48K Spectrum

RAMDOS

Random Access
Micro Drive
Operating
System

ROYBOT

RANDOS - RANDOM ACCESS MICRO DRIVE OPERATING SYSTEM

CONTENTS

ROYBOT, 45 Hullbridge Road, RAYLEIGH, SS6 9NL
Telephone 0268 771663

RAMDOS - RANDOM ACCESS MICRO DRIVE OPERATING SYSTEM

RAMDOS provides extensions to BASIC for microdrive file handling, error recovery and text processing. This package comprises the RAMDOS machine code, a description of each facility and three programs to show how to use the main extensions provided for business type processing. A second package, the ROYBOT FREE TEXT DATABASE, uses the extensions to provide a general purpose home information system and a third , the RAMDOS UTILITIES, for microdrive management and error recovery.

Appendix 1 gives details of each RAMDOS function, entered in program format to ensure that the syntax is correct. To make the statements similar to Spectrum BASIC, keywords with a resemblance to the desired function have been used: a list of these is given at the end of the appendix. Appendices 2 to 4 give listings of the demonstration programs.

It is assumed that users of this software have a reasonable knowledge of BASIC and the normal microdrive functions.

COPYING THE SOFTWARE

The first step on obtaining the software is to make a back-up copy on a new cartridge. In order to make use of the RAMDOS foolproof mechanism for using multiple cartridges on one drive, each cartridge should be formatted with a unique name e.g. cart1, cart2: this should not be "RAMDOS", which is the name of the cartridge supplied.

1. Insert the ROYBOT cartridge in drive 1 and type LOAD *"m";1;"copy" to load the copy program. The listing for this is given in Appendix 2.

2. On entering RUN, the RAMDOS machine code is loaded and initialised (lines 30 to 40).

3. The next step is to select drives for the two cartridges and to open them to read the catalogs (lines 80 to 190): a description of these activities is given for demonstration 1.

4. The files are then copied from RAMDOS to the new cartridge (line 240). If one drive is being used, RAMDOS issues instructions to change cartridges at the appropriate times for both copying and verifying, the final step (line 280).

OTHER COPY FUNCTIONS

Other copy functions also require the cartridges to be open. To make a further copy of of the first demonstration program as a "run" file, enter GOTO 320. Lines 330 to 340 copy and verify the file.

Tape back-up copies of files can be made in the special RAMDOS format, which copies large files in parts, depending on the buffer size specified (b$ line 230). To produce a tape copy of the first program, enter GOTO 380, starting the tape for saving when requested (line 390). After saving, "start tape" and "Comp" (for compare) apppear on the screen - rewind the tape and start for reading and verification (line 400).

Restoring a tape file can be carried out in a similar manner by specifying the file name or, as at line 410, which indicates "read the next tape file and save it on the microdrive using the same name". Rewind the tape after the above verification, press ENTER and start the tape for reading. This demonstrates another RAMDOS feature, as the file read is already on the cartridge - either press ENTER to cancel the copy or input a new name.

A cartridge can be dumped to tape (10 minutes for a full cartridge) in the same sort of way. In this case, the catalog is copied first. Enter GOTO 450 to dump the new cartridge. On dumping, the start tape message is given to help to ensure that there is a gap between files so that, on reading or verifying, the tape can be stopped as the microdrive searches for a file. Lines 460 and 470 dump and verify the cartridge. A whole cartridge can be restored as at line 480, where existing files can again be ignored. The catalog at the start of the tape can be read into an array by the TAB CAT TO t$ function and displayed or printed as shown in demonstration 1.

Besides using the copy functions for back-up purposes, a wise precaution is to save RAMDOS and a variation of the copy program on tape, in the normal format. To do this, see line 610 in the copy program.

ERRORS

If the program stops with the usual microdrive error messages or through a verification failure, either delete the offending file or reformat the cartridge. Detailed treatment of errors is outside the scope of this package but is fully covered in the ROYBOT RAMDOS UTILITIES. However, for the ambitious, Appendix 1 gives sufficient information for an error evaluation program to be produced. The RAMDOS features for this are:

1. a catalog of the status of each sector on the tape (line 150);

2. using a variation of random write to test any sector (line 310);

3. duplexing records or files (line 37), used as for serial write;

4. busying or freeing any sector (lines 410/420);

5. reading any sector and updating the sector catalog (line 440) or reading file records, both ignoring the normal error checking (line 460);

6. converting what can be read into a valid format (lines 900 to 960).

ERASING FILES

Files can be erased using the normal ERASE or via a special RAMDOS function, which also deletes the entry from the stored catalog. Random and duplexed files need erasing twice. The RAMDOS erase function is shown in the copy program at line 520. This is for a special case where formatting creates a spurious file record, the name starting with CHR$ 0 (shown as "?" in the RAMDOS catalog). This does not appear in the normal catalog and the feature can be used for creating "invisible files". This erase is included here as, on the rare occasions that spurious files are created, a "file not found" error may occur on copying cartridges.

DEMONSTRATION 1 (LOAD *"m";1;"demo1")

The first step for using the BASIC extensions is to load the RAMDOS machine code, without this the new lines of code cannot be typed in or run. Lines 10 to 30 in demo1 show how this is done.

CATALOG

Besides not being able to handle random access files, the major disadvantages of the Spectrum microdrive software include the inability to use multiple cartridges on one drive, to have files on more than one cartridge, to extend existing files and program stoppages due to "file not found", "writing to a read file", "microdrive full" etc. In order to find a way around these problems, RAMDOS requires the catalogs to be in memory.

So that the file handling statements can find the information, the first variables declared MUST be arrays for the catalogs. Demo1 uses DIM c$(1000) for this purpose. The array has to be large enough for information on all files - 17 bytes each, plus 17 for the cartridge name etc. Thus the example given can accommodate 57 files: specially designed databases can have quite small catalogs. For two cartridges DIM c$(2,1000) or DIM c$(1000): DIM e$(1000) could be used.

The first RAMDOS type statement is to open the cartridge (line 130) on a unique stream. Demo1 closes it first (line 120) to avoid "stream already open" when the program is rerun. Unlike the standard software, a stream is then open for all files on a cartridge, not just one.

