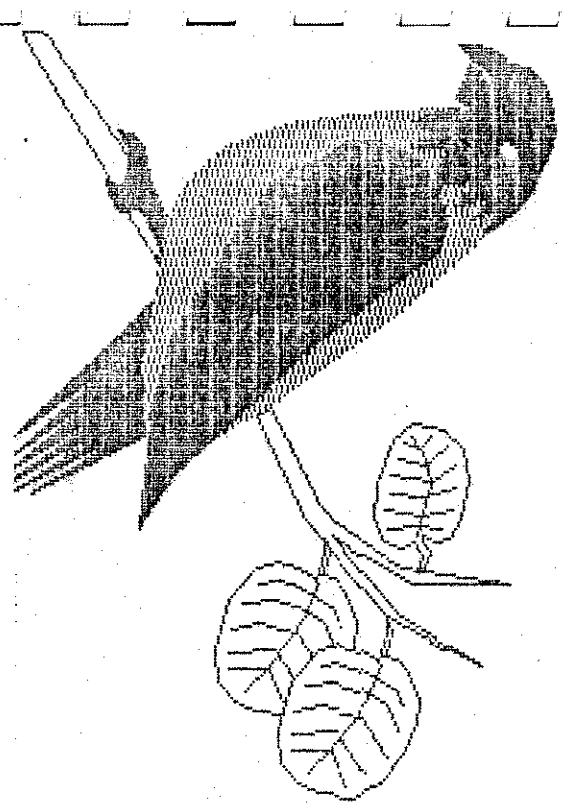
POLY4

- FILE HANDLING
- GRAPHICS
- MEMORY MAPPING

PROGENI



FILE HANDLING

Files are simply sets of data stored on disk. The operating system looks after where it is stored and catalogues each file by name. CAT shows the list of files on a disk.

Files whatever their function, all have a standard format on disk. Thus it is possible to read or List any type of file. To distinguish the functions of different files, we do assign particular extensions.

.DAT Is a data file. In a program if a data file is not given an extension, then .DAT is put onto the filename.

.BAS BASIC source files are assigned an extension of .BAS. The LOAD command expects .BAS if no extension is specified. SAVE assigns the extension .BAS.

.BAC BASIC compiled files are COMPILED with the default extension of .BAC. Run and CHAIN expect the extension .BAC.

.BAK Version 2.0 may assign a file the extension .BAK if the SAVE option is used on opening a file.

.PRT Files with an extension of .PRT are automatically printed by the operating system, as soon as they are closed. The default extension for the WTD command is .PRT

.TXT is assigned to files SAVED when editing in the TEXT mode.

.CMD Operating system commands are stored in files with a .CMD extension.

.SYS Operating system files are stored with .SYS extension.

Any extension may be used to suit the user. .PRT is the only extension for which any automatic function take place.

Information is stored on the disk magnetically in concentric circles, each circle is called a SECTOR. Each sector holds 256 bytes of information.

Disks can be recorded on both sides or on only one side, in single or double density. The Poly system can use either single or double sided disks and records in single density. Using single density, each side of the disk has 1140 sectors. When a file is created, it may contain many sectors, which may or may not be adjacent. The first 4 bytes of a sector 'point' to the next sector in the file. This leaves 252 bytes if the sector is left for data.

Files are of two types serial and random.

Serial Files

Files are made up of a number of records, each record containing a set of information. Think of a card file where each card contains information about different plants. Each card would probably contain a basic set of formatted information - the common name, the botanical name, the distribution etc. A file is the set of cards. An individual record is one card. Each record contains a set of 'fields' each containing a set of data.

A disk file is similiar. Each record may be up to 252 bytes long. Records are separated by a line feed character (CHR$(13)) on the disk.

In BASIC, before a file can be written or read, it must be opened. The OPEN statement checks the disk for the named file, stores the location of the file and sets up a 252 byte 'buffer' in the Poly's memory, into which a record is read. This buffer is simply a piece of memory set aside for that file. When a file has been finished with, it must be closed. The following are all valid OPEN statements.

1. OPEN OLD "MASTFILE" AS 1

2. OPEN NEW "CUSTFILE" AS 6

3. OPEN NEW A$ AS 12

When a file is opened, it must be specified whether the file is a NEW or OLD file. If OLD is specified, then the file must be present. Unless the filename has specified a specific drive, this will be on the current drive for that Poly. If it cannot be found, ERROR4 is returned. This would need to be trapped using an ON ERROR routine.

If NEW is specified a new file is immediately created. If a file of the same name already exists, then the old file is automatically deleted. No error is given.

In V2.0 a third form of OPEN is available.

```
OPEN SAVE "FILE1" AS 10
```

This is the same as NEW except that if an old file exists then instead of being deleted, its extension is changed to .BAK. If a file of that name exists, it is deleted. This allows an old version of a file to be kept automatically.

The filename may either be entered in quotes or be held in a string variable. The filename contains
The drive number (optional)
The filename
The file extension (optional)

If the drive number is not specified, then the current drive for that Poly is assumed.

If the file extension is not specified, then .DAT is assumed.

The filename must not be longer than 8 characters.

