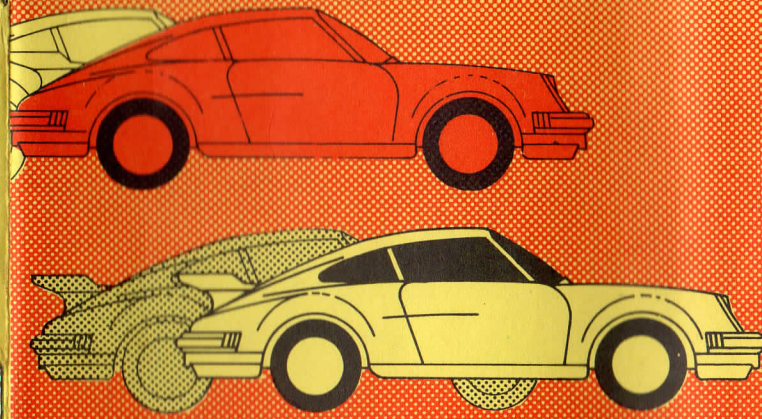
SPRINT

Basic Compiler
For the DRAGON 32



OASIS SOFTWARE

OASIS BASIC

Dr. David Gray
November 1983

Backups of the tape cannot be made and the low volume signal makes tape to tape copying extremely unreliable. Regrettably these steps must be taken if a realistic price is to be maintained to the genuine purchaser of our software.

This document describes OASIS BASIC 1.0.

CONTENTS

1. COMPILERS AND INTERPRETERS

In this manual DRAGON BASIC refers to the BASIC supplied with the DRAGON computer and OASIS BASIC refers to the BASIC implemented by the compiler.

DRAGON BASIC is implemented by an interpreter held in read only memory (ROM). An interpreter is a program which takes commands one at a time, analyses them and then performs the required action. For an interpreter to work the text of a program must be held in memory.

Implementing BASIC by means of an interpreter has several distinct advantages.

(a) Programs can be entered and executed without the need to carry out (a lengthy) translation known as a compilation.

(b) Because the text of the program is held in memory it is possible to give better diagnostics when there is an error. For example if a program stops with an execution error then the user can examine the values of variables to determine what has gone wrong.

However an interpreter has some significant disadvantages.

(a) They are slower than compiled code.

(b) Syntax errors are only detected when the code is executed.

(c) Adding comments makes the text held in memory larger and may slow down the program.

A compiler works by translating (compiling) the BASIC program into some other language; either machine code or as in the case of this compiler, some intermediate code.

Compilers have the following advantages.

(a) They produce faster code, i.e. the output of the compiler is code which executes faster than interpreting the BASIC text.

(b) The compiler does a complete analysis of the program as it is compiled. Thus syntax errors are detected before the program is executed.

(c) The use of comments does not effect the finished code.

(d) The layout of the program does not effect the speed of execution, e.g. frequently used lines need not be put at the start of the program.

However compiling the program can take a considerable amount of time and diagnostics are generally not so good. For example when an error occurs in a compiled program it is impossible to determine the values of variables since the text of the program is no longer available.

OASIS BASIC has been designed to be highly compatible with DRAGON BASIC so that programs can be written and debugged using the interpreter. Once a program is correct (!) it can be compiled to produce faster code.

2. OPERATING INSTRUCTIONS.

There are three phases in producing and using a compiled BASIC program.

1. Program development and debugging using the existing interpreter.

2. Compilation, i.e. translation into the intermediate code.

3. Executing the intermediate code.

2.1 PROGRAM DEVELOPMENT

Programs should be developed and fully tested using the DRAGON BASIC interpreter. Once this has been achieved any modifications needed to make the program compatible with OASIS BASIC should be carried out. The program should then be renumbered and saved on a cassette in ASCII format.

This can be achieved by using RENUM to give a program starting with line 0 and have line increments of 1.

For example given

```
10 REM
20 REM   A SIMPLE TEST PROGRAM
30 REM
40 CLS
50 PRINT @230,"NUMBER PLEASE";
60 INPUT N
70 IF N=255 THEN @PRINT N
80 CLS
90 END
```

We can use:- RENUM 0,10,1 to get

```
0  REM
1  REM   A SIMPLE TEST PROGRAM
2  REM
3  CLS
4  PRINT @230,"NUMBER PLEASE";
5  INPUT N
6  IF N=255 THEN @PRINT N
7  CLS
8  END
```

This can then be saved on a cassette by using:- CSAVE "TEST",A. You should remember the line number of the last line of the program.

2.2 COMPILATIOI

During compilation the program is analysed and translated into an intermediate form. If any errors are discovered during compilation these are reported and no translated form is produced. If, however, no errors are detected a complete translated program is produced and this may be saved and executed.

If errors are detected then the DRAGON BASIC interpreter will have to be used to remove them and the program resaved on cassette.

Compilation can be achieved as follows:-

(a) Insert the compiler tape.

(b) Load the CONTROL program using CLOAD.

(c) Execute this program using RUN. The compiler will then be loaded.

(d) When the compiler has loaded, the cassette containing the program to be compiled should be inserted.

At this stage the compiler tape should NOT be rewound as a linker program is now the next item on the tape.

(e) The compiler will now execute and you will be asked if output is to go to the screen or to a printer. You will also be asked for the maximum line number of the program.

(f) As the program is compiled any errors in syntax will be detected and reported. For our example program the compiler produces the following output to the printer.

```
0  REM
1  REM    A SIMPLE TEST PROGRAM
2  REM
3  CLS
4  PRINT @230,"NUMBER PLEASE";
5  INPUT N
6  IF N=255 THEN @PRINT N
*****************↑209
7  CLS
8  END
COMPILATION COMPLETE
001 error reported
```

Line 6 contains a syntax error, i.e. there should be no @ character. With the interpreter this error would only be detected if the user typed in 255 in response to the INPUT on line 5.

If the output is being produced on the screen the compiler will wait for a key to be pressed after reporting each error. A full list of compilation errors is given in Appendix A.

(g) If any errors are detected the compilation process will go no further.

(h) If no errors are detected you will be requested to re-insert the compiler tape. The LINKER will then load as two programs.