The first byte in the array gives the number of entries in binary (line 170) - the remainder are printable (line 230). Bytes 2 to 11 contain the cartridge name (line 180): this is used by RAMDOS to ensure that the correct cartridge is in a drive before reading or writing. Bytes 13 to 15 indicate the number of free sectors (line 190): RAMDOS updates this and file sizes when changes are made.

Starting at byte 18, 35 etc., each file entry has bytes 1 to 10 as the name, 12 to 14 as file size in sectors and 16 as file type identifier (B-BASIC, C-code, A-array, $-character array, D-data). The catalog is printed (lines 170 to 230).

Sometimes it may be necessary to access or manipulate the catalog from BASIC, so demo1 shows various ways of doing it: this also illustrates RAMDOS text processing capabilities on typical tables.

FIND LINE (demo1 290 to 340)

This shows finding any line in a table from a constant starting point and to get the line in a variable (a$), in this case to display the catalog.

FIND ARRAY ADDRESS OF START OF LINE (350 to 410)

This time line 1 is specified each time, the line is obtained in a$ and start indicates the start address of the next line. The catalog is again dislayed.

SEARCH TABLE FOR FILE NAME (430 to 542)

A file name has to be typed in and the first 10 columns of each entry are scanned for a match. If the file is found (line 530), the file length is displayed. On exit, file "newfile", which is used later, is searched for and, if it has been written its size is extracted (line 542).

TABLE SORT

Lines 570 to 610 sort the catalog by size and display it; 620 to 660 sort by file type and 670 to 710 by file name.

RAMDOS FILES

RAMDOS files have physical record sizes of 512 bytes, corresponding to the microdrive sector size. Logical records can be any size, determined by the BASIC program. The number of records can be obtained from the catalog but it may be necessary to keep a count of the number of entries or length of text. The latter can be in the first few elements of arrays saved or written as separate records or files.

The first records written for serial and random files are written as serial files (lines 730 to 800). The RAMDOS machine code searches the catalog: new files must start with record 1 and the file name should not be in the catalog. If it is in the catalog you are given the opportunity to type in a new one - used in RAMDOS copy (on running demo1 for a second time this option will be demonstrated - press ENTER to ignore). Files are extended as serial and the first record extension must correspond to the next catalog record. If the rules are not followed "parameter error" is indicated.

At line 760, data is entered in 4 records. Line 780 sets the channel to indicate data and the 4 records are written at 790. Lines 810 to 850 show that the catalog has been updated.

DIFFERENT CARTRIDGES

Line 870 temporarily changes the cartridge name to show the effects of the wrong cartridge being in the drive required for reading: in this case, press c to cancel ("Break into program, 880:1" indicated). Press CONT to continue at line 900. The ability to change the cartridge name (and file details) provides the mechanism to extend databases over a number of cartridges.

READ SERIAL FILE AND EXTEND

Lines 860 to 900 demonstrate reading of the 4 records written earlier: this should take about 7 seconds.

The file "newfile" is extended by 4 records at lines 940 to 1000 and the catalog displayed again. This time, for reasons explained later, the records are written one at a time. To avoid an error stoppage, if the program is rerun, this part is missed out via line 948

RANDOM READ

Once written, the 512 byte records can be read in any order, the main constraint for the demonstration being the time taken. Lines 1010 to 1120 read records 2, 4 and 6 (skipped sequential) to illustrate the timing problem. During normal writing, as records 1 to 4, the software is likely to write in every other sector on the tape. Besides the motor start/stop time, RANDOS adds a further delay. Thus, when records are written individually, as 5 to 8, they are placed 11 to 12 sectors apart and, on being read one at a time, sequentially, they will be picked up quickly. Reading records 2 and 4 will probably take 7 seconds each, then record 6 in less than a second. With the RANDOS technique, the records become more randomised as, after 15 have been written, record 1 position will have been reached again.

RANDOM WRITING

In order to update a record, it is necessary to know the physical sector number on the tape. To do this, firstly the file is assigned to a pointer array (line 1140) and, secondly a random read format is used to read the data and set the pointers (line 1150). The pointers are displayed at line 1100, via subroutine 2010 which extracts the data from the pointer array. Note that sector numbers are in reverse order to record numbers and note the spacing between sectors.

Two bytes are obtained for each record, the sector number and a flag which initially contains the stream on which reading took place (see LARGE FILES). The main use of the flag byte is for identifying the records to be updated, where bit 7 has to be set (128 added). The reason for this is to enable skipped sequential updating in the fastest time. This is demonstrated in lines 1210 to 1240. Line 1210 inserts extra data in records 5 and 2 via 2100 and changes the flag pointers via 2040. Line 1215 puts data in record 3 but does not change the flag. The update is carried out at 1220. Afterwards the flags are reset via 2070. The other use of the flag is for error conditions, where the sector to update is not found - bit 7 is set (64 added). This is tested at line 2080.

Records can be updated one at a time by inserting the data to be written in a 512 byte array and by specifying this and the record number in the write statement.

The file is re-read and displayed at lines 1252 to 1280. Note that the extra data for record 3 is not included.

FILE STRUCTURE

There are many options for designing the file structure and considerations should include how much data can be in memory at the same time and the read, add, update ratios (bearing in mind the 7 second delay of the tape loop).

For example, a name and address file, all on one cartridge, could have 1500 variable length entries. With no more than 100 for one index letter, about 6000 bytes in memory and 12 cartridge sectors would be the maximum requirement. Records 1 to 26 could be pre-wrtten, one for each letter with the first 4 bytes set aside for data length and the next 36 for the number of any additional data records, entered when the file is extended. With all entries in memory, a new one can be inserted in the appropriate place. Bearing in mind the records have just been read, the writing procedure should be - write any additional records, rewrite the first record index (if changed), rewrite records with changes in and following ones if affected by insert or delete, then, optionally, verify the newly written records.