The following are valid filenames.

```
"1. MASTFILE.TXT"
"XXX.DAT"
"T"
"0.XXXXX"
```

When creating filenames, they should be meaningful. Pupils should always put their initials as the first three characters.

In BASIC, a program may use up to 12 files at any one time. Each file is assigned a channel number between 1 and 12. The OPEN associates the file with a particular channel. This channel number is then used in all other commands which reference the file.

Reading and Writing Serial Files

WARNING. In Versions of the operating system prior to V2.0, errors may occur if more than 2 files are open for write at the same time. No error is flagged but disk area corruption may occur. V2.0 allows multiple files to be written simultaneously at any time.

Serial files are written using PRINT# and read using INPUT# or INPUTLINE#

Each PRINT# command writes data to the file starting in the next record. Each INPUT# or INPUTLINE# reads the data starting in the next record on the file. A record is terminated by a CHR$(13) automatically.

For Example:

```
PRINT#1, A$;B$;C
```

The 1 is the channel number specified in the OPEN. This must be followed by a ,. Thereafter the fields to be written to the file are specified separated as in the PRINT statement.

```
A = -2
A$ = "AA"
PRINT#1, "XXX"; X$; "ZZZ"
PRINT#1, A$, A
```

This writes 2 records on the file. The first contains

```
XXXAAZZZ
```

and the second contains.

```
AA     - 2
```

The printed representation of the numeric field is stored. Note that the -2 is "tabbed" to the next column and has 6 spaces preceding it.

The physcial record on the disk would contain

```
XXXAAZZ*AA      -2*
```

To further format the record, USING may be used.

```
PRINT#1, USING "$$##.##", 123.456
```

writes a record on the file containing

```
$123.46
```

The maximum string length of data written cannot exceed 255 characters. No error is griven if this length is exceeded. If the string contains CHR$(0) characters, these are deleted.

Reading a File

INPUT# and INPUTLINE# are used for reading a sequential file.

```
INPUT#1, A$
```

puts into A$, all the data in the next record up to the next CHR$(13). If multiple fields are to be read from a record, these must be written separated by commas.

```
10   OPEN NEW "MYFILE" AS 1
20   A$"AAA":B$="BBB"
30   PRINT#1, A$;B$
40   PRINT#1, A$,";B$
50   CLOSE 1
60   OPEN OLD "MYFILE" AS 1
70   INPUT#1, A$, B$                    ERROR
80   PRINT A$, B$
90   INPUT#1, A$, B$
100  PRINT A$, B$
110  CLOSE 1
```

An error occurs at line 70 because there is only one field to input before reach the end of the record - there was no , to break it up into fields.

WARNING

1. If the Data written using a PRINT# statement contains any CHR$(0) (nulls), then these are deleted before the data is written to disk. To store data containing nulls, random files must be used. Not this when using CVT functions.

INPUTLINE# acts in a similiar fashion to INPUT# except that it reads all the data up to the end of the record (CHR$(13) including any commas. Whereas INPUT# may specify a number of variables, INPUTLINE# may only specify a single string variable.

```
INPUT LINE#1, A$
```

Reading and Writing Files in Programs

In most programs, it is necessary to put in a separate INPUT# or PRINT# for every record on a file, but these would be placed within a loop.

```
20    OPEN NEW "MYFILE" AS 1
30    CLS : PRINT@ (10,0);
35    N$= ""
40    INPUT "NAME"; N$
45    IF N$ = """ TURN 110
50    INPUT "ADDRESS LINE 1"; L1$
60    INPUT "ADDRESS LINE 2"; L2$
70    INPUT "ADDRESS LINE 3"; L3$
80    INPUT "TELEPHONE NO";
90    PRINT#1, N$, ", ", L1$, ",", L2$, ", ", L3$, ",", T$
100   GO TO 30
110   CLOSE 1
```

To read this file

```
5     ON ERROR GO TO 90
10    OPEN OLD "MYFILE" AS 1
20    INPUT#1, A$, B$, C$, D$, E$
30    CLS
40    PRINT A$!B$!C$!D$!E$
50    A$ = INCH$(0)
60    GO TO 20
70    CLOSE 1
80    END
90    RESUME 70
```

V2.0 has a new command to check for end of file

```
      15 ON END 1 GO TO 70
```

This would replace the ON ERROR routine specified above.

V2.0 also allows a INPUT LINE statement to specify the lenght of the fold to be entered - the cursor cannot move outside this area.

```
      INPUT LINE(4), A$
```

If a ! is used in a PRINT#, this effectively creates separate records for each field.

Exercise

Because of the possible errors in multiple units writing to disk in V1.6, these exercises are only centred around reading serial files. As long as not more than 1 person is writing to the disk at a time, there should be no problems.

BASIC source files are serial files, each line being separated by a CHR$(13).

Write a program to read a BASIC source file, and list it on the screen.

SAVE the program and then run it.

