.

```
100 REMark  PRETTY PRINT ROUTINE
110 REMark       BY B.G.MERRICK
120 REMark          (c)1984
130 REMark
140 INPUT#0,"Source file
? ";Source$\"Destination file ? ";Dest$
150 IF (Dest$=''OR Dest$=Source$)AND Source$(1 TO 3)=="MDV"
160   Dest$=Source$
170   Source$=Source$&'_tmp'
180   COPY Dest$ TO Source$
190   DELETE Dest$
200   temp=1
210 ELSE
220   temp=0
230 END IF
240 OPEN_IN#3,Source$
250 OPEN_NEW#4,Dest$
260 Spc=1
270 REPeat Loop
280   IF EOF(#3) THEN EXIT Loop
290   INPUT#3,line$
300   Num line$
310   Before=0:After=0
320   Spaces line$
330   Spc=Spc-Before
340   IF Spc<0:Spc=0
350   PRINT#4,FILL$(' ',Spc);line$
360   Spc=Spc+After
370 END REPeat Loop
380 CLOSE#3
390 CLOSE#4
400 IF temp THEN DELETE Source$
410 STOP
420 :
430 DEFine PROCedure Num(a$)
440   LOCal n$,i
450   n$='':i=1
460   REPeat loop2
470     IF a$(i)<>' ' THEN EXIT loop2
480     i=i+1
490   END REPeat loop2
500   REPeat Loop3
510     IF a$(i)<'0'OR a$(i)>'9' THEN EXIT Loop3
520     n$=n$ & a$(i)
530     i=i+1
540   END REPeat Loop3
550   PRINT#4,FILL$(' ',5-LEN(n$));n$;
560   REPeat loop4
570     IF a$(i)<>' 'THEN EXIT loop4
580     i=i+1
590   END REPeat loop4
600   a$=a$(i TO)
610 END DEFine
620 :
630 DEFine PROCedure Spaces(l$)
640   LOCal N,Colon,thenin,Stmt$
650   RESTORE
660   FOR N=1 TO 10
670     READ Stmt$
680     IF l$(1 TO LEN(Stmt$))=Stmt$ THEN EXIT N
690   END FOR N
700   Colon=':'INSTR l$
710   IF Colon=LEN(l$) THEN Colon=0
720   SELect ON N
730    =1,2
740     Diff 1,1
750     IF Colon THEN Spaces l$(':'INSTR l$+1 TO)
760    =3 TO 6
770     IF NOT Colon THEN Diff 0,2
780    =7
790     thenin='THEN'INSTR l$
800     IF thenin THEN thenin=LEN(l$)>thenin+4
810     IF NOT(Colon OR thenin) THEN Diff 0,2
820    =8:Diff 2,2
840    =9:Diff 2,0
850   END SELect
860 END DEFine
870 :
880 DEFine PROCedure Diff(a,b)
890   Before=Before+a
900   After=After+b
910 END DEFine
920 :
930 DATA '=','ON','REPeat','FOR','DEFine'
940 DATA 'SELect','IF','ELSE','END','*'
```

Into overdrive

Malcolm Bryant shows how Spectrum Microdrives can be used with the QL.

Sinclair claim that the Spectrum Microdrives are not compatible with the QL. This means that you cannot take a cartridge with a Spectrum program and read it into the QL, which is not really surprising.

However, it is possible to connect Spectrum Microdrives directly to the QL and they will then work exactly as if they were additional QL Microdrives.

Proceed as follows. Take your Spectrum Microdrive(s) along with their connecting cable and plug into the QL Microdrive expansion port on the right-hand side of the computer. By putting a single twist in the cable, the Spectrum Microdrive(s) can sit on top of the QL Microdrives. You will now find that these drives can be accessed as MDV3, MDV4 and so on. This can be extremely useful. A practical example of how four drives can be used is while running the Psion packages. The back-up commands (eg, *Archive*) are almost worthless with only two drives, since a file cannot be copied from MDV2 on to a new cartridge, unless it is to MDV1 which is not normally what is required. Now files can be backed up from MDV2 to MDV3 — far more convenient.

A further tip when running *Quill*. To improve the speed of the Microdrive operations, keep your document on MDV3 (if you also have MDV4 then put your back-up cartridge in there). The work file *def-doc* will still be written to MDV2 and file reading and writing then be faster, particularly if you can keep a lot of free sectors on MDV2.

Another string . . .

*Use this program by **Richard Snowdon** to edit and write text using Quill.*

This program is very useful as it allows you to edit or even write programs using the word processor, *Quill* and then convert the text to a machine readable form so that the edited program can be tested.

The following is a fools guide to using this *Quill* utility.

1) Load program to be edited, eg, load mdv2_invader (if the program is called 'invader')
2) Save this program with the extension '_lis' so that it can be loaded into *Quill*. eg *save mdv2_invader_lis*
3) Boot *Quill*, eg, lrun mdv1_boot
4) Choose the import option on *Quill* (under 'Files' on the second command screen) to load the program to be edited, eg, *Import, invader_lis* (making sure invader_lis is on mdv2)
5) When it's loaded, press ESC to leave 'Files' and edit the program using any of the ample features of QUILL:
   I used QUILL with this program to replace short variable names with long meaningful ones and split multi-command lines. It could also be used to move blocks of code to the end of a program and head them as procedures, etc.
6) When finished editing, use the 'Save' option (on the first command screen) to save the program, and then *Quit Quill*.
7) Load this utility, and enter the drive number of the program and the name you saved it as, and the file name you want the loadable version to be.
8) When the utility is finished, you will be able to load the final version of the edited program.

This utility is needed because *Quill* saves text (documents and programs) with excess codes (linefeeds, page markers, margin information) padded around the text. This utility strips all of these characters from the program proper, enabling it to be loaded. I have taken advantage of the fact that *Quill* saves text unjustified. This is the reason it takes such a long time to *Save* and *Load* — when a document is loaded, the characters are read one at a time from a file, and each line has to be justified as it comes. Similarly, *Quill* has to reformat/unjustify text before filing.

As the present generation of mainframe/mini users have found, it is much easier to write and edit programs using a word processor. I hope that with the help of this utility, other QL users will realise this too.

```
100 REMark QUILL utility
110 REMark by Richard Snowdon
120 MODE 4:PAPER 0:STRIP 2
130 DIM text$(1000),search$(1):count=0:memory=0
140 REPeat validate
150  PRINT #0,"Which drive ? ";:drive$=KEY$(1)
160  IF drive$="1" OR drive$="2" THEN EXIT validate
170 END REPeat validate:PRINT #0,drive$
180 INPUT #0,"What is the name of the Quill
file ?"!quill$
190 IF LEN(quill$)>3 THEN
200    last$=quill$(LEN(quill$)-3 TO)
210    IF last$<>"_doc" AND last$<>"_DOC" THEN
220       quill$=quill$&"_doc"
230    END IF
240  ELSE quill$=quill$&"_doc"
250 END IF
260 INPUT #0,"What do you want the resulting
file to be called ?"!final$
270 OPEN #6,"mdv"&drive$&"_"&quill$
280 DELETE "mdv"&drive$&"_"&final$
290 OPEN_NEW #7,"mdv"&drive$&"_"&final$
300 REPeat check
310  first$=KEY$(6)
320  IF CODE(first$)=110 THEN count=count+1
:ELSE count=0
330  IF count=3 THEN count=0:EXIT check
340 END REPeat check
350 REPeat main_loop
360  text$=KEY$(6)
370  asc=CODE(text$)
380  IF asc=0 THEN count=count+1:ELSE count=0
390  IF count=3 THEN EXIT main_loop
400  SELect ON asc
410   =49 TO 57
420    PRINT text$;
430    REPeat rest
440     char$=KEY$(6)
450     IF CODE(char$)=0 THEN
460        length=LEN(text$)
470        IF length=1 THEN EXIT rest
480        memory=memory+length
490        PRINT #7,text$:PRINT
500        EXIT rest
510     END IF
520     PRINT char$;
530     text$=text$&char$
540    END REPeat rest
550  END SELect
560 END REPeat main_loop
570 CLOSE #6:CLOSE #7
580 PRINT \"Finished.....Program length ";
INT(memory/1024*100)/100;" Kb"
590 DEFine FuNction KEY$(chan)  :
RETurn INKEY$(#chan,-1)
```

Just your type....

Bored with that same old type face? Take heart, and take a look at ***R Snowdon's*** *Definer program for the QL*

This program allows the user to choose a character he or she wants to define, then presents a blow-up of this character, as well as the actual size while defining. This is very useful when designing gothic or modern character sets.

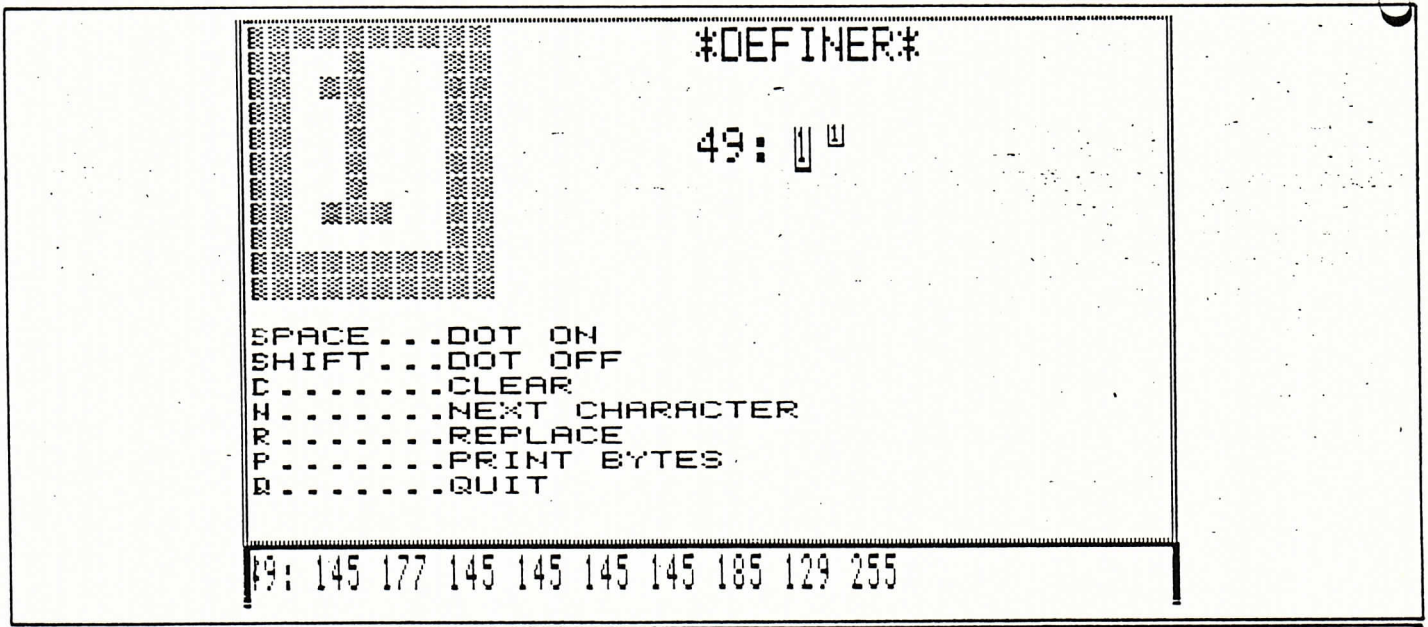
It is quite simple to operate. When the code of the character to be defined is input, the character size must be entered, *CSize* ?,0. This can be in the range nought to three. This is necessary because the QL uses a different range of pixels in each character row, depending on the pre-set character size. For instance, character size one gives the maximum 8 pixels across, and character size three gives 5 pixels.

When these graphics are used in your own programs, you must use the *CSize* command with the character size (which you entered when defining the character) as its first argument, eg, *CSize* 1,0 or *CSize* 1,1. Failure to do this may cause odd things to happen. After this, the user can experiment with the character definition on the blow-up grid.

Commands

Space-bar: light pixel at cursor position
Shift: remove pixel at cursor position
C: clear grid
N: next character definition
R: replace old character definition
P: print out definition bytes
Q: leave program

If space has already been reserved on the QL (*Respr*(100) ... etc), I recommend the QL is reset before running *Definer*.




```
1 REMark : QL Graphics Definer
2 REMark : by Richard Snowdon (*SnowSoft*)
100 WINDOW #1,450,210,33,6:PAPER 0
110 WINDOW #0,450,40,33,216:PAPER #2,0
120 MODE 4:SCALE 256,0,0:BORDER
2,175:BORDER #0,2,2
130 moveset
140 ask
150 initiate
160 setup
170 display
180 newset
190 start
200 STOP
210 DEFine PROCedure initiate
220 CSIZE 2,0
230 DIM pix(9,8)
240 FOR f=1 TO 9
250 FOR e=1 TO 8
260 pix(f,e-(size>1)) = ((PEEK(charbase+f)
&&(2^(8-e)))=0)*32
270 NEXT e:NEXT f
280 x=1:y=1
290 END DEFine
300 DEFine PROCedure display
310 FOR f=1 TO 9
320 FOR e=1 TO across
330 AT f,e:PRINT CHR$(pix(f,e))
340 NEXT e:NEXT f
350 END DEFine
360 DEFine PROCedure start
370 REPeat mainloop
380 k=KEYROW(1)
390 AT y,x:PRINT CHR$(pix(y,x))
400 IF KEYROW(6)&&8 THEN CSIZE size,1:STOP
410 IF KEYROW(2)&&8 THEN
420 FOR f=1 TO 9:FOR e=1 TO 8:pix(f,e)=32:
NEXT e:POKE charbase+f,0:NEXT f:displaY
430 END IF
440 IF k&&2 AND x>1 THEN x=x-1
450 IF KEYROW(5)&&16 THEN
460 FOR replace=1 TO 9:POKE charbase+
replace,PEEK(oldbase+10+(a-32)*9+replace):N
EXT replace:initiate:display
470 END IF
480 IF k&&16 AND x<across THEN x=x+1
490 IF k&&4 AND y>1 THEN y=y-1
500 IF k&&128 AND y<9 THEN y=y+1
510 IF KEYROW(7)&&1 AND pix(y,x)=0 THEN
pix(y,x)=32:POKE charbase+y,(PEEK(charba
se+y))^^2^(8-x-(size>1))
520 AT 4,22:CSIZE size,1:PRINT CHR$(a);' ';
:CSIZE size,0:PRINT CHR$(a):CSIZE 2,0
530 IF k&&64 AND pix(y,x)=32 THEN pix(y,x)
=0:POKE charbase+y,(PEEK(charbase+y))!
!2^(8-x-(size>1))
```

Continued over the page

```
540 IF KEYROW(7)&&64 THEN RUN
550 IF KEYROW(4)&&32 THEN prit
560 AT y,x:PRINT CHR$(32+(pix(y,x)>0)*255)
570 END REPeat mainloop
580 DEFine PROCedure setup
590 CLS
600 CSIZE 2,1:AT 0,18:PRINT "*DEFINER*":AT
2,18:PRINT a;':':CSIZE 2,0
610 AT 12,0:PRINT "SPACE...DOT ON"\"SHIFT.
..DOT OFF"\"C.......CLEAR"\"N.......NE
XT CHARACTER"\"R.......REPLACE"\"P.......
PRINT BYTES"\"Q.......QUIT"
620 INK 2:CSIZE #0,1,1
630 across=8-2*(size=0)-3*(size>1)
640 AT 0,0:PRINT FILL$(CHR$(255),across+2)
650 FOR y=1 TO 9:AT y,0:PRINT CHR$(255);
FILL$(" ",across);CHR$(255)
660 PRINT FILL$(CHR$(255),across+2)
670 INK 7
680 END DEFine
690 DEFine PROCedure moveset
700 set=167722
710 IF RESPR(0)>261120 THEN
720 oldbase=PEEK_L(set)
730 newbase=RESPR(875)
740 FOR m=0 TO 875 STEP 4
750 POKE_L newbase+m,PEEK_L(oldbase+m)
760 NEXT m
770 END IF
780 oldset
790 END DEFine
800 DEFine PROCedure ask
810 CLS:CSIZE 0,0
820 PRINT "Select the character to be
re-defined"\"by entering its code (32-127)
830 INPUT \"Character code? ";a
840 IF a<32 OR a>127 THEN STOP
850 PRINT \"What character
size (0 to 3)? ";
860 REPeat vet
870 size=INKEY$(-1)
880 SELect ON size=0 TO 3:EXIT vet
890 END REPeat vet
900 PRINT size
910 charbase=newbase+10+(a-32)*9
920 END DEFine
930 DEFine PROCedure newset
940 POKE_L set,newbase
950 END DEFine
960 DEFine PROCedure oldset
970 POKE_L set,oldbase
980 END DEFine
990 DEFine PROCedure prit
1000 LOCal answer$,chan
1010 CLS#0:PRINT #0,"To printer (y/n)
";:answer$=INKEY$(-1)
1020 chan=(answer$=="Y")*8
1030 IF chan THEN OPEN #8,ser1
1040 PRINT #chan,\a;':';:FOR B=1 TO
9:PRINT #chan;' ';PEEK(charbase+B);:NEXT B
1050 END DEFine
```

Open Forum

We are always actively seeking programs for publication — either for Open Forum, the machine pages or Star Game. When sending in a program for consideration, a clear program listing should be sent, together with, wherever possible, a saved copy on cassette. Documentation — usually not more than 1000 words — should start with a general description of the program, what it does, and then some detail of how the program itself is constructed. We pay very competitive rates, according to the length and nature of the program and the quality of the accompanying documentation.

Clock

on QL

This is a program utilising the QL 'clock — whilst it can't show the seconds actually ticking by while we're working with another program it keeps going and is always spot on whenever we type in *time* to set it in motion again.

I've used high line numbers for the Procedure time and put it on channel 13 so it's out of the way of everything else. (Hence all those hash-13 designators which are essential in this kind of exercise.)

Lines 32738 to 32741 give a narrow green strip slightly above the usual TV display panel. This position can be modified to suit individual receivers by altering the last figure (the y co-ordinate) of 32738. I find 4 works best for me, but you can go up or down in steps of one or two until you find the setting that's best for your particular equipment.

The *Repeat* cycle displays the date and time in yellow characters on a black background. *Day$* seems to supply the day of the week automatically; but *Date$* needs to be reset each time you power up. The formula is *Sdate* (ie, set date) followed by the year, month, day, hour, minute and second.

For example: *Sdate* 1984,8,13,18,0,0
stands for: 1984 Aug 13 18:00:00

Lines 32725 to 32729 open a window for the standard displays of channels 1 and 2 with the addition of a green border. When you are working on a program, *Break* and *time* let you see the seconds going by without disturbing your display in the main window.

```
32720 :
32721 REMark ====
32722 REMark TIME
32723 REMark ====
32724 :
32725 WINDOW 490,200,12,16
32726 BORDER 2,4
32727 PAPER 0
32728 INK 6
32729 CLS
32730 CLS #0
32731 time
32732 :
32733 REMark ==========
32734 REMark definition
32735 REMark ==========
32736 :
32737 DEFine PROCedure time
32738  OPEN #13,scr_480x14a16x4
32739  BORDER #13,2,4
32740  PAPER #13,4
32741  CLS #13
32742  REPeat cycle
32743   AT #13,0,7: PAPER #13,0: INK
#13,6: PRINT #13,DAY$!DATE$
32744  END REPeat cycle
32745 END DEFine time
32746 :
32747 REMark ========
32748 REMark END TIME
32749 REMark ========
32750 :
32751 REMark copyright
32752 REMark francis cameron
32753 REMark 840813
32754 :
```

Clock
by F Cameron

The Music Box

Play back

Those of you who are interested in pursuing the field of computer music further could do worse than buy a copy of Hal Chamberlin's mammoth book — *Musical Applications of Microprocessors* (Hayden Book Company, New Jersey, 1980). I shan't be plugging my own book until it comes out under the PaperMac imprint next year.

Chamberlin's book, although already showing its age and not aimed at the hobbyist, probably gives you the most comprehensive possible introduction to the field. Read it, and you'll understand just how exciting and complex the subject is. It weighs in (and I do mean weigh) at 661 pages and will set you back the price of a reasonably good meal for two (wine included, though unfortunately, not with the book).

Now comes the news that the author of the bible has been busy soldering and has produced the DigiSound 16, a 2-channel, 16-bit digital processing unit for use with any computer that has two eight-bit parallel ports (available, for example, on any computer using a 6522 interfacing chip, which is to say any 6502- or 6510-microprocessor-based computer such as the BBC or Commodore).

What's exciting about the DigiSound is that it offers parallel processing which has got to be the next big thing, since serial processing, although electrically more reliable (you can use longer leads), is either too slow to be very useful or becomes unreliable.

The synthesiser manufacturers' MIDI standard for plugging computers into electronic instruments is a serial system which compromises between speed and reliability by using a data transmission rate of 31.5 Kbaud. This is above the maximum rate for RS-232 and similar standards, which is 19.2 Kbaud, and so requires additional circuitry. In principle, a parallel system (as used, for example, with Centronics printers) should be fairly easy to implement and could be fast enough to provide really sophisticated sound processing. The chief problem will be that you wouldn't want to take a parallel system on stage with you. For that purpose, MIDI — with its simple cables which can be several meters long without loss of signal — will doubtless remain, although a serial-to-parallel converter, allowing MIDI instruments to be controlled by parallel-wired computers, should not be beyond the wit of the hackers and hobbyists.

Anyway, to find out about the DigiSound 16 (which can be used for digitising, sampling, sequencing and playing back sounds), write to Micro Technology Unlimited, 2806 Hillsborough Street, Raleigh, North Carolina 27607, USA.

Gary Herman

The Music Box is a new weekly column with news, reviews and readers comments on all aspects of micros and music.

Any readers with experience of computer music making or companies with new product news are invited to write to: drop a line explaining what they're doing to: Gary Herman, The Music Box, 12-13 Little Newport Street, London WC2R 3LD.

QL Clock

by Andrew Pepper

'Clock' is a simple program which displays an analogue and digital clock on the QL screen. It must be run in TV mode: that is, after resetting or switching on the QL, select F2.

When run, you will be asked for the time of day in HHMMSS format: type in the current time in 12-hour clock format. For example, if the time is 8:32:40 in the evening type: 083240.

After this has been entered a real-time clock will be displayed. To terminate this program type CTRL+SPACE.

```
100 MODE 8
110 PAPER 2:INK 7
120 CLS
130 CSIZE 3,1
140 AT 9,1:PRINT"Clock 1.0";
150 CSIZE 0,0
160 AT 10,5:PRINT Andrew Pepper";
170 INK 5
180 AT 8,7:PRINT"Enter time (HHMMSS):                    ";
190 AT 29,7
200 INPUT t$
210 IF LEN(t$) <> 6 THEN GO TO 180
220 hr = t$(1 TO 2)
230 mn = t$(3 TO 4)
240 se = t$(5 TO 6)
250 oh=hr:om=mn:os=se
260 SDATE 1984,6,3,hr,mn,se
270 INK 4:CLS:CSIZE 2,1
290 FILL 0
300 t = 0
310 CSIZE 0,0
```

```
315 INK 4:FILL 1:CIRCLE 86,47,35:FILL 0
317 INK 4,7
320 FOR i = PI/6 TO 2*PI STEP PI/6
330 t = t + 1
340 IF t > 12 THEN GO TO 370
350 x = 32*SIN(i) : y = 30*COS(i)
360 CURSOR x+85,y+50,0,0:PRINT t;
370 END FOR i
372 CIRCLE 86,47,35
380 CSIZE 2,1
385 INK 7
387 drwhr
390 updig:drwsc:drwhr:drwmn
400 GO TO 390
410 DEFine PROCedure updig
420 LOCal t$
430 t$ = DATE$
440 BEEP 100,10
450 IF t$ = DATE$ THEN GO TO 450
460 hr = t$(13 TO 14): mn = t$(16 TO 17): se = t$(19 TO 20)
470 hr = hr + mn/60
480 AT 14,0:PRINT t$(12 TO 20);
490 BEEP 100,20
500 END DEFine
510 DEFine PROCedure drwhr
520 LOCal i
530 h = -1
540 i = oh*PI/6:drwln i,4:i = hr*PI/6:drwln i,7:oh=hr
550 h = 0
560 END DEFine
570 DEFine PROCedure drwmn
580 LOCal i
590 i=om*PI/30:drwln i,4:i=mn*PI/30:drwln i,7:om=mn
600 END DEFine
610 DEFine PROCedure drwsc
620 LOCal i
630 i=os*PI/30:drwln i,4:i=se*PI/30:drwln i,7:os=se
640 END DEFine
650 DEFine PROCedure drwln(i,c)
660 LOCal x,y
670 INK c
680 x=25*SIN(i):y=24*COS(i)
690 IF h THEN x = 20*SIN(i) : y = 18*COS(i)
700 LINE 85,50 TO x+85,y+50
710 END DEFine
```

A global option

This useful utility by ***Martin Amess*** *provides a string search facility*

One useful feature that is sadly missing from the QL is the facility to search for and/or replace a certain string contained in a program.

The following program implements this feature allowing you to search for and replace a chosen string contained in a selected program stored on a microdrive. There are three options to choose from.

The first is a search only option. The second goes one stage further and allows you to replace your selected string if required. The third is a global replace option which will replace all occurances of the selected string.

When either option 2 or 3 is chosen then a new program with the alterations will be stored in the file *File_Bak*. At the beginning, any existing file with the name *File Bak* will be deleted. When the search is completed you can copy the file *File_Bak* to your own program name if you are satisfied with it using the *Copy* command.

Program Notes

Lines 115 to 225

This repeat loop contains the main menu. You are requested to enter your chosen option number (1 to 3). Any other number will result in the program ending.

You will then be asked to enter the string that is to be searched for. If you selected options 2 or 3 you will then be asked to enter the new string that will replace it.

Finally you will need to enter the file that is to be searched, in the format *Mdv?_Name_of_File*, where *?* is the Microdrive number and *Name_of_File* is the name of the file stored on microdrive.