(i) If all goes well you will be given the option of saving the compiled program and then given the option of executing it.

Once a program has been executed it may or may not be possible to re-execute it using EXEC.

If the program is saved on a tape then it can be loaded using CLOADM and executed using EXEC.

During compilation problems may arise if your program is too large or there is a tape failure. A tape failure will be indicated by a suitable message. If your program is too large the compiler may fail with the following message:-

```
0 compiler failure
```

This means that your program has too many variable names to be handled by the compiler.

2.3 PROGRAM EXECUTION

Once a translated program has been executed it may terminate in one of three ways.

(a) It may terminate correctly and may be re-executed using EXEC.

(b) It may terminate correctly but have corrupted the DRAGON BASIC system so that no commands can be executed. In particular EXEC will not work.

(c) The program may stop because of an execution error. A list of possible errors are given in Appendix B. Provided the N- directive (see 3.8) has not been used, the line number of the line causing the error will also be reported. In some cases the normal DRAGON BASIC execution errors will be produced.

3. DETAILS

In theory it should be possible to write a compiler to compile any DRAGON BASIC program; however in practise there are many considerations which make such an undertaking extremely difficult. OASIS BASIC is highly compatible with DRAGON BASIC but there are differences and certain DRAGON BASIC features have not been implemented. A description of OASIS BASIC is given in Appendix C and a summary of the differences is given in Appendix D.

In this section we want to look at these differences in more detail and discuss their effects.

3.1 INTEGERS

Version 1.0 of OASIS BASIC uses integers rather than floating point to represent numbers, i.e. all numbers must be whole numbers in the range -32768 to 32767. This means that any program which must use real numbers or numbers outside this range will not work with this version of OASIS BASIC.

Even if a program uses real numbers it may be possible to scale all the numbers to make them integers. For example a program which manipulates sums of money held as a number of pounds could be changed to hold sums of money as a number of pence. Thus instead of holding £1.40 as 1.40 it could be held as 140.

Many programs use real numbers with RND (e.g. IF RND (0) > 0.22 THEN..) however these can easily be changed (e.g. IF RND (100) > 22 THEN..).

3.2 VARIABLE NAMES ARE NOT AVAILABLE AT RUN TIME

The code produced by the compiler does not contain the names of the variables used in the program; just their addresses in memory. Thus commands which must use variable names cannot be implemented.

In particular the X subcommand of PLAY and DRAW will not work and will cause an execution error. This problem can be overcome by either using multiple PLAY's and DRAW's, or by concatenating strings.

For example we could replace

```
.                   DRAW "XA$;XA$;XA$"
```

```
.                   by DRAW A$ + A$ + A$
```

```
.                   or DRAW A$ : DRAW A$ : DRAW A$
```

3.3 STRINGS

With DRAGON BASIC strings may be between 0 and 255 characters long; the system allocates the exact amount of space needed to hold a string variable. With OASIS BASIC each string variable or component of a string array is given a fixed location in memory. Allocating 255 bytes for each string, so that it could hold up to 255 characters would be extremely wasteful.

Instead OASIS BASIC allocates a smaller default number of bytes (32) with the option that the programmer can request more or less. To ensure compatibility with DRAGON BASIC a special kind of comment called a DIRECTIVE is used to inform the computer of the size required.

A directive is a comment whose first character is $. To specify the size of a string variable or a string array we simply enclose the size in square brackets. For example.

```
.                REM$ A$ [255] , X$ (10,10) [10]
```

causes the compiler to leave 255 bytes for A$ and 10 bytes for each component of the two dimensional array X$.

For such a directive to be effective it must be the first textual occurrence of the string variable or string array identifier. In the case of a string array such a directive serves as a DIM definition, but an ordinary DIM should be included for compatibility with DRAGON BASIC.

3.4 DATA

Since the text of the program is not available in the compiled code, all items in data statements must be converted to their internal forms during compilation. This means that it is impossible to treat an integer in a data statement as a string by reading it into a string variable.

To avoid ambiguity OASIS BASIC requies all strings in data statements to be enclosed in quotes and any attempt to read a string as an integer, or an integer as a string causes an execution error.

For example

```
.                   DATA 12,XYZ,13,A
```

must be written as

```
.                   DATA 12, "XYZ", 13, "A"
```

and any attempt to read 12 or 13 as strings will fail. Of course we could write the above as

```
.                   DATA "12", "XYZ", "13", "A"
```

but then we could not read "12" and "13" as integers.

3.5 INPUT/OUTPUT

Those input and output commands of DRAGON BASIC which are implemented in OASIS BASIC have essentially the same meaning However, there are two minor differences.

(a) The expression in the INPUT and LINE INPUT commands of OASIS BASIC are slightly more general than in DRAGON BASIC. They can be arbitrary string expressions enclosed in parentheses. For example

```
.                   INPUT (LEFT$(A$,N)); X
```

(b) In DRAGON BASIC an INPUT command requesting values for several variables can be satisfied by typing values separated by commas. Thus in response to

```
.                   INPUT X,Y,Z
```

We can type

```
.                   12, 13, 14 <RET>
```

In OASIS BASIC such an input is divided into 3 separate inputs, but typing a comma is equivalent to pressing RETURN. We can still type 12, 13, 14 <RET> but the visual effect will be slightly different.

3.6 MISSING FEATURES

Those features of DRAGON BASIC which have not been included in OASIS BASIC can be divided into three classes.

(a) System commands (e.g. CONT, EDIT).

Strictly speaking these commands are not part of BASIC, but are commands to the underlying BASIC interpreter. As such it is inappropriate to compile them.

(b) Cassette and printer commands

These commands may be included in a future version.

(c) Floating point functions (SIN,SQR etc)

Since the compiler uses integers there is no point including these functions.

3.7 GRAPHICS COMMANDS

The graphics commands of OASIS BASIC are essentially the same as those of DRAGON BASIC. As already indicated (3.2) the X subcommand will not function with DRAW. However the major difference is in the syntax of the CIRCLE command.