Random Files

Serial files may only be read or written starting from the 1st record and moving through the file. Random Files may be read or written in any record order. V1.6 sets all record sizes at 252 characters (ie a sector) but in V2.0 a record size may be specified which is less, allowing several records to be packed into a sector. Serial files can be OPEN either for reading or writing but not for both. Once OPEN, a RANDOM file may be either read or written to.

```
OPEN OLD RANDOM "CUSTFILE" AS 1
```

The OPEN statement is the same as for serial files, except that the word RANDOM must be inserted. The CLOSE statement is the same.

```
CLOSE 1
```

Instead of PRINT# and INPUT#, random files use GET# and PUT#.

```
GET#, REOCRD 10
```

GET# gets a specific record and places it in the "buffer" area for the file. It does not put the record directly into a variable. If the RECORD NO is not specified the next record on the file is returned. The FIELD statement is used to assign variable names to the contents of the buffer.

```
FIELD#1, 24 AS A$, 20 AS B$, 30 AS C$
```

This assigns the first 24 characters to A$, the next 20 to B$ and the next 30 as C$.

After GETTING the record, we can now process the information in the various variables.

```
5    ON ERROR GO TO 110
10   OPEN OLD RANDOM "CUSTNAME" AS 1
20   DIM A$(4)
30   FIELD#1, 20 AS A$(0), 20 AS A$(1), 20 AS A$(2), 20 AS A$(3), 6 AS
     A$(4)
35   INPUT "RECORD NO"; I
40   GET#1, RECORD 1
50   CL
60   FOR I = 0 TO 4
70   ? A$(I)
80   NEXT
100  GO TO 35
110  CLOSE 1
```

Writing RANDOM Files

PUT# is used to write a random file record from the file buffer. The program must first put the data into the buffer. The field names are assigned using the FIELD statement as in the GET#. However, the values cannot be put into the variables in the usual way.

```
10   FIELD#1, 20 AS A$, 20 AS B$
20   A$ = "MYFILE"
```

Line 10 assigns a 20 character field to A$. Line 20 assigns A$ a new value "MYFILE" somewhere else in the string area and the value of A$ in the buffer is lost.

Two special commands LSET and RSET which do not shift the current position of A$, must be used.

```
FIELD#1, 20 AS A$, 20 AS B$
LSET A$ = "MYFILE"
RSET B$ = "YOURNAME"
```

LSET puts the value "MYNAME" into the current A$ string, right justifying it and padding the end of A$ to spaces.

RSET put "YOURNAME" into the right hand end of the current B$ string, and puts spaces preceding it. Once the buffer has been set up, it is then PUT# to the file.

```
10   CLS
20   OPEN OLD "NAMEFILE" AS 1
25   FIELD#1, 20 AS A$, 10 AS B$
30   INPUT "NAME; A$
40   INPUT "TELEPHONE", B$
50   INPUT "RECORD NO"; C
60   LSET A$ = A1$
70   LSET B$ = B1$
80   PUT#1, RECORD C
90   CLS: GO TO 30
```

WARNING

Writing a random record which extends the file may cause disk corruption in V1.6. V2.0 corrects this.

GRAPHICS

TOOLS

SELECT
MIX
SET
RESET
LINE
POINT
LDES$
DRAW
DRAW@
FILL
FILL@
STORE
Extensions in V2.0
DRAW#
FILL#
STORE#

These notes discuss the use of the fine graphics on the Poly. To use them most effectively, some knowledge of the way in which they are stored and accessed is necessary.

Each of the 2 graphics screens occupy an 8K block of memory. Screen 2 is outside of addressable memory and screen 4 is in the top 8K of the programming area. If used, screen 4 reduces the available memory for programming by 8K. In V1.6, a SELECT of screen 4 cannot be made from with a loop or subroutine. It is wise therefore to select screen 4 at the start of the program if it is going to be used. Be aware that if secondary colours are used on screen 2, then screen 4 is automatically assigned.

Colours

Each of the graphics screens have only the four primary colour available, ie white, red, blue and green. The secondary colours, yellow, magenta and cyan can be achieved in several ways by using MIX.

By using MIX on an individual screen a choice of any one of 4 sets of 4 colours can be made ie either

1.	RED	RED	YELLOW	MAGENTA
2.	GREEN	YELLOW	GREEN	CYAN
4.	BLUE	MAGENTA	CYAN	BLUE
7.	WHITE	WHITE	WHITE	WHITE
		(+RED)	(+GREEN)	(+BLUE)

Note that the COLOUR command still demands that the primary set is used to define the colour.

SEE LISTING GR9.BAS

If there is sufficient memory to use both graphics screens, then secondary colours may be obtained more simply by using MIX ON.

SEE LISITNG GR10.BAS

If at line 90 screen 4 has not been selected, it is selected automatically and memory is reduced. (V1.6 has trouble at the moment if this occurs inside a FOR/NEXT loop or within a subroutine). Screen 4 is not automatically displayed or cleared or MIX put ON. If secondary colours are required on the graphics screen a careful consideration of the MIX type required needs to be made.

On the fine graphics screens it is not possible to have every pixel a separate colour. Each group of 6 horizontal pixels must be the same colour. This can be understood by looking at the way in which the screen is represented in memory. The whole screen is stored in an 8K block, starting with the top left hand side, each row is stored serially. Each six pixels are stored in an 8 bit byte, the first 2 bits giving the colour, and the last 6 bits having0 to represent a pixel off and 1 to represent a pixel on.