Lines 230 to 310

The procedure *Search* is defined. This procedure will search line by line for any occurence of your selected string. When a line is found it will be displayed on the screen and the prompt *Any More (Y/N)?* displayed. Answer *Y* to continue the search or *N* to return to the menu.

Lines 315 to 475

The procedure *Search_Replace* is defined. This procedure is similar to *Search* except that when a line is found containing the selected string you will be asked *Replace ? (Y/N)* after the line has been displayed on the screen. Answering *Y* will replace the string in this line with that of the new data entered at the beginning.

After entering *Y* or *N*, the program line will be checked further to see if there are any more occurences of the selected string. If there are you will be asked again whether or not to replace it.

Once the line is completed the program will continue checking the next line. When the end of the file is reached you will be returned to the menu.

Lines 480 to 620

The procedure *Global* is defined. This procedure works in the same way as the *Search_Replace* procedure except that the user is not asked whether or not to replace individual occurences of the selected string. All occurences are automatically replaced. As each line is processed by the program it is displayed on the screen.

A problem with the QL is that when a line of data is inputted from a file it must not be longer than approx. 125 characters, otherwise the *Buffer full* error will be given.

Many program lines are often larger than 125 characters and therefore this program would not necessarily work with all program files. So to avoid this problem, data is inputted one character at a time using the *Inkey$* keyword. Although this will resolve this problem, it does mean that it slows the program down.

The procedure *line_Input* (Lines 640 to 675) inputs the data from the file one character at a time, searching for the control character *Chr$(10)* which represents the end of each particular line of data.

However, should you be sure that your program does *not* contain any lines over 125 characters, then lines 265,365 and 530 can be replaced by the line *Input #11, Line$*, which will enter data from the file line by line and therefore speed the program up considerably.

```
100 REMark SEARCH & REPLACE
105 REMark © 1984 MARTYN AMESS
110 INK 7 : PAPER 0
115 REPeat MENU_LOOP
120 CLS #0:CLS #1
125 REMark MAIN MENU
130 AT 2,10:PRINT "MAIN MENU":AT 3,10:PRINT "----------"
135 AT 8,0 PRINT "Enter required option,":AT 10,5:PRINT
    "1. SEARCH ONLY":AT 12,5 :PRINT "2. SEARCH & REPLACE"
    :AT 14,5:PRINT "3. GLOBAL REPLACE":AT 16,2:INPUT "EN
    TER NUMBER ";OPTION
140 IF OPTION < 1 OR OPTION > 3 THEN EXIT MENU_LOOP
145 CLS
150 IF OPTION <> 1 THEN DELETE MDV1_FILE_BAK
155 AT 10,5:PRINT "ENTER STRING TO SEARCH FOR ":
    AT 12,7:INPUT SEARCH_STRING$
160 IF OPTION= 2 OR OPTION =3 THEN
165 AT 12,5:INPUT "ENTER NEW STRING ";NEW_STRING$
170 OLD_LEN=LEN(SEARCH_STRING$) : NEW_LEN=LEN
    (NEW_STRING$)
175 END IF
180 IF SEARCH_STRING$ ="" THEN STOP
185 CLS :AT 10,1:PRINT "ENTER FILE NAME TO SEARCH":AT
    11,5:PRINT "E.G 'MDV1_TEST
    _BAS'":AT 13,0:INPUT FILE_NAME$
190 CLS
195 SELect ON OPTION
200 =1 : SEARCH
205 =2 : SEARCH_REPLACE
210 =3 : GLOBAL
215 END SELect
220 END REPeat MENU_LOOP
225 STOP
230 REMark SEARCH FOR A STRING
235 DEFine PROCedure SEARCH
240 AT 0,10:PRINT "SEARCH ONLY":AT 1,10:
    PRINT "-----------"
245 OPEN #10,FILE_NAME$
250 REPeat SEARCH_LOOP
255 AT 10,10:PRINT "Searching..."
260 IF EOF(#10) THEN EXIT SEARCH_LOOP
265 LINE_INPUT
270 SET=SEARCH_STRING$ INSTR LINE$
275 IF SET THEN
280     SCR_CLEAR:AT 10,0:PRINT LINE$
285     AT 16,0:INPUT "ANY MORE (Y/N) ?";QUES$
290 IF QUES$="N" THEN EXIT SEARCH_LOOP
295 SCR_CLEAR
300 END IF
305 END REPeat SEARCH_LOOP
310 CLOSE #10:END DEFine
315 REMark SEARCH AND REPLACE A GIVEN STRING
320 DEFine PROCedure SEARCH_REPLACE
    :PRINT "-----------------"
325 AT 0,10:PRINT "SEARCH & REPLACE":AT 1,10
330 OPEN #10,FILE_NAME$
335 OPEN_NEW #11,MDV1_FILE_BAK
340 SCR_CLEAR
345 REPeat LOOP
350 AT 10,10:PRINT "Searching..."
355 IF EOF(#10) THEN EXIT LOOP
360 SET=0
365 LINE_INPUT
370 SET=SEARCH_STRING$ INSTR LINE$
375 IF SET THEN LINE_FOUND
380 PRINT #11,LINE$
385 END REPeat LOOP
390 CLOSE #10 : CLOSE #11
395 END DEFine SEARCH_REPLACE
400 DEFine PROCedure LINE_FOUND
405 SCR_CLEAR
410 AT 10,0:PRINT LINE$
415 AT 15,0:INPUT "REPLACE ? (Y/N) ";QU$
420 SCR_CLEAR
425 IF QU$ <> "Y" THEN RETurn
430 NEW_LINE$=FILL$(" ",((LEN(LINE$) +
    (NEW_LEN-OLD_LEN))))
435 NEW_LINE$(1 TO (SET-1)) = LINE$(1 TO (SET-1))
440 NEW_LINE$(SET TO)=NEW_STRING$
445 IF (SET + NEW_LEN) > LEN(LINE$) THEN GO TO 235
450 NEW_LINE$((SET + NEW_LEN) TO)=LINE$((
    SET+OLD_LEN) TO)
455 LINE$=NEW_LINE$
460 SET1=SET SET=SEARCH_STRING$ INSTR LINE$
    ((SET+NEW_LEN) TO)
465 IF SET <> 0 THEN SET=SET+SET1+OLD_LEN
470 IF SET THEN GO TO 235
475 END DEFine
480 REMark GLOBAL SEARCH & REPLACE
485 DEFine PROCedure GLOBAL
490 AT 0,10:PRINT "GLOBAL REPLACE":AT 1,10:
    PRINT "---------------"
495 AT 3,2:PRINT "Processing line,"
500 OPEN #10:FILE_NAME$
505 OPEN_NEW #11,MDV1_FILE_BAK
510 SCR_CLEAR
515 REPeat GLOBAL_LOOP
520 IF EOF(#10) THEN EXIT GLOBAL_LOOP
525 SET=0
530 LINE_INPUT
535 SET=SEARCH_STRING$ INSTR LINE$
540 IF SET THEN AMEND_LINE
545 PRINT #11,LINE$
550 SCR_CLEAR:AT 10,0:PRINT LINE$
555 END REPeat GLOBAL_LOOP
560 CLOSE #11 : CLOSE #10
565 END DEFine
570 DEFine PROCedure AMEND_LINE
575 NEW_LINE$=FILL$(" ",((LEN(LINE$) +
    (NEW_LEN-OLD_LEN))))
580 NEW_LINE$(1 TO (SET-1))=LINE$(1 TO (SET-1))
585 NEW_LINE$(SET TO)=NEW_STRING$
590 IF (SET + NEW_LEN) > LEN(LINE$) THEN GO TO 235
595 NEW_LINE$((SET + NEW_LEN) TO)=LINE$
    ((SET+OLD_LEN) TO)
600 LINE$=NEW_LINE$
605 SET1=SET:SET=SEARCH_STRING$ INSTR LINE$
    ((SET+NEW_LEN) TO)
610 IF SET <>0 THEN SET=SET+SET1+OLD_LEN
615 IF SET THEN GO TO 320
620 END DEFine
625 DEFine PROCedure SCR_CLEAR
630 AT 3,0:CLS 2
635 END DEFine SCR_CLEAR
640 DEFine PROCedure LINE_INPUT
645 LINE$=""
650 REPeat LOAD_LOOP
655 A$=INKEY$(#10)
660 IF A$=CHR$(10) THEN EXIT LOAD_LOOP
665 LINE$=LINE$ & A$
670 END REPeat LOAD_LOOP
675 END DEFine LINE_INPUT
```

Detailed examination

Stop Poking around in the dark and take a look at the QL Disassembler by **Keith Poole**

Any serious code programmer at some point will want to examine the QL's memory in detail, maybe with a view to using a Rom routine, or perhaps from idle curiosity.

This program, *QL Disassembler*, will allow you to do just that, and should prove useful to any QL enthusiast. The listing itself will be printed over two weeks, with accompanying notes on the program and the 68000 chip.

Program Notes

The instruction set of the 68000 can be split into 13 groups depending on the top four bits of the op-code. Each of these groups defines a certain set of operations.

Group 0: Immediate and bit instructions (ADDI etc)
Group- 1-3: Move (1=bytes, 2=long words, 3=words)
Group 4: Miscellaneous instructions
Group 5: Quick, set and decrement branch instructions
Group 6: Branch instructions
Group 7: Moveq instruction
Group 8: Arithmetic 1 (or, div and sub)
Group 9: Subtract
Group A&F: User defined instructions
Group B: Arithmetic 2 (eor and cmp)
Group C: Arithmetic 3 (and & multiply)
Group D: Add
Group E: Shift and rotate

Addressing Modes

The 68000 has 12 addressing modes. These are shown in the normal motorola assembler format, except for the program counter relative mode (eg, 14A(pc)) which is followed by the effective address separated by !, ie, jsr 2068(pc)! 80C5.

This is not available for the other pc-relative mode because this depends on the contents of address or data registers which cannot be determined by the disassembler.

```
20 CLEAR
30 INK 7
40 agh=0:DIM cc$(15,2):setup:lgt=0
50 digit$="0123456789ABCDEF"
60 MODE 4:CSIZE 1,0:CLS #0:BORDER #0,3:CLS:AT 8,15:PRINT
"Disassembler v1.0":AT
9,14:PRINT "1984 LRY Software"
70 op$="":lin=18
80 INPUT #0,"start address"!a$:s=dec(a$)
90 IF s/2<>INT(s/2) THEN s=s-1
100 REPeat loop
110 top=PEEK(s) DIV 16:bot=PEEK(s) MOD
16:rest=PEEK(s+1):op=2
120 SELect ON top
130 ON top=0:group0
140 ON top=1 TO 3:groupm
150 ON top=4:group4
160 ON top=5:group5
170 ON top=6:group6
180 ON top=7:group7
190 ON top=8:group8
200 ON top=9:group9
210 ON top=10,15:user
220 ON top=11:groupb
230 ON top=12:groupc
240 ON top=13:groupd
250 ON top=14:groupe
260 END SELect
270 IF agh=1 THEN op=2:agh=0
280 col=20:col=col-LEN(hex$((s)))
290 PRINT hex$((s));" ";
300 FOR i=0 TO op-1 STEP 2
310 t$=hex$(256*PEEK(s+i)+PEEK(s+i+1)):IF LEN(t$)
<4 THEN t$=FILL$("0",4-LEN(t$))
;t$
320 PRINT t$;:col=col-LEN(t$):END FOR i
330 asc$
340 PRINT FILL$(" ",col);op$
350 op$="":lgt=0
360 s=s+op
370 lin=lin-1:IF lin=0 THEN dly:lin=18
380 END REPeat loop
390 DEFine FuNction dec(a$)
400 LOCal i,tot
410 tot=0
420 FOR i=1 TO LEN(a$):tot=tot*16+val(a$(i))
430 RETurn tot
440 END DEFine
450 DEFine FuNction val(s$):PEMark return hex value of s$
460 LOCal t
470 t=s$ INSTR "abcdef":IF t>0 THEN RETurn t+9
480 t=s$ INSTR "ABCDEF":IF t>0 THEN RETurn t+9
490 IF s$<"0" OR s$>"9" THEN
500    RETurn 0
510   ELSE
520    RETurn s$
530 END IF
540 END DEFine
550 DEFine FuNction hex$(a)
560 LOCal str$,main,b
570 str$="":REPeat main
580 a=a/16:b=(a-INT(a))*16:a=INT(a)
590 str$=digit$(b+1)&str$
600 IF a=0 THEN EXIT main
610 END REPeat main
620 RETurn str$
630 END DEFine
640 DEFine PROCedure group0
650 IF slice(rest,3,3)=1 AND bot MOD 2=1 THEN
660    op$="movep"
670    IF slice(rest,6,1)=1 THEN
680       op$=op$&".l":lgt=1
690    ELSE
700       op$=op$&".w"
710    END IF
720    p=PEEK_W(s+op)
730    reg$="d"&(bot DIV 2):ad$=hex$(p)&"
(a"&(rest MOD 7)&")":op=op+2
740    IF rest DIV 128=1 THEN
750       op$=op$&"  "&reg$&","&ad$
760    ELSE
770       op$=op$&"  "&ad$&","&reg$
780    END IF
790 ELSE
800 op$=type$(slice(rest,6,2))
810 IF slice(rest,6,2)=2 THEN
820    p=PEEK_L(s+op):op$=op$&"  #"&hex$(p):op=op+4
830 ELSE
840    p=PEEK_W((s+op))
850    IF slice(rest,6,2)=0 THEN p=p&&255
960    op$=op$&"  #"&hex$(p):op=op+2
870 END IF
880 op$=op$&","&raddr$(slice(rest,3,3),slice(rest,0,3))
890 SELect ON bot
900 ON bot=0:op$="addi"&op$
910 ON bot=2:op$="andi"&op$
920 ON bot=12:op$="cmpi"&op$
930 ON bot=10:op$="eori"&op$
940 ON bot=0:op$="ori"&op$
950 ON bot=4:op$="subi"&op$
960 ON bot=REMAINDER
970    op=2
980    xx=slice(rest,6,2)+4*(bot MOD 2)
990    SELect ON xx
1000      ON xx=5:op$="bchg":im=0
1010      ON xx=1:op$="bchg":im=1
1020      ON xx=6:op$="bclr":im=0
1030      ON xx=2:op$="bclr":im=1
1040      ON xx=7:op$="bset":im=0
1050      ON xx=3:op$="bset":im=1
1060      ON xx=4:op$="btst":im=0
1070      ON xx=0:op$="btst":im=1
1080      END SELect
1090 IF slice(rest,3,3)<>1 THEN
1100 IF im=1 THEN
1110    op$=op$&"  "&hex$(PEEK_W(s+op))&","
1120 op=op+2
1130    op$=op$&addr$(slice(rest,3,3),slice(rest,0,3))
1140 ELSE
1150    op$=op$&"  d"&hex$(bot DIV 2)&","&raddr$(slice
(rest,3,3),slice(rest,0,3))
1160 END IF
1170 ELSE
1180    op$=op$&"  error"
1190 END IF
1200 END SELect
1210 END IF
1220 END DEFine
1230 DEFine PROCedure groupm
1240 op$="move"
1250 SELect ON top
1260 ON top=1:op$=op$&".b"
1270 ON top=2:op$=op$&".l":lgt=1
1280 ON top=3:op$=op$&".w"
1290 END SELect
```



```
1300 op$=op$&"  "&addr$(slice(rest,3,3),slice(rest,0,3))
&","&addr$(slice(rest,6,
2)+4*(bot MOD 2),bot DIV 2)
1310 END DEFine
1320 DEFine PROCedure group4
1330 SELect ON bot
1340    ON bot=2:op$="clr"&type$(slice(rest,6,2))&"
"&addr$(slice(rest,3,3),slice(rest,0,3))
1350    ON bot=4
1360       IF slice(rest,6,2)=3 THEN
1370          op$="move.w  "&addr$(slice(rest,3,3),slice
(rest,0,3))&",ccr"
1380       ELSE
1390          op$="neg"&type$(slice(rest,6,2))&"
"&addr$(slice(rest,3,3),slice(rest,0,3))
1400       END IF
1410    ON bot=6
1420       IF slice(rest,6,2)=3 THEN
1430          op$="move.w  sr,"&addr$(slice
(rest,3,3),slice(rest,0,3))
1440       ELSE
1450          op$="negx"&type$(slice(rest,3,3))&"
"&addr$(slice(rest,3,3),slice(rest,0,3))
1460    END IF
1470    ON bot=6
1480       IF slice(rest,6,2)=3 THEN
1490          op$="move.w  "&addr$(slice(rest,3,3)
,slice(rest,0,3))&",sr"
1500       ELSE
1510          op$="not"&type$(slice(rest,3,3))&"
"&addr$(slice(rest,3,3),slice(rest,0,3))
1520    END IF
1530    ON bot=8:sub8
1540    ON bot=10
1550       IF slice(rest,6,2)=3 THEN
1560          op$="tas  "&addr$(slice(rest,3,3)
,slice(rest,0,3))
1570       ELSE
1580          op$="tst"&type$(slice(rest,6,2))&"
"&addr$(slice(rest,3,3),slice(res
t,0,3))
1590    END IF
1600    ON bot=14:sub14
1610    ON bot=12:movem
1620    ON bot=REMAINDER
1630       xx=slice(rest,6,2)+4*(bot MOD 2)
1640       SELect ON xx
1650          ON xx=6:op$="chk":r$="d"
1660          ON xx=7:op$="lea":r$="a"
1670          ON xx=REMAINDER :op$="error":r$="a"
1680       END SELect
1690    op$=op$&"  "&addr$(slice(rest,3,3),slice
(rest,0,3))&","&r$&(bot DIV 2)
1700 END SELect
1710 END DEFine
1720 DEFine PROCedure group5
1730 IF rest DIV 64=3 THEN
1740    IF bot=1 THEN
1750       op$="ra"
1760    ELSE
1770       op$=cc$(bot)
1780    END IF
1790    IF slice(rest,3,3)=1 THEN
1800       op=PEEK_W(s+op):op=op+2
1810       op$="db"&op$&"  d"&(rest MOD 8)&"
,"&hex$(s+op+4)
1820    ELSE
1830       IF op$="ra" THEN op$="et"
1840       op$="s"&op$&"  "&addr$(slice(rest,3,3)
,rest MOD 8)
1850    END IF
1860 ELSE
1870    IF bot MOD 2=1 THEN
1880       op$="subq"
1890    ELSE
1900       op$="addq"
1910    END IF
1920    p=bot DIV 2:IF p=0 THEN p=8
1930    op$=op$&"  #"&hex$(p)&","&addr$(slice(rest,3,3)
,rest MOD 8)
1940 END IF
1950 END DEFine
1960 DEFine PROCedure group6
1970 IF rest=0 THEN
1980    oper=PEEK_W(s+op):op=op+2
1990    IF oper>32767 THEN oper=-(65536-oper)
2000 ELSE
2010    oper=rest
2020    IF oper>127 THEN oper=-(256-oper)
2030 END IF
2040 op$="b"&cc$(bot)&"  "&hex$(s+oper+2)
2050 END DEFine
2060 DEFine PROCedure group7
2070 IF (bot MOD 2) THEN
2080    op$="error":agh=1
2090 ELSE
2100    op$="moveq  #"&hex$(rest)&",d"&(bot DIV 2)
2110 END IF
2120 END DEFine
2130 DEFine PROCedure group8
2140 IF slice(rest,6,2)=3 THEN
2150    IF (bot MOD 2)=1 THEN
2160       op$="divs"
2170    ELSE
2180       op$="divu"
2190    END IF
2200    op$=op$&"  d"&(bot DIV 2)&","&addr$(slice
(rest,3,3),slice(rest,0,3))
2210 ELSE
2220    IF slice(rest,3,5)=1 AND (bot MOD 2)=1 THEN
2230       op$="sbcd  d"&(rest MOD 8)&",d"&(bot DIV 2)
2240    ELSE
2250       op$="or"&type$(slice(rest,6,2))
2260       IF (bot MOD 2)=1 THEN
2270          op$=op$&"  d"&(bot DIV 2)&","&raddr$(slice
(rest,3,3),slice(rest,0,3))
2280       ELSE
2290          op$=op$&"  "&addr$(slice(rest,3,3),slice
(rest,0,3))&",d"&(bot DIV 2)
2300       END IF
2310    END IF
2320 END IF
2330 END DEFine
2340 DEFine PROCedure group9
2350 IF (bot MOD 2)=1 AND slice(rest,3,3)=0 THEN
2360    op$="subx"&type$(slice(rest,6,2))&"
d"&(rest MOD 8)&",d"&(bot DIV 2)
2370 ELSE
2380    IF slice(rest,6,2)=3 THEN
2390       op$="suba"
2400       IF bot MOD 2=0 THEN
2410          op$=op$&".w  "
2420       ELSE
2430          op$=op$&".l  "
2440       END IF
2450       op$=op$&addr$(slice(rest,3,3),slice
(rest,0,3))&",a"&(bot DIV 2)
2460    ELSE
2470       op$="sub"&type$(slice(rest,6,2))
2480       IF (bot MOD 2)=1 THEN
2490          IF slice(rest,3,3)=1 THEN op$="error":agh=1
2500          op$=op$&"  d"&(bot DIV 2)&","&raddr$(slice
(rest,3,3),slice(rest,0,3))
2510       ELSE
2520          op$=op$&"  "&addr$(slice(rest,3,3),slice
(rest,0,3))&",d"&(bot DIV 2)
2530       END IF
2540    END IF
2550 END IF
2560 END DEFine
2570 DEFine PROCedure user
2580 op$="udef"
2590 END DEFine
2600 DEFine PROCedure groupb
2610 IF (rest DIV 64)=3 THEN
2620    op$="cmpa"
2630    IF bot MOD 2=1 THEN
2640       op$=op$&".l":lgt=1
2650    ELSE
2660       op$=op$&".w"
2670    END IF
2680    op$=op$&"  "&addr$(slice(rest,3,3)
,slice(rest,0,3))&",a"&(bot DIV 2)
2690    IF type$(rest DIV 64)=".l" THEN lgt=1
2700 ELSE
2710    IF bot MOD 2=1 THEN
2720       op$="cmpm"&type$(rest DIV 64)&"
(a"&(rest MOD 8)&")+,(a"&(bot MOD 2)
2730    ELSE
2740       op$="cmp"&type$(slice(rest,6,2))&"
"&addr$(slice(rest,3,3),slice(res
t,0,3))&",d"&(bot DIV 2)
2750    END IF
2760 END IF
2770 END DEFine
2780 DEFine PROCedure groupc
```

continued next week

Take the money and run

A J Laurance presents a utility to display and auto-run programs on the QL

Boot is not a device for kicking th dog but a programe to display a menu of programs and autoload the selected program for the Sinclair QL.

The program is activated by the autoload facility of the QL on powerup or reset. Any program called *Boot* on microdrive one will be automatically loaded and run after the powerup initialisation. The display on this program has been organised for the "TV" display option but could be modified for the monitor option if needed.

To create a datafile, type in *GOTO 2000*. To edit a single entry, type *change*. To save changed data, you type *save*.

Note that in the listing hash appears as a £ sign, and underscore as ←. Also the QL needs an 'intelligent' printer to produce listings, and my printer will not automatically overspill on to another line, but stops printing, and needs both carriage return and line feed which the QL does not produce. As a result I had to produce the listing as a data file to import to the Quill word processor, in order to use the excellent Quill printer driver. Due to Quill's word wrap which you cannot switch off, a new line is produced too soon, so as not to break up words. Thus, the listing does not look exactly as it does on the screen (eg, Line 2100).

The method by which I achieved this might be of interest, as I have seen it stated that this feat is not possible. The following actions are done in direct mode so as not to appear in the listing itself.

Firstly, *OPEN ← new£5, mdv1 ← listing ← exp*. A channel to microdrive is opened called listing←exp (for export to Quill it will not work without the EXP). Next *LIST £5*. The listing is sent to the microdrive file. Finally, CLOSE£5; (Very important as the lack of end of file marker will cause the whole of Quill to crash).

Load *Quill*, enter *"Files"* via the commands and press *I* for import, followed by th filename *"listing"* (without the exp). The listing will then be on a valid Quill file.

Program Notes

Line 1000 to 1040 Initialises screen and creates arrays for data. Up to nine programs can be placed on the menu. By omitting the commentary, more could be displayed on screen.

Line 1060 to 1300 Reads data for titles and commentary from file called Boot data. Not all nine possible choices have to be filled.

Line 1320 to 1350 Reads keyboard for option one to nine.

Line 1360 to 1400 Loads and runs selected program.

Line 2000 to 2100 Creates data for title and description.

Line 3000 to 3050 Procedure for data input.

Line 3100 to 3130 Procedure to change data.

Line 4000 to 4080 Procedure to save data.

```
99 :
100 REMark Boot programm
101 :
102 REMark wafer must be in drive 1
999 :
1000 REMark display
1001 :
1005 MODE 4:CSIZE 2,1:PAPER 0:INK 7
1010 PRINT'              BOOT Autoload'
1020 CSIZE 1,0
1030 DIM title$(9,16)
1040 DIM description$(9,80)
1060 OPEN←IN £6,mdv1←boot←data
1070 FOR n=1 TO 9
1100  INPUT£6,title$(n)
1110  INPUT£6,description$(n)
1130  CSIZE 2,0:AT n*2,14: PRINT n! title$(n)!n
1140  CSIZE 1,0: PRINT description$(n)
1300 END FOR n
1310 CSIZE£0; 2,0:PRINT£0;'press number key for required
program':CSIZE £0;0,0
1320 REPeat inputkey
1330  LET n$=INKEY$(-1):LET n=CODE(n$)
1340  IF n<56 AND n>47 THEN EXIT inputkey
1350 END REPeat inputkey
1360 CLS:CSIZE 2,1:AT 5,8:PRINT "loading ";title$(n$)
1400 LRUN 'mdv1←'&title$(n$)
1999 :
2000 REMark create datafile
2001 :
2030 DIM title$(10,16)
2040 DIM description$(10,80)
2050  FOR n=1 TO 9
2060  input←data
2070  IF title$(n)=''THEN EXIT n
2080  END FOR n
2100 PRINT£0;'Change any data? ':LET a$=INKEY$(-1):IF a$=
'y' OR a$='Y'THEN change:GO TO 2100
2110 save←data
2200 STOP
2999 :
3000 REMark data input
3010 DEFine PROCedure input←data
3030 PRINT£0;n!'program name-';:INPUT title$(n):PRINT
 n!title$(n)
3040 PRINT£0;n!'comment- (80 chars)'\:INPUT description$
(n):PRINT title$(n)
3050 END DEFine
3100 DEFine PROCedure change
3110 PRINT£0;'input number to change ':LET n=INKEY$(-1)
3120 input←data
3130 END DEFine
3999 :
4000 REMark save data
4001 :
4005 DEFine PROCedure save←data
4010  DELETE mdv1←boot←data
4020  OPEN←NEW £6,mdv1←boot←data
4030   FOR n=1 TO 9
4040   PRINT £6, title$(n)\description$(n)
4060  END FOR n
4070  CLOSE£6
4080 END DEFine
4100 :
30000 SAVE mdv1←boot
```

Tracing a line

Andrew Pennell makes use of the QL's multi-tasking to give the machine a Trace *facility*

This program uses the most powerful feature of the QL — multi-tasking — to add a *Trace* ability to SuperBasic.