In DRAGON BASIC the last three parameters of CIRCLE (h,s and e) must be real numbers. Clearly in OASIS BASIC we cannot supply real numbers. Instead OASIS BASIC accepts integer values for these parameters which are 64 times the equivalent DRAGON BASIC values.

For example if we require 0.5 for DRAGON BASIC we must supply 32 for OASIS BASIC.

To remain compatible with DRAGON BASIC, OASIS BASIC has a special format for these parameters, namely

```
.                   (e)/64 where e is a numeric expression
```

In OASIS BASIC this expression produces an integer which is used directly by CIRCLE; the 64 is ignored. In DRAGON BASIC however the value of e is divided by 64 and passed to CIRCLE.

The normal DRAGON BASIC command

```
.       CIRCLE (10,10),10,,2,0.5,1
```

can be written as

```
.       CIRCLE (10,10),10,,(128)/64,(32)/64,(64)/64
```


3.8 DIRECTIVES

As already indicated in 3.3 OASIS BASIC supports directives (i.e. special comments) which are used by the compiler. In 3.3 we saw how directives could be used to give details of the maximum size of string variables to the compiler.

The current version of OASIS BASIC also supports three other kinds of directives known as compiler options. These options have the form of a letter (C,N or L) followed by + to enable the option and - to disable the option.

THE L OPTION

The L option controls the production of a listing from the compiler.

Normally the compiler produces a listing to either the screen or a printer. L- can be used to stop the listing and L+ to start the listing again.

THE C OPTION

Normally the compiler generates code to check that the subscripts of an array are within range. For example given

```
.                   DIM X (10)
```

The assignment

```
.                   X(11) = 15
```

will generate a runtime error (C) since 11 is not a valid subscript.

This checking requires extra code and slows down the execution of the compiled program. The C option can be used to direct the compiler not to generate these checks. C- will cause the compiler to stop generating checks and C+ will cause it to resume generation.

WARNING: If checks are switched off then using an invalid subscript will not be detected and will produce some unpredictable effect. Checking should only be removed if you are certain (?) that the program is correct and speed is very important.

THE N OPTION

Normally the compiler generates code to record the line number of the currently executing line. Again this code requires space and slows down the execution of the program. The compiler can be directed to stop generating this code by using N- and to resume by using N+

Again stopping the generation of this code should only be done if absolutely necessary. If an error occurs on a line for which this code has not been generated then the error message will contain the number of the last line executed for which the code was generated.

For example consider the following program.

```
0  CLEAR 2000
1  CLS
2  DIM INFORMATION$(9)
3  PRINT
4  INPUT "STRING";ST$
5  INPUT "INDEX";INDEX
6  CLS
7  INFORMATION$(INDEX)=ST$
8  FOR I= 0 TO 9
9  PRINT I;" -- "; INFORMATION$(I)
10 NEXT I
11 GOTO 3
```

This program keeps details of 10 strings and allows the user to change any of them. This is not a very well written program since the user can make it fail by typing an invalid index value. In DRAGON BASIC an invalid subscript generates the error

```
                    ? BS ERROR IN 7
```

With OASIS BASIC we get an error C in line 7. Also OASIS BASIC will generate error E on line 4 if any attempt is made to input a string longer than 32 characters, i.e. ST$ can only hold strings up to 32 characters long. We can add directives to this program to get the following.

```
0  CLEAR 2000
1  REM$ L-,N-,C-,ST$[254]
2  CLS
3  DIM INFORMATION$(9)
4  PRINT
5  INPUT "STRING";ST$
6  INPUT "INDEX";INDEX
7  CLS
8  REM$ INFORMATION$(9) [254]
9  INFORMATION$(INDEX)=ST$
10 FOR I= 0 TO 9
11 PRINT I;" -- ";INFORMATION$(I)
12 NEXT I
13 REM$ L+,N+,C+
14 GOTO 4
```

As far as DRAGON BASIC is concerned this program behaves in exactly the same way as the original. However with OASIS BASIC the directives instruct the compiler to produce slightly different code than for the original.

Firstly the directives in line 1 instruct the compiler to switch off listing, range checking and line number generation. These are switched on again by the directives in line 13. It should be noted that these operations are performed during compilation and NOT during execution. Thus lines 2 to 13 will contain no code to check subscripts nor will it contain code to generate line numbers. When an execution error occurs the error will be reported on either line 1 or line 14 depending on which one was last executed. Inputting an invalid subscript will have an unpredictable effect.

The compiler output produced for this program is given in the following listing. The L- directive in line 1 switches off listing but line 1 itself is listed. The L+ directive in line 13 switches on listing but line 13 itself is not listed. Line 8 is listed because it contains a warning message. Lines containing errors or warnings are always listed.

```
0  CLEAR 2000
1  REM$ L-,N-,C-,ST$[254]
8  REM$ INFORMATION$(9)[254]
**********************↑233
14 GOTO 4
COMPILATION COMPLETE
no errors reported.
```

The directive ST$ [254] in line 1 instructs the compiler to leave room for strings of up to 254 characters for ST$. However the directive in line 8 occurs textually after the first occurrence of INFORMATION$ in line 3 and thus is ignored. Thus the components of INFORMATION$ can only be up to 32 characters long. Now if a string of greater than 32 characters is input there is sufficient room in ST$ but not in INFORMATION$ (INDEX) and the assignment on line 9 will fail.

3.9 MACHINE CODE PROGRAMMING

One of the benefits of using a compiler is that programs execute faster. However there are still times when they are not fast enough or when there are things to be done which cannot be done in BASIC.

To achieve this, machine code has to be used. In this section we look at the layout of compiled program and how machine code may be incorporated into compiled programs.

3.9.1 PROGRAM LAYOUT

When a program is compiled using OASIS BASIC it will use all the RAM memory except that reserved by PCLEAR. A program containing PCLEAR 0 will use all available RAM. A program containing no PCLEAR will use all memory except graphics pages 1 to 4.

With OASIS BASIC the CLEAR command is ignored and cannot be used to reserve memory for machine code routines. Any attempt to place machine code at the top of RAM will corrupt the compiled program and have an unpredictable effect on the excution of the program.