For Example:

1	Colour
0	Colour
1	On
1	On
0	Off
0	Off
1	On

Screen 4 can be looked at from BASIC by PEEKING at the memory from 32768 to 40959. Screen 2 can be mapped into memory and looked at as well but this is considerably more difficult (see section on Memory Mapping)

When using SET, LINE, DRAW or FILL the rules used within BASIC are as follows:

Setting a pixel on:

If the byte is of the correct colour already or if all pixels are off then the pixel is set on and the colour set if necessary.

If the byte has the incorrect colour set, then all pixels in the byte are set off, the colour is set and the new pixel set on.

Setting a pixel off:

The bit value is simply set to 0

SEE LISTING GR1.BAS

Notice how the green line clears a 6 pixel wide highway through the other colours. As you now understand the way in which the graphics work, then it will help you in producing good graphics programs.

The screen coordinates are numbered from the top left hand side corner.

The columns are numbered from 0 to 239 and the rows from 0 to 203. In all specifications of positions, the row is given first followed by th column.

Hence (0,0) is in the top left hand corner and (203,239) is row 203, column 239 and is in the bottom right hand corner.

Most of the commands accept negative values or values over the edge of the screen. Lines are drawn to these points as if they were there.

SET and RESET need no further comment except to say that the brackets are not necessary. Hence SET (100,10) is the same as SET 100,10.

POINT is a function which returns the colour of any particular pixel. If the pixel is OFF the POINT returns value 0. POINT does not take into account any mix which may be on but returns the primary colour of the pixel on that screen.

LINE draws a line between the sets of points defined. Note that the coordinates may be defined using variables. By changing the variables the "LINE" can be moved around the screen. LINE may have the coordinates written paired with brackets or the brackets may be omitted.

```
LINE 0,0,0,30,10,35,10,50,20,50
LINE (0,0),(0,30),(10,35),(10,50),(20,50)
```

These are both permisable.

SEE LISTING GR2.BAS

Note that LINE quite happily accepts negative coordinates so that the car can appear and disappear easily.

The logic for this program is for each position of the car across the screen.

```
Set colour to green
Draw the car
Set colour to blank
Draw the car
```

This gives a jerky movement as we can visually see the car being drawn and undrawn. If these actions can take place on a screen that is turned off, then the movement can be much clearer.

SEE LISTING GR3.BAS

Use of LDES$ and DRAW are a lot easier to code. LDES$ can define the picture anywhere on the screen. DRAW moves it for you.

Listing GR5.BAS shows this done using one screen and GR4.BAS shows it using both screens.

This is a simple example of animation. When doing any animation, carefully work out each of the alternate actions that you want. In this case each of the drawings were the same. If it was a martian moving across the scteen there would need to be at least 2 different diagrams. Or maybe only a small piece of the diagram needs to be changed.

SEE LISTING GR6.BAS

When using LDES$ it is always worthwhile defining shapes in a clockwise direction so that it can also be used with FILL. Let's go back to our car and replace the DRAW with a FILL.

SEE LISTING GR7.BAS WHICH SHOWS FILL ON 1 SCREEN AND GR8.BAS WHICH USES TWO SCREENS.

LDES$ does not need to define a continous line - the line may be broken into several pieces by separating the coordinates with a ; in place of a ,. For example:

```
A$ = LDES$ (0,0,10,10;100,100,0,100)
```

If the LDES$ contains more points than can be accomodated in a single BASIC line, then the description may be broken into several strings and then added together. Note that the line is broken at the point where the strings are joined. To make a continuous line, the end points need to be duplicated.

For example:

```
A$ = LDES$(0,0,100,100,200,200)
B$ = LDES$(200,200,100,50,0,0)
A$ = A$ + B$
DRAW A$
```

One for the main restrictions of LDES$ and LINE is that they only allow DRAW to draw the picture in a single colour. Each colour in the picture needs to be handled separately. STORE enables a block of the graphics screen to be stored in a string and then be drawn again anywhere on the screen.

```
STORE (10,10),(100,100) CAR$
```

Stores the rectangle with the opposite corners (10,10) and (100,100). Some care needs to be taken with use of STORE as the data stored may be extremely large. A full screen may require up to 8K of memory. When storing the screen, multiple blanks and repeated patterns are compressed.

Although the coordinates for STORE are given for the fine graphics screen, only complete bytes are stored. In the above example the start column requested is 10. The start of that byte is column 6.

The end column is 100, and the end of that byte is 101. So

```
STORE (10,6),(100,101) CAR$
```

would store the same area.

The block can then be redrawn in the same place with DRAW CAR$
or elsewhere with DRAW@ (120,0) CAR$

Small pictures drawn from a stored picture reproduce reasonably quickly

SEE THE G11.BAS LISTING WHICH INTERCHANGES 2 PICTURES.

For a moving object, it is worthwhile storing a blank space alongside the picture so that the previous picture is erased when the new one is drawn.

SEE G12.BAS LISTING WHICH USES THE FIGURE FROM THE PRESENTATION.