To my knowledge, not only is this the first QL machine-code program to be published, but it must be the first to use multi-tasking. What it does is to set up a small program that constantly monitors Basic, and prints the current line number at the top of the screen. It can do this as it seemingly runs at the same time as the Basic interpreter.

To use it, firstly type in the listing, and save it before running. Next, *Run* it, and you should be greeted with *"loaded OK"*. If you get *"wrong data"*, then you must have made a mistake in the data somewhere. To turn the *Trace* on, you have to *Call 261120* (It's important that you only do this call once). You should get "−0" printed at the very top left of the screen. Next you should *Call 261192,8*, which sets the speed of the trace to an average value. From now on, any program that *Runs* should be accompanied by a display along the top of the screen of the line number each time it changes, separated by dashes.

On my television, there is quite a gap between the top of the screen, and the top of the listing window. If you don't have such a gap on yours, you can change Line 160 to position your *Trace* window at a more convenient place, but try not to make it clash with any other windows. If you changed it to *a$="ser1"* then the trace will appear on a printer, but firstly set the printer's width using control codes, or else it will all be printed on one line of paper!

A machine-code program that runs under multi-tasking is known as a 'job', and normally on the QL only one job is running — the Basic interpreter. However, what the machine-code does is to set up a second job, the sole purpose of which is to print line numbers every time they change. Each job has a speed factor, from 1 to 32, and this determines how fast it runs compared to the other jobs. Basic runs at the maximum of 32, but *Trace* works OK at a speed of 8, and this is what the *CALL 261192* is for — you can change the speed of the *Trace*. Normally 8 is OK, but sometimes 16 gives better results, and if you want to be extravagant you could get it to run at 32. Note that the faster you make the *Trace*, the slower Basic runs at, so that a *Trace* speed of 32 will make Basic half its normal speed. A speed of 0 will switch *Trace* off, and make Basic run normally.

With *Trace* enabled, the *Respr* function cannot be used, and will give a *'not complete'* error. Also, during some I/O operations, you can get strange numbers printed, and a *Mode* instruction makes the print-out disappear, for (as yet) unknown reasons. After you've run the loader program, you can safely do a *New* — *Trace* will remain intact, and possibly running.

An assembly language listing of this program, along with masses of other information, will shortly be available in my forthcoming book *Assembly Language Programming on the Sinclair QL* published by Sunshine Books.

```
10 REMark ****************************
20 REMark ********* TRACE ***********
30 REMark **(C) Andrew Pennell 1984**
40 REMark ****************************
50 REPeat makeroom
60 IF RESPR(0)<=261120 THEN EXIT makeroom
70 a=RESPR(1024)
80 END REPeat makeroom
90 t=0
100 RESTORE
110 FOR i=261120 TO 261305
120 READ a
130 POKE i,a:t=t+a
140 NEXT i
150 IF t<>12642 THEN PRINT #0;"wrong
data":STOP

160 a$="scr_400x12a40x4"
170 POKE_W 261220,LEN(a$)
180 FOR i=1 TO LEN(a$):POKE 261221+i,
CODE(a$(i))
190 PRINT "TRACE loaded OK"
210 PRINT "to setup: CALL 261120"
220 PRINT "speed   : CALL 261192,?"
230 DATA 114,0,36,60,0,0,0,62
240 DATA 66,131,34,67,112,1,78,65
250 DATA 67,250,0,76,34,129,34,124
260 DATA 0,3,252,124,36,60,0,0
270 DATA 0,61,16,217,81,202,255,252
280 DATA 112,1,118,2,65,250,0,54
290 DATA 78,66,67,250,0,38,34,136
300 DATA 16,60,0,10,34,58,0,32
310 DATA 116,1,118,0,78,65,78,117
320 DATA 34,121,0,2,128,104,34,105
330 DATA 0,4,19,65,0,19,66,128
340 DATA 78,117,0,0,0,0,0,0
350 DATA 0,0,255,255,0,15,83,67
360 DATA 82,95,52,48,48,88,49,50
370 DATA 65,52,48,88,52,0,0,0
380 DATA 0,0,0,0,46,124,0,4
390 DATA 0,0,32,121,0,3,252,90
400 DATA 34,121,0,2,128,16,50,41
410 DATA 0,208,178,121,0,3,252,98
420 DATA 103,232,51,193,0,3,252,98
430 DATA 18,60,0,45,118,255,112,5
440 DATA 78,67,50,57,0,3,252,98
450 DATA 52,121,0,0,0,206,78,146
460 DATA 96,200
```

Round the clock

Ian Logan presents a clock program that demonstrates a large number of the features of Superbasic

The Superbasic of the QL is very different from the popular Sinclair Basic found in the Spectrum; and it will take some time for a new owner of a QL to become fluent in its use.

The following program produces both 'analog' and 'digital' clocks and shows a large number of the features of Superbasic. Initially, you might think that the listing is more like one for the BBC microcomputer; and with respect to the manner in which Superbasic allows the use of named procedures, you would be correct. But, Superbasic allows a lot more than just the use of procedures.

So, taking each section of the program in turn:

Lines 160-290:
The procedure *Set* is defined. Mode 4 — the high definition mode — is selected so as to take full advantage of the QL's potential.

WINDOW 412,256,0,0 — creates a window of maximum size.
BORDER 30 — creates a border within this window of width 30 pixels.
CSIZE 3,1 — selects the largest of the standard type sizes.
CURSOR 0,100 — moves the cursor down 100 pixels within the available window.
SDATE 1984,6,1,h,m,s, — this program cheats by using the function *Dates* to store the current time; and this line sets *Dates* to the required time (on 1 June, 1984).

Lines 310-560:
The procedure *Face* is defined.
SCALE 200,-150,-100 — this line 'scales' the output window to give it 200 graphic lines (instead of 100) on its vertical scale. The origin of the bottom left corner is given the coordinates -150,-100 (so as to make the centre of the clock face be 0,0).
The screen is then cleared and three circles are drawn to represent a clock face.
The *For a — End for* structure in lines 400 to 550 uses the 'turtle graphics' of the QL to draw minute marks on the clock face at the required positions. The numbers are added to the clock face after first setting the print cursor to a suitable position (line 530).

Lines 580-680:
The procedure *Time* is defined.
This procedure contains a *Repeat — END REPeat* structure (lines 610 670) from which there is no exit. And, if the value of *Dates* is found to be changed — as it will be every second — the procedures *Watch* and *Digital* are called.

Lines 700-790:
The procedure *Watch* is defined. This procedure determines which hands of the clock face are to be re drawn. If a hand is to be moved, then it is first overdrawn with white ink, before being re drawn at its new position with black ink.

Lines 810-890:
The procedure *Hand* is defined. This procedure requires three parameters; i - the colour of ink to be used,
r - the angle *Turned* at the centre of the clock
l - the length of the hand to be drawn.
The hands are drawn using the 'turtle graphics' of the QL.

Lines 910-960:
The procedure *Digital* is defined. This simple procedure prints the appropriate slice of the *Newtimes*, after first suppressing an initial zero in this string of characters.

```
100 REMark      QL CLOCK
110 set
120 face
130 time
140 STOP
150 REMark ......................
160 DEFine PROCedure set
170 MODE 4
180 WINDOW 512,256,0,0
190 PAPER 0
200 INK 7
210 CLS
220 BORDER 30
230 CSIZE 3,1
240 CURSOR 0,100
250 PRINT "Enter"
260 PRINT "Hours, Minutes, Seconds"
270 INPUT h,m,s
280 SDATE 1984,6,1,h,m,s
290 END DEFine
300 REMark ......................
310 DEFine PROCedure face
320 SCALE 200,-150,-100
330 BORDER 0
340 PAPER 7
350 INK 0
360 CLS
370 CIRCLE 0,0,85
380 CIRCLE 0,0,52
390 CIRCLE 0,0,4
400 FOR a=59 TO 0 STEP -1
410 LINE 0,0
420 TURNTO a*6
430 PENUP
440 MOVE 46
450 PENDOWN
460 INK 0
470 MOVE 2+3*NOT(a MOD 5)
480 END FOR a
490 FOR a=330 TO 0 STEP -30
500 n=3-a/30+12*(a/30>2)
510 x=60*COS(a*PI/180)
520 y=63*SIN(a*PI/180)
530 CURSOR x-9*(n>9)+4*(n>11),v
,-7,-9 540 PRINT n
550 END FOR a
560 END DEFine
570 REMark ......................
580 DEFine PROCedure time
590 s1=99: m1=99: h1=99
600 oldtime$=""
610 REPeat update
620 newtime$=DATE$
630 IF newtime$=oldtime$ THEN NEXT
update
640 watch
650 digital
660 oldtime$=newtime$
670 END REPeat update
680 END DEFine
690 REMark ......................
700 DEFine PROCedure watch
710 s=newtime$(19 TO 20)
720 m=.5*(s>29)+newtime$(16 TO 17)
730 h=INT(5*(m/60+newtime$(13 TO 14
))) 740 IF hJ<>h THEN hand 7,hJ,25
750 IF mJ<>m THEN hand 7,mJ,40
760 hand 7,s1,15: hand 0,s,15
770 hand 0,m,40: hand 0,h,25
780 s1=s: hJ=h: mJ=m
790 END DEFine
800 REMark ......................
810 DEFine PROCedure hand (i,r,l)
820 LINE 0,0
830 TURNTO -6*r+90
840 PENUP
850 MOVE 5
860 PENDOWN
870 INK i
880 MOVE l
890 END DEFine
900 REMark ......................
910 DEFine PROCedure digital
920 digit$=newtime$(13 TO)
930 IF digit$(1)="0" THEN digit$(1)
=" "940 CURSOR 40,220
950 PRINT digit$
960 END DEFine
970 REMark ......................
```


One file at a time

***Malcolm Bryant** demonstrates a file copy program for the QL*

This program will copy the entire contents of one microdrive cartridge onto another. The *Clone* programs supplied with the QL only copy the Psion software cartridges and are not general-purpose utilities.

The QL manuals specifically recommend that microdrive cartridges are backed-up frequently. This file copy program can prevent the tedious business of typing in a separate *Copy* command for each file on the cartridge. The user has the option of copying the whole cartridge in one operation, or of being prompted by the program for one file at a time.

The file copy program works by sending a directory listing to a special temporary file on the 'destination' microdrive. This file is subsequently read back and the information is used for the copying process. Finally, the temporary file is then deleted and the contents of the 'destination' cartridge are listed on the screen.

Note that, the symbols printed in the listing as £ signs should be typed in as hash (#) marks.

```
100 CLS:PRINT"File copy program"
110 PRINT"-----------------"
120 REPeat drv
130 INPUT"Which drive are you copying from? "\d:IF d=1 OR d=2 THEN EXIT drv
140 END REPeat drv
150 s$=d:t$=3-d
160 PRINT"Press ENTER to copy everything or anyother key to copy individual files"
170 e=CODE(INKEY$(-1))
180 PRINT"Accessing microdrive"
190 n$="mdv"&t$&"_temp"
200 OPEN_NEW £5,n$
210 DIR £5,"mdv"&s$&"_"
220 CLOSE £5
230 OPEN £5,n$
240 INPUT £5\a$\a$:
250 REPeat loop
260 INPUT £5\a$:
270 f=10
280 IF NOT e=10 THEN
290 PRINT"Press ENTER to copy file "\a$
300 f=CODE(INKEY$(-1))
310 END IF
320 IF f=10 THEN
330 PRINT"Copying "\a$
340 COPY "mdv"&s$&"_"&a$ TO "mdv"&t$&"_"&a$
350 ELSE
360 PRINT"File "\a$:" not copied"
370 END IF
380 IF EOF(£5) THEN
390 CLOSE £5:DELETE n$
400 DIR "mdv"&t$&"_":EXIT loop
410 END IF
420 END REPeat loop
```

Sounding off

Dilwyn Jones sounds off 'syntactically' using his QL...Beep...Beep...Beep

The QL's *Beep* command can accept a variable number of parameters to produce a one-channel complex sound.

As well as being so versatile, this means that, initially at least, complex sounds are just that. Indeed, the QL manual states unhelpfully that the "command is best used experimentally rather than syntactically".

Reading that made me think that some kind of sound development program was called for. The program I have written enables you to have an on-screen display of the current parameter values and to play or cancel any sound using the values displayed. Any value can be changed providing that it does not cause an overflow error.

Briefly, these are the eight values used with the SuperBasic *Beep* command, in order:

Duration controls the time for which the sound is played, from a value of 0 playing the note continuously until cancelled to the shortest note with a value of 1.

Pitch 1 sets the pitch. A low value is a high pitch (short period). The use of just the two parameters *Duration* and *Pitch-1* gives the simplest type of one-note bleep, as on the ZX Spectrum.

Pitch-2 is the other limit of a note of varying pitch. This may have a higher or lower value than *Pitch-1*. If suitably set up, the sound can alternate between *Pitch-1* and *Pitch-2*.

Grad-X is the rate at which the sound changes by steps of *Grad-Y*. Contrary to what is said in the manual, this accepts values in the range 0 to 32767.

Grad-Y is the step value used while alternating between *Pitch-1* and *Pitch-2*. This gives the change of pitch between every sound played. It takes either a negative or positive value corresponding to ascending or descending notes.

Wrap determines whether the sound goes up and down in pitch or just in one direction, and how many times. A value of 15 means 'wrap forever'.

Fuzzy changes the pitch randomly every cycle. A high value (eg, 15) tends to make most sounds end up like white noise.

Random changes the gradient randomly. The effect depends on other parameters, but is often quite noticeable!

To simulate notes without the full range of eight parameters, any redundant values should be assigned a zero value. When you run the program, it displays the eight *Beep* parameters, their current values and the limits which the values can take. At the bottom of the screen is a display of all the controls available.

At any time the parameter you are working on is displayed in reversed video (ie, red numbers on a white strip). You can move up and down the list with the up and down cursor keys at the bottom of the keyboard. The white strip moves to show which parameter you're currently working on.

The function key *f1* at the left of the keyboard is used to increase the value of the parameter. Pressing *f1* alone increases the value by 1. Pressing *Ctrl f1* increases the value by 10. Pressing *Shift f1* increases the value by 100. Pressing *Ctrl shift f1* increases the value by 1000. Decreasing values is done with the *f2* function key in the same steps as for *f1*. This is all displayed at the bottom of the screen as a constant reminder of the controls available.

To hear the sound, press either the *f3* function key or the *P* (play) key. QL sound is quite loud, so you need a way of aborting any long sound. Pressing *f4* or the *C* key will cancel the current sound. If you want to quit the program simply press the *Q* key. This does not cancel any sound already set so be sure to do this first, or you may have to enter *Beep* alone as a direct command to make your QL shut up!

The issue of QL that I have been using has an annoying arithmetic habit of making −1 −1 and −2 −2 both equal to 0, but any other numbers are evaluated correctly (eg, −3 −3 gives −6). If this happens on your QL as well, you may be unable to step down negative values from −1 to −2 with the program, so you should step down by, for example, −10 then step back up again in steps of 1.

The program uses many of the QL's facilities, with not a *Goto* in sight. Long variable names are used throughout, which means that there is a lot of typing to be done, but don't let this discourage you. I find this program very easy to use and very useful for developing sounds for SuperBasic games.

```
100 REMark Sound Development Program
110 :
120 REMark    (C) DILWYN JONES 1984
130 initialise
140 REPeat program_loop
150   key=CODE(INKEY$(-1))
160   IF key=99 OR key=67 OR key=244
THEN cancel_sound
170   LET previous=parameter
180   IF key=208 AND parameter>0 THEN
 LET parameter=parameter-1
190   IF key=216 AND parameter<7 THEN
 LET parameter=parameter+1
200   IF previous<>parameter THEN
210     AT 11,previous*6
220     PAPER 2
230     INK 7
240     PRINT value(previous);FILL$('
 ',6-LEN(value(previous)))
250   END IF
260   IF key=232  THEN LET value(para
meter)=value(parameter)+1
270   IF key=233 THEN LET value(param
eter)=value(parameter)+10
280   IF key=234 THEN LET value(param
eter)=value(parameter)+100
290   IF key=235 THEN LET value(param
eter)=value(parameter)+1000
300   IF value(parameter)>limit(highe
st,parameter)  THEN LET value(paramete
r)=limit(highest,parameter)
310   IF key=216 THEN LET value(param
eter)=value(parameter)-1
320   IF key=217 THEN LET value(param
eter)=value(parameter)-10
330   IF key=218 THEN LET value(param
eter)=value(parameter)-100
340   IF key=219 THEN LET value(param
eter)=value(parameter)-1000
350   IF value(parameter)<limit(lowes
t,parameter) THEN LET value(parameter
)=limit(lowest,parameter)
360   IF key=113 OR key=81 THEN EXIT
program_loop
370   IF key=112 OR key=80 OR key=240
 THEN play_sound
380   IF key=99 OR key=67 OR key=244
THEN cancel_sound
390   AT 11,parameter*6
400   PAPER 7
410   INK 2
420   PRINT value(parameter);
430   INK 7
440   PAPER 2
450   PRINT FILL$(' ',6-LEN(value(par
ameter)))
460 END REPeat program_loop
470 STOP
480 DEFine PROCedure play_sound
490   BEEP value(0),value(1),value(2)
,value(3),value(4),value(5),value(6),
value(7)
500 END DEFine play_sound
510 DEFine PROCedure cancel_sound
520   IF BEEPING THEN BEEP
530 END DEFine cancel_sound
540 DEFine PROCedure initialise
550   INK 7
560   PAPER 2
570   MODE 4
580   CLS
590   CSIZE 1,0
600   PRINT'BEEP duration,pitch_1,pit
ch_2,grad_x,grad_y,wrap,fuzzy,'rand
om'
610   CSIZE 2,0
620   AT 0,3
630   PRINT'-----------------------
----------'
640   PRINT'Parameter   Value    Limit
s'
650   PRINT'-----------------------
----------'
660   PRINT'duration    0        -3276
8 to 32767'
670   PRINT'pitch_1     0        0 to
255'
680   PRINT'pitch_2     0        0 to
255'
690   PRINT'grad_x      0        0 to
32767'
700   PRINT'grad_y      0        -8 to
 7'
710   PRINT'wrap        0        0 to
15'
720   PRINT'fuzzy       0        0 to
15'
730   PRINT'random      0         3276
8 to 32767'
740   PRINT'-----------------------
----------'
750   PRINT
760   CSIZE 1,0
770   PRINT'   value   1      10
100         1000       up/down'
780   PRINT'increase  f1  CTRL f1  SH
IFT f1  CTRL-SHIFT f1  cursor'
790   PRINT'decrease  f2  CTRL f2  SH
IFT f2  CTRL-SHIFT f2  keys'
800   PRINT'f3/p play sound      f4/c
 cancel sound      q quit';
810   LINE 0,20 TO 165,20
820   LINE 0,15 TO 139,15
830   LINE 0,5 TO 165,5
840   LINE 27,5 TO 27,20
850   LINE 39,5 TO 39,20
860   LINE 65,5 TO 65,20
870   LINE 95,5 TO 95,20
880   LINE 139,5 TO 139,20
890   LINE 54,0 TO 54,5
900   LINE 125,0 TO 125,5
910   CSIZE 2,0
920   DIM limit(1,7),value(7)
930   LET lowest=0:LET highest=1
940   LET duration=0:LET random=7
950   RESTORE
960   FOR parameter=duration TO random
970     READ limit(lowest,parameter),
limit(highest,parameter)
980   END FOR parameter
990   DATA -32768,32767,0,255,0,255,0
,32767,-8,7,0,15,0,15,-32768,32767
1000  parameter=duration
1010  LET previous=parameter
1020  PAPER 7
1030  INK 2
1040  AT 11,parameter*6
1050  PRINT 0
1060  INK 7
1070  PAPER 2
1080 END DEFine initialise
```

Proceeding nicely

Microdrive operation made easy with a useful procedure utility written by **Bob Gavin**

The program defines eighteen short utilities making QL operation simpler and combines them in single procedure operator program — *User Procedures*.

After the program is loaded any of the routines — for example printing a microdrive directory or formatting a blank cartridge — can be carried out using only a couple of simple key-presses.

A while back when I was using my Sinclair Spectrum I decided that it would be a good idea to define numeric variables such and set them equal to the respective starting line numbers of a *Gosub* routines within a program that executed such 'procedures' — the idea being to call such procedures or *Gosub* routines as required.

Then along came the QL with its *DEFine PROCedure* command, an answer to all my problems.

Wouldn't it be nice, I thought, if I could just type *D1* for a directory listing of microdrive 1 instead of the long-winded command necessary.

See for yourself what the results of such thinking by typing in the listing below. It may not be the best solution but I find *User Procedures* very useful. It would be easy to extend the program to include additional procedures.

Points you should be aware of include:

1) I have an early version of the QL where the co-ordinates of the *At* command are reversed. If your machine is without the dohgle swap these values round for each *At* command.

2) Beware of using procedures or variables with the same name as those in *User Procedures*. Errors may be abundant.

3) Type *Help* if you find yourself in difficulties, this will display the Help Screen.

4) Occasionally the *Save* procedures do not work first time around. If this occurs use the *Retry* command. I am told this is not a problem on the most recent versions of the QL.

5) The *Clear* command tends to sort out problems if your machine is acting up. Use it when all else fails.

6) The program is called 'boot' and as such may be loaded in the same manner as the packages supplied with the QL.

7) The £ sign in the listing should be typed in as a# sign.

8) *User Procedures* starts at line 29999. No program should be loaded subsequently which use these line numbers greater than 29999.

The following is a list of key-presses and their functions.