3.9.2 MACHINE CODE STORAGE

A machine code routine can still be used with OASIS BASIC by either placing it in a reserved graphics page or by reading it into a numeric array within the program. If code is to be placed into a numeric array we must place two bytes per component, i.e we must assign a value (256* byte one + byte two) to each component of the array.

For example, the following machine code routine can be used to invert the text screen.

```
8E,04,00,A6,84,81,80,25,04,88,0F,20,02,88,40,A7,80,8C
05,FF,23,ED,39
```

We can either read it into a graphics page (say page 1) or we can read it into a numeric array (say X). Since this routine has 23 bytes the array will need 12 components, i.e DIM X(11). The following program reads the routine into page 1 of graphics memory.

```
0  PCLEAR 1
1  REM
2  REM READ ROUTINE IN GRAPHICS PAGE 1
3  REM
4  FOR I = 1536 TO 1536 + 22
5  READ CODE
6  POKE I,CODE
7  NEXT I
8  REM
9  REM TEST THE ROUTINE
10 REM
11 FOR I = 1 TO 10
12 EXEC 1536
13 FOR J = 1 TO 100 : NEXT J ' WAIT
14 NEXT I
15 END
16 REM
17 REM DATA FOR THE ROUTINE
18 REM
19 DATA &H8E,&H04,&H00,&HA6,&H84,&H81,&H80,&H25,&H04,&H88,&H0F
&H20
20 DATA &H02,&H88,&H40,&HA7,&H80,&H8C,&H05,&HFF,&H23,&HED,&H39
COMPILATION COMPLETE
no errors reported.
```

The next program reads it into the array X.

```
0  PCLEAR 1
1  DIM X(11)
2  REM
3  REM READ ROUTINE IN GRAPHICS PAGE 1
4  REM
5  FOR I = 0 TO 11
6  READ X(I)
7  NEXT I
8  REM
9  REM TEST THE ROUTINE
10 REM
```



```
11 FOR I = 1 TO 10
12 EXEC @X
13 FOR J = 1 TO 100 : NEXT J ' WAIT
14 NEXT I
15 END
16 REM
17 REM DATA FOR THE ROUTINE
18 REM
19 DATA &H8E04,&H00A6,&H8481,&H8025,&H0488,&H0F20
20 DATA &H0288,&H40A7,&H808C,&H05FF,&H23ED,&H3900
COMPILATION COMPLETE
no errors reported
```

3.9.3 EXECUTION OF MACHINE CODE

If a routine is held in a graphics page or indeed somewhere with a known address (e.g. a routine in ROM) then an ordinary EXEC can be used to execute the routine. For example our routine in Page 1 of graphics memory can be executed by using

```
.                EXEC 1536
```

If a routine is held in an array then it can be executed by using a special OASIS BASIC version of EXEC. For example if our routine is held in the array X we can execute it by using

```
.                EXEC @X
```

3.9.4 ADDRESSES

Since OASIS BASIC only permits integers in the range -32768 to 32767 then all 65535 locations of memory cannot be addressed with positive numbers. Memory locations 0 to 32767 are addressed with integers 0 to 32767 and locations 32768 to 65535 are addressed with the integers -32768 to -1 respectively. In fact the hexidecimal numbers &H8000 to &HFFFF represent the integers -32768 to -1 respectively.

3.9.5 USR FUNCTIONS

The machine code for USR functions can be held either in graphics memory or in a numeric array. To specify which, a USR definition can have one of two forms. For example

```
.                DEF USR0 = 1536
.                DEF USR0 = X
```

The first states that the machine code for USR0 is at the start of graphics page 1 and the second states that it is in array X.

When the code for a USR function is executed register D contains the parameter and the result should be left in D. VAPTR can be used to leave the address of a variable in D rather than its contents.

For example the following program places the machine routine

```
.                ADDD  # 1
.                RTS
```

into graphics page 1 and associates it with USR9. Note that this routine simply adds one to the value in D and returns that value. Thus X =USR9(X) is equivalent to X = X + 1 except that 32767 + 1 causes overflow but USR9(32767) gives -32768.

```
0  PCLEAR 1
1  REM
2  REM READ ROUTINE INTO GRAPHICS PAGE 1
3  REM
4  FOR I = 1536 TO 1536 + 3
5  READ CODE
6  POKE I,CODE
7  NEXT I
8  DATA &HC3,&H00,&H01,&H39
9  REM
10 REM DEFINE USR9
11 REM
12 DEFUSR9 = 1536
13 REM
14 REM TEST USR9
15 REM
16 INPUT "NUMBER PLEASE";X
17 IF X = 0 THEN END
18 PRINT "USR9(";X;") = ";USR9(X)
19 PRINT "X = ";X
20 GOTO 16
COMPILATION COMPLETE
no errors reported
```

By contrast the following program places the machine code

```
.                       TFR  D,X
.                       LDD  , X
.                       ADDD # 1
.                       STD  , X
.                       RTS
```

into the array X and associates it with USR9. Note that this routine actually adds one to the contents of a variable. Thus Z = USR9(VAPTR(X)) is equivalent to X = X + 1 : Z = X except that overflow cannot occur.

```
0  PCLEAR 1
1  DIM X(4)
2  REM
3  REM READ ROUTINE INTO X
4  REM
5  FOR I = 0 TO 4
6  READ X(I)
7  NEXT I
8  DATA &H1F01,&HEC84,&HC300,&H01ED,&H8439
9  REM
10 REM DEFINE USR9
11 REM
12 DEFUSR9 = X
13 REM
14 REM TEST USR9
15 REM
```



```
16 INPUT "NUMBER PLEASE"; X
17 IF X = 0 THEN END
18 PRINT "USR9(";X;") = ";USR9(VAPTR(X))
19 PRINT "X = ";X
20 GOTO 16
COMPILATION COMPLETE
no errors reported
```

3.9.6 VARIABLE REPRESENTATIONS

An integer variable is represented as two bytes. A string variable is represented as N + 1 bytes where N is the maximum size of string the variable can hold. The first byte is the size of the current string value held in the variable. Normally a string variable will be represented as 33 bytes.