When setting up strings using LDES$, is often convenient for them all reference a common point so that positioning them with DRAW@ follows a set of standard positions. For example, if a set of special characters are created by LDES$ for the graphics screen, it is sensible to base them on a rectangular area and to position them on the screen relevant to the top left hand corner of the rectangle.

To do this, all "characters" should be defined in the top left hand corner of the screen with a "dummy" (0,0) point first. This should be followed with a ;.

SEE G13.BAS LISTING

LDES$ creates strings. Each pair of coordinates are stored in 3 bytes, the first byte denoting the row, and bytes 2 and 3 the column. STORE saves an image of the screen area into the string. Ahead of the screen dump is a 4 byte header. Where multiple spaces or multiple patterns occur these are stored in a compressed format.

Printing Graphics Screens

The Epson printer has 8 vertical pins. The dots must be presented to it as a vertical set of 8 dots. Considerable manipulation has to take place in order to format them in this way.

The only way of doing this reasonably effectively in BASIC is by the use of the logical operators.

LISITNG G14.BAS SHOWS THE LISTING OF A ROUTINE TO PRINT THE TWO GRAPHICS SCREENS.

MEMORY MAPPING

The 6809 hardware is capable of addressing 64K bytes of memory. However, a feature called Dynamic Memory Allocation has been incorporated into the POLY design which allows more than 64K to be used.

This feature allows the user to switch into addressable memory any of 8 blocks of 8K memory, from a maximum of 16 blocks. These may be placed in any order in any of the 8 positions in addressable memory. Two memory maps are available, and the user indicates which is currently in use. Memory Map 1 is fixed at initialization of the system, but the user may alter Memory Map 2 to suit his requirements by using Software Interrupt 29 ($ID). The selection of Memory Map is made using Software Interrupts 27 ($IB) and 28 ($IC).

The Physical memory in the POLY is allocated as follows:

Blocks 10 to 15	not present
Blocks 8 and 9	16K RAM Bank 4
Blocks 6 and 7	16K BASIC ROM
Blocks 4 and 5	16K RAM Bank 3 Block 4 contains Graphic Screen 4
Blocks 2 and 3	16K RAM Bank 2 Block 2 contains Graphic Screen 2
Blocks 0 and 1	16K RAM Bank 1

The System ROM and teletext screens are within a protected area not accessable directly by the user. These may only be accessed via software interrupts.

Memory Map 1 is set up as follows:

LOWER Address	Physical Block	
$FFFF		
$E000	7	BASIC ROM
$C000	5	OPERATING SYSTEM RAM
$A000	6	BASIC ROM
$8000	4	(GRAPHICS 4 OR USER RAM)
$6000	8	RAM
$4000	3	RAM
$2000	1	RAM
$0000	0	Contains BASIC extensions up to $1700 approximately.

Memory Map 2 may vary depending upon the last use of it.

Setting and Calling Memory Map 2

1. Set up a string of 8 bytes, each byte containing the physical block number to be put into the map, starting from address 0.

2. Call software interupt 29 ($ID) giving the address of the 8 bytes as parameter 1.

3. Call software interupt 28 ($IC) to select memory Map 2.

CALLING MEMORY MAP1

1. Call software interupt 27 ($1B) to select memory Map 1.

For example

In BASIC

```
100 A$ = CHR$(0)+CHR$(1)+CHR$(3)+CHR$(8)+CHR$(2)+CHR$(6)+CHR$(5)+CHR$(7)
110 A% = SWI (29, DPEEK (PTR(A$)))
120 A% = SWI (28)
```

This puts the graphics screen 2 in place of graphics screen 4 in addressable memory. In BASIC, care must be taken not to call graphics commands while using Memory Map 2 as all graphics commands use Memory Map 2 and will overwrite the memory map set up by the user.

In PASCAL

PASCAL has no direct command to call Software Interupts. These may be called from Assembler which may be embedded in the PASCAL program by preceding it with a ! The code to set up Memory Map 2 the same as in the BASIC example is

```
        !  PSHS D                    Save register D
        !  LDD #MAP2                 Points D to MAP2
        !  SWI
        !  FCB 29                    Write to MAP2
        !  SWI
        !  FCB 28                    Select MAP2
        !  PULS  D
        !  BRA NEXT
!MAP2 FCB 0,1,3,8,2,6,5,7
!NEXT EQU*
```

Software interrupts are called by

```
    !SWI
    !FCB n
```

When n is the software interrupt number.

GRAPHIC SCREEN MEMORY MAPPING

Each graphics screen is memory mapped into an 8K block of memory, screen 2 occupying block 2 and screen 4 memory block 4.

Display of the screens is controlled by software interupts 15 and 16 where 16 reads the 16 bit control word and 15 sets to the value put into the D register. (See the BASIC manual for a description of the bit values)

Each byte of the screen memory contains the display information for a group of 6 horizontal pixels. The first 2 bit contain the colour information and the last 6 bits show whether a pixel is on or off - 1 indicates ON and 0 off. The colour has 00 = blue, 01 = green, 10 = red and 11 = white.