Press	Routine
L1	Loads from mdv1
L2	Loads from mdv2
D1	Directory of mdv1
D2	Directory of mdv2
S1	Saves program to mdv1
S2	Saves program to mdv2
DEL1	Delete from mdv1
DEL2	Delete from mdv2
SPROC	Save current user procedures
DPROG	Deletes current program from memory
DTE	Sets clock
R	Run program
L	List program
F2	Format mdv2
C1	Copy from mdv1 to mdv2

```
 29999 INIRO
30000 REMark ***********************************************
30005 DEFine PROCedure INIRO
30010 MODE 4
30015 SCREEN1
30020 BLOCK 511,95,0,160,4
30025 CSIZE 3,1
30030 AT 9,1
30035 PRINT "W E L C O M E"
30040 AT 14,3
30045 PRINT "I N"
30050 AT 12,5
30055 PRINT "U S E R"
30060 AT 6,7
30065 PRINT "P R O C E D U R E S"
30070 CSIZE 0,0
30075 INK 0
30080 PAPER 4
30085 AT 20,17
30090 PRINT "THE SYSTEM CLOCK IS SET AS"
30095 AT 31,19
30100 PRINT DATE$
30105 AT 26,21
30110 PRINT "Press  SPACE' to change date or"
30115 AT 26,22
30120 PRINT "Press any other key to continue"
30125 AT 34,24
30130 PRINT "VERSION 1.01"
30135 IF INKEY$="" THEN GO TO 30095
30140 IF KEYROW(1)=64 THEN DTE
30145 HELP
30150 END DEFine INIRO
30155 REMark ***********************************************
30160 DEFine PROCedure DTE
30165 SCREEN2
30170 PRINT "CHANGE DATE"
30175 PRINT
30180 INPUT "YEAR          (YYYY) = ";YEAR
30185 INPUT "MONTH           (MM) = ";MONTH
30190 INPUT "DAY             (DD) = ";DAY
30195 INPUT "HOURS           (HH) = ";HOURS
30200 INPUT "MINUTES         (mm) = ";MINUTES
30205 PRINT "ONCE THE VALUE FOR SECONDS IS INPUT THEN THE SYSTEM
CLOCK WILL BE SET"
30210 INPUT "SECONDS         (ss) = ";SECONDS
30215 SDATE YEAR,MONTH,DAY,HOURS,MINUTES,SECONDS
30220 PRINT
30225 PRINT "THE SYSTEM CLOCK IS SET AS"
30230 AT 0,11
30235 PRINT DATE$
30240 PRINT
30245 PRINT "Press any key to continue"
30250 IF INKEY$="" THEN GO TO 30230
30255 HELP
30260 REMark ***********************************************
30265 DEFine PROCedure ENTER
30270 CLS
30275 PRINT "ENTER FILE NAME "!
30280 END DEFine ENTER
30285 REMark ***********************************************
30290 DEFine PROCedure L1
30295 LOCal X$
30300 ENTER
30305 PRINT "TO BE LOADED FROM MDV1_"
30310 INPUT X$
30315 MERGE "MDV1_"&X$
30320 RUN
30325 END DEFine L1
30330 REMark ***********************************************
30335 DEFine PROCedure L2
30340 LOCal X$
30345 ENTER
30350 PRINT "TO BE LOADED FROM MDV2_"
30355 INPUT X$
30360 MERGE "MDV2_"&X$
30365 RUN
30370 END DEFine L2
30375 REMark ***********************************************
30380 DEFine PROCedure D1
30385 CLS
30390 PRINT "DIRECTORY MDV1_"
30395 DIR MDV1_
30400 END DEFine D1
30405 REMark ***********************************************
30410 DEFine PROCedure D2
30415 CLS
30420 PRINT "DIRECTORY MDV2_"
30425 DIR MDV2_
30430 END DEFine D2
30435 REMark ***********************************************
30440 DEFine PROCedure S1
30445 LOCal X$
30450 ENTER
30455 PRINT "TO BE SAVED TO MDV1_"
30460 INPUT X$
30465 SAVE "MDV1_"&X$, 10,29999
30470 HELP
30475 END DEFine S1
30480 REMark ***********************************************
30485 DEFine PROCedure S2
30490 LOCal X$
30495 ENTER
30500 PRINT "TO BE SAVED TO MDV2_"
30505 INPUT X$
30510 SAVE "MDV2_"&X$, 10,29999
30515 HELP
30520 END DEFine S2
30525 REMark ***********************************************
30530 DEFine PROCedure DEL1
30535 LOCal X$
30540 ENTER
30545 PRINT "TO BE DELETED FROM MDV1_"
30550 INPUT X$
30555 DELETE "MDV1_"&X$
```



```
30560 HELP
30565 END DEFine DEL1
30570 REMark ***********************************************
30575 DEFine PROCedure DEL2
30580 LOCal X$
30585 ENTER
30590 PRINT "TO BE DELETED FROM MDV2_"
30595 INPUT X$
30600 DELETE "MDV2_"&X$
30605 HELP
30610 END DEFine DEL2
30615 REMark ***********************************************
30620 DEFine PROCedure SPROC
30625 CLS
30630 PRINT "SAVING CURRENT PROCEDURES..."
30635 DELETE adv1_boot
30640 SAVE adv1_boot
30645 HELP
30650 END DEFine SPROC
30655 REMark ***********************************************
30660 DEFine PROCedure DPROG
30665 CLS
30670 PRINT "DELETEING CURRENT PROGRAM..."
30675 LRUN adv1_boot
30680 END DEFine DPROG
30685 REMark ***********************************************
30690 DEFine PROCedure R
30695 RUN
30700 END DEFine R
30705 REMark ***********************************************
30710 DEFine PROCedure L
30715 LIST
30720 END DEFine L
30725 REMark ***********************************************
30730 DEFine PROCedure F2
30735 LOCal X$
30740 CLS
30745 PRINT "FORMAT MDV2_"
30750 PRINT "ENTER MICRODRIVE FORMAT NAME"
30755 INPUT X$
30760 FORMAT "MDV2_"&X$
30765 HELP
30770 END DEFine F2
30775 REMark ***********************************************
30780 DEFine PROCedure HELP
30785 SCREEN2
30790 PRINT "HELP"
30795 PRINT "INIRO     Introduces user procedures and option to
set clock"
30800 PRINT "L1        LOAD from mdv1_"
30805 PRINT "L2        LOAD from mdv2_"
30810 PRINT "D1        DIRectory of mdv1_"
30815 PRINT "D2        DIRectory of mdv2_"
30820 PRINT "S1        SAVE to mdv1_"
30825 PRINT "S2        SAVE to mdv2_"
30830 PRINT "DEL1      DELETE from mdv1_"
30835 PRINT "DEL2      DELETE from mdv2_"
30840 PRINT "SPROC     SAVE current user procedures"
30845 PRINT "DPROG     DELETE current program from memory"
30850 PRINT "DTE       Sets clock"
30855 PRINT "R         RUN program"
30860 PRINT "L         LIST program"
30865 PRINT "F2        FORMAT mdv2_"
30870 PRINT "C1        COPY from mdv1_ to mdv2_"
30875 PRINT "HELP      Displays this screen"
30880 PRINT "Choose desired option"
30885 END DEFine HELP
30890 REMark ***********************************************
30895 DEFine PROCedure C1
30900 ENTER
30905 PRINT "TO BE COPIED FROM MDV1_ TO MDV2_"
30910 INPUT X$
30915 COPY "MDV1_"&X$ TO "MDV2_"&X$
30920 HELP
30925 END DEFine C1
30930 REMark ***********************************************
30935 DEFine PROCedure SCREEN1
30940 WINDOW #1,512,256,0,0
30945 PAPER 2
30950 INK 7
30955 CLS
30960 END DEFine SCREEN1
30965 REMark ***********************************************
30970 DEFine PROCedure SCREEN2
30975 SCREEN1
30980 WINDOW #0,452,32,30,224
30985 WINDOW #1,452,208,30,16
30990 WINDOW #2,452,208,30,16
30995 PAPER #0,2
31000 INK #0,7
31005 CLS #0
31010 PAPER #2,0
31015 INK #2,4
31020 CLS #2
31025 CLS
31030 END DEFine SCREEN2
31035 REMark ***********************************************
```

Dragon Slayer

Rid the land of ferocious dragrons armed with only a sturdy bow, in this arcade game by Richard Snowdon for the QL

Many years ago, when dragons roamed the forests, a typical day's work may have involved ridding the land of herds of ferocious dragons. But this is no easy task, as the dragons are devious, and powerful and will soon defeat you if you lack concentration.

Here, the QL becomes your own personal hunting trainer to tune up your reflexes. A direct hit is required to destroy a dragon, but plan your attacks well, as you have only six arrows. Once this stock is depleted, you must venture across the river and enter th castle to re-stock.

Use the cursor keys to move, and the *AL* key to fire. To enter the castle, stand just i front of the castle door and step inside whe it opens.

```
10 REMark     Dragon Slayer
20 REMark        *SnowSoft*
30 REMark   Richard Snowdon
120   moveset
130   readchars
140   castle 280,166
150   door 280,166,5
160   lake
170   init
180   play
190 STOP
200 DEFine PROCedure castle(x,y)
210 WINDOW #1,465,210,32,2:WINDOW #0,465,46,32,210:BEEP
220 FOR i=0 TO 2:BORDER #i,0,0:SCALE #i,256,0,0:INK
#i,7:PAPER #i,0:CLS #i:OVER #i,0
230 MODE 8
240 LINE x,y:INK 2:FILL 1:LINE_R 50,0 TO 0,40 TO -50,0 TO 0,-40
250 INK 2,3:FILL 1:LINE_R 0,40 TO 10,5 TO 50,0 TO -10,-5
260 INK 2,0:FILL 1:LINE_R TO 10,5 TO 0,-40 TO -10,-5 TO 0,40
270 INK 0:FILL 1:LINE x,y
280 LINE_R 3,10 TO 0,20 TO 3,5 TO 3,-5 TO 0,-20 TO -6,0
290 FILL 1:LINE x+38,y:LINE_R 3,10 TO 0,20
TO 3,5 TO 3,-5 TO 0,-20 TO -6,0
300 INK 3:FILL 1:LINE x,y:LINE_R 0,40 TO 5,10 TO 5,-10 TO -10,0
310 INK 3,0:FILL 1:LINE_R 5,10 TO 10,5 TO 5,-10 TO -10,-5 TO -5,10
320 INK 3:FILL 1:LINE x+38,y:LINE_R 0,40 TO 5,10 TO 5,-10 TO -10,0
330 INK 3,0:FILL 1:LINE_R 5,10 TO 10,5 TO 5,-10 TO -10,-5 TO -5,10
340 FILL 0:OVER -1:CSIZE 3,1:CSIZE #0,0,1:OVER #0,1
350 FOR i=10 TO 0 STEP -2:INK #0,1+6*(i=0):CURSOR #0,
i,(10-i):PRINT #0,"QL Dragon Slayer"
360 OVER #0,0:END DEFine
370 DEFine PROCedure door(x,y,colour)
380 LINE x+25,y:OVER 0:INK colour:FILL 1
390 LINE_R -10,0 TO 0,30 TO 10,5 TO 10,-5 TO 0,-30 TO -10,0:OVER -1
400 END DEFine
410 DEFine PROCedure init
420 x=20:y=150:dirn=33:fx=330:fy=50:lives=4:dras=-1
430 dx=270:dy=80:po=37:fire=0:arr=6:inc=0
440 END DEFine
450 DEFine PROCedure play
460 knight x,y:dra dx,dy,po:arrow
fx,fy:arrowsleft:livesleft:drasleft
470 REPeat loop
480 k1=KEYROW(1)
490 IF k1&2 :IF x>8 THEN knight x,y:x=x-8:dirn=35:knight x,y
500 IF k1&16 :IF x<420 THEN knight x,y:x=x+8:dirn=33:knight x,y
510 IF k1&4 :IF y>8 THEN knight x,y:y=y-8:knight x,y
520 IF k1&128 :IF y<180 THEN knight x,y:y=y+8:knight x,y
530 IF KEYROW(7)&4 :IF arr>0 :IF NOT fire THEN fire=1:
fx=x:fy=y:fp=10*(dirn=33)-10*(dirn=35):in
c=dirn=33:arr=arr-1:arrowsleft:arrow fx,fy:BEEP 0,10,-2,100,-2,100
540 IF fire :IF fx<10 OR fx>420 THEN fire=0:arrow fx,fy:fx=330:fy
=50:BEEP:ELSE arrow fx,fy:fx=fx
+fp:arrow fx,fy
550 IF fx<dx+50 :IF fx>dx :IF fy>dy-10 :IF fy<dy+10 THEN bump
dx,dy,4:drasleft
560 IF arr=0 :IF y<80 :IF x>326 AND x<334 :IF y>40 THEN door
280,166,0
570 IF arr=0 :IF y<72 :IF y>60 :IF x>326 AND x<334 THEN arr=6:
knight x,y:x=20:y=150:door 280,166
,5:knight x,y:arrowsleft:FOR s=255 TO 10 STEP -1:BEEP 16000,s,
-255,500,-255,500
580 r=dot(x,y)
590 IF r THEN bump x,y,1:livesleft
600 IF fire :IF dy>y-10 :IF dy<y+10 THEN dra dx,dy,po:dy=(dy+2)*
(dy<170):dra dx,dy,po
610 IF RND(40)<5 THEN
620 IF fire THEN arrow fx,fy
630 breath dx,dy:dra dx,dy,po
640 IF dx>x THEN po=37:dx=dx-8
650 IF dx<x THEN po=42:dx=dx+8*(dx<366)
660 IF dy>y :IF dy>18 THEN dy=dy-8:IF dot(dx+
(po=42)*40,dy) THEN dy=80
670 IF dy<y THEN dy=dy+8:IF dot(dx+(po=42)*40,dy) THEN dy=130
680 dra dx,dy,po
690 IF fire THEN arrow fx,fy
700 END IF
710 END REPeat loop
720 END DEFine
730 DEFine PROCedure arrow(x,y)
740 INK 3:CURSOR x,y:PRINT CHR$(138+inc)
750 END DEFine
760 DEFine PROCedure arrowsleft:AT #0,0,20:PRINT #0,"Arrows:
";:FOR i=1 TO arr:PRINT #0,CHR$(189)
770 CLS #0,4:END DEFine
780 DEFine PROCedure breath(bx,by)
790 LOCal w,p
800 BEEP 16000,255,-200,20,-200,20,9,0
810 IF po=37 :p=RND(40):p=p*(p<bx):w=bx-p :ELSE p=bx+50:w=RND
(320 TO 440)-p:IF w<p THEN w=440-p
820 IF by>20 :IF by<72 :IF bx<320 :IF po<>37
THEN w=300-p:w=w*(p<290)
830 BLOCK w,40,p,by-10,2,0
840 IF x>p :IF x<p+w :IF y>by-30 :IF y<by+30
THEN bump x,y,2:livesleft
850 BLOCK w,40,p,by-10,2,0
860 IF fire THEN BEEP 0,10,-2,100,-2,100
870 END DEFine
880 DEFine PROCedure dra(x,y,d)
890 INK 4
900 CURSOR x,y:PRINT CHR$(d):CURSOR x+10,y:PRINT CHR$(d+1)
:CURSOR x+20,y:PRINT CHR$(d+2):CURSOR
x+30,y:PRINT CHR$(d+3)
910 END DEFine
920 DEFine PROCedure bump(cx,cy,colour)
930 LOCal p,r,f
940 BEEP 0,2,50,50,50,50,7,7:INK colour
950 LINE cx/440*420,(190-cy)/190*256:FILL 0
960 FOR p=1 TO 2
970 FOR r=1 TO 50 STEP 5:CIRCLE_R 0,0,r
980 FOR r=50 TO 1 STEP -5:CIRCLE_R 0,0,r
990 NEXT p:BEEP
1000 END DEFine
1010 DEFine PROCedure livesleft:lives=lives-1:AT #0,1,16:PRINT #0,
"Lives:";:FOR i=1 TO lives:PRI
NT #0,CHR$(134);:NEXT i:PRINT #0," "
1020 knight x,y:x=20:y=150:knight x,y:IF lives=0 THEN gameover
1030 END DEFine
1040 DEFine PROCedure drasleft:dras=dras+1:AT #0,1,26:PRINT
#0,"Dragons:";dras;
1050 dra dx,dy,po:dx=270:dy=80:po=37:dra dx,dy,po:fire=0:arrow fx,
fy:fx=330:fy=50:IF dras=10 THEN gameover
1060 END DEFine
1070 DEFine FuNction dot(dx,dy)
1080 p=131072+(dx+32)*.2530612+(dy+2)*128
1090 RETurn PEEK(p)+PEEK(p+1)+PEEK(p+2560)+PEEK(p+2561)
1100 END DEFine
1110 DEFine PROCedure lake
1120 INK 1:FILL 1
1130 LINE 0,160 TO 52,150 TO 43,130 TO 0,130 TO 0,160:FILL 1
1140 LINE 75,145 TO 390,110 TO 390,80 TO 64,125 TO 75,150
1150 END DEFine
1160 DEFine PROCedure screen
1170 OVER -1:CSIZE 0,0:BEEP 0,255,-60,100,-25,100
1180 FOR t=1 TO 2
1190 FOR p=10 TO 0 STEP -1
1200 AT p,p+1:FOR x=p TO 36-p:INK (x MOD 7)+1:PRINT CHR$(41);
1210 FOR y=p TO 19-p:AT y,37-p:INK ((y+3) MOD 6)+1:PRINT CHR$(41)
1220 FOR x=37-p TO p STEP -1:AT 20-p,x:INK
(x MOD 7)+1:PRINT CHR$(41);
1230 FOR y=20-p TO p STEP -1:AT y,p:INK ((y
+3) MOD 6)+1:PRINT CHR$(41)
1240 NEXT p:BEEP 0,0,-200,200,-200,200:NEXT t:BEEP
1250 END DEFine
1260 DEFine PROCedure gameover
1270 AT #0,0,0:CSIZE #0,1,1:CLS #0:BEEP 0,255,-60,-70,-25,100
1280 IF dras=10 THEN PRINT #0,"The Knight won";:PAUSE 50:PRINT
#0," this time":ELSE PRINT #0,"Th
e Dragon won";:PAUSE 50:PRINT #0," as per usual"
1290 PAUSE 50:CSIZE #0,3,1
1300 PRINT #0,"Another game (Y/N)";:i$=INKEY$(3000)
1310 IF i$=="Y" THEN RUN 140:ELSE screen:STOP
1320 END DEFine
1330 DEFine PROCedure moveset
1340 set=167722
1350 oldbase=PEEK_L(set)
1360 newbase=RESPR(875)
1370 FOR m=0 TO 875 STEP 4
1380 POKE_L newbase+m,PEEK_L(oldbase+m)
1390 NEXT m
1400 POKE_L set,newbase
1410 END DEFine
1420 DEFine PROCedure readchars
1430 RESTORE
1440 numberofchars=13
1450 FOR char=1 TO numberofchars
1460 READ c:charbase=newbase+10+(c-32)*9
1470 FOR dat=1 TO 9
1480 READ d:POKE charbase+dat,d
1490 NEXT dat
1500 NEXT char
1510 END DEFine
1520 REMark characters in mode 8 need size 3 when defining.
1530 DATA 33,24,24,16,52,92,20,24,36,72
1540 DATA 34,24,24,16,52,92,52,24,40,68
1550 DATA 35,48,48,16,38,116,30,56,72,36
1560 DATA 36,48,48,16,38,116,38,48,40,68
1570 DATA 37,0,16,96,16,104,4,4,0,0
1580 DATA 38,0,0,4,24,48,64,64,48,12
1590 DATA 39,0,40,84,0,0,0,0,0,124
1600 DATA 40,0,0,68,44,60,56,32,32,64
1610 DATA 41,124,68,84,84,84,84,84,68,124
1620 DATA 42,0,0,68,72,104,120,120,8,4
1630 DATA 43,0,40,84,0,0,0,0,0,124
1640 DATA 44,0,0,112,8,4,4,8,48,64
1650 DATA 45,0,16,12,16,28,96,64,0,0
1660 DEFine PROCedure knight(x,y)
1670 INK 7:CURSOR x,y:PRINT CHR$(dirn+x/8 MOD 2)
1680 END DEFine
```

```
(T 28)) 31 30 31 30 31 31 30 31 30 31))
(SETQ TEMP MONTHS)
(SETQ A MONTH)
(LOOP
(SETQ B (CAR TEMP))
(SETQ TEMP (CDR TEMP))
(UNTIL (ZEROP (SETQ A (SUB1 A)))))
(COND
((GREATERP DAY B)(SETQ DATE-OK 0)))))
(COND
((EQ DATE-OK 0)(PRINC 'ERROR))
(T
(SETQ DAYS (QUOTE (SUN MON TUE WED THU FRI SAT)))
(SETQ MONS (QUOTE (JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC)))
(SETQ A (QUOTE (5 4 3 2 1 0 6)))
(CALC-DATE)
(PRINT-DATE)
)))


(DEFUN FIND-DATE (DATE1)
(COND ((EQ DATE1 NIL)(SETQ DATE1 (QUOTE (-1)))))
(SETQ DAY (CAR DATE1))
(COND ((LISTP (CDR DATE1))(SETQ DATE1 (CDR DATE1)))
(T (SETQ DATE1 (QUOTE (-1)))))
(SETQ MONTH (CAR DATE1))
(COND ((LISTP (CDR DATE1))(SETQ DATE1 (CDR DATE1)))
(T (SETQ DATE1 (QUOTE (-1)))))
(SETQ YEAR1 (CAR DATE1))
(COND ((LISTP (CDR DATE1))(SETQ DATE1 (CDR DATE1)))
(T (SETQ DATE1 (QUOTE (-1)))))
(SETQ YEAR2 (CAR DATE1)))


(DEFUN CALC-DATE ()
(SETQ T1 (ADD1 (REMAINDER YEAR1 7)))
(LOOP (SETQ D1 (CAR A))
(SETQ A (CDR A))
(UNTIL (ZEROP (SETQ T1 (SUB1 T1)))))
(SETQ T1 (REMAINDER (TIMES 5 (QUOTIENT YEAR2 4)) 7))
(SETQ D1 (SKIP (PLUS D1 T1)))
(SETQ T1 (REMAINDER YEAR2 4))
(COND ((GREATERP T1 0)
(SETQ D1 (SKIP (PLUS D1 T1 1)))))
(SETQ N 0)
(SETQ L MONTH)
(COND ((GREATERP L 1)
(SETQ L (SUB1 L)))
(LOOP (SETQ N (PLUS N (CAR MONTHS)))
(SETQ MONTHS (CDR MONTHS))
(UNTIL (ZEROP (SETQ L (SUB1 L))))))
(SETQ N (REMAINDER (PLUS N DAY -1) 7))
(SETQ D1 (ADD1 (SKIP (PLUS D1 N))))
(LOOP (SETQ TODAY (CAR DAYS))
(SETQ DAYS (CDR DAYS))
(UNTIL (ZEROP (SETQ D1 (SUB1 D1))))))


(DEFUN PRINT-DATE ()
(PRINTC)
(PRINC TODAY BLANK (QUOTE THE) BLANK DAY BLANK
(COND ((OR (EQ DAY 1) (EQ DAY 21) (EQ DAY 31))
(QUOTE ST))
((OR (EQ DAY 2) (EQ DAY 22))
(QUOTE ND))
((OR (EQ DAY 3) (EQ DAY 23))
(QUOTE RD))
(T (QUOTE TH)))
BLANK
(QUOTE OF) BLANK)
(LOOP (SETQ MON (CAR MONS))
(SETQ MONS (CDR MONS))
(UNTIL (ZEROP (SETQ MONTH (SUB1 MONTH)))))
(PRINC MON BLANK YEAR1 YEAR2))


(DEFUN SKIP (X)
(COND ((GREATERP X 6)
(DIFFERENCE X 7))
(T X)))
```

QL Palette
by Edward Leigh

QL Palette is a simple utility for selecting colours on the Sinclair QL. The program allows the user to view all the colour combinations for two of the four stipples (the number of each stipple is displayed in the black boxes at the top left-hand corner of the grid). The display can be toggled between the other two stipples by pressing space bar. The program works in colour modes four and eight, and can also be used to view a screen filled with only one colour.


```
100 DIM stc(2,2):MODE 4:CLS #0:PRINT:INPUT "Which mode (4 or 8) ? "; m
110 INPUT "Which two stipples together ( 0 and (1 to 3) ? "; stc
120 IF stc(1 OR stc)3 THEN PRINT "out of range":GO TO 110
130 PRINT:PRINT "On the Y-axis are the background colours"
140 PRINT "On the X-axis are the foreground colours (2 stipples to each)"
150 PRINT:PRINT "Press any key":z$=INKEY$(-1)
160 stc(1,1)=0:stc(1,2)=stc:IF stc=3 THEN stc(2,1)=1:stc(2,2)=2
170 IF stc=2 THEN stc(2,1)=1:stc(2,2)=3
180 IF stc=1 THEN stc(2,1)=2:stc(2,2)=3
190 stn1=1:stn2=1:MODE m
200 FOR y=0 TO 7
210  FOR x=0 TO 15
220   WINDOW 27,25,40+27*x,16+25*y:PAPER y: INT(x/2),stc(stn1,stn2)
230   CLS:BORDER 1,7:IF INT(x/2)=0 AND y=0 THEN PRINT stc(stn1,stn2)
240  stn2=3-stn2:END FOR x
250 END FOR y
260 WINDOW 448,210,32,6:PAPER 0:INK 7:CLS #0
270 CSIZE #0,0,0:CSIZE 0,0:IF m=4 THEN CSIZE #0,2,0:CSIZE 2,0
280 FOR t=0 TO 7:CURSOR 0,25*t+20:PRINT t;CURSOR 54*t+37,0:PRINT t;NEXT t
290 PRINT #0," Press space bar for other stipples"
300 PRINT #0,"Press 'V' to view a particular colour"
310 PRINT #0,"    Press 'R' to re-run the program"
320 PRINT #0,"    Press 'T' to stop the program"
330 z$=INKEY$(-1):IF z$=" " THEN stn1=3-stn1:GO TO 200
340 IF z$="T" OR z$="t" THEN CLS #0:STOP
350 IF z$="R" OR z$="r" THEN RUN
360 IF z$="V" OR z$="v" THEN
370  CLS #0:PRINT #0,"background colour number ";a:IF a)7 THEN GO TO 370
380  INPUT #0,"foreground colour number ";b:IF b)7 THEN GO TO 380
390  INPUT #0,"          stipple number ";c:IF c)3 THEN GO TO 390
400  CLS #0:PAPER a,b,c:CLS:PRINT #0,"Press any key to return to palette"
410 END IF
420 GO TO 330
```

Brimstone Part One
by Paul Gallagher

Brimstone is a three-part adventure game for the Dragon 32; parts two and three will be published in the next two issues of *PCW*. Part one will be of particular interest to Advanced Dungeons and Dragons players, as it provides a means of evolving new characters by closely following the AD+D format. It was designed by an AD+D player to generate new characters, and also allows preparation of the characters' subsequent development when group AD+D sessions are not in session. It's so comprehensive that it represents a game in its own right, as well as being a useful tool for AD+D players.

The program creates and equips a single character at a time, and then allows the final result to save as a file. No knowledge of Dungeons and Dragons' character formation is required as full instructions are given in the program. Several characters can be thus generated and combined into a variety of adventure parties.

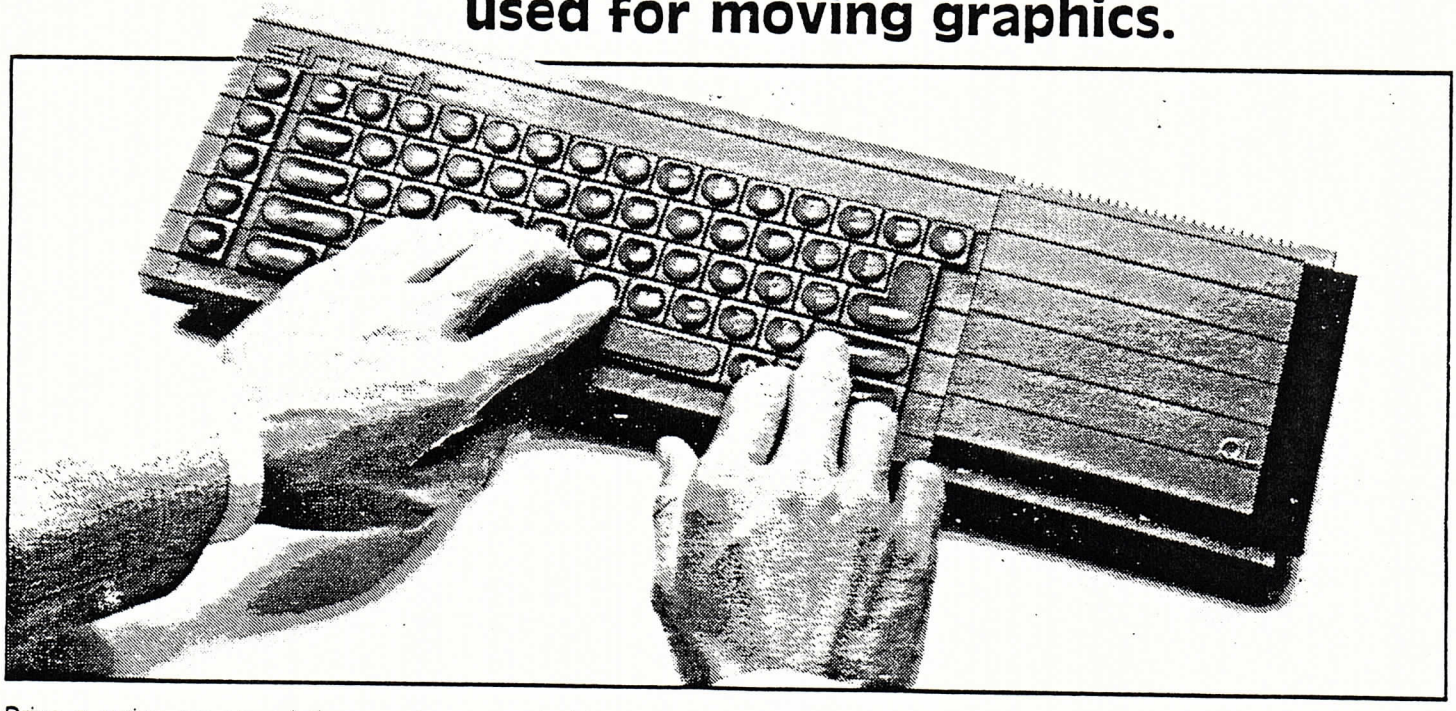
Next month we'll publish the first scenario for your newly-developed character. This will be a traditional dungeon which uses the basic AD+D rules and 'dice' percentages. Up to four characters can adventure together, so you may like to get a team of characters together with friends to tackle the 75-room, seven-floor dungeon that's coming. Alternatively, one person could control between one to four characters making up an adventure party. Survivors of that dungeon can progress to part three, 'The Tavern', in two months' time.