Another approach, for the same problem, is to simply add the new address to the end of the last record and maintain an index of names e.g. average length 18 bytes (12 for the name, 3 for the record, 3 for the position in the record). With the index in memory, and using RAMDOS search, this method will be faster for unusual names where the first letter is in doubt.

Sometimes data may be saved and be required to be updated shortly afterwards. In this case, the records will have to be re-read to get the sector pointer. In other cases, it may be desirable to write particular records without reading them first. For this, a large number of records could be written first as blank ones, the file read to obtain the sector pointers, the pointer array saved as the record 1 and this read first on running the program.

LARGE FILES

Microdrive record numbers can be up to 255. A series of files of this size can be written over a number of cartridges. For example, if a cartridge is full when a file is 100 records long, change the cartridge name in the catalog to that of a new one then write record 101: RAMDOS will tell you to change the cartridge. (the catalog of the second one will indicate a file length of 101). The program and an index will need to identify this arrangement. When the program is re-run and the first cartridge opened, the file size and cartridge name will need changing in the catalog for further extensions. If more than one drive is available, the cartridges can be opened on different ones as different streams. When the appropriate records are read for random updating, the stream number is included in the pointer array. On updating, the records must be written one at a time and the stream number can be extracted from the pointer array, if required.

Larger files can be produced, in a similar manner, by changing the file name (e.g. data1, data2) and the file sizes in the catalog, as appropriate. This, of course, needs careful control by the program.

ERASING FILES

Serial files, which are not extended, follow the standard conventions. When files are extended, more than one end of file marker will be written. This does not affect RAMDOS operation but erase may have to be carried out twice to delete a file completely.

DEMONSTRATION 2

This shows display, input and editing of forms, then display, input and editing of text.

To load, type in: LOAD *"m";1;"demo2"

In this case, RAMDOS machine code is loaded at line 8000.

FORMS DESIGN AND DATA ENTRY

On loading, a blank form is displayed. This is produced by a subroutine at line 1500, which can handle both blank and completed forms. The data for the blank form is inserted at lines 120 to 150. The form has 6 lines of 24 characters, one of 1, one of 3 and one of 128. Data is typed in, left and right cursor keys can be used (and up/down in the longer entry, when more than 32 characters are in). Delete can also be used and data is inserted at the cursor position. To end one line, press ENTER with the cursor anywhere in the line.

Lines 170 to 230 select the screen line and column and maximum length of an entry, then call subroutine 2009 for the screen edit function. An ENTER character is added to the line at 2020: this would not be required for fixed length entries but even the first 6 lines are treated as variable, with ENTER identifying the end. For later use, a maximum length of line has to be defined - lwide at 170.

The data is added to the end of array d$ at line 2050. An important aspect of the data array is the initial insertion of an end of text marker (CHR$ 136) at line 170.

When the last line has been typed in, the data is displayed at line 280.

EXTRACT DATA

Data is extracted from array d$ at lines 310 to 410, using the RAMDOS find function, and put in the same variables as used for the blank form. The actual length is also noted in array l. The completed form is displayed at line 410 via subroutine 1500.

SELECT LINE FOR EDITING

Normal BASIC functions are used to select any line from the form that needs changing at 420 to 550, using the up and down cursor keys, ENTER to select for changing and EDIT when all are satisfactory.

EDIT LINE (560 to 690)

One of 590 to 620 is selected according to the line to be edited, and subroutine 650 called, where the screen edit function is used. The old line is deleted, using the delete function, at 670 (the contents are saved in t$ for reinsertion, if required). The new data is inserted at 680, using the RAMDOS insert function.

After the line has been changed, press ENTER to return to the line selection routine. Changed lines are highlighted via the edit function attributes parameter at 650. The value of attr can be obtained by e.g. typing in PAPER 1:INK 7:CLS:PRINT PEEK 23693 whilst in BASIC entry mode.

At the end of editing, the data is displayed again at line 700.

DISPLAYING TEXT (720 to 900)

Information is inserted in array d$ at line 740 and also the end of text marker. The following routines show one use of this marker, that is in identifying the last line to display.

With o$(1) initially blank, each line of text is extracted at 770 and displayed at 780, left justified. The output array can also be printed using LPRINT.

The display is repeated at 810 to 860, using a different line width and left/right justification, governed by o$(1) being initially "R" for the find function at 830.

EDIT TEXT

Free format text is edited in a way that shows end of line characters and allows a number of lines to be typed in one edit, if required.

Whilst displaying the text, the start of each line is noted in array s at 820. The last line is selected for editing at 920 and is deleted from the data array at 930, with the deleted line in o$ and length as dlen. If an end of line (or end of text) character is found, dline is returned as 0 and the end of line indicator (CHR$ 137) should be inserted in the editing array as at 940.

The editing array t$ is displayed, using the edit function, at screen line 12, column 1 (see 960). This time a$(1) is "I" to indicate that the end of line characters will be displayed and exit from the routine is by pressing EDIT. The box can be filled with data or the chosen line changed.

The length of the line edited is dlen+1 (see 940), to allow the cursor to go past the last character for deletion (for adding new data, o$ would be blank and elen=1). In the demonstration, move the cursor to the end of the line and type 2 or 3 characters then ENTER, repeating to produce 10 short lines.

If the text had been deleted at 960, elen would be returned as less than 2: in this case, the following insert section would not be carried out.

Line 970 displays the data as edited. This is then passed to variable string j$ at 990 and inserted at the point of deletion by the RAMDOS insert function at 1000.