****

LISTING OF GR1.BAS

****

```
10CLS:PRINT@(10,10);
20SELECT 4:CLS:DISPLAY4
30A$=LDES$(0,0,0,239,40,239,40,0,0,0)
35FILL@(10,0)A$
40COLOUR 1
55FILL@(80,0)A$
60COLOUR 2
70LINE 0,4,200,239
75STOP
80FORI=32768 TO 40959 STEP4
90DPOKE I,RND(32767)
100NEXTI
110STOP
```

****

****

LISTING OF GR2.BAS

****

```
10CLS
20SELECT 2:CLS:DISPLAY2
30FOR col=-30 TO 260 STEP 10
40FOR colo = 2 TO 0 STEP -2
50COLOUR colo
60LINE100,col,100,col+30,110,col+35,110,col+50,120,col+50,120,col,100,col
70NEXT:NEXT
80GOTO30
```

****

****

LISTING OF GR3.BAS

****

```
10CLS
20SELECT 2:CLS:DISPLAY2
30SELECT 4:CLS:DISPLAY4
40FOR col=-30 TO 260 STEP 5
50FOR scr=2TO4 STEP 2
60DISPLAY scr OFF
70SELECT scr
80FOR colo = 0 TO 2 STEP 2
90COLOUR colo
100LINE100,col,100,col+30,110,col+35,110,col+50,120,col+50,120,col,100,col
110col=col+10
120NEXT
130col=col-15
140DISPLAY scr
150NEXT:col=col-5
160NEXT
170GOTO40
```

****

```
**********************************************************************************

            LISTING OF   GR4_BAS

**********************************************************************************

 10CLS
 20SELECT 2:CLS:DISPLAY2
 30car$=LDES$(0,0,0,30,10,35,10,50,20,50,20,0,0,0)
 40FOR col=-30 TO 260 STEP 5
 50FOR colo = 2 TO 0 STEP -2
 60COLOUR colo
 70DRAW@(100,col)car$
 80NEXT:NEXT
 90GOTO40

**********************************************************************************
```

```
**********************************************************************************

            LISTING OF   GR5.BAS

**********************************************************************************

 10CLS
 20SELECT 2:CLS:DISPLAY2
 30SELECT 4:CLS:DISPLAY4
 35car$=LDES$(0,0,0,30,10,35,10,50,20,50,20,0,0,0)
 40FOR col=-30 TO 260 STEP 5
 50FOR scr=2TO4 STEP 2
 60DISPLAY scr OFF
 70SELECT scr
 80FOR colo = 0 TO 2 STEP 2
 90COLOUR colo
 100DRAW@(100,col)car$
 110col=col+10
 120NEXT
 130col=col-15
 140DISPLAY scr
 150NEXT:col=col-5
 160NEXT
 170GOTO40

**********************************************************************************
```

```
**********************************************************************************

                LISTING OF   GR6.BAS

**********************************************************************************

10CLS
15DIM fig$(1)
20SELECT 2:CLS:DISPLAY2
25fig$(0)=LDES$(4,20,2,18,0,20,2,22,4,20,20,20,40,35;20,20,30,10,30,0)
26fig$(1)=LDES$(4,20,2,18,0,20,2,22,4,20,20,20,30,30,30,40;20,20,40,5)
30FOR col=-30 TO 260 STEP 5
40FOR colo = 2 TO 0 STEP -2
50COLOUR colo
60DRAW3(100,col)fig$(x%)
70NEXT
75x%=ABS(x%-1)
76NEXT
80GOTO30

**********************************************************************************
```

```
**********************************************************************************

                LISTING OF   GR7.BAS

**********************************************************************************

 10CLS
 20SELECT 2:CLS:DISPLAY2
 30car$=LDES$(0,0,0,30,10,35,10,50,20,50,20,0,0,0)
 40FOR col=-30 TO 260 STEP 5
 50FOR colo = 2 TO 0 STEP -2
 60COLOUR colo
 70FILL@(100,col)car$
 80NEXT:NEXT
 90GOTO40

**********************************************************************************
```

```
*****************************************************************************

              LISTING OF   GR8.BAS

*****************************************************************************

 10CLS
 20SELECT 2:CLS:DISPLAY2
 30SELECT 4:CLS:DISPLAY4
 35car$=LDES$(0,0,0,30,10,35,10,50,20,50,20,0,0,0)
 40FOR col=-30 TO 260 STEP 5
 50FOR scr=2TO4 STEP 2
 60DISPLAY scr OFF
 70SELECT scr
 80FOR colo = 0 TO 2 STEP 2
 90COLOUR colo
 100FILL@(100,col)car$
 110col=col+10
 120NEXT
 130col=col-15
 140DISPLAY scr
 150NEXT:col=col-5
 160NEXT
 170GOTO40

*****************************************************************************
```

****

LISTING OF GR9.BAS

****

```
10CLS
20SELECT2:CLS:DISPLAY2
30MIX 2,1
40COLOUR 1
50LINE 0,100,200,100
60COLOUR 2
70LINE 0,110,200,110
80COLOUR 4
90LINE 0,120,200,120
100WAIT 20
110MIX 2,2
120WAIT 20
130MIX 2,4
140WAIT 20
150MIX 2,0
160WAIT 20
170GOTO30
```