For example given A$ represented as 33 bytes A$ = "ABC" will put 3 in the 1st byte, ASC ("A") in the 2nd, ASC ("B") in the 3rd and ASC ("C") in the 4th. The contents of the other bytes will be undefined.

An array variable with N components is represented as N consecutive blocks of bytes. In the case of integer arrays each block is two bytes long. In the case of string arrays the size of the block will be 1 + the maximum string size for the array.

3.10 A WORKED EXAMPLE

The following program draws a disk at the centre of the screen and plays a tune.

```
10  PCLEAR 8
20  REM
30  REM      SET UP GRAPHICS MODE
40  REM
50  PMODE4,1:PCLS:SCREEN1,1
60  REM
70  REM      DRAW DISC
80  REM
90  CIRCLE(127,95),50,,0.5
100 PAINT(127,95),5,5
110 REM
120 REM    PLAY TUNE
130 A$ = "O2L4GG;L2GDL4BB;L2BGL4GB;O3L2DDL4CO2B;L1AL4AB;
O3L2CCO2L4BA;L2BGL4GB;L2ADL4F#A;L1G;"
140 REM
150 FOR I= 8 TO 16 STEP 2
160 PLAY "T"+STR$(I)+"XA$;XA$;"
170 NEXT
180 CLS:END
```

While this program is a correct DRAGON BASIC program, it is not a correct OASIS BASIC program. To see this, the program can be renumbered to give the following program and then saved in ASCII format for compiling.

```
0  PCLEAR 8
1  REM
2  REM     SET UP GRAPHICS MODE
3  REM
4  PMODE4,1:PCLS:SCREEN1,1
5  REM
6  REM    DRAW DISC
7  REM
8  CIRCLE(127,95),50,,0.5
9  PAINT(127,95),5,5
10 REM
11 REM   PLAY TUNE
12 A$ = "O2L4GG;L2GDL4BB;L2BGL4GB;O3L2DDL4CO2B;L1AL4AB;
O3L2CCO2L4BA;L2BGL4GB;L2ADL4F#A;L1G;"
13 REM
14 FOR I= 8 TO 16 STEP 2
15 PLAY "T" +STR$(I)+"XA$;XA$;"
16 NEXT
17 CLS:END
```

Compiling this program produces the following listing.

```
0  PCLEAR 8
1  REM
2  REM       SET UP GRAPHICS MODE
3  REM
4  PMODE4,1:PCLS:SCREEN1,1
5  REM
6  REM    DRAW DISC
7  REM
8  CIRCLE(127,95),50,,0.5
**********************^008
9  PAINT(127,95),5,5
10 REM
11 REM     PLAY TUNE
12 A$ = "O2L4GG;L2GDL4BB;L2BGL4GB;O3L2DDL4CO2B;L1AL4AB;
O3L2CCO2L4BA;L2BGL4GB;L2ADL4F#A;L1G;"
13 REM
14 FOR I= 8 TO 16 STEP 2
15 PLAY "T"+STR$(I)+"XA$;XA$;"
16 NEXT
******^036
17 CLS:END
COMPILATION COMPLETE
002 errors reported
```

From this listing we can see that the program contains two errors.

. Line 8 needs the special syntax for CIRCLE. we need

. 8 CIRCLE (127,95),50,, (32)/64

. Line 16 needs an identifier after the NEXT. We need

. 16 NEXT I

Changing these lines, the compilation then works and produces the following listing.


```
0  PCLEAR 8
1  REM
2  REM    SET UP GRAPHICS MODE
3  REM
4  PMODE4,1:PCLS:SCREEN1,1
5  REM
6  REM    DRAW DISC
7  REM
8  CIRCLE(127,95),50,,(32)/64
9  PAINT(127,95),5,5
10 REM
11 REM    PLAY TUNE
12 A$ = "O2L4GG;L2GDL4BB;L2BGL4GB;O3L2DDL4CO2B;L1AL4AB;
O3L2CCO2LBA;L2BGL4GB;L2ADL4F#A;L1G;"
13 REM
14 FOR I= 8 TO 16 STEP 2
15 PLAY "T"+STR$(I)+"XA$;XA$;"
16 NEXT I
17 CLS:END
COMPILATION COMPLETE
no errors reported
```

However executing the resultant program causes an execution error E on line 12. The string variable A$ is not large enough to hold an 82 character string value. This can be fixed by inserting a directive to tell the compiler to leave space for 82 characters in A$,i.e. we need the directive REM$ A$[82]. Compiling this program we get the following listing.

```
0  PCLEAR 8
1  REM
2  REM    SET UP GRAPHICS MODE
3  REM
4  PMODE4,1:PCLS:SCREEN1,1
5  REM
6  REM    DRAW DISC
7  REM
8  CIRCLE(127,95),50,,(32)/64
9  PAINT(127,95),5,5
10 REM
11 REM   PLAY TUNE
12 REM$ A$[82]
13 A$ = "O2L4GG;L2GDL4BB;L2BGL4GB;O3L2DDL4CO2B;L1AL4AB;
O3L2CCO2L4BA;L2BGL4GB;L2ADL4F#A;L1G;"
14 REM
15 FOR I= 8 TO 16 STEP 2
16 PLAY "T"+STR$(I)+"XA$;XA$;"
17 NEXT I
18 CLS:END
COMPILATION COMPLETE :
no errors reported
```

However executing this program causes the execution error

. ?FC ERROR IN 16

This is a DRAGON BASIC error and indicates an attempt to use the X subcommand in PLAY on line 16.

Line 16 can be changed to 16 PLAY "T" +STR$(I)+ ";"+A$ + A$ to give the following completely correct program.

```
0  PCLEAR 8
1  REM
2  REM      SET UP GRAPHICS MODE
3  REM
4  PMODE4,1:PCLS:SCREEN,1
5  REM
6  REM    DRAW DISC
7  REM
8  CIRCLE(127,95),50,,(32)/64
9  PAINT(127,95),5,5
10 REM
11 REM  PLAY TUNE
12 REM$ A$ [82]
13 A$ = "O2L4GG;L2GDL4BB;L2BGL4GB;O3L2DDL4CO2B;L1AL4AB;
O3L2CCO2L4BA;L2BGL4GB;L2ADL4F#A;L1G;"
14 REM
15 FOR I= 8 TO 16 STEP 2
16 PLAY "T"+STR$(I)+";"+A$+A$
17 NEXT I
18 CLS:END
COMPILATION COMPLETE  :
no errors reported
```

4 PROBLEM REPORTING

This product is sold on an "AS IS" basis and OASIS Software make no warrant that the software programs or user manual will be free from error or will meet any requirements. However OASIS Software will replace any cassette which fails to load.