It's also easy to create your own dungeons, so if the following scenarios are a success we'll publish more dungeons sent in by other 'Dungeon masters'.

```
5 'BRIMSTONE PART 1:copywrite P.Gallagher 1984
10 DIM E$(22),F$(1,22):GOTO50
20 K$=INKEY$:IFK$=""THEN20ELSERETURN
30 PRINT@257,STRING$(31,32):PRINT:PRINT:PRINT:PRINT:PRINT:RETURN
40 '**************************
50 CLS:PRINT@167,"DUNGEON & DRAGONS":PRINT@230,"CHARACTER GENERATOR"
60 GOSUB28 'SDELAY18
70 PRINT:PRINT"THIS SUB-PROGRAM GENERATES SIX  INITIAL CHARACTERS FOR YOU TO  C
HOOSE ONE & THEN ADAPT IT TO   YOUR REQUIREMENTS."
80 PRINT:PRINT"THIS PROCESS CLOSELY FOLLOWS THEA.D.& D.PLAYER'S HANDBOOK."
90 PRINT:PRINT"YOU CAN THEN SAVE YOUR CHARACTERFOR SUBSEQUENT ADVENTURES IN A  P
ARTY OF UP TO FOUR."
100 'ESBASIC ABILITIES ROUTINES
110 FORI=1TO3
120 B(1)=B(1)+RND(6):B(2)=B(2)+RND(6):B(3)=B(3)+RND(6):B(4)=B(4)+RND(6):B(5)=B(5
)+RND(6):B(6)=B(6)+RND(6)
130 I(1)=I(1)+RND(6):I(2)=I(2)+RND(6):I(3)=I(3)+RND(6):I(4)=I(4)+RND(6):I(5)=I(5
)+RND(6):I(6)=I(6)+RND(6)
140 W(1)=W(1)+RND(6):W(2)=W(2)+RND(6):W(3)=W(3)+RND(6):W(4)=W(4)+RND(6):W(5)=W(5
```

Moving Graphics On The QL

Drive a racing car around the screen in this fast-moving game from Tim Hartnell, which demonstrates how effectively SuperBASIC can be used for moving graphics.



Drive a racing car around the screen in this fast-moving game from Tim Hartnell, which demonstrates how effectively SuperBASIC can be used for moving graphics.

Although the speed of the QL has been criticised, it is still possible to produce highly satisfactory moving graphics programs as you'll see when you run this program.

In QL RACER you drive a little racing car (which looks remarkably like an arrow) around a race track. You'll discover that the game, although it starts off running fairly slowly, is almost impossible to play. If you manage to get around the track once without crashing, it will speed up, and will continue to increase its speed for twenty games.

The program, which comes from my book *Tim Hartnell's QL Games Compendium* (Interface Publications, £5.95), makes the most of a number of features which are unique to the QL, such as the real-time clock.

You travel from the top left hand corner round the course clockwise, then up the left hand edge to your starting position, where you'll be given a new car. You must avoid all the edges to stay in the race. Your score is related to how long you manage to keep the car in action, and also to the 'difficulty level' which is set at the start of a game.

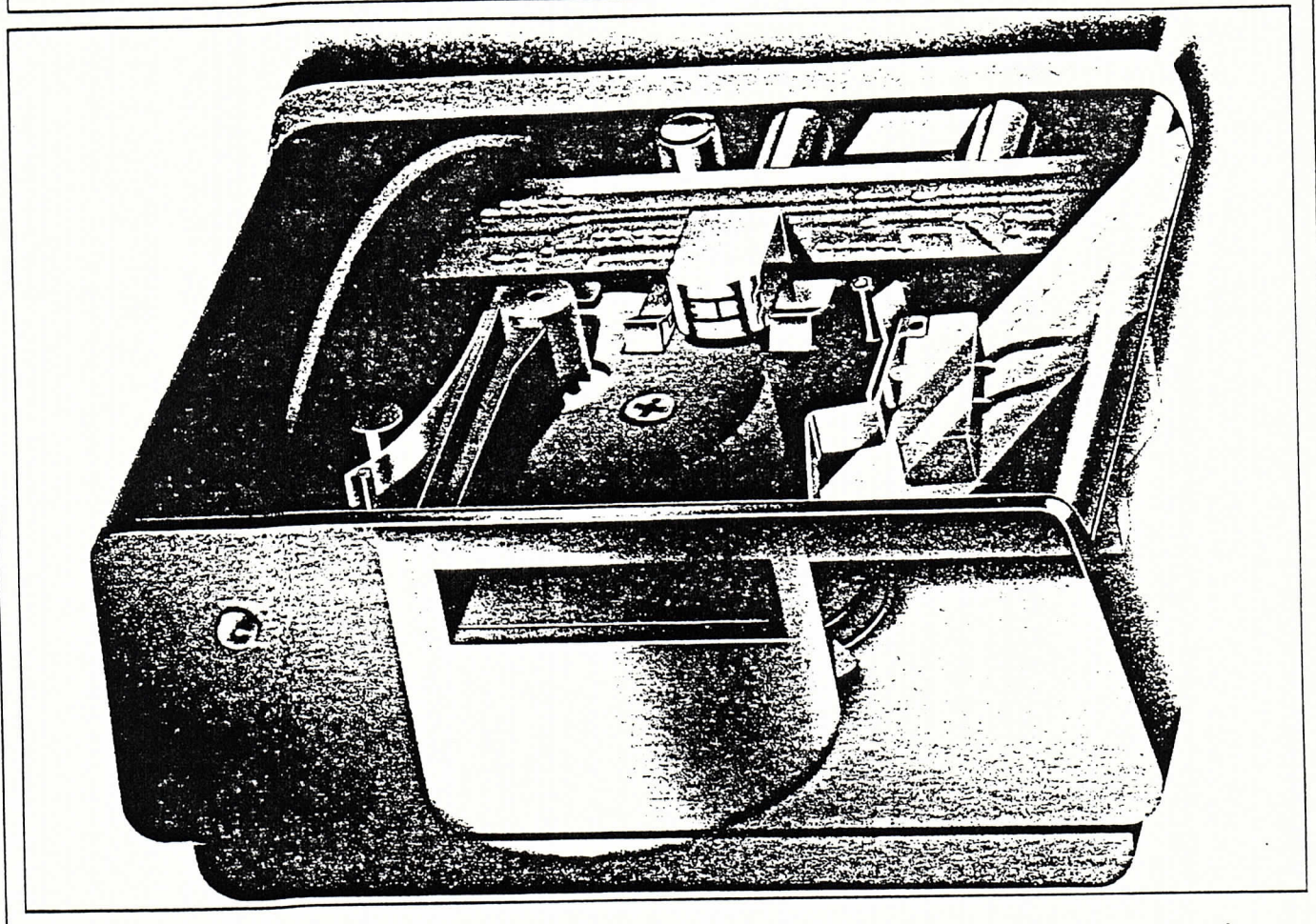
As I pointed out, the real-time clock is used in this program. The QL's internal clock is used to give a readout which shows how long you have survived. The clock, and the score, is updated using the procedure defined in lines 470 to 560.

```
10 REMark QL Racer
20  high_score=0
30  difficulty=21
40 REPeat cycle
50  initialise
60   REPeat race
70   increment_score
80    read_keyboard
90    erase_old_car
100    check_if_smash
110     IF smash=1 THEN EXIT race
120    place_new_car
130    BEEP 1000,RND(240 TO 250)
140   FOR delay=1 TO difficulty
150   END FOR delay
160   END REPeat race
170 REMark ---------------
180 REMark Smash sequence
190 FOR j=1 TO 70
200 AT car_down,car_across
210 INK RND(1 TO 7)
220 PRINT "*"
230 BEEP 10,RND(1 TO 20)
240 AT car_down,car_across
250 PRINT "+"
260 BEEP 10,RND(40 TO 70)
270 AT car_down,car_across
280 PRINT CHR$(254)
290 END FOR j
300 AT 0,0
```

```
310 STRIP 1:INK 7:PAPER 1
320 FLASH 1
330 PRINT "Your score is ";score
340 IF score>high_score THEN
350  high_score=score
360 END IF
370 AT 18,16
380 PAPER 2
390 PRINT "High score is ";high_score
400 FLASH 0
410 FOR y=1 TO 100
420   BEEP 50,y
430   BEEP 70,200-2*y
440 END FOR y
450 END REPeat cycle
460 REMark ------------------
470 DEFine PROCedure increment_score
480  score=score+1
490 AT 1,32
500 PRINT score
510 get$=DATE$
520 INK 7:PAPER 2
530 AT 9,5
540 PRINT get$(16 TO 20)
550 PAPER 0
560 END DEFine increment_score
570 REMark ------------------
580 DEFine PROCedure place_new_car
590 INK 6
600 AT car_down,car_across
610   PRINT car$
620 END DEFine place_new_car
630 REMark ----------------
640 DEFine PROCedure check_if_smash
650 smash=c(car_down,car_across)
660 END DEFine check_if_smash
670 REMark --------------
680 DEFine PROCedure erase_old_car
690 AT erase_down,erase_across
700 PRINT " "
710 END DEFine erase_old_car
720 REMark --------------
730 DEFine PROCedure read_keyboard
740 erase_across=car_across
750 erase_down=car_down
760 new$=INKEY$
770 IF new$="" THEN new$=old$
780 IF new$=CHR$(192) THEN
790  car_across=car_across-1
800  car$=CHR$(188)
810 END IF
820 IF new$=CHR$(200) THEN
830  car_across=car_across+1
840  car$=CHR$(189)
850 END IF
860 IF new$=CHR$(208) THEN
870  car_down=car_down-1
880  car$=CHR$(190)
890 END IF
900 IF new$=CHR$(216) THEN
910  car_down=car_down+1
920  car$=CHR$(191)
930 END IF
940 old$=new$
950 END DEFine read_keyboard
960 REMark --------------
970 DEFine PROCedure initialise
980 PAPER 0:INK 3:BORDER 4,2
990 CLS:CLS #0
1000  score=0
1010 IF difficulty>1 THEN difficulty=difficulty-1
1020 PAPER 7:INK 2
1030 AT 0,30:PRINT "Score:"
1040 AT 4,32:PRINT " ";difficulty;" "
1050 DIM a(233),b(233),c(20,30)
1060 REMark Build racetrack
1070 PAPER 6:INK 4
1080 CSIZE 2,0
1090 RESTORE
1100 FOR j=1 TO 233
1110  READ a(j),b(j)
1120  c(a(j),b(j))=1
1130 AT a(j),b(j)
1140 PRINT CHR$(254)
1150 END FOR j
1160 PAPER 1
1170 FOR j=1 TO 29
1180 READ x,y
1190 AT x,y
1200 INK RND(2 TO 7)
1210 PRINT CHR$(253);:BEEP 100,5*j
1220 END FOR j
1230 REMark Place car
1240 car_across=3:car_down=3
1250 erase_across=car_across
1260 erase_down=car_down
1270 old$=CHR$(200)
1280 smash=0
1290 PAPER 0:INK 6
1300 FOR y=1 TO 50:BEEP 100,y
1310 SDATE 1984,7,3,0,0,0
1320 END DEFine initialise
1330 REMark --------------
1340 REMark Track data
1350 DATA 1,4,1,5,1,6,1,7,1,8,1,9,1,10,1,11
1360 DATA 1,20,1,21,1,22,1,23,1,24,1,25,1,26
1370 DATA 1,27,1,28,1,29
1380 DATA 2,2,2,3,2,4,2,11,2,12
1390 DATA 2,18,2,19,2,20,2,29,2,30
1400 DATA 3,1,3,2,3,12,3,16,3,17,3,18,3,30
1410 DATA 4,1,4,6,4,7,4,8,4,12,4,13
1420 DATA 4,16,4,23,4,24,4,25,4,26,4,30
1430 DATA 5,1,5,4,5,5,5,6,5,7,5,8,5,9
1440 DATA 5,13,5,14,5,15,5,16,5,21,5,22
1450 DATA 5,23,5,26,5,27,5,30
1460 DATA 6,1,6,4,6,9,6,19,6,20,6,21
1470 DATA 6,22,6,23,6,24,6,25,6,26,6,30
1480 DATA 7,1,7,4,7,9,7,10,7,19,7,20
1490 DATA 7,21,7,22,7,30
1500 DATA 8,1,8,4,8,10,8,11,8,12,8,13,8,14
1510 DATA 8,15,8,16,8,17,8,18,8,19
1520 DATA 8,20,8,28,8,29,8,30
1530 DATA 9,1,9,4,9,10,9,11,9,12,9,13
1540 DATA 9,14,9,15,9,16,9,17,9,18
1550 DATA 9,24,9,25,9,26,9,27,9,28,9,29
1560 DATA 10,1,10,4,10,11,10,18,10,19
1570 DATA 10,20,10,23,10,24
1580 DATA 11,1,11,4,11,5,11,6,11,7,11,8
1590 DATA 11,11,11,19,11,20,11,24,11,25
1600 DATA 11,26,11,27,11,28,11,29
1610 DATA 12,1,12,7,12,8,12,9,12,10,12,11
1620 DATA 12,14,12,15,12,19,12,20,12,21
1630 DATA 12,28,12,29
1640 DATA 13,1,13,10,13,11,13,14,13,15,13,16
1650 DATA 13,19,13,20,13,21,13,22
1660 DATA 13,29,13,30
1670 DATA 14,1,14,2,14,6,14,10,14,11
1680 DATA 14,15,14,16,14,19,14,20,14,22
1690 DATA 14,23,14,24,14,25,14,26,14,30
1700 DATA 15,2,15,4,15,5,15,6,15,7,15,10
1710 DATA 15,11,15,12,15,15,15,16
1720 DATA 15,20,15,21,15,22,15,23,15,24
1730 DATA 15,25,15,29,15,30
1740 DATA 16,2,16,3,16,4,16,7
1750 DATA 16,15,16,16,16,17,16,28,16,29
1760 DATA 17,7,17,8,17,14,17,15
1770 DATA 17,17,17,18,17,28
1780 DATA 18,8,18,9,18,10,18,11,18,12,18,13
1790 DATA 18,14,18,18,18,19,18,20
1800 DATA 18,21,18,22,18,23,18,24
1810 DATA 18,25,18,26,18,27,18,28
1820 DATA 14,21,7,7,9,5,6,5,9,8
1830 DATA 10,9,7,5,11,10,6,6
1840 DATA 9,6,5,25,6,8,9,7
1850 DATA 10,7,6,7,5,24,7,6
1860 DATA 7,8,11,9,8,5,10,10
1870 DATA 8,6,10,8,8,7,10,6
1880 DATA 8,8,8,9,10,5,9,9
```

MOBILE MICRODRIVE

Simon Goodwin presents a compendium of useful information for the Sinclair QL system and a handy program for making back-up copies for your Microdrive cartridges.



This week we're taking a brisk tour of the QL system, with a collection of hints and tips about the Microdrives, keyboard and display.

Duplicart

Duplicart is a general-purpose program which can create a back-up copy of your QL cartridge. Just put the original cartridge in drive 1, a blank in drive 2, and RUN the program. A minute or so later you can pour coffee over the original cartridge, secure in the knowledge that you've got a duplicate.

You have probably used the 'clone' programs supplied with each QL package. These were specially written to copy the files on a specific cartridge so that, for instance, the Quill clone can't copy the Archive files, and vice versa.

Duplicart does not have this restriction and will save you a lot of work typing COPY statements whenever you want to make a security copy of a cartridge. This in turn saves a lot of work when your cartridge decides to (literally) get knotted. Microdrives are not the most reliable of gadgets, and anything that makes it easier to back them up must make the Sinclair QL more usable.

Duplicart works by formatting the blank cartridge and copying the directory of the original into a temporary file there. This file is read into memory and the names are extracted to generate a set of COPY statements which transfer the files automatically.

Faulty piping

Ideally I would have used a 'pipe' to store the directory, instead of a temporary file. A pipe is a temporary file which is created in memory rather than on cartridge. It should be possible to 'pour' data into one end of a pipe and read it out later (perhaps while the pouring is still going on), but in practice I had no joy at all in reading data back from the pipe. The commands:

```
OPEN #3,pipe_2000
DIR #3,mdv1_
```

Will happily squirt the directory listing into a pipe 2,000 characters long, but no amount of coaxing would bring the data out of the pipe. Perhaps this secret feature of the QL will become more useful when Sinclair tell us how to use the machine's 'multi-tasking' facilities.

As it is, Duplicart reads the directory listing from cartridge into the unimaginatively named array NAME$, and then the directory file is deleted. The number of files is shown when copying begins — a maximum of 50 files can be copied by the program at one go. The names are printed one by one as files are duplicated.

A simple procedure has been defined to make it easy for you to copy individual files while Duplicart is loaded. If your file is called PCN, you need only type:

```
c PCN
```

to copy the file from drive 1 to drive 2.

You may find that your QL works better if files are copied from the right hand drive (number 2) to the left hand one.

Early QLs had undersized cooling plates behind the second Microdrive which could lead to overheating and unreliable saveing on that drive. The reverse is true on other machines, which suffer from interference between drive 1, (on the left) and the logic array on the circuit board nearby. If in doubt, swap over the drive names throughout the

listing and see if that increases the speed at which files are copied.

Make sure that you change the message on lines 190-210 if you reverse the copying sequence. If you confuse the source and destination cartridges you could end up scrubbing the data you are trying to duplicate.

In the interests of speed Duplicart only formats a cartridge once before copying onto it. Repeated formatting can condition the tape so that it will hold more data, so it is a good idea to use a couple of explicit FORMAT commands before you copy a cartridge which is very full.

Remember that the capacity of QL cartridges does vary, although not as much as their Spectrum counterparts, so it is not a good idea to fill cartridges completely — you could end up having trouble finding a backup cartridge which will accommodate all of the data.

There are a number of ways in which Duplicart could be improved. A question and answer sequence could be added to allow files to be selected for copying, and the program could be adapted to handle other devices. As it stands, Duplicart is a short, efficient program which takes a lot of the hassle and worry out of using the QL.

Key notes

If you find the QL keyboard irritating you may be interested in a few POKEs which allow you to alter its characteristics.

Should the auto-repeat rate be too fast for your tastes, use POKE 163983, N to alter the delay between repetitions of a keypress. The normal value of N is 2, which represents a delay of 1/25 second. The value is in multiples of 1/50 second (or 1/60 second on US models), so that POKE 163983,5 would reduce the repeat rate to a rather more pedestrian 10 characters per second.

The delay before repetition starts is controlled by the value at address 163981. Again the delay is in units of 1/50 second. The normal value is 30, which means that characters start to repeat after they have been held down for 3/5 second. Use POKE 163981,50 to select a one second delay, or POKE W 163980,32000 to turn off the repetition altogether.

It is possible to select Caps Lock from within a program. This can be useful if you want to save yourself the trouble of converting input strings into capital or small letters, use POKE 163976,1 to select Caps Lock and POKE 163976,0 to turn it off.

Sadly, we can't find a POKE which stops the plastic legs falling off the back of the computer.

Closing the windows

Quite a few QL users seem to have problems reading all the characters on the screen, even if F2 is pressed when the computer is turned on, selecting the TV display. This is because the computer tries to display characters at or beyond the left-hand margin of the TV screen.

The following commands give a clear and readable screen on an aging Hitachi TV:

```
MODE 1
BORDER 4,0
BORDER #0,4,0
BORDER #2,4,0
```

The first statement selects smaller characters, although still using the narrow TV display area. Paradoxically this makes the text easier to read on most TVs we have tested — the large characters otherwise used are so crudely-formed that they are hard to read.

The BORDER statements aren't well explained in the QL manual. The first of these adds a black border to window 1, which is used by PRINT statements. The next two commands give the same treatment to window 0 (the command area, at the bottom of the screen) and window 2, used for program listings.

If your TV doesn't cope very well with MODE 1, try the compromise of MODE 1 together with CSIZE #0,1,0: CSIZE 1,0: CSIZE #2,1,0. This spaces out the characters in each window, making them easier to read.

Program listing — Duplicart

```
80 REMark Duplicart (c) 1984 Simon N Goodwin
90 REMark Version 0.2 26th August 1984
100 MODE 1
120 DIM name$(50,32)
130 BORDER 4,110
140 BORDER 8,128
150 CSIZE 3,1
160 AT 1,8
170 PRINT "DUPLICART!"\\
180 CSIZE 1,0
190 PRINT"Put the cartridge to be copied in
    the LEFT drive and"
200 PRINT"the blank cartridge to be filled in
    the RIGHT drive."\\
210 PRINT"Press ENTER when you are SURE you're
    ready to start."
220 INPUT a$
230 FORMAT #0,mdv2_
235 REMark Read source directory onto
    destination cart.
240 OPEN_NEW #3,mdv2_direct
250 DIR #3,mdv1_
260 CLOSE #3
265 REMark Extract filenames
270 PAPER 0
280 OPEN #3,mdv2_direct
290 INPUT #3,n$:PRINT "Cartridge Name: ";n$;
300 i=0 :REMark First 'name' is sector data
310 REPeat get_names
320   INPUT #3,name$(i)
330   IF EOF(#3) THEN EXIT get_names
340   i=i+1
350   END REPeat get_names
360 CLOSE #3
370 PRINT" (";i;" files)",
380 DELETE mdv2_direct
385 REMark Copy each file
390 FOR j=1 TO i:PRINT !name$(j),:c name$(j)
400 PRINT"FINISHED!"
410 FOR i=0 TO 30:BEEP 100,i
420 STOP
430 DEFine PROCedure c(a): COPY "mdv1_" & a TO
    "mdv2_" & a: END DEFine c
```

WISE UP ON WINDOWS

The QL's windows are easy to use and versatile once you understand them, which you will after reading this illuminating article by Tom Short.

One of the most attractive features of the QL is its ability to divide the physical screen into a number of 'mini-screens', or windows. The contents of these windows can be manipulated by using facilities available in SuperBasic. But before looking into QL windows it is worth describing how SuperBasic handles the physical screen.

There are two screen modes on the QL. In the lowest resolution mode, the screen is divided into 256x256 pixels and can display eight distinct colours (black, blue, red, magenta, green, cyan, yellow, and white). In this mode, flashing is available as an option, but there is a limitation on the smallest size of character which can be dislayed (see below).

This mode is set using either:

MODE 256

(*ie* 256 pixels across the screen) or

MODE 8 (*ie* eight colours)

The higher resolution mode divides the screen into 512 (horizontal) x 256 (vertical) pixels and can display four colours (black, red, green, and white).

This mode is set using either:

MODE 512 or

MODE 4

Coordinate systems

There are three distinct ways in which the screen can be viewed: (a) the pixel coordinate system; (b) the graphics coordinate system; and (c) a modification of (a) that I call the character coordinate system.

The pixel coordinate system originates at the top left hand corner. The y-axis proceeds downwards from 0 to 225 and the x-axis proceeds to the right from 0 to 511. The division of the screen horizontally into 512 units is true for both screen modes. The system automatically adjusts to 256 pixels in the lower resolution mode.

The graphics coordinate system has its origin in the bottom left-hand corner of the screen and the y-axis proceeds upwards from 0 to 100 units. The x-axis proceeds to the right from 0 to 148 units, assuming that the whole of the physical screen is being used. Both the value of the origin and the number of vertical divisions can be redefined using: SCALE. The default setting is equivalent to SCALE 100,0,0. The first parameter is the number of divisions in the vertical direction and the next two are the x and y values of the origin.

Therefore SCALE 200,50,70 will divide the vertical distance into 200 units and the origin in the bottom left-hand corner will be (50,70). The horizontal axis scale will adjust in proportion so that any figure plotted with a change of scale will change in size but not have its shape distorted.

A number of graphics commands are provided in SuperBasic that use this coordinate system (see Table 1). Note that the execution of the scale command does not rescale images already plotted on the screen, but only affects those plotted subsequently.

The character coordinate system stands at the top left of the screen like the pixel coordinate system. The screen is, however, now divided into rows and columns. Since the character size can be varied under software control on the QL, the number of rows and columns that take up the whole screen at any one time depends on the character height and width.

In the 256 mode there are potentially 42 columns and 25 rows for characters with the default size, while in 512 mode default sized characters are organised as 85 columns and 25 rows.