****

****

LISTING OF GR10.BAS

****

```
10CLS
15MIX ON
16SELECT4:CLS:DISPLAY4
20SELECT2:CLS:DISPLAY2
40COLOUR 1
50LINE 0,100,200,100
60COLOUR 2
70LINE 0,110,200,110
80COLOUR 3
90LINE 0,120,200,120
100COLOUR 4
110LINE 0,130,200,130
120COLOUR 5
130LINE 0,140,200,140
140COLOUR 6
150LINE 0,150,200,150
160COLOUR 7
170LINE 0,160,200,160
180STOP
```

****

LISTING OF GR11.BAS

```
10CLS
20SELECT2:CLS:DISPLAY2
30A$=LDES$(0,0;24,2,11,5,11,0,6,6,0,7,0,13,6,14,11,20,11,15,24,18)
40B$=LDES$(0,0;24,2;2,9,2,11,6,11,6,9,2,9)
50COLOUR 2
60DRAW@(0,9)A$
70FILL@(0,9)B$
80COLOUR 4
90FOR c= 0TO2
100SET9,17+2*c
110NEXT
120STORE(0,0),(24,41)M$
130COLOUR 1
140DRAW@(0,9)A$
150FILL@(0,9)B$
155COLOUR 7
160FOR c= 0TO2
170SET9,17+2*c
180NEXT
190STORE(0,0),(24,41)T$
200DRAW@(100,106)M$
210DRAW@(100,100)T$
220GOTO200
230STOP
```

LISTING OF GR12.BAS

```
10CLS
20SELECT2:CLS:DISPLAY2
30A$=LDES$(0,0;24,2,11,5,11,0,6,6,0,7,0,13,6,14,11,20,11,15,24,18)
40B$=LDES$(0,0;24,2;2,9,2,11,6,11,6,9,2,9)
50COLOUR 2
60DRAW@(0,9)A$
70FILL@(0,9)B$
80FOR c= 0TO2
90SET9,17+2*c
100NEXT
110STORE(0,0),(24,29)M$
120REM *** Bring class on ***
130FORr=110TO180 STEP30
140FORs=220TO100 STEP-25
150FORc=5 TO s STEP6
160DRAW@(r-24,c-7)M$
170NEXT:NEXT:NEXT
180STOP
```

LISTING OF GR13.BAS

```
5CLS
10A$=LDES$(0,0;0,1,6,1,6,4,2,4,2,1)
20SELECT 2:CLS:DISPLAY2
30DRAWA$
40STORE(0,0),(9,5)a$:CLS
41A$=LDES$(0,0;0,1,6,1;4,2,1,4;5,2,6,4)
42DRAWA$
43STORE(0,0),(9,5)b$
50DISPLAY0OFF
55SELECT 5:CLS:DISPLAY5
60FORi=0TO194STEP10
70FOR j=0TO476STEP12
80DRAW@(i,j)a$
85DRAW@(i,j+6)b$
90NEXT:NEXT
95PRINT@(22,0);
100STOP
```

****

LISTING OF PRINT.BAS

****

```
10CLS:PRINT@(1,8)"¦CMPOLY SCREEN PRINT":PRINT@(5,0);:INPUT"Start column (0-239)
";scol%:INPUT"End column (0-239)";ecol%:INPUT"Start row (0-203)";srow%:INPUT"END
 row (0-203)";erow%:INPUT"File name     ";ptemp$:INPUT"Screens (24)";screen%
20IFptemp$=""ORscol%<00Recol%>239ORecol%<scol%ORsrow%<00Rerow%<srow%ORerow%>239T
HENPRINT"¦BERROR":WAIT100:GOTO10
30both%=0:IFscreen%=24ORscreen%=42THENscreen%=2:both%=-1:DISPLAY4ELSEIFscreen%<>
2ANDscreen%<>4THENPRINT"¦BERROR":WAIT100:GOTO10
40DISPLAYscreen%:PRINT"¦BNo of rows =";(erow%-srow%)DIV8:SELECTscreen%:OPENNEWRA
NDOMptemp$AS11:FIELD_11,252ASz5$:IFzE%<>99THENDIMz%(7):zE%=99:FORz1%=0TO7:z%(z1%
)=2^z1%:NEXT
50scol%=scol%DIV6*6:ecol%=ecol%DIV6*6+5:zF%=32768+scol%DIV6+40*srow%:z1%=ecol%-s
col%+1:z4$="¦[A¦H¦M¦[K" +CHR$(z1%MOD256)+CHR$(INT(z1%/256)):z3$=STRING$(8):zB%=D
PEEK(PTR(z3$)):z3%=DPEEK(PTR(z5$))+8
60LSETz5$=z4$+STRING$(242,0)+"¦[2":FOR z2%=0TO(erow%-srow%)DIV8:PRINTz2%;:zA%=0:
FORz4%=0TO(z1%-1)DIV6:z6%=scol%+z4%*6:FORz5%=0TO7:STORE(srow%+z5%,z6%),(srow%+z5
%,z6%+5)z1$:z9%=ASC(RIGHT$(z1$,1)):IFboth%THENz9%=z9%ORPEEK(zF%+z4%+z5%*40)
70POKEzB%+z5%,z9%:NEXT:FORz5%=0TO5:z7%=0:z8%=z%(5-z5%):FORz9%=7TO0STEP-1:IF(PEEK
(zB%+7-z9%)ANDz8%)=z8%THENz7%=(z7%ORz%(z9%))
80NEXT:POKEz3%+zA%,z7%:zA%=zA%+1:NEXT:NEXT:srow%=srow%+8:PUT_11:zF%=zF%+320:NEXT
:CLOSE11
```