Every effort has been made to ensure that the OASIS BASIC compiler and run time system are correct and free from error. However with software of this complexity it is impossible to ensure absolute correctness.

OASIS Software intend to maintain and develop OASIS BASIC and will correct any problems in future versions of this software. To aid in this process, response from users would be extremely useful. If you have any problems or comments about the software or its manual please let us know.

If you have any problems compiling or executing a Program please send a copy of the program plus a complete description of the problem. Cassettes will be returned but you should retain sufficient copies of the program.

5. THE DEMONSTRATION PROGRAM

In order to demonstrate the full power of the compiler we have included an example of a moderately comprehensive BASIC program before and after compiling.

You will notice that sound commands are NOT speeded up and so sections of code which are rich in sound do not seem to benefit as much as others.

To run the BASIC program on side A of the demonstration tape type :-

CLOAD"" followed by ENTER

This is the loader program so when this is loaded type:-

RUN followed by ENTER

Once the loader has loaded type :-

RUN followed by ENTER to execute.

To run the compiled program on side B repeat the above but it will be found that only the first 'RUN' is required.

Instructions to the game itself are included in the program.

Appendix A Compilation error codes

Code		Error description
000	:	Integer constant expected.
001	:	String constant expected.
002	:	"(" expected.
003	:	")" expected.
004	:	"[" expected
005	:	"]" expected.
006	:	"," expected.
007	:	";" expected.
008	:	":" expected.
009	:	"@" expected.
010	:	ELSE expected
011	:	TO expected
012	:	THEN expected
013	:	STEP expected
014	:	End of line expected
015	:	Name reserved in DRAGON BASIC.
017	:	NOT expected
018	:	AND or OR expected.
019	:	Relational operator expected.
020	:	Multiplication operator expected.
021	:	Adding operator expected.
036	:	Integer identifier expected.
037	:	String identifier expected.
200	:	Integer too large.
201	:	No closing quotes for string.
202	:	DRAGON BASIC feature not supported.
203	:	Program too large.
204	:	Unary + or - can only be applied to integers.
205	:	This operation may not be applied to strings.
206	:	Operands of different types.
207	:	Integer expression expected
208	:	String expression expected.
209	:	Invalid start of command.
210	:	Line number not in sequence
211	:	Invalid item in DATA statement
212	:	Line number expected.
213	:	2nd PCLEAR or value out of range.
214	:	Line number out of range.
215	:	Action in PUT wrong.
216	:	Array in PUT or GET must have two dimensions.
217	:	G parameter expected.
218	:	PSET or PRESET expected.
219	:	B or BF expected.
220	:	"-" expected
221	:	"=" expected.
222	:	GOTO or GOSUB expected.
223	:	String or string variable expected.
224	:	";","," or end of statement expected.
225	:	Identifier expected.
226	:	Expression and variable of different types.

227	:	Error in FACTOR
228	:	FN or USR function not defined.
229	:	TIMER may only be set to zero.
230	:	DATA string too long.
231	:	"/" expected.
232	:	"64" expected.
233	:	Array already in use -- definition ignored.
234	:	Invalid directive.
235	:	String variable encountered before - definition ignored
236	:	FN or USR definition expected.
237	:	Function already defined.
238	:	A directive must be the last command on a line.
239	:	String too large.
240	:	INLINE integer must be between 0 and 255.
241	:	Invalid INLINE item.
242	:	String variable expected.
243	:	"H" expected.
244	:	"ELSE" or end of line expected.
245	:	Hex digit expected.
246	:	Too many dimensions to an array.
247	:	Too few subscripts.
248	:	Too many subscripts.
249	:	Invalid put option.

Appendix B Execution error codes

Code		Error description
A	:	Integer overflow
B	:	Runtime stack overflow.
C	:	Range violation (e.g. subscript of array out of range.)
D	:	RETURN with no GOSUB.
E	:	String value too long.
F	:	Division by zero.
G	:	Value less than zero (e.g. ON-GOSUB or ON-GOTO value less than zero.)
H	:	Illegal character (e.g. trying to use CHR$ on an integer which is too large.)
I	:	Empty string value.
J	:	RETURN missing.
K	:	Zero step in FOR command.
L	:	NEXT without FOR
M	:	
N	:	Error in READ; no data left.
O	:	Attempt to read string as an integer.
P	:	Attempt to read an integer as a string.
Q	:	Argument to standard function or command out of range.
R	:	Error in PMODE; insufficient memory for graphics mode and/or graphics page used by program.

NOTE: Standard DRAGON BASIC error messages may also be produced.

Appendix C OASIS BASIC description

This appendix summarizes the syntax of OASIS BASIC using EBNF (Extended Backus-Naur Form). EBNF is used to give a series of (production) rules for the various objects of the program. Thus section 1 defines how programs are constructed out of lines.

With this notation the following conventions are used.

. Objects in double quotes represent themselves.

E.G. "READ" implies that READ will be part of the program.

. Alternatives are separated by | .

```
E.G compound-command = if-command|
.                      for-command |
.                      next.|
```

implies that a compound-command can either be an if-command, a for-command or a next.

. Definitions are terminated by a period.

. Items enclosed in { } may be repeated zero or more times.

. Items enclosed in [] may be repeated zero or once.

. Items of the form (a|b|..|z) mean that one of a,b,..,z must be selected.

The following is a description of OASIS BASIC.