Width and height parameters are related to numbers of pixels as follows:

Height	Pixel Positions
0	10
1	20

Width	Pixel Positions
0	6
1	8
2	12
3	16

Character size can be changed using:

CSIZE width, height

It is important to realise that for the purpose of calculation, the screen is assumed to consist of 512 pixels across the screen in both modes. In 256 mode the smallest character size is 2.0 or 12x10 pixels. A string of characters can be placed on the screen using the character coordinate system by means of the AT facility. For example:

AT 20, 10: PRINT "A character string"

The string will be printed with the first character at a position 20 characters from the left and 10 characters from the top. If you are unfortunate enough to be using a first release QL with version FB SuperBasic, the two parameters following the AT keyword must be reversed.

Windows

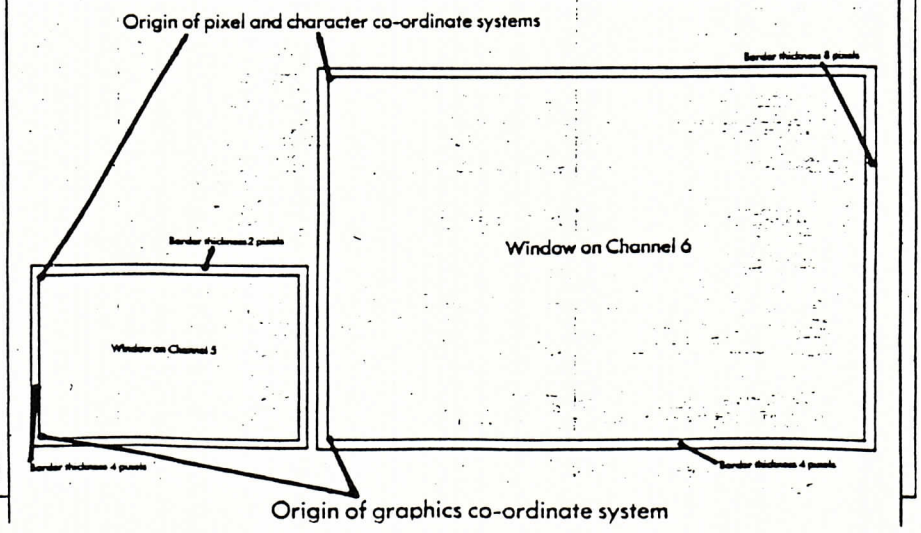
Windows are like 'mini-screens' placed on the physical screen and images within them can be manipulated using SuperBasic. A maximum of 16 windows can be defined, although in some circumstances this is reduced.

Windows can only be rectangular with their sides parallel to the physical screen, so in order to set one up its dimensions and position only need be specified. We must also have some way of referring to it, to distinguish it from others. This is achieved by using a channel number and the window is created with an OPEN statement. As an example, suppose we want to create a window 100 pixels wide, 50 pixels deep, positioned 40 pixels from the left edge of the screen and 20 pixels from the top. A possible OPEN statement is:

OPEN#5,SCR_100X50A40X20

Here we are using channel number 5. The SCR is a standard QDOS device name and stands for screen output. The 100X50 indicates the window size and A40X20 is the position. The x can be thought of as 'by' and the A as 'at position'.

Figure 1



We can now operate on this window in SuperBasic. For example, we can set the background colour with:

```
PAPER£5,7
```

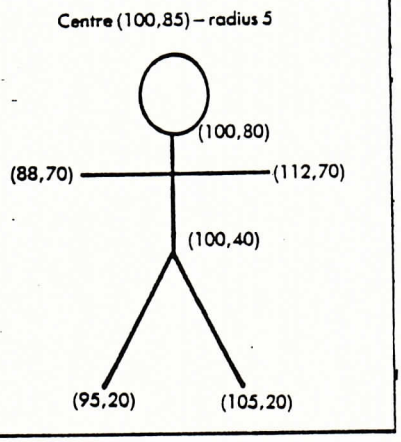
This sets the window on channel 5 to colour 7 (white). This becomes apparent when we clear the window with:

```
CLS£5
```

Table 1 indicates the SuperBasic commands that will accept a channel number in order to manipulate the contents of a window.

In order to appreciate how some of these commands work with windows, let us set up two windows on channels 5 and

Figure 2



6 with the following specifications: (see Figure 3)

	Channel 5	Channel 6
Window size	100x50	200x100
Window position	50x75	175x25
Background (paper) colour	blue	magenta
Foreground (ink) colour	white	black

The following procedure will set up the windows:

```
1000DEFine PROCedure wsetup
1010 MODE 8
1020 OPEN£5,SCR_100X50A50X75
1030 OPEN £6, SCR_200X100A175X25
1040 PAPER£5,1
1050 PAPER£6,3
1060 INK£5,7
1070 INK£6,0
1080 CLS£5
1090 CLS£6
1999 END DEFine wsetup
```

The procedure can be executed by simply typing its name: wsetup

A coloured border can be added to each window by inserting the following lines:

```
1100 BORDER£5,2,6
1200 BORDER£6,4,5
```

The two parameters after the channel number indicate the thickness of the border (in pixels) and its colour. The thickness specified is actually that of the horizontal components of the border. The vertical components at the sides are twice the specified thickness. Therefore, the smaller window (channel 5) will have a border thickness of two pixels horizontally, four pixels vertically and colour yellow (6) and the larger (channel 6) will have a border of four pixels horizontally, eight pixels vertically and colour cyan (5).

The addition of a border takes place within the inner edge of the window and therefore decreases its effective size. The notes on the pixel and graphics coordinate systems apply equally well to individual windows. The origins of the coordinate systems in the current example are shown in Figure 1.

To illustrate the effect of executing graphics operations in windows, we will define a procedure to draw a simple stick figure (see Figure 2):

```
2000 DEFine PROCedure stick(channel)
2010 REMark Draw head radius 5, centre (100,85)
2020 CIRCLE£channel,100,85,5
2030 REMark Draw arms
2040 LINE£channel,88,70 TO 112,70
2050 REMark Draw body
2060 LINE£channel,100,80 TO 100,40
2070 REMark Draw legs
2080 LINE£channel,95,20 TO 100,40 TO 105, 20
2999 END DEFine stick
```

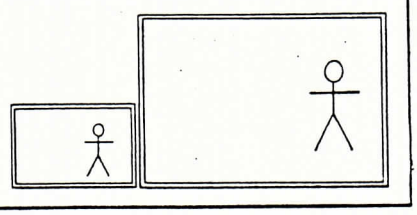
This stick figure can be placed in each of our example windows by the program:

```
100 wsetup
110 REMark Draw figure in window on channel 5
120 stick 5
130 REMark Draw figure in window on channel 6
140 stick6
```

The result of running this program is shown in Figure 3. The following points should be noted.

1 Each window has its own graphics coordinate system. The same figure has been drawn in each but it has been scaled so that the window height (excluding the border) is 100 units on the graphics coordinate system. This means procedures that use graphics facilities can be written independently of the final window into which they are to be drawn. It is worth inserting a SCALE statement into the above program to see the effect. Since SCALE can take a channel number, the scaling can be handled differently in different windows.

Figure 3



2 Each window is twice as long as it is wide in pixel units. Since each window is 100 graphics units high, you might expect that drawing the body of the figure at x=100 would place it in the centre of the window. Unfortunately this is not the case. The reason is that each pixel is not square but rectangular and, therefore, 50 pixels horizontally does not cover the same distance on the screen as 50 pixels vertically. The graphics coordinate system, however, does use the same scale horizontally as vertically.

3 We can try changing the position of one of the windows in the above program so that the two windows overlap. This can be done by modifying the appropriate OPEN statement in wsetup or alternatively using WINDOW.

The following program lines will redefine and clear the window on channel 5 so that it overlaps with that on channel 6:

```
104 WINDOW£5, 200,50,250,50
106 106 CLS£5
```

The window has been redefined so that it is 150 pixel units from the left of the screen. If a border is required on this redefined window, it must be added again. For example:

```
108 BORDER£5,2,6
```

Running the modified program will still show the image of the original window on the screen with the redefined window covering it. The two images in the window overlap as shown in Figure 4.

Back to the beginning

You can now appreciate that all screen activity on the QL takes place in windows. The system uses three predefined windows on channels 0, 1 and 2. The default arrangement of these depends on whether the TV or monitor option was chosen when the QL was initialised. With the monitor option, the default mode is 512 and the three default windows are shown in Figure 5.

With the TV option, the default mode is 256 and windows 1 and 2 coincide in their positions on the screen as shown in Figure 6. In this case, the window sizes are smaller to take account of the fact that most TVs do not display the full extent of the screen.

Each window has a particular use. The channel 0 window contains the current command or program line as it is entered, the edit line, and also displays the error messages. Channel 1 is the default channel. Most program opera-

Figure 4

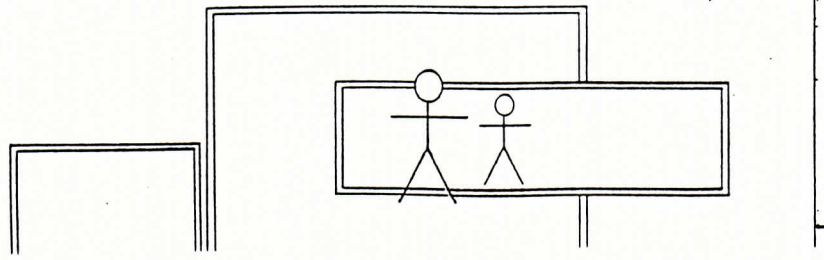
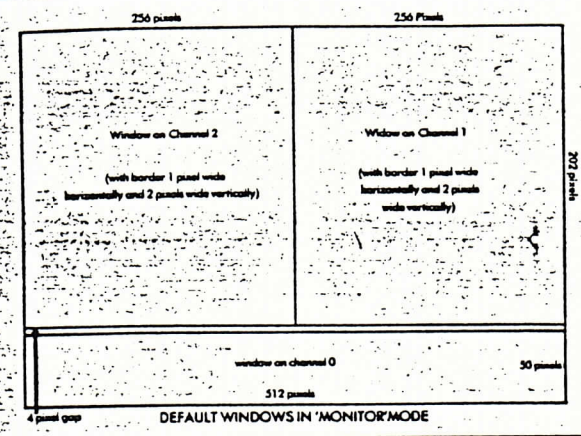
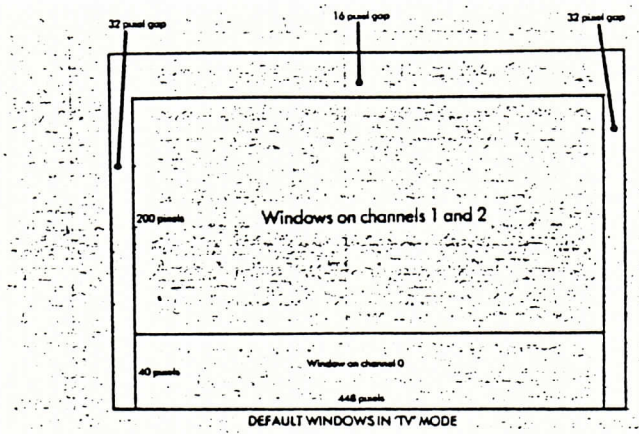


Figure 5



DEFAULT WINDOWS IN 'MONITOR' MODE

Figure 6



DEFAULT WINDOWS IN 'TV' MODE

tions such as PRINT, CLS, INPUT, FLASH, BORDER will operate on channel 1 if no other number is specified. Channel 2 displays the program as it builds up and is also the default for the LIST command.

In through the window

If we wish to input to a window by means of an input statement, an alternative form of OPEN must be used because scr_ is a write-only device.

The alternative that allows both output and input is the console device, con_. The method used for opening this is similar to scr_, except that the size of the type-ahead buffer associated with the window must also be specified. For example, an alternative to line 1020 in the procedure wsetup could have been:

```
1020 OPEN#5, con_100X50A50X7580
```

The figure 80 means that 80 characters can be typed before the type-ahead buffer overflows and characters are lost. Input statements, such as the following, can now be executed:

```
INPUT#5,value
```

This will wait for input to be provided in the window on channel 5.

All opened devices should be closed before a program terminates. In SuperBasic this is: CLOSE#5

Although its image may still appear on the screen, the window no longer exists.

From these simple examples, the versatility of the QL windows can be seen. Examining Table 1, you will see that there are other powerful facilities for manipulating the contents of windows. These include the ability to pan and scroll in either direction, to see part or all of the contents of a window, and draw hollow and filled figures.

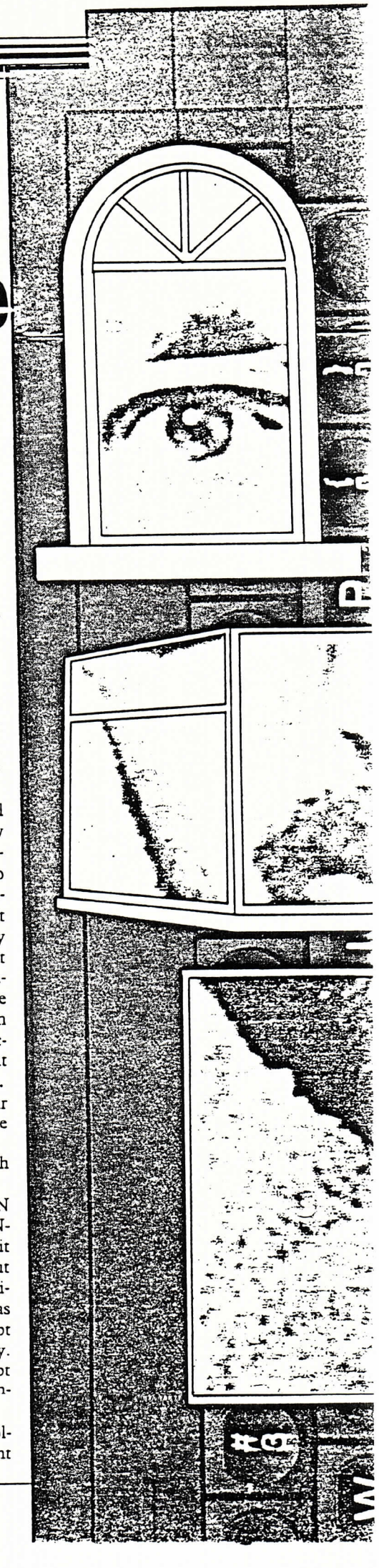
Table 1

Keywords that will accept a window channel number

KEYWORD	DEFAULT WINDOW	COORDINATE SYSTEM	ACTION	NOTES
ARC	1	graphics	Draws a circular arc	Joins two points with a circular arc. Curvature indicated by specifying the angle turned through.
ARC_R	1	graphics	Relative ARC	Like ARC but point is taken relative to the last point.
AT	1	character	Positions text cursor	In version FB parameters are reversed. In version PM, only works on channel 1.
BLOCK	1	pixel	Draw filled rectangle	Dimensions, position of top left hand corner and colour need to be specified.
BORDER	1	pixel	Adds border to window	Thickness and colour must be specified.
CIRCLE	1	graphics	Draws circles/ellipses	Centres, radii, eccentricities and angles of orientation must be specified.
CIRCLE R	1	graphics	Relative CIRCLE	Uses relative coordinates for centres.
CLOSE	1	—	Close window	De-assigns channel number to window.
CLS	1	—	Clear window	Specifies which part to clear. Default is whole window.
CSIZE	1	character	Sets character size	Sets size of characters printed in window.
CURSOR	1	pixel graphics	Position cursor	Can use combination of graphics and pixel coordinates.
DIR	1	—	Lists Microdrive files	
ELLIPSE	1	graphics	Same action as circle	
ELLIPSE_R	1	graphics	Same as CIRCLE_R	
FILL	1	—	Fills solid area	Switches filling on and off.
FLASH	1	—	Character flashing	Switches flashing on and off. Only in mode 8. Only text flashes.
INK	1	—	Set foreground colour	
INKEYS	1	—	Input character	Function returns value entered. Optional wait period specified.
INPUT	1	—	Inputs data	Optional prompt.
LINE	1	graphics	Draws straight line	Two points specified. Also used to move graphics cursor.
LINE_R	1	graphics	Relative LINE	
LIST	2	—	Lists program	All or part of program listed.
MOVE	1	graphics	Moves graphics cursor	Turtle graphics.
OPEN	1	—	Creates window	See text for details.
OVER	1	—	Sets overprinting	Allows printing of one character over another, combining the two. Also sets strip colour.
PAN	1	pixel	Pans window contents	Whole or part of screen panned left or right.
PAPER	1	—	Sets background colour	
PENDOWN	1	—	Sets 'write' mode	Turtle graphics.
PENUP	1	—	Unsets 'write' mode	Turtle graphics.
POINT	1	graphics	Plots points	One or more points can be specified.
POINT_R	1	graphics	Relative POINT	
SCALE	1	graphics	Change scale	See text for details.
SCROLL	1	pixel	Scrolls window contents	Scrolls all or part of window up or down.
STRIP	1	—	Sets strip colour	Sets local character background colour. See also OVER.
TURN	1	graphics	Relative TURNTO	Turtle graphics.
TURNTO	1	graphics	Turns turtle	Turns turtle through specified number of degrees.
UNDER	1	—	Sets underlining	Set character underlining on or off.
WINDOW	1	pixel	Redefines window	Specifies new dimensions and position of existing window.

John Gilbert demonstrates how to open windows on the QL and disproves the popular myth that they are necessarily linked with multi-tasking

Making the frame fit



IF YOU ASK anybody what a window does he would probably give one of two answers. The first would be that it lets light into a room and the second that it is something through which you can look onto another scene.

The latter is true of the window facility on the QL. You can define a window to look either at a SuperBasic listing or at the results of a program when it is run. QL windows are minature versions of the large screen display. Conversely, the latter is just another window which has been set up by the QL.

When the machine is switched on, or reset, it offers two display options. If you go into monitor, or 80-column, mode you will find that the screen is split into three windows. The one on the left shows the listing of a program and the one on the right produces the results when it is run. At the bottom of the display is the workspace window which is used for entering and editing SuperBasic text.

In television, or 40-column, mode the same windows are displayed in different positions. The workspace window still occupies the bottom of the screen but the listing and run windows have been merged. The run window has been put under the list window and only CLS #2 or RUN. Both windows have been created so that they go into action when the relevant calls, RUN, LIST or CLS, are made to them.

Each window can be addressed using a number prefixed by a hash mark, such as #2 which corresponds to the listing display. Those are channel numbers and by using them you can reference, OPEN and CLOSE data channels and streams.

You may know that the QL sets up its own channels to deal with microdrive operation and sending data to a printer. What may not be so obvious is that the screen is also treated as a device to which you can attach channels. The whole screen display consists of one big window device which is produced on the screen using the OPEN command.

The channel number, attached by means of a hash mark to the OPEN instruction, must be within the range of zero to 15. That means that the QL will allow the use of 16 channels at one time. The QL uses channels zero, one and two to produce the editing, listing and runtime windows. When you are first experimenting with those numbers it might be best not to use those three values to OPEN or CLOSE channels.

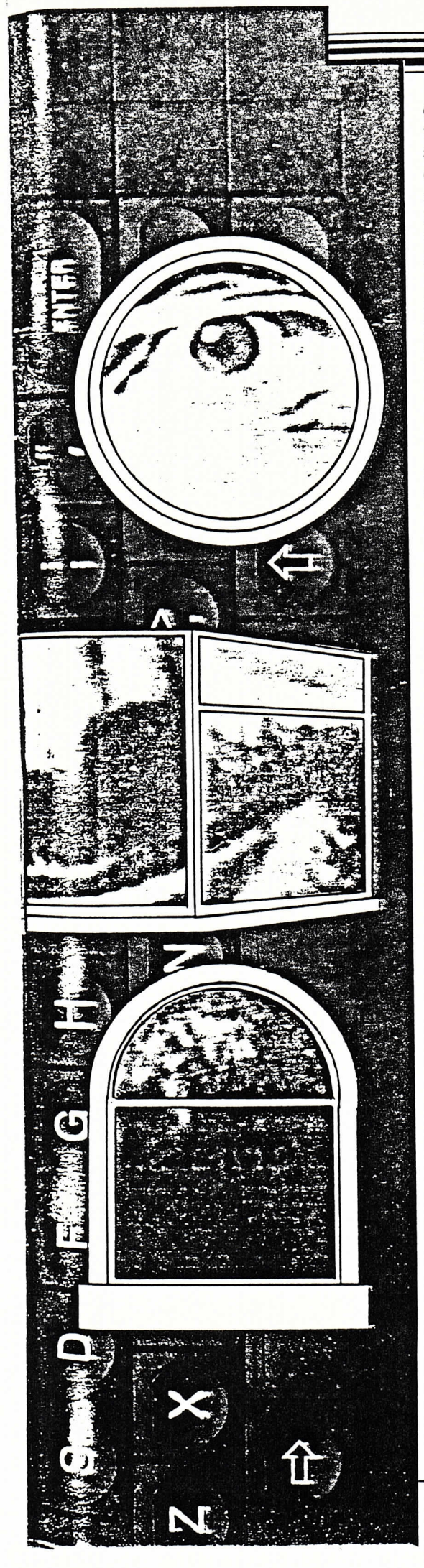
If, for instance, you closed channel zero, which is connected to the window through which the editing of instructions is done, you would be unable to type anything into the machine as window zero is used to accept your input from the keyboard. You can try it by using the instruction CLOSE #0 but make sure that you have nothing important in the memory before you close off the vital visual link to the main processor. The computer will still accept information from the keyboard but that information will not be displayed.

The instruction for OPENing your own windows on the screen uses the format:

OPEN# channel_number, CON_width × height a horizontal × down.

CON tells the QL that the OPEN instruction is to be applied to a CONsole device which is an entry or exit point for a channel through which input and output can be accepted. Theoretically the microdrives can be set up as CONsole devices as they can accept input and output from a file in memory. In the same way a window can accept the input display of characters or graphics and output it onto the screen.

The four numeric values which follow CON_ set up the width and height



of the window followed, after the 'a', by the x,y co-ordinates at which it will appear on the screen. Windows are displayed using the pixel co-ordinate system which consists of 257 pixels running down the screen and 513 pixels running across the display from the lefthand side.

When you think about the positioning of a window it should be at least 32 pixels away from either edge of the screen. If it is not you will find that the window disappears off the edge of the display. The problem is that the QL screen format is larger than that with which a television can cope.

The origination point of any window is at its top lefthand corner. For instance, if you used the co-ordinates 50,50 that corner would be located at a point 50 pixels from the top of the screen and 50 across from the left. If a window is defined at that point you can safely give it a size of 130 pixels both in width and depth.

The instruction is:

OPEN # 3, CON_130x130a50x50

When opening the window through channel three make sure that you enter the 'x's and 'a's within the statement and not make the mistake of using commas which nearly all the other commands relating to SuperBasic graphics use. Think of the 'x' as meaning 'by' in carpenters' terms and 'a' meaning 'across the display' in terms of position from the top of the screen.

If you type in the OPEN # 3 statement as a direct command you will have to type in CLS and CLS # 3 to see the results of your work against the red background of the runtime screen. To see the effects of the windows on the screen you can define another window. Position it at 200,50 which is 150 pixels to the right of the first and give it the same dimension of 130,130.

OPEN # 4, CON_130x130a200 × 50

When you clear the screen again and then CLS # 4 you will see that the new window has appeared by the side of the first. You can give them different tasks to do and you will see that each responds almost immediately.

You can list a program in any window by typing the LIST instruction followed by the # suffix, which was used in the OPEN statement which defined it followed by its channel number. Enter the following program, or use one of your own, and then produce a listing of it in both windows # 3 and # 4.

```
10 PRINT # 0, "Sinclair User": Pause 50
20 PRINT # 0, "shows how to produce": Pause 50
30 PRINT # 0, "windows on": Pause 50
40 PRINT # 0, "the QL": Pause 50
```

If you type in the LIST commands on the same line, ENTERing them at the same time, you will see the delay between finishing one task in a window and starting another.

LIST # 3: LIST # 4

The delay and the way in which you entered the LIST instructions disproves the popular myth that windows and multi-tasking are somehow linked. As Sinclair Research has explained windows can be used to multi-task in machine code but just because you can output different listings and displays to windows you cannot run two programs concurrently in SuperBasic.

That is not to say windows are a waste of time when used within SuperBasic. You can, for instance, set up several display areas using windows some of which are used for the input of information, some for responses and some for displaying the status of the program. Such formats could be used in business programs, such as Archive and Easel, or in complex adventure games in which compartmentalised status displays are required.

The use of graphics within different windows is not as complex as it may seem in the User Guide. When the two windows # 3 and # 4 were defined the pixel co-ordinate system — Figure 1 — was used with a scale that ran down the screen from zero to 256. When producing graphics, such as lines, arcs and circles within a window you will need to use the graphics display co-ordinate system which exists in parallel with the pixel display — Figure 2.

It might seem complicated to have two systems operating on the screen to do different tasks but the graphics scale is more flexible than the pixel. The pixel scale is fixed but you can change the graphics scale from its default range of 0 to 100 co-ordinates to any other range. For instance, you could rescale it to 150 or 200.

You can see the change in scale by drawing a line up the lefthand side of window # 3. That is done by using the command

LINE 0,0 TO 0,100

The first set of values in the LINE command marks the x,y co-ordinates of the point of origin of the line and the last two are the destination co-ordinates. The scale has initially been set by the

continued on page 144

continued from page 143

QL at 100 and so the line should touch the top of the window display.

If, however, you change the scale the results will be different.

The instruction is:

SCALE # 3, 200,0,0

doubles the scale of the window # 3 to 200 instead of its original 100 pixels in depth. The whole window is affected by the change as you are using 0,0 co-ordinates as the start point of the change but you could make the scaling even more complex by starting the 200 scale somewhere else in the window which would leave the 100 scale still partially in effect. For instance, if you rescaled at 0,50 the new scale would

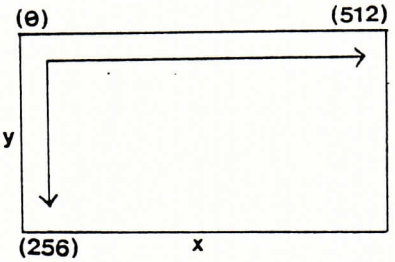


Figure 1. Pixel co-ordinate system.

come into effect halfway up the window.

