THE SPECTRUM OF ZX BASIC

BASIC has become the standard language of microcomputers, but almost every machine has its own variation — or dialect. In this series of articles we will be looking at some of these variations and their functions, as well as explaining how they can be 'translated' from one dialect to another. This first article looks at the most widely used dialect — Sinclair BASIC.

We begin with variable names — always a source of confusion between BASIC dialects. In Sinclair BASIC, string variable names must have only one letter, and there is no distinction between upper and lower case letters. This means that the variables aS and AS refer to the same memory location. String array names follow the same rules as simple variables, and pre-empt them, so that once you've DIMensioned the string array H$, all further mentions of H$ in the program will be taken as referring to the array H$. This follows from the fact that Sinclair BASIC regards all string variables as array-type variables, some of them formally DIMensioned, and others not.

Numeric variable names are less constrained than those of string variables: they must begin with a letter, and they must consist of letters or digits, but they may be any length. They may include spaces, and they may be a mixture of upper and lower case letters, but although these factors are helpful to the programmer, they are of no significance to the machine, which will ignore them. Some valid numeric variable names are:

```
qwert, ub40, advanced computer course
```

and the following are exactly equivalent:

```
QWERT, UB 40, Advanced Computer Course
```

Numeric array names must be single letters, but this does not preclude numeric variables of the same name: the array variable v(8) is quite distinct from the simple numerical variable v. Single-letter non-array numerical variables such as v must be used as the counters of FOR...NEXT loops, so FOR V=1 to 9...NEXT V is legal, but FOR loop=1 TO 9 is illegal.

The main differences between the Sinclair dialect and other BASICS lie in the treatment of string quantities. Let us start with the effect of the DIM statement. In Sinclair BASIC, when the statement DIM a$(12) is executed, 12 bytes of memory are set aside exclusively for the use of the variable a$, and these bytes are initialised with spaces. Each of these bytes can be referred to as a subscripted variable, or the whole 12 bytes can be referred to collectively as a$. The length of this variable will always be 12, and assignments to it will be padded with spaces or truncated on the right as necessary to preserve this length. Suppose we write:

```
DIM a$(12): LET a$="123456789"
```

then a$ will actually contain the characters '123456789' followed by three spaces, making 12 characters in all. If we write instead:

```
DIM a$(12):LET a$="ABCDEFGHIJKLMN"
```

then a$ will actually contain only the 12 characters 'ABCDEFGHIJKL' — the string quantity 'ABCD EFGHIJKLMN' has been truncated on the right to fit into the DIMensioned length of a$. If we now write:

```
LET a$(2 TO 5)="1234"
```

then a$ will contain 'A1234FGHIJKL'. This shows the power of Sinclair string handling — all strings are treated as single-dimension string arrays, the arrays can be subscripted or not, and individual elements of an array can be accessed — singly or as part of a sub-string — by subscripts. It also shows another major divergence from other versions of BASIC. Elsewhere DIM a$(12) creates 12 separate string variables called a$(1), a$(2), etc., each of which has the length of the expression assigned to it. If nothing has been assigned to a particular string variable, then its length is 0, and it contains only the null string, "".

In other BASICS this way of handling strings requires the various string functions, LEFT$, RIGHT$, MID$, and sometimes INSTR, to enable sub-string manipulation and string-slicing in the way demonstrated. But this is not so in Sinclair BASIC. The Sinclair equivalents of these string functions are:

```
LEFT$(A$,N) = A$( TO N)
```

(meaning the N leftmost characters of A$);

```
RIGHT$(A$,N) = A$(LEN A$-N+1 TO )
```

(meaning the N rightmost characters of A$); and

```
MID$(A$,P,N) = A$(P TO P+N-1)
```

(meaning the N characters from position P onwards in A$).

```
LET S=INSTR(A$,"teststring")
```

(meaning find the starting position in A$ of the substring "teststring") can be replaced by:

```
LET Y$=A$:LET Z$="teststring":GOSUB 9900:LET
  S=POSN
9900 LET ZL=LEN Z$:LET SL=LEN Y$-ZL+1:LET
```