Lines 1010 to 1060 display the data again, this time 10 characters wide. If you keep pressing ENTER, the text scrolls down to the end and then scrolls up, a varying number of lines via the scroll up function at 1090.

```
100>REM : RAMDOS FUNCTIONS
110 REM :
115 REM :     OPEN
116 REM :
120 IN dr; OPEN #st;CHR$ ; CAT  TO c$
130 REM : Open cartridge in drive dr, as stream st. Put catalog
131 REM : in array c$. c$ MUST be first variable declared with
132 REM : DIM at least 17 + 17 * number of files. CODE c$(1) =
133 REM : number of files; c$(2 TO 11) = cartridge name; c$(13
134 REM : TO 15) = free sectors; file 1 - c$(18 TO 27) = file
135 REM : name; c$(29 TO 31) = length in sectors; c$(33) = file
136 REM : type - B BASIC, C CODE, A Array, $ Character array,
137 REM : D Data; file 2 - c$(35 TO 44) = file name etc.
140 REM :
150 IN dr; OPEN #st;CHR$ ; CAT  TO c$; DATA d$
160 REM : As OPEN but details of sectors and status to d$.
161 REM : DIM d$(17*256); starting at 18 for sector 1 - d$(18)
162 REM : = Y or N header checksum correct or not; d$(19) = V
163 REM : valid busy sector, I invalid, F free; d$(20) = E end
164 REM : of file; d$(21) = file type as catalog; CODE d$(22) =
165 REM : record number 0 to 255; CODE d$(23)+256*CODE d$(24) =
166 REM : record length; d$(25 TO 34) = file name; sector 2 35+
170 REM :
171 REM :    CLOSE & ERASE
172 REM :
180  OUT #st; CLOSE #
190 REM : Close stream st. Use instead of CLEAR #. Do not use
191 REM : CLOSE #st (it saves an end of file record).
199 REM :
200  OUT #st; ERASE f$
210 REM : Erase file f$. Use instead of ERASE "m";1;f$ to
211 REM : erase from CAT in c$.
212 REM : RAMDOS and duplexed files may need erasing twice.
220 REM :
223 REM :    SET FILE TYPE
225 REM :
230  OUT #st;MD:  REM data
240  OUT #st;MP:  REM program or code
250 REM : Set before writing; not required on COPY.
260 REM :
261 REM :    READ & WRITE SERIAL FILE
265 REM :
270 IN #st;f$;Rr1,r2;BIN  TO b$
275 REM :
280  OUT #st;f$;Rr1,r2;BIN  TO b$
290 REM : Read or write cartridge opened on stream st, file f$,
291 REM : 512 character records r1 to r2, from or to buffer
292 REM : input array b$. Writing new files r1 should be 1.
293 REM : Extending a file r1 should be existing length + 1.
300 REM :
301 REM :    RANDOM FILES
305 REM :
310 POINT f$ TO p$
320 REM : First assign file f$ to pointer array p$ (DIM >30).
```

```
321>REM ! p$(1 TO 10) hold the name f$, then 2 bytes are used
322 REM ! for each record. On reading, the first byte gets the
323 REM ! sector number and the second gets the stream number.
324 REM ! On writing, the latter needs bit 7 setting (add 128)
325 REM ! to update a record; it also returns with bit 6 set
326 REM ! if a sector is not found. See detailed instructions.
329 REM !
330 IN #st;RND f$;Rr1,r2;BIN  TO b$;POINT  TO p$
335 REM !
340  OUT #st;RND f$;Rr1,r2;BIN  TO b$;POINT  TO p$
350 REM ! Read or write random file. Only existing records can
351 REM ! be written. Use serial write to extend a file then
352 REM ! random read to get sector pointer.
360 REM !
361 REM !      SPECIAL WRITE/READ
362 REM !
370  OUT #st;f$;D Rr1,r2;BIN  TO b$
380 REM ! Duplex records r1 to r2.
385 REM !
410  OUT #st;SQR Ss1,s2;F
420  OUT #st;SQR Ss1,s2;B
430 REM ! Free or busy sectors s1 to s2, usually one at a time.
435 REM !
440 IN #st; Ss1,s2;BIN  TO b$; CAT  DATA d$
450 REM ! Read sectors s1 to s2, put status in d$ (see OPEN).
455 REM !
460 IN #st;SQR f$; Rr1,r2;BIN  TO b$;POINT  TO p$
470 REM ! Sector query, read file f$, records r1 to r2, indic-
471 REM ! ate sumcheck errors in p$ - set bit 5 in byte 2; also
472 REM ! byte 1, sector number to 0 if not found.
475 REM !
480 REM !      COPY DATA (CODE), COMPARE (COS)
485 REM !
490 CODE #st1; f$  TO #st2; g$ BIN  TO b$
500 COS #st1; f$  TO #st2; g$ BIN  TO b$
510 REM ! Copy/compare file f$, cartridge open as stream st1 as
511 REM ! file g$ on cartridge open as st2 - can be on same
512 REM ! drive. DIM b$(512*n) for data buffer, n as large as
513 REM ! possible but not dependent on file sizes.
515 REM !
520 CODE #st1; CHR$   TO #st2; BIN  TO b$
530 COS #st1; CHR$   TO #st2; BIN  TO b$
540 REM ! Copy/compare all files on two cartridges.
545 REM !
550 REM !      TAPE BACK-UP (TAB)
551 REM !
560 CODE #st1; f$  TO TAB  g$ BIN  TO b$
570 CODE TAB  f$  TO #st2; g$ BIN  TO b$
580 COS TAB  f$  TO #st2; g$ BIN  TO b$
590 REM Copy/compare file to/from cassette tape - RAMDOS format.
595 REM !
600 CODE #st1; CHR$   TO TAB g$ BIN  TO b$
610 CODE TAB  f$  TO #st2; CHR$ BIN  TO b$
620 COS TAB  f$  TO #st2; CHR$ BIN  TO b$
```

```
630>REM : Copy/compare cartridge to/from cassette tape.
635 REM :
640 TAB  CAT  TO t$
650 REM : Tape catalog to array t$. This is copied to the start
651 REM : of the tape with CODE TAB.
660 REM :
662 REM :     ACCESS TEXT (AT)
663 REM :
670 REM : parameters MUST be variables, not expressions or
671 REM : values as some variables are updated by RAMDOS.
680 REM :
681 REM :     FIND LINE OF TEXT
682 REM :
690 AT t$; st, ed, F, ln, len, o$
700 REM : Find line ln in array t$, line 1 starting at t$(st),
701 REM : end is at t$(ed) which is graphics character CHR$
702 REM : 136; DIM o$(lwide) where lwide is maximum length of
703 REM : line; line is returned in o$ (excluding ENTER chara
704 REM : if one in line), length of line up to any ENTER chara
705 REM : returned as len, start of next line as st, ln as 1.
707 REM : Initial values >0. Initial value of o$(1)=" ", line
708 REM : returned left justified; o$(1)="R" returned left and
709 REM : right justified.
710 REM :
712 REM :     EDIT LINE
713 REM :
715 AT i$; dline, dcol, E, len, attr, a$
716 REM : Edit line, maximum length of data defined by DIM i$,
717 REM : at screen line dline and column dcol; initial length
718 REM : of data len, temporary colour attributes attr. New
719 REM : length of data returned as len. Initial a$(1)=" ",
720 REM : press ENTER to end edit anywhere in data, no ENTER
722 REM : chara included. Intial a$(1)="R", press ENTER to end
723 REM : edit, ENTER inserted at cursor position. Initial
730 REM : a$(1)="I", press EDIT to end edit, ENTER inserted as
733 REM : graphics chara 137 on screen but as ENTER chara on exit.
740 REM :
741 REM :     DELETE, INSERT LINE
742 REM :
750 AT t$; start, end, D, dline, dlen, a$
760 REM : Delete (see FIND) line dline from array t$, line 1 at
761 REM : t$(start), maximum length DIM of a$. Deleted line
762 REM : returned in a$, length as dlen, dline as 0 if ENTER
763 REM : chara was found; new end returned.
765 REM :
770 AT t$; start, end, I, iline, lwide, j$
780 REM : Insert data from variable string j$ in array t$, at
781 REM : line iline, line 1 is t$(start), lwide is maximum
782 REM : line length. New end returned.
790 REM :
791 REM :     SCROLL UP
792 REM :
800 AT t$; start, end, U, up, lwide, a$
810 REM : Go back up lines, return new start.
```