When you have reset the range on the window type in the LINE command using 100 as its length. You will see that the line will now only go halfway up the window. The graphics system has been adapted for use with the 200 scale.

As well as redefining the SCALE with which window graphics can be plotted it is also possible to redefine the positions of windows, which either you or the QL have brought into existence, without using the CLOSE command to close a channel and re-opening it at another position. The instruction to do that is WINDOW and it will enlarge or shrink the existing window and relocate it on the screen if necessary.

You might, for instance, want to put the editing facilities of the window # 0 onto the main part of the screen so that it overlays both the runtime and the listing windows. That would mean you would have to CLS # 0 every time you wanted to bring the edit window to the top of the stack instead of relying on the QL to do it automatically.

The width of the redimensioned # 0 window would be 448 on the x-axis and 180 along the y-axis. The origination of the window is not as you might think 0,0 because of the obliteration problem mentioned earlier. It is 32 for the x-axis and 16 for the y-axis, counting down from the top of the screen. The full definition is:

WINDOW # 0, 448,180,32,16

You will find that once you have entered that as a direct command you will have a whole screen in which to edit information instead of the few lines given to you by the QL original editing window. Unfortunately it looks messy as you have three colours on the screen

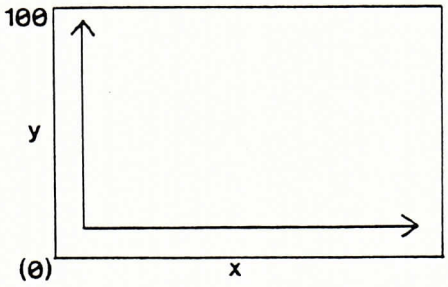


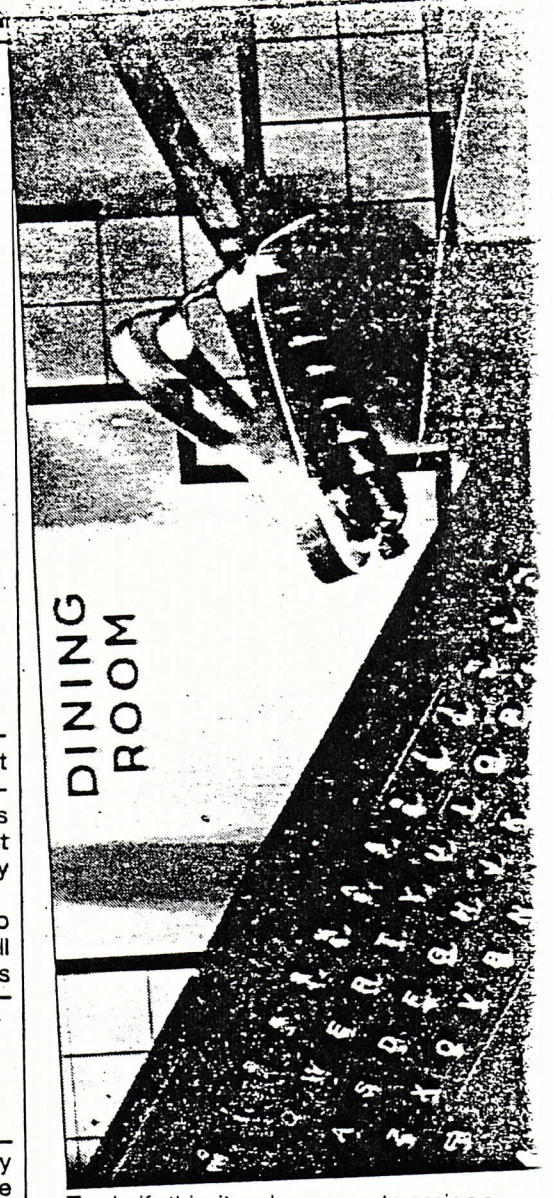
Figure 2. Graphic co-ordinate system.

— red, black and blue. To clear all those problems you can use the program below to get rid of any text which might have been left at the bottom of the screen when you re-located window # 0. Make sure, however, that the first instruction in all your programs which use the technique is CLS # 0.

```
10 CLS # 0
20 WINDOW # 0,448,180,32,16
30 FOR K= 0 TO 2
40 PAPER # K,0: CLS # K
50 NEXT K
60 CLS # 0
70 PRINT # 0,"ready"
```

Extending QL SuperBasic

Adam Denning, of Micronet 800, shows that QL Basic can be enhanced by way of extensions to the machines procedure list.



Although the QL's SuperBasic is in many respects a wonderful programming language it is somewhat deficient when seen as the QDOS command language. Where are essential disc commands like BACKUP and RENAME?

Luckily the mechanism for extending the procedure list is simple—in Basic! The procedures to do these extra operations are also generally easier to write in Basic, so that's what we're going to do here.

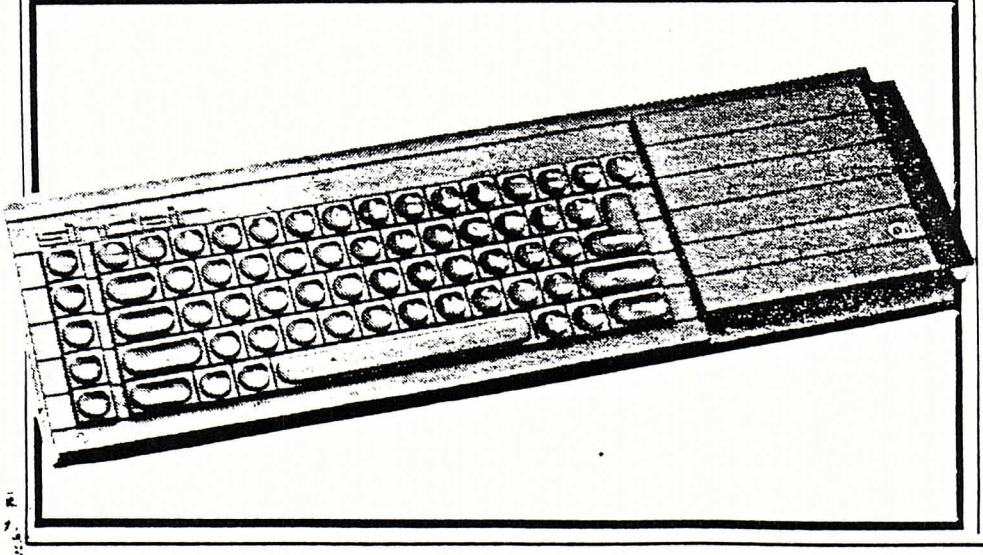
The author is currently working on the machine code versions of these procedures so that these can be linked in as part of the 'permanent' procedure/function list on boot. The QDOS documentation explains reasonably clearly how to do this, but the lack of a QL assembler makes things rather awkward at the moment.

When adding procedures and functions to SuperBasic from **within** SuperBasic, one has to choose a line numbering such that as little conflict as possible occurs with the user's own program and the easiest way to do this is by using the highest line numbers possible. The list of procedures described here starts at line 31500, and with the default AUTO line number of 100 it would take a very extensive program indeed to overrun the procedures. Naturally those procedures that are not required within a particular application may be deleted if the program does get too big, but if you do this you must ensure that procedures that are called by other procedures are not deleted.

As a number of QL owners are likely to be new to Basic programming it is as well to get the distinction between procedures and functions clear at the start. In many respects they are the same in that they are both defined blocks of code that perform a particular operation or sequence of operations.

"... the mechanism for extension is simple".

The fundamental difference is that a function returns a result while a procedure does not. This issue can become a little clouded when a procedure is written which alters the value of a variable *declared outside the procedure,* as it then appears to return a result, but in such cases the result is implicit rather than explicit.



To clarify this, it makes sense to assign a variable to a function, but a similar operation within a procedure would be syntactic nonsense. In other words if **func(param)** is a function then one can write

```
result = func(param)
```

but if it were a procedure one could not. As an example, let's consider some SuperBasic functions and procedures. **SIN, INT, RESPR** and **CHR$** are examples of functions as all the lines below make perfect sense:

```
PRINT SIN(PI)
x=INT(RAD)(45))
space=RESPR(1024)
a$=FILL$(CHR$(26),34)
```

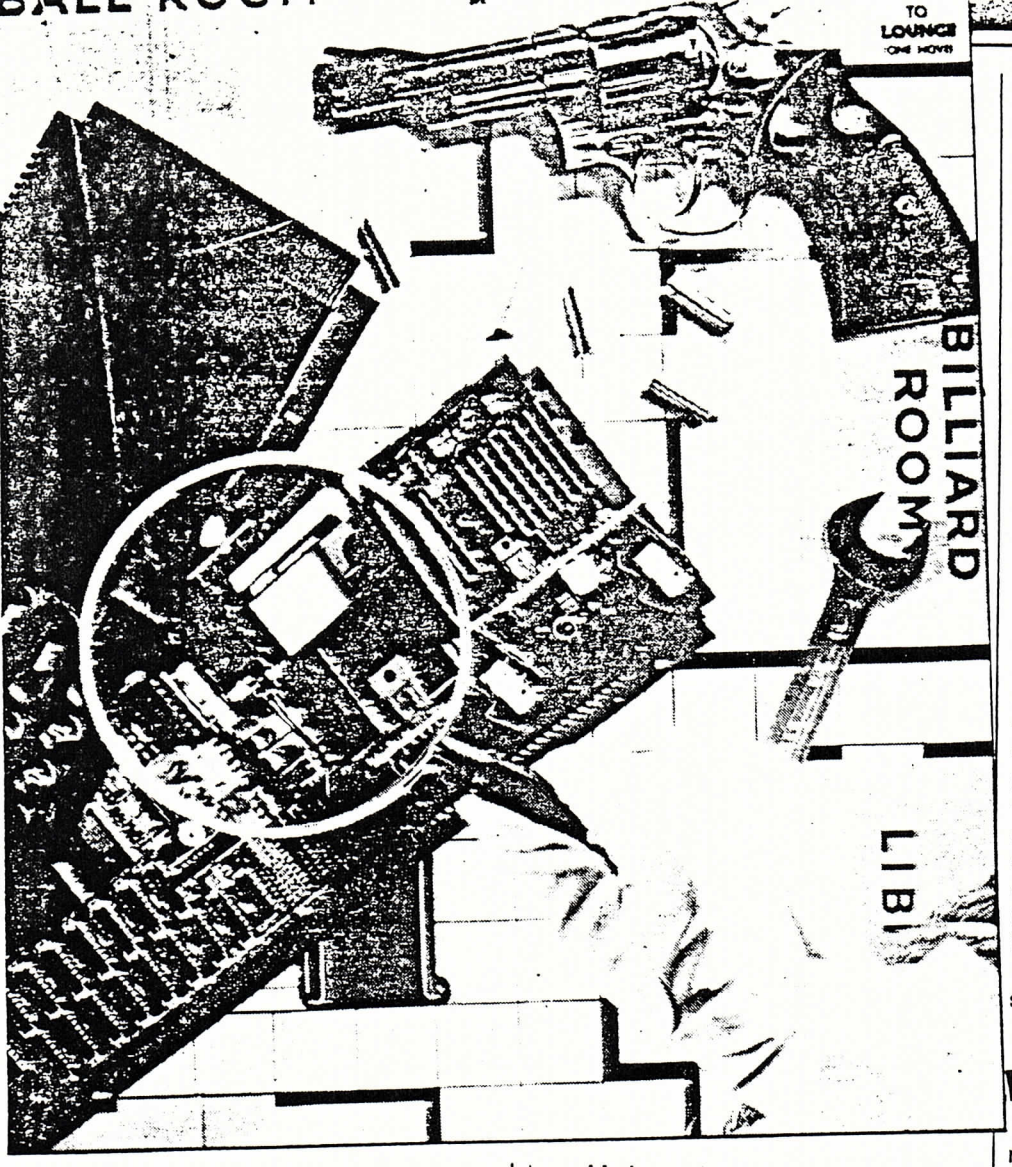
SuperBasic procedures are things like **CLS, PRINT, LIST** and **POKE.** That's because you can say things like:

```
PRINT "I'm a procedure"
```

while things like

```
a=CLS
```

are pure rubbish! Notice how SuperBasic functions, whether inbuilt or user defined, *always* have their parameters in brackets while procedures in general don't. We say 'in general' because it is quite valid to include parameters in brackets, but doing so alters the way they are treated by



SuperBasic. A parameter of x is subtly different to a parameter of (x) – the first case results in the variable itself being the parameter while parenthesised parameters pass the value.

This is only of significance to us when a procedure alters a global variable – by passing the variable in brackets it won't get altered. Sometimes!

While we're defining things we might as well make the distinction between operators and procedures/functions too. An operator returns a result by operating on one or more operands, so that in an operator that returns the negative of the operand following it, and + returns the result of adding the operands on either side of it and **MOD** produces the remainder of an integer division of the left hand operand by the righthand operand. Notice that, unlike most other computers that supply this function, **INSTR** in SuperBasic is an operator and not a function.

Now onto our defined procedures and functions. Notice that in many cases we have declared certain of the parameters as being typed (ie said that they are integer, string or whatever) despite the fact that formal parameters are defined as typeless in SuperBasic. This has been done simply to clarify the use of each parameter and it is not necessary to adhere to this nomenclature. **Listing 1** is a procedure to spin a specified microdrive cartridge until the ESCAPE key is pressed. Although this may at first sound entirely fatuous it has proved to be of immense value when trying to recover data from microdrive cartridges that seem intent on always returning **bad or changed medium** error reports. It will by no means always recover a set of data but a good spin of sixty seconds or so does seem to do a cartridge wonders! The procedure also raises a few interesting points about SuperBasic.

"SuperBasic is somewhat deficient when seen as the QDOS command language".

As with other procedures it is good programming practice to reduce the scope of variables within a procedure as much as possible – in other words, make them local as the first line of a procedure or function definition. This has been extended to include loop identifiers (used in REPeat, FOR and SELect) because the scope of these matches are that of a variable.

The **SPIN** procedure first builds up a string to represent the specified microdrive cartridge, putting the result in **drive$**. It then opens the smallest screen window possible to channel 200. Such a high channel number was chosen simply because it is unlikely to conflict with channel definitions elsewhere in a program. This line also shows up to points worth noting. Firstly, although the QL User guide implies that window definitions can use steps of 1 pixel this in fact seems to be 2 pixels in reality. Secondly the guide again implies that only channel numbers #0 to #15 are valid in Basic (as is the case with the Spectrum). This also turns out to be untrue, although we have yet to discover the highest channel number that is available.

Once this window is open we enter a loop in which the cartridge is repeatedly DIRd to this channel until the ESCAPE key is pressed. As the window is so small and out of the way it is, to all intents and purposes, invisible. Once ESCAPE has been pressed we clear and close channel 200 and then return to whatever called the procedure. Believe me, it really isn't as facile as it seems!

The procedure of **Listing 2** is short and sweet and merely serves to increase the friendliness of SuperBasic. It allows us to type **CAT n** to catalogue a drive rather than having to go through the eventually rather tedious process of having to type **DIR mdvn_** each time. As the procedure stands it can only catalogue to the default channel (1) but it is a simple matter to alter it to accept a second parameter representing the channel to catalogue to.

The procedure is straightforward and self explanatory.

What's in a name

Listing 3 shows a procedure that is of rather more immediate use and simply adds a **RENAME** procedure to SuperBasic. It has three parameters – the file name to be renamed, the new file name and the drive. As it stands the procedure will not allow you to generate a copy of a file on a different drive, which is standard RENAME practice. If you really want to do this, try the QL's COPY procedure! RENAME works by building up two strings – one to define the old file and one to define the new file. The old file is then copied to the new file and subsequently deleted, which is the nearest we can get to a RENAME. Certainly its effect is to rename a file, but if the file to be renamed is longer than the space left on a cartridge it obviously can't do its job. It would probably be possible to write a machine code procedure to rename every sector of a file, but it would be of dubious value as it would decrease the life of the cartridge, which is short enough now!

Note that there is no explicit error checking within the procedure, so if the old file does not exist or the new file does already exist, a standard SuperBasic error message will be generated and the procedure will be aborted. For this reason error

checking was deemed unnecessary.

The most complicated procedure is that which performs a drive-to-drive backup. It uses a fairly devious technique to discover which files exist on the source drive's cartridge, and it includes the option to delete the original files as it goes. This option would be taken if a cartridge were proving unreliable and you wanted to reformat it but keep the files it contains intact. This option is selected by making the third parameter anything other than zero. A value of zero for **switch%** forces the procedure to perform the more usual backup facility in which files are simply copied across. This procedure is shown in **Listing 4.**

The first thing this program does is define three strings – **d$, a$** and **b$.** The first holds the filename 'dir_tmp' on the destination drive, the second holds the source drive definition and the last holds the destination drive definition. Channel 3 (you may alter this of course) is then opened as a new file and the source drive is catalogued to it. This file is then closed and re-opened for read only. Two dummy strings are then input from the directory, the first representing the cartridge name (which we don't need) and the second representing the number of sectors available on the cartridge. Again, we do not need this information. Everything left in the file is then a file name to be copied across, so we enter a REPeat loop (controlled by **movefiles**) to copy each file. The first thing this loop does is check to see whether we have reached the end of the directory file, and if so leave the file. The end of the file is checked first because if we attempted to backup an empty drive this occurrence would be trapped without error before it happened. In other words the loop is set up to act as a zero-case trap loop, something which is tedious to implement on a BBC Micro!

LISTING 1:

```
31500 DEFine PROCedure SPIN(dr%)
31510 LOCal loop,drive$
31520 drive$="mdv"&dr%&"_"
31530 OPEN#200,scr_2x2a0x0
31540 REPeat loop
31550   DIR#200,drive$
31560   IF INKEY$=CHR$(27):EXIT loop
31570 END REPeat loop
31580 CLS#200:CLOSE#200
31590 END DEFine
```

LISTING 2:

```
31600 DEFine PROCedure CAT(dr%)
31610 LOCal f$
31620 f$="mdv"&dr%&"_"
31630 DIR f$
31640 END DEFine
```

LISTING 3:

```
31650 DEFine PROCedure RENAME(old$,new$,dr%)
31660 LOCal a$,b$
31670 a$="mdv"&dr%&"_"&old$
31680 b$="mdv"&dr%&"_"&new$
31690 COPY a$ TO b$
31700 DELETE a$
31710 END DEFine
```

LISTING 4:

```
31720 DEFine PROCedure BACKUP(dr1%,dr2%,switch%)
31730 LOCal a$,b$,d$,f$,movefiles
31740 d$="mdv"&dr2%&"_dir_tmp"
31750 a$="mdv"&dr1%&"_"
31760 b$="mdv"&dr2%&"_"
31770 OPEN_NEW#3,d$
31780 DIR#3,a$
31790 CLOSE#3
31800 OPEN_IN#3,d$
31810 INPUT#3;f$;f$
31820 REPeat movefiles
31821   IF EOF(#3):EXIT movefiles
31830   INPUT#3;f$
31840   PRINT#0;'Copying ';f$
31850   COPY a$&f$ TO b$&f$
31860   IF switch%:DELETE a$&f$
31880 END REPeat movefiles
31890 CLOSE#3
31900 DELETE d$
31910 END DEFine
```

BACKUP then prints a message to channel zero (the command channel) telling us which file it is copying, and then it copies that file from source drive to destination drive. Finally, if **switch%** was given a non-zero value then it will be considered as **TRUE,** so the source drive file just copied will be deleted. If the parameter was zero then no files will be erased. Once the end of the directory file has been reached the loop is ended and channel 3 closed. The directory file is then deleted and the procedure ends. Again there is no explicit error checking, as common sense and SuperBasic ought to prevail!

Classic coding

The next piece of code is a classic conversion function that returns a decimal number corresponding to a string parameter that is taken to represent a number in any base between 2 and 36. It works by zeroing an accumulator (the variable total) and progressively adding each converted digit to base times the accumulator. The method is not the fastest but it is the easiest to understand. It makes use of the QL's **INSTR** operator to find the position of each digit in a string comprising all possible digits for that base, and returning its representative decimal value to the variable **temp.** If the tested digit does not occur in the string then it is treated as invalid and results in **temp** having a value of —1.

Once the value has been obtained invalid digits are trapped in such a way that the function terminates immediately and returns a value of zero. In valid cases the process is repeated for the length of the number string. **Listing 5** shows the procedure.

The reverse of this function is provided by the next function **OFDEC$.** This is passed as a decimal number and a number between 2 and 36 to represent the base. The decimal number is then converted to a string representing that number in the given base, and this is returned as the function's result. It works by using a process analogous to splitting a number up into the number of units, tens, hundreds, thousands and so on that form the number, but instead of splitting it into powers of ten it splits it into powers of the specified base. The resulting number is then converted to ASCII and added to the front of an initially null accumulator string (unfortunately called **hex$** here!). The end effect is to produce a string that accurately represents the number. **OFDES** is shown **Listing 6**

Numbers to the base 16 (hexadecimal or just hex) are so useful in computing that a function to allow us to use hex numbers within a SuperBasic program would be very useful. Our **DFC** function would do this quite happily but it always has to be passed two parameters the nex number and 16 to represent the base. So this short function, called HEX, calls up DEC and automatically adds the base as 16 during the call. This means that we can now use hex numbers very easily, like so:

POKE_W HEX ('20000'), HEX('4E75')

Our final manipulation of the two base conversion functions results in a routine that will convert a number in one base to a number in another base. We use one of our functions as a parameter to the other, resulting in a base to base conversion involving decimal as an intermediate stage. This is by far the easiest method, and the function thus defined is known as **BASES.** It is passed by three parameters – the number to be converted (passed as a string) and the two bases. The first is the base that we are converting from and the second is the base to which we want to convert. This procedure is shown in **Listing 8.**

The final procedure shown in **Listing 9** is purely of interest to those with an Epson dot matrix printer. It opens a specified channel to the printer (which is assumed to be connected to serial port one) and then prompts for an option string. This option string can consist of any combination of the digits 0, 1 and 2. Option 0 initialises the printer by sending ESC @, option 1 sets the UK character set (so that £ signs get printed properly) and option 2 sets up the USA character set (so that the # sign gets printed correctly).

This set of nine procedures and functions merely serves to demonstrate the power of SuperBasic, and the user can easily write a whole bunch more for all sorts of applications.

LISTING 5:

```
31920 DEFine FuNction DEC(num$,base)
31930 LOCal total,temp,loop
31940 total=0
31950 FOR loop=1 TO LEN(num$)
31960   temp=(num$(loop) INSTR "0123456789ABCD
      EFGHIJKLMNOPQRSTUVWXYZ"(1 TO base)
)-1
31970   IF temp<>-1
31980     total=total*base+temp
31990   ELSE
32000     total=0
32010     EXIT loop
32020   END IF
32030 END FOR loop
32040 RETurn total
32050 END DEFine
```

LISTING 6:

```
32060 DEFine FuNction OFDEC$(number,base)
32070 LOCal num,t1,t2,hex$,buildstring
32080 num=number:hex$=""
32090 REPeat buildstring
32100   IF num<base:EXIT buildstring
32110   t1=INT(num/base):t2=t1
32120   t1=num-t1*base:num=t2
32130   hex$=CHR$(t1+48+7*(t1>9))&hex$
32140 END REPeat buildstring
32150 hex$=CHR$(num+48+7*(num>9))&hex$
32160 RETurn hex$
32170 END DEFine
```

LISTING 7:

```
32180 DEFine FuNction HEX(hex$)
32190 RETurn DEC(hex$,16)
32200 END DEFine
```

LISTING 8:

```
32210 DEFine FuNction BASE$(num$,base1,base2)
32220 RETurn OFDEC$(DEC(num$,base1),base2)
32230 END DEFine
```

LISTING 9:

```
32240 DEFine PROCedure EPSON(chan%)
32250 LOCal uk$,usa$,init$,choice$,ch,loop
32260 uk$=CHR$(27)&'R'&CHR$(3):usa$=uk$
32270 OPEN#chan%,ser1c
      (1 TO 2)&CHR$(0):init$=CHR$(27)&'@'
32280 REPeat loop:INPUT#0;'Initialise
      (0), Set UK (1) or Set USA (2)?'!choice$:I
F choice$<>'':EXIT loop
32290 FOR loop=1 TO LEN(choice$)
32300   ch=choice$(loop)
32310   SELect ON ch
32320     =0
32330       PRINT#chan%;init$;
32340     =1
32350       PRINT#chan%;uk$;
32360     =2
32370       PRINT#chan%;usa$;
32380     =#REMAINDER
32390       PRINT#0;'Invalid parameter'!loop!''
32400   END SELect
32410 END FOR loop
32420 PRINT#0;'Printer channel 'chan%!'now set up'
32430 END DEFine
```

Multitasking made easy

Adam Denning says there's nothing to QL multitasking – he explains in his crash course on getting the most out of the QL.

Multi-tasking on the QL is easy. There really is nothing to it. All the rumours about how difficult it is to write programs which will run concurrently with others are pure fabrication. Oh sure, you have to be able to program in 68008 machine code, but let's face it if you're reading this magazine then it's a fair bet that you're half way there already.

Each multi-tasking program is referred to as a **job** by QDOS, and each is simply a self-contained program. Each job has its own record of all its registers, so whenever the system switches the processing from one job to another all it has to do is save the registers in a pre-defined place, load the registers of the next job, and away it goes.

Each job runs for the most part in the 68008's User mode, so that the task of processing each job can be looked after by the scheduler, which runs in Supervisor mode. If at any time a job wanted to go into Supervisor mode then it is fully able to do so, but the act of entering this mode immediately makes the current job the only task running until the system reverts to user mode.

The SuperBasic interpreter is itself a job, but is a little special in that it can alter the amount of room it takes up dynamically, while other jobs are required to declare their RAM requirements before they are activated. The SuperBasic interpreter is known as job 0, as it has a job number of 0 and a tag of zero. Don't worry, these words will be explained very soon.