1. PROGRAMS

A program consists of a series of numbered lines 0,1,..,N. Each line consists of the line number followed by one or more commands separated by colons.

```
program = line {line}.

line = line-number command-sequence.

command-sequence = command {":" command}.

line-number = integer.

integer = digit {digit} | "&H" hex-digit {hex-digit}.

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
        "9".

hex-digit = digit | "A" | "B" | "C" | "D" | "E" | "F" .
```

NOTE

THE N+1 lines of the program must be numbered 0,1,2,..,N. This can be achieved by using RENUM.

2. EXPRESSIONS

An expression can compute either an integer or a string value.

```
expression = and-expression {"OR" and-expression}.

and-expression = sub-expression {"AND" sub-expression}.

sub-expression = ["NOT"] sub-expression |
.                relational-expression.

relational-expression = simple-expression {relational-operator
.....                                      simple-expression}

simple-expression = term {adding-operator term}.

term = factor {multiplying-operator factor}.

factor = unary-operator factor | "(" expression ")" |
.        function-call | variable | integer | string.

relational-operator = "<" | ">" | "<=" | " >=" | "=" | "<>" |
.                     "><" | "=>" | "=<".

adding-operator ="+" | "-".

multiplying-operator = "*" | "/" | "\".

unary-operator = "+" | "-".

variable = identifier [subscripts].

subscripts = "(" expression {"," expression} ")".

string = '"' any-sequence-of-characters-except-" '"'.

function-call = identifier parameter-list.

parameter-list = "(" expression {"," expression} ")".

identifier = letter {(letter | digit)} ["$"].

letter = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
.        "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
.        "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
```

The available functions are:-

ASC	CHR$	INKEY$	STR$	VAL	ABS
FIX	INT	JOYSTK	PEEK	POINT	PPOINT
RND	SGN	VAPTR	LEFT$	RIGHT$	MID$
LEN	POS	HEX$	INSTR	STRING$	
USR0 ..	USR9	FNA .. FNZ			

The following standard variable exists :-

TIMER

Notes

(a) FNA..FNZ and USR0.. USR9 are user defined functions.

(b) Numeric expressions and variables use integer values in the range -32768..32767.

(c) "/" is integer division and "\" is integer remainder.

(d) INT and FIX are the identity function, i.e. INT (e) = FIX(e) = e. However they are included for compatibility, e.g. the following code can still be used to find the remainder of dividing X by Y.

```
X - INT(X/Y) * Y.
```

(e) As with DRAGON BASIC only "+" can be used with strings.

(f) RND may only take parameters >= 1.

(g) PEEK takes one integer parameter in the range -32768..32767. Negative values map onto values in the range 32768..65535,i.e. -1 = 65535 ,.., -32768 = 32768.

3. COMMANDS

command = BASIC-command | sound-command | graphics-command.

3.0 SOUND COMMANDS

Both PLAY and SOUND are supported.

(a) SOUND integer-expression, integer-expression.

(b) PLAY string-expression.

Note

All facilities of PLAY except execution of substrings are supported.

3.1 GRAPHICS COMMANDS

All graphics commands are supported.

(a) CIRCLE (x,y),r,c,h,s,e
. where x,y,r and c are integer expressions, and h,s and e
. are of the form (integer-expression) / 64.

(b) COLOR integer-expression, integer-expression.

(c) DRAW string-expression.

(d) GET (x1,y1)-(x2,y2),array,G

. PUT (x1,y1)-(x2,y2)array,action

. where x1,x2,y1 and y2 are integer expressions and
. "array" is a two dimesional integer array variable.

(e) LINE (x1,y1) - (x2,y2),a,b

. where x1,x2,y1 and y2 are integer expressions.

(f) PAINT (x,y),c,b.

. where x, y, c and b are integer expressions.

(g) PCLS c

. where c is an integer expression.

(h) PCOPY a TO b where a and b are an integer expression.

(i) PMODE m,p.

. where m and p are an integer expression.

(j) PRESET (x,y)

. PSET (x,y,c)

. RESET (x,y)

. SET (x,y,c)

. where x, y and c are an integer expressions.

(k) SCREEN t,s

. where t and s are an integer expression.

(l) PCLEAR integer.

Notes

(a) In DRAGON BASIC the last parameters of CIRCLE may be real
. values. However in this version of BASIC we require
. integer values. Using the above syntax maintains
. compatibility, i.e. (32)/64 represents 0.5 in DRAGON
. BASIC and 32 in this version of BASIC.

(b) All facilities of DRAW except the execution of substrings are supported.

(c) PCLEAR takes an integer parameter in the range 0 to 8.

3.2 BASIC COMMANDS

```
BASIC-command = definition | simple-command |
.               compound-command | read-command |
.               control-command | machine-comand |
.               comment | inline-command.
```

(a) Definitions

```
definition = "DEFFN" letter "(" identifier ")""=" expression |
             "DEFUSR" digit "="([("+" | "-")] integer |
.            @identifier )
.            ("REM$"|"'$'") directive {"," directive} |
.            "DIM" array-spec {"," array-spec}.
array-spec = identifier "(" integer {"," integer} ")".
directive = string-def | compiler-option.
string-def = (identifier | array-spec) "[" integer "]".
compiler-option = ("C" | "L" | "N")  ( "+" | "-").
```

Notes

(a) The dimensions of an array may not be defined by variables
(b) DEFUSR may be defined with an integer value in the range
. -32768..32767.

(c) A string definition is used to specify the size of a
. string variable or the size of components of the string
. array. If a string variable is encountered before such a
. definition it is given a default size (i.e. 32.)

For example REM$ A$[10], B$(20)[30] defines a variable A$ which can hold strings up to 10 characters long and a string array B$ of 21 components each of which can be a string of up to 30 characters. To be effective REM$ definitions should contain the first "textual" occurance of the variables being defined.

A compiler option is used to control the compiler. "+" is used to turn on the given option and "-" is used to switch it off.

C - controls generation of runtime checks.

L - controls the production of a listing.

N - controls the generation of code to record line numbers.

A directive command must be the last command on a line.