must DIM $(512)

```
820>REM :
821 REM :    SORT TABLE
822 REM :
830 AT t$; start, noe, S, rwidth, stcol, a$
840 REM : Sort table in t$, starting at t$(start), field width
841 REM : to sort on defined by DIM a$, number of entries to
842 REM : sort=noe, record width rwidth, starting column stcol.
850 REM :
852 REM :    SEARCH/QUERY TEXT OR TABLE
853 REM :
860 AT t$; start, noe, Q, rwidth, stcol, h$
861 REM : Parameters as sort except h$ is variable text to
862 REM : find. On return stcol=0 if not found. If found stcol
870 REM : is number of entry containing match. For text, noe is
871 REM : length from current value of start, rwidth and stcol
872 REM : should equal 1. On return stcol is starting position
873 REM : of match.
880 REM :
882 REM :     RECOVERY ACCESS
883 REM :
890 AT t$; stblock, noblk, B
900 REM : Convert noblk 512 character blocks, starting at block
901 REM : stblock, into a valid BASIC program format.
905 REM :
910 AT t$; stblock, noblk, M
920 REM : Convert to machine code saving format.
925 REM :
930 AT t$; stblock, noblk, A
940 REM : Convert to printable (ASCII) data.
945 REM :
950 AT t$; stblock, noblk, P
960 REM : Display as BASIC lines.
961 REM :
962 REM :     SPECTRUM KEY WORDS USED
963 REM :
970 REM : KEYWORD  MODE  SHIFT  KEY  MEANING
971 REM :
972 REM : AT        K     SYMBOL  I   Access Text
974 REM : BIN       E             B   Buffer Input
976 REM : CAT       E     SYMBOL  9   Catalog
978 REM : CHR$      E             U   Cartridge
979 REM : CLOSE #   E     SYMBOL  5   Close stream
980 REM : CODE      E             I   Copy Data
981 REM : COS       E             W   Compare Sectors
982 REM : DATA      E             D   Data for sector status
983 REM : ERASE     E     SYMBOL  7   Erase file
985 REM : IN        E     SYMBOL  I   Input drive or stream
986 REM : OPEN #    E     SYMBOL  4   Open cartridge
987 REM : OUT       E     SYMBOL  O   Output drive or stream
988 REM : POINT     E     SYMBOL  8   Point to sector
989 REM : RND       E             T   Random file
990 REM : SQR       E             H   Sector query
991 REM : TAB       E             P   Tape Back up
993 REM : TO        K     SYMBOL  F   To
```