"Oh sure, you have to be able to program in 68008 machine code".

Each job has a priority, which is a number between 0 and 127 which determines how often the job will be executed in a given period of time. QDOS multi-tasks by allocating each active job a slice of the processor's time, and it decides how often to give a job a time slice from that job's priority. If the priority is zero then the job will never be executed, it is said to be inactive. Other levels of priority mean that job is active and will therefore be given processor time in due course.

However a job can be active but in a state of suspension, either because of some deliberate act or because it is waiting for some action to finish. It may for

LISTING 1. The header file – header ASM.

```
        NOLIST

* Operating system vectors

UT_CON    EQU     $C6
UT_SCR    EQU     $C8
UT_ERRO   EQU     $CA
UT_ERR    EQU     $CC
UT_MINT   EQU     $CE
UT_MTEXT  EQU     $D0
UT_CSTR   EQU     $E6
CN_DATE   EQU     $EC
CN_DAY    EQU     $EE
BP_INIT   EQU     $110
CA_GTINT  EQU     $112
CA_GTFP   EQU     $114
CA_GTSTR  EQU     $116
CA_GTLIN  EQU     $118
BV_CHRIX  EQU     $11A
RI_EXEC   EQU     $11C


* Operating system offsets and equates

CH_LENCH  EQU     $28
BV_CHBAS  EQU     $30
BV_RIP    EQU     $58
ERR_NO    EQU     -6
ERR_EF    EQU     -10
ERR_BP    EQU     -15
ERR_OV    EQU     -18
ERR_BL    EQU     -21
RET_STR   EQU     1
RET_FP    EQU     2
RET_INT   EQU     3
OPEN_INX  EQU     0
OPEN_INS  EQU     1
OPEN_NEW  EQU     2
OPEN_OVR  EQU     3
OPEN_DIR  EQU     4

* Trap keys
* Trap 1

MT_CJOB   EQU     1
MT_JINF   EQU     2
MT_FRJOB  EQU     5
MT_SUSJB  EQU     8
MT_RELJB  EQU     9
MT_ACTIV  EQU     $A
MT_PRIOR  EQU     $B
MT_DMODE  EQU     $10
MT_IPCOM  EQU     $11
MT_RCLCK  EQU     $13
MT_ALCHP  EQU     $18
MT_RECHP  EQU     $19
* Trap 2

IO_OPEN   EQU     1
IO_CLOSE  EQU     2

* Trap 3

IO_FBYTE  EQU     1
IO_FLINE  EQU     2
IO_FSTRG  EQU     3
IO_SBYTE  EQU     5
IO_SSTRG  EQU     7
SD_WDEF   EQU     $D
SD_CURS   EQU     $F
SD_POS    EQU     $10
SD_TAB    EQU     $11
SD_NCOL   EQU     $14
SD_CLEAR  EQU     $20
SD_FOUNT  EQU     $25
FS_POSAB  EQU     $42
FS_POSRE  EQU     $43
FS_HEADR  EQU     $47
FS_LOAD   EQU     $48

* RI operation keys

RI_FLOAT  EQU     -8
RI_ADD    EQU     $A
RI_MULT   EQU     $E
```

instance be waiting for another, subsidiary job to terminate, or it may want to read the keyboard but is unable to because another job is doing so.

In the November issue of *E&CM* I explained that most of the calls to the QDOS operating system are made by using one of five 68008 trap instructions: TRAP #0 to TRAP #4. Each of the individual traps is delegated a certain group of tasks, and in the case of traps with more than one function to provide (TRAPs #1, #2 and #3) the actual function is determined by the value of register D0 on entry to the trap. A trap is just a 68000 instruction which switches the processor to an exception handling routine, which is essentially similar to the software interrupt found on smaller processors like the 6502 and 6809. It also invokes supervisor mode, so anything that is trap invoked is likely to be executed within that job's current time slice. In certain cases this is guaranteed and is then described as an atomic action.

TRAP #1 is used by QDOS for all its manager routines, such as allocating memory to jobs, creating jobs, talking to the 8049 co-processor and so on. TRAP #2 is used to allocate RAM for I/O (in other words it looks after channel open and close) and TRAP #3 looks after other I/O operations such as the actual transfer of data, the screen drivers and the file system.

A job is normally held in a file, and would be executed from SuperBasic using either the EXEC or EXEC_W commands. The former loads and activates the job and then returns to the Basic interpreter, whilst the latter loads and executes the job but suspends the SuperBasic interpreter pending completion of the thus loaded job. Let's look at EXEC and EXEC_W in more detail, so that we can get a little more insight into QDOS and job control.

Open given filename for reading
Load the file's header into RAM
Read its length and the length of its data area
Create a job with these specifications
Load the file into the space allocated
Close the file
Activate the job with priority 32, and a timeout of 0 if the command was EXEC or —1 if the command was EXEC_W

That't the basic outline of how these two SuperBasic commands are implemented. Now let's examine each stage in greater detail. Remember that both these keywords are passed a filename as their parameter. We'll call this file job file. Having invoked the procedure it first checks to see if it has the correct number of parameters and whether they (or rather, it) is in the cor-

LISTING 2: Printer spooler routines

```
* By Adam Denning
* Copyright (C) 1984 Adam Denning

        GET     "adv1_header_asm"
        SIZE    150

* A printer spooler routine
* started 23rd September 1984
* Altered from byte-by-byte transfer to 100 bytes at a time
* on Wednesday 26th September
* Altered from 100 byte at a time transfer to 4096 bytes at a time
* transfer on 10th October 1984

BUF_LEN   EQU     100             Length of input buffer
HEAP_ROOM EQU     4096            Length of common heap area wanted

        BRA.S   START_P           Ignore standard format code
        DC.L    0
        DC.W    $4AFB             Standard format identification
        DC.W    7                 Program name
        DC.B    'SPOOL_1',0

* Use MT_PRIOR to set the priority of this job to 1 so that it does not
* slow Basic down too noticeably. MT_PRIOR sets the priority of the job
* whose ID is held in D1 to the value held in D2. It preserves all
* registers except A0 which is left holding the base of this job's job
* control area. If D1 was passed as -1 (for current job) then on return
* it will hold the true job ID

START_P MOVEQ   #MT_PRIOR,D0      Set priority
        MOVEQ   #-1,D1            of this job
        MOVEQ   #1,D2             to 1
        TRAP    #1

* Use the UT_CON utility routine to open a console device (CON_) with the
* parameters specified in the definition block pointed to by A1. It returns
* with the ID of the newly opened channel in A0 and corrupts D0 - D3 and
* A1 - A3. If there are no errors then D0 is returned as 0

        LEA.L   PBLOCK,A1         Open up specified console device
        MOVE.W  UT_CON,A2
        JSR     (A2)
        TST.L   D0                Error?
        BNE     JOB_END           Yes, so leave

* Use the UT_MTEXT utility routine to print a prompt on the screen. This
* utility needs the ID of the channel to which the message is to be
* printed in A0 and the base address of the string in A1. It corrupts all
* registers except A0 (A4 - A7 are safe)

GET_FILE LEA.L  MESSAGE1,A1       Print 1st message
        MOVE.W  UT_MTEXT,A2
        JSR     (A2)

* Use the IO_FLINE trap to collect a filename from the keyboard (the
* input device to our console channel). IO_FLINE requires the ID of the
* channel from which the data is to be read in A0, a timeout in D3, the
* length of the buffer to which the line will be collected in D2 and
* the actual address of the buffer in A1. All registers are preserved
* except D1 which holds the number of bytes collected (including the L/F
* which terminated the input) and A1 which is updated to the next free
* buffer position. The error 'buffer full' can be returned but it is
* unlikely to be in this case; we ignore it anyway!

        MOVEQ   #BUF_LEN,D2       Fetch filename from channel
        MOVEQ   #-1,D3
        LEA.L   BUFFER,A1
        MOVEQ   #IO_FLINE,D0
        TRAP    #3
        MOVE.L  A0,-(A7)          Save console channel ID

* Now open the collected filename for shared input (so that other jobs
* can examine the file if necessary) using IO_OPEN. This trap requires
* the job ID in D1 (or -1 for current job) and the open access in D3 as
* a byte key. A0 must point to the name of the device to be opened. All
* registers are preserved except D0 (error return), D1 (true job ID if -1
* was passed) and A0, which holds the ID of the newly opened channel

        LEA.L   BUF_POS,A0        Get ready for IO_OPEN call by
        SUBQ.L  #1,D1             converting line fetched to a
        MOVE.W  D1,(A0)           string, removing LF from count
        MOVEQ   #OPEN_INS,D3      Open this file for input
        MOVEQ   #-1,D1
        MOVEQ   #IO_OPEN,D0
        TRAP    #2
        TST.L   D0                Error?
        BEQ.S   GOT_FILE          No - so continue

* If the call to IO_OPEN above resulted in an error then the error code
* is in D0. The UT_ERR utility routine will print out the error message
* corresponding to this code to the channel whose ID is held in A0. All
* registers are preserved by the call

        MOVE.L  (A7)+,A0          Else retreive console channel ID
        MOVE.W  UT_ERR,A2         and write requisite error message
        JSR     (A2)              to it. Then try again.
        BRA.S   GET_FILE

* Console channel ID is currently on the stack; swap with file ID held in
* A0, putting the console ID in A0 so that IO calls use that channel

GOT_FILE MOVE.L (A7)+,A1          Put console channel ID in A1
```

rect format.

It then uses trap #2 to open the file. The actual function performed is IO_OPEN, which involves D0 holding the code for IO OPEN (1), D1 holding the ID of the job for which the file is being opened (the SuperBasic interpreter – job 0 – in this case), D3 holding the code under which the file will be opened (probably 1 for shared read only) and A0 pointing to the filename. The trap is then executed and the channel is opened. Or not. If there was some error, such as the file not being found or not enough memory, the D0 is returned holding some value other than 0. This applies to all QDOS traps, D0 being equal to zero is the only indication of success. Any other values of D0 are actual error codes, which would normally be passed back to Basic and reported.

Assuming success, all other registers except A0 are returned unharmed, but A0 holds what is known as the channel ID. This, like the job ID is a long word (four bytes), and is of fundamental importance. To communicate with a channel you must have its ID. Note that there is no direct connection between the #channel numbers used in Basic and the ID returned by IO_OPEN.

Having successfully opened our file we must now read its header into RAM. A header is in this case 14 bytes long (assuming a microdrive file) and is laid out as shown:

long word	file length
byte	access (unused)
byte	type (0 for all except SEXECd files, when it is 1)
long word	data space length for SEXECd files or zero for normal files
four bytes	currently unused

All EXEC / EXEC_W needs to do is check whether or not the file type byte is 1 and if it is read the file length into register D2 and the data space length into D3. The header would be read into RAM with the FS_HEADR trap, which is trap #3 with D0 = #47. This requires D2.W to hold the buffer length (minimally 14, but 15 for safety), D3.W to hold the timeout (see below), A0 to hold the channel ID and A1 to point to the address of the buffer into which the header will be read. Assuming no errors the file length and data lengths can then be read with two simple MOVE.L instructions, and away we go.

The next trick is to create the job. The manager trap MT_CJOB does this, and it is invoked by loading D0 with 1 and doing a trap #1. It requires D1 to hold the ID of the job which will own the new job, and as this is going to be the SuperBasic interpreter this would normally be zero. D2 and D3 hold the code length and data lengths respectively and A1 holds an absolute starting address or zero. This would normally be zero as we don't want to rely on absolute addresses in a 68000 system. Putting 0 in A1 makes QDOS allocate

LISTING 2 – Continued

```
          MOVE.L    A0,-(A7)            save file channel ID on stacl
          MOVE.L    A1,A0               make console current channel

* Use UT_MTEXT and IO_FLINE as before to collect the device name of the
* channel to which the data is to be printed

GET_OUTP  LEA.L     MESSAGE2,A1         Print 2nd message to console
          MOVE.W    UT_MTEXT,A2         to it.
          JSR       (A2)
          MOVEQ     #BUF_LEN,D2         Get printer device specification
          MOVEQ     #-1,D3              from console and open it as file
          LEA.L     BUFFER,A1
          MOVEQ     #IO_FLINE,D0
          TRAP      #3
          MOVE.L    A0,-(A7)            Save console channel ID
          LEA.L     BUF_POS,A0          Point to start of string
          SUBQ.L    #1,D1               'Remove' trailing L/F by reducing
*                                       the character count by 1
          MOVE.W    D1,(A0)             Save count at start of string

* Use IO_OPEN again to open the printer device for exclusive output

          MOVEQ     #OPEN_NEW,D3
          MOVEQ     #-1,D1
          MOVEQ     #IO_OPEN,D0
          TRAP      #2
          TST.L     D0                  Error?
          BEQ.S     GOT_OUTP            No - so continue

* If IO_OPEN above failed then use UT_ERR as before to print error message
* to console channel; then collect name again and repeat until no error

          MOVE.L    (A7)+,A0            Retrieve console channel ID
          MOVE.W    UT_ERR,A2           print requisite error message
          JSR       (A2)
          BRA.S     GET_OUTP            and try again

* We no longer need the console so we can choose that channel. First
* swap its channel ID at the top of the stack with that of the newly
* opened printer device and then call IO_CLOSE. This trap just needs the
* channel ID in A0 and all registers except A0 are preserved

GOT_OUTP  MOVE.L    (A7)+,A1            Swap console and printer channel
          MOVE.L    A0,-(A7)            IDs on stack and close the
          MOVE.L    A1,A0               console device.
          MOVEQ     #IO_CLOSE,D0
          TRAP      #2

* Now allocate some common heap space for a large buffer to transfer
* bytes from the file to the printer. We first try to claim as many bytes
* as we can (HEAP_ROOM), bbut if this fails then D0 will contain an
* error code. If this is so then we divide the amount of room we require
* by 2 and repeat the process until we have space or until D1 is less
* than or equal to 1, when we return with no transfer as there is not
* enough room. MT_ALCHP requires the job ID in D2 (or -1 for the current
* job) and the number of bytes required in D1. All of D0 to D3 and A0 to
* A3 are corrupted, with D1 holding the number of bytes allocated and A0
* pointing to the beginning of the area allocated

          MOVE.L    #HEAP_ROOM,D1
GET_ROOM  MOVEQ     #-1,D2
          MOVEQ     #MT_ALCHP,D0
          TRAP      #1
          TST.L     D0                  Error?
          BEQ.S     GOT_AREA            No

          LSR.L     #1,D1               Divide space required by 2
          CMPI.L    #1,D1               If D1(= 1 then stop with error
          BGT.S     GET_ROOM            else try again
          BRA.S     FILE_END

* Once area is allocated save address of area in A3 and bytes in area in
* A2, as neither of these registers are corrupted by later traps

GOT_AREA  MOVE.L    D1,A2
          MOVE.L    A0,A3

* Collect input file ID from stack and use IO_FSRTG to read bytes from it
* into common heap buffer. IO_FSTRG fetches a string of bytes from the
* channel whose ID is in A0. It will fetch up to D2.W bytes depending on
* the timeout in D3.W and EOF being reached. The base address of the
* buffer must be in A1. It returns with the number of bytes collected in
* D1.W, an error code or zero in D0, and A1 updated to point to the next
* free position in the buffer. All other registers are preserved

FILE_P    MOVE.L    4(A7),A0            Get file channel ID from stack
          MOVEQ     #IO_FSTRG,D0        Fetch (HEAP_ROOM bytes from file
          MOVE.L    A2,D2               Put area length in D2
          MOVEA.L   A3,A1               and area start in A1
          MOVEQ     #-1,D3              infinite timeout
          TRAP      #3
          MOVE.L    D0,-(A7)            Save error return

* Use IO_SSTRG to send these bytes to the printer device. This trap
* requires the ID of the channel to which the bytes are to be sent in A0,
* the timeout in D3 (it's preserved from IO_FSTRG, so we don't need to
* set it again) and the base of the buffer from which the bytes are to be
* fetched in A1. It returns with an error report or zero in D0 (we ignore
* it anyway), the number of bytes sent in D1.W and the updated buffer
* position in A1. All other registers are preserved.
```

LISTING 2—Continued

```
        MOVEQ   #IO_SSTRG,D0        Send these bytes to printer
        MOVE.L  4(A7),A0            Retrieve printer channel ID
        MOVEA.L A3,A1               put area address in A1
        MOVE.L  D1,D2               and bytes collected in D2
        TRAP    #3

* If IO_FSTRG returned EOF then the process is over, so we can leave

        MOVE.L  (A7)+,D0            Retrieve error return
        TST.L   D0                  End of file?
        BEQ.S   FILE_P              No, so continue

* Use IO_CLOSE to close the printer channel then the file channel, both
* of who's IDs are on the stack.

FILE_END MOVE.L (A7)+,A0            Yes, so close printer
        MOVEQ   #IO_CLOSE,D0
        TRAP    #2
        MOVE.L  (A7)+,A0            and close file
        MOVEQ   #IO_CLOSE,D0
        TRAP    #2
        BRA.S   END_JOB

* Now force release the job (a bit heavy, perhaps, but it certainly
* ensures success!). Also, if one of the earlier errors causes a jump to
* this point then we can use UT_ERR0 to send the error message to the
* command channel. UT_ERR0 is identical to UT_ERR except that it requires
* no channel ID as it automatically uses channel 0 (if it's free, channel
* 1 if not).

JOB_END MOVE.W  UT_ERR0,A2
        JSR     (A2)

END_JOB MOVEQ   #MT_FRJOB,D0        Then kill this job.
        MOVEQ   #-1,D1
        TRAP    #1

* Console device specification

PBLOCK  DC.W    0                   No border
        DC.W    4                   black paper green ink
        DC.W    440                 width
        DC.W    30                  height
        DC.W    36                  X position
        DC.W    15                  Y position

MESSAGE1 DC.W   12
        DC.B    'Print file:

MESSAGE2 DC.W   16
        DC.B    'Printer device:

BUF_POS DC.W    0

BUFFER  EQU     *

        END
```

an area for the job, and this will be its starting address. The trap returns with the ID of the new job in D1 and the base address of the area allocated in A0.

Now we need to load the code of the job from **job_file** into the area allocated. We use another trap #3 routine here, FS_LOAD. This is D0 = $48, with D2 containing the length of the file, D3 containing the timeout, A0 holding the channel ID and A1 holding the address to which the file will be loaded.

We now call IO_CLOSE (trap #2, D0 = 2) to close the file whose channel ID is in A0 when we make the trap, and all we need to do now is activate the job. This is done with another trap #1 routine, MT_ACTIV (D0 = $A), which requires the ID of the job to be activated in D1, the priority with which it is to be activated in D1, the priority with which it is to be activated in D2 and the timeout in D3. Here the timeout is zero if the command invoking all this was EXEC, or –1 if we used EXEC_W.

That's it, really, but there's little point in us writing a program to do all this when all we need to do is type EXEC filename, is there? Instead we're going to explain job ID2, channel IDs and timeouts a little more and then write a program which will multi-task.

IDs and timeouts

To recap, whenever a channel is opened or job created, QDOS returns a number which uniquely identifies that job or channel to the system. This is of course its ID, which in both cases is a long word of data. This long word can be further split up into two words, the first of which bears a relationship to the number of channels opened or jobs created since the machine was last reset, and the lower word of which has something to do with the actual number of jobs or channels currently present.

The SuperBasic interpreter has an ID of zero, and we can never change this – we can't remove the interpreter from the job list and then recreate it later to give it another job ID. The next job to be created will be the given the ID $00000001, which corresponds to a job number of 1 and a 'tag' of zero. If we created another job we would find that its ID is $00010002, which means that it is job number two with a tag of 1. If we then killed our first job and loaded in another, this last job would be given a job ID of $00020001, as the tag still rises but the job number is allocated according to the number of jobs in the machine at the moment.

Later on we'll find that it is often useful to talk of a job in terms of its job number and its tag, as these two numbers individually are a lot easier to remember than the entire long word ID – especially if we choose to represent it in the rather more natural decimal.

An awful lot of the QDOS routines require a timeout. This is simply a measure of the time that the processor will spend doing the job of the particular trap before it returns to the calling program with failure or partial success. If the timeout is zero then the routine will, in most cases, try to do what it can. For example if we called a trap to collect bytes from a channel with zero timeout then it would collect as many bytes as there are to get. If we called the routine with infinite timeout (ie –1) then the trap would not return until it has either finished completely or had an error condition. The only real exception to this is MT_ACTIV, which has only two valid timeout values – 0 and –1. A timeout of zero in this case activates a job and resumes, while a timeout of –1 activates a job and waits for it to finish. The upshot of this is that the two most common timeouts that we are likely to use are 0 and –1. Naturally most of the routines which accept timeouts become non-atomic if the timeout is not zero.

The printer spooler

Now for our program. This is a very simple printer spooler routine which allows you to print a document (or indeed copy a file to any other device or file) while other operations such as a Basic program are going on. By setting the priority of this job to 1 we ensure that it is the most background of background tasks and is therefore unable to alter the speed of the SuperBasic interpreter very much. If on the other hand we wanted the print to continue as the foreground task whilst Basic or whatever proceeded as something less important, all we need to do is change the priority to some other value. Remember that until we do something about it the SuperBasic interpreter and all jobs activated by it have a priority of 32.

"The routine allows you to print a document as a background or foreground task".

For the purposes of this article we must assume that you have an assembler, and in fact if you have the Metacomco assembler then you will not need to change any of the source. Other assemblers follow slightly different conventions, particularly in the area of assembler directives, so check your assembler documentation carefully.

The file mdv1_header_asm **(Listing 1)** is first included in the assembly. This file just consists of all the QDOS declarations, keys and equates which we use later on. We then declare the data size of the program as being 150 bytes. If your assembler doesn't have a directive like this, the best thing to do is to load

the assembled code into a safe area of RAM and then resave it using SEXEC.

The program itself starts at the BUF_LEN declaration. The symbol BUF_LEN is 100 and the symbol HEAP_ROOM is 4096; we'll be using these a little later. The first instruction in the program is a BRA which jumps over a few bytes which declare the job as being in standard QDOS format (ie the word starting at the 6th byte is $4AFB and this is followed by the job name).

The main code starts at START_P. The MT_PRIOR trap is used to set the priority of the current job (indicated by a job ID of –1) to 1. We then use one of the thoroughly useful utility routines to open a console device. The various parameters of the window of this device are declared in a parameter block further down in memory called PBLOCK. UT_CON calls IO_OPEN and returns the ID of the newly opened channel in A0. If there was an error then D0 is non-zero, so we test for this. Assuming everything is OK we then print a message on our console which says 'Print file: '. This is a message to the user telling him to type in the name of the file which he wishes to print. To collect the filename we use the IO_FLINE trap, which is very useful as it allows the line to be edited before ENTER is pressed, in just the same way as in a line of Basic. The program's data space starts at BUFFER, to collect the filename.

"At any one time I've had up to 61 jobs running on a machine simultaneously".

When IO_FLINE returns D1 contains the number of characters collected, including the line feed which terminated the input. We're going to use the filename in a call to IO_OPEN, so we first have to get the string into QDOS format. A QDOS string is a string of characters preceded by a word containing the character count; a word is reserved at BUF_POS where we can put the string length. The first thing to do is decrease the length of the string by 1, as we don't want to send the line feed to IO_OPEN. We now store this string length in BUF_POS and call IO_OPEN to open the file. Notice that D3 contains the code for open or shared read, so that other programs can examine (but not write to) the file at the same time as this one.

If all succeeds here then save all the channel IDs on the stack and print another message, 'Printer device: '. This is a prompt for the channel on which we have the printer. We could enter ser1c, as I do for my Epson, or mdv1_dump to save the text to a file called mdv1_dump. Any valid QL device which supports output can be entered in response to this prompt. We again use IO_FLINE to read in the new string, taking the same precautions as before. Once a valid filename is obtained it is opened for exclusive output and the console closed as it is no longer needed.

The next bit is rather flash. The area between the system variables and the transient program area is called the common heap space, and is available for use by any job that needs space. So we ask for 4096 bytes of it. If we don't get it, we divide the amount we're asking for by 2 until eventually we either get enough room or discover that there is absolutely none available. In the latter instance we just kill the job and return to Basic. If we can get the room then we enter a loop in which data is read from the input file using IO_FSTRG and sent to the output file using IO_SSTRG. IO_FSTRG is very similar to IO_FLINE except that we tell it how many bytes we want to collect, and a line feed is not taken as a terminator. IO_SSTRG is exactly its converse – it sends a predetermined number of bytes to a channel. When IO_FSTRG has reached the end of the input file, it returns End Of File, but nothing is done about that until AFTER the IO_SSTRG call as there may be some residual bytes to send.

When all is well and the end of file has been reached, we close our two files and kill the job. Killing a job requires a call to MT_FRJOB ('force release job'), which removes the specified job and any subsidiaries from the job table, making the space which they occupy available to other jobs. The act of killing a job also releases any common heap space which it might have owned.

When this program has been assembled and saved to drive, possibly with the filename print_exec, it can be loaded and run at any time simply by typing EXEC mdv1_print_exec and pressing ENTER. To switch between active cursors on the screen, press CTRL-C. This will circulate around all the jobs awaiting keyboard input, but as you're only likely to have SuperBasic and this job resident when you first try it out, you only need one press of CTRL-C to switch between the two jobs.

Further issues of *E&CM* will contain many examples of multi-tasking QL programs, but remember the cardinal rule – ANY program which is self contained and does not require to do anything naughty, such as clearing the whole of RAM, will multi-task. It really is that simple.

Important things to remember are that when a job is first activated registers A4, A5 and A6 are set up to particularly useful values. A6 holds the address of the start of the job, (A6,A4.L) points to the start of the data area and (A6,A5.L) points to the end of the data area – that is, the end of the job. Also, remember that each job has its stack at the top of its data area – and this grows downwards!

When there are a large number of jobs running in a machine at any one time (and I've had 61 so far!), it would be useful to be able to have some degree of control over them from Basic. I've just written five procedures and one function to do just this, and they'll all appear in the next issue of *E&CM*.

"Any program which is self contained and does not require anything naughty, such as clearing the whole of RAM, will multitask".