(b) SIMPLE COMMANDS

```
simple-command = "CLS" [expression] | "END" | "STOP" |
.                "CLEAR" integer ["," integer] |
.                input-command | print-command | assignment.
input-command =  "INPUT" [expression ";"] variable {","
.                variable} |
.                "LINE" "INPUT" [expression ";"] variable.
print-command =  ("PRINT"|"?") ["@" expression ","]
.                [print-list].
print-list =     expression {(";" | ",") expression}.
assignment =     ["LET"] variable "=" expression.
```

Notes

i) END and STOP both cause the program to terminate.

ii) CLEAR has no effect.

iii) The expression in an INPUT command is restricted to be
. a simple variable, a string or an arbitrary expression
. in parenthesis.

(c) CONTROL COMMANDS

```
Control-command = "GOSUB" line-number | "GOTO" line-number |
     expression "GOSUB" line-list |
"ON" expression "GOTO" line-list | "RETURN".
line-list = line-number {"," line-number}.
```

(d) COMPOUND COMMANDS

```
compound-command = if-command | for-command | next.
next = "NEXT" identifier {"," identifier}.
for-command = "FOR" identifier "=" expression "TO" expression
.             ["STEP" expression].
if-command = "IF" expression "THEN action ["ELSE" action].
action = line-number | command-sequence.
```

(e) MACHINE COMMANDS

```
machine-command = "EXEC"(expression |"@"identifier) |
                  "POKE"  expression "," expression.
```

Note

All expressions in machine commands are integer expressions in the ranges - 32768..32767 or 0..255. Negative values map onto integers in the range 32768..65535.

(f) COMMENTS

comment = ("REM" | "'") any-sequence-of-characters.

(g) READ COMMANDS

Read-command = "READ" variable {"," variable} | "RESTORE" |
"DATA" data-item {"," data-item}.

data-item = [("+" | "-")] integer | string.

Note

String items must be enclosed in quotes.

(h) INLINE COMMANDS

inline-command = "INLINE" inline-item {"," inline-item}.

inline-item = integer | identifier ["("")"].

An inline command can be used to embed p-codes in a program. An inline-item which is an integer must be in the range 0 to 255. If an inline-item is an identifier then the address of the variable is generated as 2 bytes.

Appendix D Summary of differences from DRAGON BASIC

The main differences are as follows:-

(a) Integer values are used; not floating point.

(b) String variables have a fixed maximum size. This size can be user defined or be the default size.

(c) Variables names cannot be used at runtime.

(d) Some new features (directives and the INLINE command) have been added.

(e) Some features of DRAGON BASIC have been omitted.

(f) Certain conventions on program layout need to be observed. For example reserved words and identifiers may not contain spaces.

Some detailed differences are as follows:-

Statement	comments
CLEAR•n,h	No effect. n and h must be integers.
CLS c	Same.
DATA	String values must be in quotes
DEF FN	same
DEFUSRnn=address	Address must be an integer in the range-32768.. 32767 or an array variable.
DIM	same
END	same
EXEC address	Address must be present and be in the range -32768..32767
FOR-NEXT	Same
GOSUB	Same
GOTO	Same
IF THEN ELSE	Essentially the same.
INPUT	Prompt expression more general.
LET	Same.
LINE INPUT	Prompt expression more general
ON GOSUB	Same
ON GOTO	Same
POKE location,value	Location must be an expression yielding a value in the range - 32768..32767.
PRINT	Same
PRINT TAB	Not included
PRINT USING	Not included
PRINT @	Same
READ	A numeric item may not be read as a string
RESTORE	Same
RETURN	Same
STOP	Same as END. CONT cannot be used to continue the program.
PLAY string	The string may not contain the X command. There will also be a limit on the size of the string.
SOUND	Same
AUDIO	Not included
CLOAD	Not included
CLOADM	Not included
CLOSE	Not included
CSAVE	Not included
CSAVEM	Not included
EOF	Not included
INPUT#-1	Not included
MOTOR	Not included
RENUM	Not included
NEW	Not included
LLIST	Not included
CONT	Not included
DEL	Not included
EDIT	Not included
LIST	Not included
RUN	Not included
OPEN	Not included
PRINT#-1	Not included

SKIPF	Not included
TRON	Not included
TROFF	Not included
PRINT#-2	Not included
CIRCLE(x,y),r,c,h,s,e	Special syntax for h,s and e.
COLOR	Same
DRAW string	The string may not contain the X command. There will also be a limit on the size of the string.
.	
.	
GET	Same
PUT	Same
LINE	Same
PAINT	Same
PCLEAR n	n must be an integer in the range 0..8
PCLS	Same
PCOPY	Same
PMODE	Same
PRESET	Same
PSET	Same
RESET	Same
SCREEN	Same
SET	Same
ASC	Same
CHR$	Same
HEX$	Same
INKEY$	Same
INSTR	Same
LEFT$	Same
LEN	Same
MID$	Same
RIGHT$	Same
STRING$	Same
STR$	Same
VAL	Same
ABS	Same
ATN	Not included
COS	Not included
EXP	Not included
FIX	Same
INT	Same
JOYSTK	Same
LOG	Not included
MEM	Not included
PEEK(n)	n must be an expression in the range -32768..32767
.	
POINT	Same
POS	Can only be used with the display
PPOINT	Same
RND(n)	n>=1
SGN	Same
SIN	Not included
SQR	Not included
TAN	Not included
TIMER	Same
USRnn	Same
VAPTR	Same

Appendix E Choice of Cassette recorder

To use Sprint effectively you will require a tape recorder which has a remote socket or which can be in some way controlled from the Dragon. Almost all tape recorders have this facility.

Some tape recorder mechanisms have more inertia than others and occasionally the tape will 'over-run' and ASCII programs will not load properly.

It is advisable therefore to remove the remote plug or type 'MOTOR ON' before making the ASCII copy of the BASIC program in the manner described on page 3. This is the only time the tape recorder should be used without the remote control.

If problems are still experienced it is possible to increase the size of the header of each block whilst ASCII saving by typing:-

POKE 144,4 followed by ENTER

before doing the ASCII save.

If problems are still experienced try POKE 144,5 then if that doesn't work POKE 144,6 and so on. If problems persist you will need to beg, borrow, buy or steal another cassette recorder.

Printed by **alpha★print**, Whitecross Road, Weston-super-Mare