APPENDIX 2

```
  10 REM Load machine code and initialise
  20 REM
  30 CLEAR 57599: CLS : PRINT "Loading machine code"
  40 LOAD *"m";1;"scode"CODE : RANDOMIZE USR 57600: RANDOMIZE USR 57607
  45 CLEAR : DIM c$(2,1000): REM catalogs
  50 REM
  60 REM Select drives
  70 REM
  80 INPUT "Input drive number for ROYBOT   cartridge ";drive1
  90 REM
 100 REM Open ROYBOT cartridge
 110 REM
 120 CLS : PRINT "RAMDOS": PRINT : PRINT "Insert cartridge in drive ";driv
e1;".    Press ENTER when done.": INPUT a$
 125  OUT #4; CLOSE #
 130 IN drive1; OPEN #4;CHR$ ; CAT  TO c$(1)
 140 REM
 150 REM Open new cartridge
 160 REM
 170 CLS : INPUT "Input drive number for new      cartridge ";drive2
 180 CLS : PRINT "New cartridge": PRINT : PRINT "Insert cartridge in drive
";drive2;".    Press ENTER when done.": INPUT a$
 185  OUT #5; CLOSE #
 190 IN drive2; OPEN #5;CHR$ ; CAT  TO c$(2)
 200 REM
 210 REM Copy cartridge
 220 REM
 230 DIM b$(512*40): REM data buffer
 240 CODE #4;CHR$  TO #5; BIN  TO b$
 250 REM
 260 REM Verify
 270 REM
 280 COS #4;CHR$  TO #5; BIN  TO b$
 290 STOP
 310 REM Copy file and verify
 320 CLS
 330 CODE #4;"copy" TO #5;"run" BIN  TO b$
 340 COS #4;"copy" TO #5;"run" BIN  TO b$
 350 STOP
 370 REM Dump file to tape, verify and restore
 380 CLS
 390 CODE #5;"copy" TO TAB "copy" BIN  TO b$
 400 COS TAB "copy" TO #5;"copy" BIN  TO b$
 410 INPUT "Press ENTER ";a$: CODE TAB "" TO #5;"=" BIN  TO b$
 420 STOP
 440 REM Dump cartridge to tape, verify and restore
 450 CLS
 460 CODE #5;CHR$  TO TAB "copycart" BIN  TO b$
 470 COS TAB "" TO #5;CHR$  BIN  TO b$
 480 INPUT "Press ENTER ";a$: CODE TAB "" TO #5;CHR$  BIN  TO b$
 490 STOP
 500 REM Erase invisible file
 510 REM
 520 LET f$=c$(2,18 TO 27): OUT #5; ERASE f$
 530 STOP
 600 REM Save RAMDOS and copy program
 610 SAVE "RAMDOS" CODE 57600,7936: CLEAR : SAVE "COPY": CLS : PRINT "REWI
ND TO VERIFY": VERIFY "RAMDOS" CODE : VERIFY "COPY": STOP
```

```
  3>REM Load machine code and initialise
  4 REM
 10 CLEAR 57599: CLS : PRINT "Load machine code"
 30 LOAD *"m";1;"scode"CODE : RANDOMIZE USR 57600
 40 REM
 50 REM First variable MUST be array for catalog
 60 REM
 70 DIM c$(1000)
 80 REM
 85 CLS : PRINT "Open cartridge"
 90 REM Open cartridge (close first)
100 REM
110 LET drive=1: LET stream=4
120 OUT #stream; CLOSE #
130 IN drive; OPEN #stream; CHR$ ; CAT  TO c$
140 REM
150 REM Catalog entries
160 REM
170 LET entries=CODE c$(1)
180 LET n$=c$(2 TO 11)
190 LET free=VAL c$(13 TO 15)
200 CLS : PRINT "Catalog": PRINT : PRINT "Cartridge    ";n$
210 PRINT "Entries       ";entries
220 PRINT "Free sectors ";free
230 LET end=17*(entries+1)+1: PRINT : PRINT c$(18 TO end)
240 INPUT "Press ENTER ";j$
250 REM
260 REM Access Text extensions
270 REM Variable names MUST be used
280 REM
285 REM Find line
286 REM
290 LET len=1: DIM a$(17)
295 CLS : PRINT "Catalog using Access Text Line": PRINT
300 FOR l=1 TO entries: LET start=18: LET line=l
310 AT c$;start,end,F,line,len,a$
320 PRINT a$
330 NEXT l
340 INPUT "Press ENTER ";j$
350 CLS : PRINT "Catalog using Access Text Start": PRINT
360 LET c$(end)=CHR$ 136
370 LET start=18: LET line=1
380 AT c$;start,end,F,line,len,a$
390 IF a$(1)=CHR$ 136 THEN  GO TO 410
400 PRINT a$: GO TO 380
410 INPUT "Press ENTER ";j$
420 REM
430 REM Search text
440 REM
450 LET name=10
460 CLS : PRINT "Search"
470 PRINT : PRINT c$(18 TO end)
480 LET start=18: LET rwidth=17: LET stcol=1
490 DIM f$(10): INPUT "Input file name. Just ENTER end ";f$
500 IF f$="          " THEN  GO TO 540
510 AT c$;start,entries,0,rwidth,stcol,f$
520 CLS : IF stcol=0 THEN  PRINT f$;" Not found": GO TO 470
530 LET pos=start+stcol*rwidth-rwidth: PRINT f$;" ";VAL c$(pos+12 TO pos+
14);" Sectors long": GO TO 470
```

```
540>LET f$="newfile": AT c$;start,entries,0,rwidth,stcol,f$
541 LET nfu=0: IF stcol=0 THEN  GO TO 550
542 LET pos=start+stcol*rwidth-rwidth: LET nfu=VAL c$(pos+12 TO pos+14)
545 REM
550 REM Sort
560 REM
570 CLS : PRINT "Sort by size": PRINT
580 LET stcol=12: DIM w$(3)
590 AT c$;start,entries,S,rwidth,stcol,w$
600 PRINT c$(18 TO end)
610 INPUT "Press ENTER ";j$
620 CLS : PRINT "Sort by file type": PRINT
630 LET stcol=16: DIM w$(1)
640 AT c$;start,entries,S,rwidth,stcol,w$
650 PRINT c$(18 TO end)
660 INPUT "Press ENTER ";j$
670 CLS : PRINT "Sort by file name": PRINT