****

OPERATING SYSTEM
ERROR MESSAGES

1 ILLEGAL FMS FUNCTION CODE ENCOUNTERED
2 THE REQUESTED FILE IS IN USE
3 THE FILE SPECIFIED ALREADY EXISTS
4 THE SPECIFIED FILE COULD NOT BE FOUND
5 SYSTEM DIRECTORY ERROR - REBOOT SYSTEM
6 THE SYSTEM DIRECTORY SPACE IS FULL
7 ALL AVAILABLE DISK SPACE HAS BEEN USED
8 READ PAST END OF FILE
9 DISK FILE READ ERROR
10 DISK FILE WRITE ERROR
11 THE FILE OR DISK IS WRITE PROTECTED!
12 THE FILE IS PROTECTED - FILE NOT DELETED
13 ILLEGAL FILE CONTROL BLOCK SPECIFIED
14 ILLEGAL DISK ADDRESS ENCOUNTERED
15 AN ILLEGAL DRIVE NUMBER WAS SPECIFIED
16 DRIVES NOT READY
17 THE FILE IS PROTECTED - ACCESS DENIED
18 SYSTEM FILE STATUS ERROR
19 FMS DATA INDEX RANGE ERROR
20 FMS INACTIVE - REBOOT SYSTEM
21 ILLEGAL FILE SPECIFICATION
22 SYSTEM FILE CLOSE ERROR
23 SECTOR MAP OVERFLOW - DISK TOO SEGMENTED
24 NONEXISTENT RECORD NUMBER SPECIFIED
25 RECORD NUMBER MATCH ERROR - FILE DAMAGED
26 COMMAND SYNTAX ERROR - RETYPE COMMAND
27 NO RESPONSE FROM MASTER UNIT
28 WRONG HARDWARE CONFIGURATION
29 DRIVE NOT YET AVAILABLE FOR WRITE

1.

ERROR MESSAGES

NUMBER	MEANING
0	EXIT key pressed
1	Illegal file request
2	Requested file is in use
3	File already exists
4	File could not be found
5	System directory error
6	System directory full
7	All disk space has been used
8	End of file error
9	Disk file read error
10	Disk file write error
11	File or disk is write protected
12	File is protected
13	Illegal file control block specified
14	Illegal disk address encountered
15	Illegal drive number specified
16	Drive not ready
17	File is protected, access denied
18	File not opened in the correct mode
19	Data index range error
20	File management system inactive
21	Illegal file specification
22	File close error
23	Sector map overflow - disk too segmented
24	Non-existent record number specified
25	Record number match error or file damaged
26	Error in command
27	Communications error
29	Drive already booked for write
31	Illegal SWI function call
40	Unbalanced parentheses
41	Illegal character
42	Source not present
43	Line too long
44	Syntax error on encode
50	Invalid syntax
51	Invalid syntax in function
52	Invalid character at line start
53	Invalid statement start
54	Invalid statement terminator
55	Label expected
56	Numeric result expected
57	String result expected
58	"(" expected
59	"," expected
60	")" expected
61	Missing or invalid item in expression
62	Mixed mode
63	Too many temporary strings
64	Subscript negative or out of range
65	Incorrect number of subscripts
66	Undimensioned array reference
67	Expression result <0 or >255

68 String variable expected
69 Different string lengths
70 RETURN without GOSUB
71 NEXT without FOR
72 RESUME not in error routine
73 Cannot continue
74 Line not found
75 Auto mode will not overwrite existing lines
76 Line number too large
77 Fatal renumbering error
80 Arithmetic overflow
81 Too large to convert to integer
82 LOG of 0 or negative number
83 SQR of negative number
84 Division by 0
85 Argument too large
90 Argument out of range
91 Out of data in READ
92 Data type mismatch in PRINT USING
93 Illegal format in PRINT USING
94 Attempt to access outside screen area
95 Bad argument in SWAP
96 Illegal parameter in SWI or USR call
97 Array already dimensioned
98 FN function not defined
99 Dimension negative or too large
100 Clock not running
101 No room for stack
102 Memory set too low or high
103 No room for new string or array
104 CRC error
105 Verify error
106 No trailer record
120 Invalid data on input
121 Number too large for integer
122 Number too large
130 Invalid "Compiled" file
131 Invalid channel number
132 Channel not open
133 Channel already in use
134 Value mismatch in FETCH
135 Cannot merge compiled files
136 Field sizes exceed record size
137 Illegal DOS command from BASIC or TEXT
138 Random file used sequentially
139 Sequential file used randomly
140 Cannot have two files open for write
150 Graphics screen not selected
151 Invalid string for graphics

PROGENI