680 LET stcol=1: DIM w$(10)
690 AT c$;start,entries,S,rwidth,stcol,w$
700 PRINT c$(18 TO end)
710 INPUT "Press ENTER ";j$
720 REM
730 REM Write serial file
740 REM
750 CLS : PRINT "Write serial file"
760 DIM d$(4*512): LET s=1
770 FOR d=1 TO 4: LET d$(s TO s+511)="New file record "+STR$ d: LET s=s+5
12: NEXT d
780  OUT #stream;MD
790  OUT #stream;"newfile";R1,4;BIN  TO d$
800 INPUT "Press ENTER ";j$: CLS : GO SUB 810: GO TO 860
810 LET entries=CODE c$(1)
820 LET free=VAL c$(13 TO 15)
830 PRINT "New catalog - see newfile"
840 LET end=17*(entries+1)+1: PRINT : PRINT c$(18 TO end)
850 INPUT "Press ENTER ";j$
855 RETURN
856 REM
860 CLS : PRINT "Reading attempt wrong cartridge in - press c, then CONT
ENTER"
862 REM
870 LET c$(2 TO 11)="**********": DIM d$(2048)
880 IN #stream;"newfile";R1,4;BIN  TO d$
890 LET c$(2 TO 11)=n$: DIM d$(2048)
900 INPUT "Press ENTER ";j$
905 REM
910 CLS : PRINT "Read serial file - press ENTER  after 'scroll?' message"
915 REM
920 IN #stream;"newfile";R1,4;BIN  TO d$
930 PRINT AT 3,0;d$
940 INPUT "Press ENTER ";j$
945 REM
948 IF nfu>4 THEN  GO TO 1006
950 CLS : PRINT "Extend file"
955 REM
960  OUT #stream;MD
970 DIM e$(512): FOR d=5 TO 8: LET e$="Extend file record "+STR$ d
980  OUT #stream;"newfile";Rd,d;BIN  TO e$
990 NEXT d
```

```
1000>CLS :GO SUB 810
1005 REM
1006 REM Random read
1008 REM
1010 DIM e$(512): CLS : PRINT "Random read record 2"
1020 IN #stream;"newfile";R2,2;BIN  TO e$
1030 PRINT AT 2,0;e$
1040 INPUT "Press ENTER ";j$
1050 CLS : PRINT "Random read record 4"
1060 IN #stream;"newfile";R4,4;BIN  TO e$
1070 PRINT AT 2,0;e$
1080 INPUT "Press ENTER ";j$
1090 CLS : PRINT "Random read record 6"
1100 IN #stream;"newfile";R6,6;BIN  TO e$
1110 PRINT AT 2,0;e$
1120 INPUT "Press ENTER ";j$
1125 REM
1130 CLS : PRINT "Read pointers for random write"
1135 REM
1140 DIM d$(4096): DIM p$(512):POINT "newfile" TO p$
1150 IN #stream; RND "newfile";R1,8; BIN  TO d$; POINT  TO p$
1160 PRINT AT 2,0;"Point data file ";p$(1 TO 10)
1170 PRINT : PRINT " Record  Sector  Stream"
1180 FOR d=1 TO 8: LET rec=d: GO SUB 2010: PRINT "   ";d;"        ";sector;
TAB 21;flag: NEXT d
1190 INPUT "Press ENTER ";j$
1195 REM
1200 CLS : PRINT "Update file"
1205 REM
1210 LET rec=5: GO SUB 2100: LET rec=2: GO SUB 2100
1215 LET rec=3: LET dp=512*(rec)-511: LET d$(dp+64 TO dp+95)="No update re
cord "+STR$ rec
1220  OUT #stream; RND "newfile";R1,5;BIN  TO d$;POINT  TO p$
1230 LET rec=5: GO SUB 2070: LET rec=2: GO SUB 2070
1240 INPUT "Press ENTER ";j$
1250 REM
1252 CLS : PRINT "Reread file"
1260 DIM d$(4096)
1270 IN #stream; "newfile";R1,8; BIN  TO d$
1280 PRINT AT 2,1;d$
1999 STOP
2000 REM
2001 REM get sector and flag
2005 REM
2010 LET p=9+rec*2: LET sector=CODE p$(p): LET flag=CODE p$(p+1)
2020 RETURN
2030 REM
2032 REM set flag for updating
2033 REM
2040 LET p=10+rec*2: LET p$(p)=CHR$ (CODE p$(p)+128): RETURN
2060 REM
2063 REM reset flag after write
2065 REM
2070 LET p=10+rec*2: LET p$(p)=CHR$ (CODE p$(p)-128)
2080 IF CODE p$(p)>64 THEN  PRINT "Record ";rec;" not found": STOP
2090 RETURN
2100 LET dp=512*(rec)-511: LET d$(dp+64 TO dp+95)="Update record "+STR$ re
c: GO SUB 2040: RETURN
```

```
100>DIM c$(1000):REM catalog
110 DIM d$(4096): DIM n$(6,24): DIM s$(1): DIM g$(3): DIM q$(128): DIM t$
(128)
115 DIM l(9)
120 FOR d=1 TO 6: LET n$(d)="........................": NEXT d
130 LET s$=".": LET g$="..."
140 LET l$="................................": LET q$=l$+l$+l$+l$
150 GO SUB 1500
159 REM
160 REM Enter data in form
161 REM
170 LET start=1: LET end=1: LET d$(1)=CHR$ 136: LET lwide=128
180 LET col=1: LET attr=96: DIM o$(24): LET a$=" "
190 FOR l=5 TO 9: GO SUB 2009: NEXT l
200 LET l=12: GO SUB 2009
210 LET l=4: LET col=30: DIM o$(1): GO SUB 2009
220 LET l=8: LET col=29: DIM o$(3): GO SUB 2009
230 LET l=15: LET col=1: DIM o$(128): GO SUB 2009
239 REM
240 REM Display data
241 REM
260 INPUT "Press ENTER ";i$
280 CLS : PRINT d$(1 TO end)
290 INPUT "Press ENTER ";i$
299 REM
300 REM Extract data
301 REM
310 LET starta=start
320 FOR l=1 TO 6: LET line=1: LET len=128
330 AT d$;starta,end,F,line,len,t$
340 LET n$(l)=t$(1 TO len-1): LET l(l)=len-1: NEXT l
350 AT d$;starta,end,F,line,len,t$
360 LET s$=t$(1): LET l(7)=1
370 AT d$;starta,end,F,line,len,t$
380 LET g$=t$(1 TO len-1): LET l(8)=len-1
390 AT d$;starta,end,F,line,len,t$
400 LET q$=t$(1 TO len-1): LET l(9)=len-1
410 GO SUB 1500
419 REM
420 REM Select line
421 REM
425 LET ed=1: LET edl=4: LET edc=0
430 LET j$=" "
440 PRINT AT 20,0; PAPER 1; INK 7; BRIGHT 1;"Use up/down cursor keys then
   ENTER to select, EDIT to end";TAB 32;
450 PRINT AT edl,edc; PAPER 6; FLASH 1; OVER 1;" "
460 GO SUB 1400
480 IF CODE j$=13 THEN  PRINT : GO TO 560
490 PRINT AT edl,edc; PAPER 6; FLASH 0; OVER 1;" "
495 IF CODE j$=7 THEN  GO TO 700
500 LET edc=0: IF ed>5 THEN  GO TO 520
510 LET edl=ed+3: GO TO 450
520 IF ed=6 THEN  LET edl=11: GO TO 450
530 IF ed=7 THEN  LET edl=3: LET edc=29: GO TO 450
540 IF ed=8 THEN  LET edl=7: LET edc=28: GO TO 450
550 LET edl=14: GO TO 450
559 REM
560 REM Edit data
```

```
 561>REM
 565 PRINT AT 20,0;"EDIT LINE";TAB 32;" ";TAB 32;
 570 LET l=edl+1: LET col=edc+1: LET len=l(ed): LET a$=" "
 580 IF ed>6 THEN  GO TO 600
 590 DIM o$(24): LET o$=n$(ed): GO SUB 650: LET n$(ed)=w$: GO TO 430
 600 IF ed=7 THEN  DIM o$(1): LET o$=s$: GO SUB 650: LET s$=w$: GO TO 430
 610 IF ed=8 THEN  DIM o$(3): LET o$=g$: GO SUB 650: LET g$=w$: GO TO 430
 620 DIM o$(128): LET o$=q$: GO SUB 650: LET q$=w$: GO TO 430
 647 REM
 648 REM Edit line
 649 REM
 650 AT o$;l,col,E,len,attr,a$
 652 LET w$=o$(1 TO len): LET x$=w$+CHR$ 13
 655 REM
 656 REM Delete old line
 657 REM
 660 LET dlen=1: LET l(ed)=len: LET len=len+1: LET line=ed: LET dline=ed
 670 AT d$;start,end,D,dline,dlen,t$
 675 REM
 676 REM Insert new line
 677 REM
 680 AT d$;start,end,I,line,lwide,x$
 690 RETURN
 700 CLS : PRINT d$(1 TO end)
 710 INPUT "Press ENTER ";i$
 719 REM
 720 REM Text
 721 REM
 730 LET z$="This is a line of text to show display formats, first left ju
stify at 16 charas wide and second right and left justify at 18 charas wid
e."
 740 LET d$=z$: LET end=LEN z$+1: LET d$(end)=CHR$ 136
 750 LET lwide=16: LET line=1: LET startl=1: DIM o$(lwide): LET f$=" ": LE
T len=1
 760 CLS : FOR d=1 TO 12: LET o$(1)=f$
 770 AT d$;startl,end,F,line,len,o$
 780 PRINT o$
 790 IF o$(1)=CHR$ 136 THEN  GO TO 800
 795 NEXT d
 800 INPUT "Press ENTER ";i$
 810 DIM s(20): LET lwide=18: LET line=1: LET startl=1: DIM o$(lwide): LET
 f$="R": LET len=1
 820 CLS : FOR d=1 TO 12: LET o$(1)=f$: LET s(d)=startl
 830 AT d$;startl,end,F,line,len,o$
 840 PRINT o$
 850 IF o$(1)=CHR$ 136 THEN  GO TO 900
 860 NEXT d
 900 INPUT "Press ENTER ";i$
 905 REM
 910 REM Edit data
 911 REM
 920 DIM t$(256): LET startd=s(8): LET dline=1: LET dlen=1
 930 AT d$;startd,end,D,dline,dlen,o$
 940 LET elen=dlen+1: LET t$=o$: IF dline=0 THEN  LET t$(elen)=CHR$ 137
 945 PRINT AT 20,0;"Type in 10 short lines with     ENTER at end, EDIT to
end";
 950 LET a$="I": LET eline=12: LET ecol=1
 960 AT t$;eline,ecol,E,elen,attr,a$
 965>IF elen<2THEN GO TO 1010
 970 CLS : PRINT "Edited data is now": PRINT : PRINT t$
 980 INPUT "Press ENTER ";i$
 990 LET line=1: LET j$=t$(1 TO elen-1)
1000 AT d$;startd,end,I,line,lwide,j$
1010 LET up=1: LET lwide=10: LET line=1: LET start1=1: DIM o$(lwide): LET
f$="R": LET len=1
1020 CLS : FOR d=1 TO 8: LET o$(1)=f$: LET s(d)=start1
1030 AT d$;start1,end,F,line,len,o$
1040 PRINT o$
1050 IF o$(1)=CHR$ 136 THEN  GO TO 1080
1060 NEXT d
1070 INPUT "Press ENTER (+ other to end) ";i$: IF i$<>"" THEN  STOP
1075 GO TO 1020
1080 LET up=up+1: LET line=up
1090 AT d$;start1,end,U,line,lwide,o$
1100 GO TO 1070
1399 STOP
1400 PAUSE 40: LET j$=INKEY$
1410 IF CODE j$<>7 AND CODE j$<>10 AND CODE j$<>11 AND CODE j$<>13 THEN  G
O TO 1400
1420 IF CODE j$=10 THEN  LET ed=ed+1: IF ed>9 THEN  LET ed=9
1430 IF CODE j$=11 THEN  LET ed=ed-1: IF ed<1 THEN  LET ed=1
1450 RETURN
1499 STOP
1500 CLS : PRINT TAB 9;"EXAMPLE FORM": PRINT
1501 REM
1510 PRINT "NAME AND ADDRESS";TAB 28;"SEX": PRINT TAB 29; PAPER 6;s$
1520 PRINT AT 4,0;
1530 FOR j=1 TO 5: PRINT  PAPER 6;n$(j): NEXT j
1540 PRINT : PRINT "TELEPHONE NUMBER": PRINT  PAPER 6;n$(6)
1550 PRINT : PRINT "QUALIFICATIONS & EXPERIENCE": PRINT  PAPER 6;q$
1560 PRINT AT 6,28;"AGE";AT 7,29; PAPER 6;g$  .
1570 RETURN
1999 STOP
2000 REM Enter data
2001 REM
2009 LET o$="": LET len=1
2010 AT o$;1,col,E,len,attr,a$
2020 LET x$=o$(1 TO len)+CHR$ 13
2029 REM
2030 REM Enter in array
2031 REM
2040 LET start2=end: LET line=1
2050 AT d$;start2,end,I,line,lwide,x$
2060 RETURN
8000 CLEAR 57599: LOAD *"m";1;"scode"CODE : RANDOMIZE USR 57600: RUN
```