**The BASIC Idea (Chapter 6):**
**Richard Forsyth;**
**Published by Chapman & Hall Ltd., London, 1978 (written 1976).**

[Please cite as: Forsyth, R.S. (1978). *The Basic Idea*. Chapman & Hall Ltd., London. 66-76.]

## 6. Character strings

A string is a sequence of characters. So far we have only used string constants, which are sequences of characters normally enclosed in quotes. Thus we have been able to write statements such as

```
400 PRINT "THE ANSWER IS ";A
```

but have not been able to manipulate strings in any other way. The processing of non-numeric information is, however, an important area of computing; and Basic provides ample facilities for it.

A character, for our purposes, is one of the 128 separate symbols the computer can recognize. These include the letters A to Z, the digits 0 to 9 and various punctuation marks. A table of the characters which can be used in Basic is given in Appendix A. Notice that each character has a numeric code in the range 0 to 127.

*Note*: String-handling in Basic on machines other than the DEC 10 may differ significantly from that described here. Apart from file processing (see Chapter 7), text processing is the area in which most differences between different versions of Basic occur. This is because neither string nor file processing was provided in the original Dartmouth Basic: they were grafted on later almost as an afterthought in various guises. Most modern varieties of Basic do have means of performing the operations described in this chapter, although possibly in other ways.

### 6.1 String variables and expressions

String variables can be used to hold string constants: they are named like numeric variables with a dollar sign appended, e.g. A$, A5$, Q2$ etc. A statement such as

```
10 LET B$ = "BASIC AS SHE IS SPOKE"
```

sets the current value of the string variable B$ to the string expression BASIC AS SHE IS SPOKE -- in this case a constant of 21 characters including spaces

One-dimensional string arrays are also permitted, which are vectors of strings. Thus

```
100 DIM Q$(60)
```

sets up a 61 element string vector, Q$(0) to Q$(60), and

```
200 LET A7$=Q$(7)
```

puts element 7 of Q$ into the string variable A7$. Two-dimensional string arrays, or matrices, are not allowed on the DECsystem10 BASIC; and string vectors may not appear in MAT instructions.

The operations of multiplication, division etc. are obviously meaningless when applied to strings; but strings can be concatenated using the plus sign. Thus LET A$=B$+"ING" would put "STRING" into A$ if B$ contained "STR".

Strings can be compared in IF statements. The comparison depends on the rank order of their ASCII character codes (see Appendix A). The following are examples.

```
100 IF A$="YES" THEN 50
200 IF B$<"A" THEN 650
202 IF B$>"Z" THEN 650
```

The last two statements, lines 200 and 202, would test, in effect, whether the first character of B$ was alphabetic -- going to line 650 if it was not. Trailing blanks are ignored in string comparisons so "NO"="NO " but leading blanks are not so " NO"<"NO". With strings composed entirely of letters < means 'earlier in the alphabet than' and > means 'later in the alphabet than'. With strings containing other symbols, it is necessary to know the ASCII code to determine the precise effect of string comparisons.

String constants that start with a letter and do not contain any commas, blanks or tabs need not be in quotes in DATA statements or in response to INPUT. Elsewhere they should be enclosed in quotes. (All strings may be in quotes.) Thus

```
500 DATA "DATE OF BIRTH", "2ND", OCTOBER, "1948"
```

contains four strings, three of which have to be in quotes because the first contains spaces and the other two start with a number. Note that "1948" could not be used in arithmetic expressions as it stands, although of course it can be printed and otherwise manipulated as a string: for instance, the relation "1948"<"1950" would be true in an IF statement.

There is no fixed upper limit on the length of a string, although in practice there are some problems with strings of over 132 characters.

**6.2 The CHANGE statement**

The simplest and often the most convenient way to access individual characters in a string is with the CHANGE statement. CHANGE A$ TO A converts the characters of A$ into a series of ASCII code numbers and puts them in the vector A. CHANGE A TO A$ does the reverse. Both forms of the CHANGE statement use element 0 of the vector to hold the number of characters in the string.
The following example calculates and prints the frequency of each letter in an input string, using the CHANGE statement. (Non-alphabetic characters are counted but not printed out.)

```
LETTER     14:20       26-APR-76

1 REM* EXAMPLE OF CHANGE:
10 DIM V(64),C(127),X(1)
100 PRINT "GIVE A STRING PLEASE"
110 INPUT S$
120 IF LEN(S$)<65 THEN 150
130 PRINT "TOO LONG!"
140 GOTO 100
150 CHANGE S$ TO V
160 MAT C=ZER      'SET VECTOR C TO ZEROES
```

```
170 FOR I=1 TO V(0)
180   J=V(I)       'ASCII CODE WILL BE SUBSCRIPT
190   C(J)=C(J)+1   'ADD 1 TO CHARACTER-COUNT
200 NEXT I
250 PRINT
260 PRINT "CODE", "LETTER", "FREQUENCY"
275 LET X(0)=1     '1-ELEMENT ARRAY
300 FOR I=64 TO 90       'LETTERS ONLY
310   IF C(I)<1 THEN 350  'SKIP IF NONE
320   X(1)=I
330   CHANGE X TO S$     'SINGLE LETTER
340   PRINT I,S$,C(I)
350 NEXT I
999 END

READY
RUN

LETTER      14:22       26-APR-76

GIVE A STRING PLEASE
?"TO BE, OR NOT TO BE? THAT IS THE QUESTION."

CODE    LETTER          FREQUENCY

65        A               1
66        B               2
69        E               4
72        H               2
73        I               2
78        N               2
79        O               5
81        Q               1
82        R               1
83        S               2
84        T               7
85        U               1
TIME: 0.24 SECS.
```

The method employed is to tally the occurrences of each character-code in the corresponding position of the vector C. Thus C(65) contains the number of occurrences of the character whose code is 65 (the letter A) after the execution of the loop on lines 170 to 200; C(66) equals the number of B's, and so on.

It can be seen from line 110 that INPUT can obtain strings from the user, as well as numbers. READ and DATA likewise may be used with strings as well as numbers. This example uses, on line 120, the LEN function -- which gives the number of characters in a string. It uses CHANGE on line 150 to put the ASCII numeric codes for the characters of S$ into the vector V, and the number of characters into V(0). It also uses CHANGE on line 330 to convert the single character code in the first element of vector X into a string (of one letter). One of the MAT instructions is also used on line 160 to fill the array C with zeroes (see Appendix E).

It should be clear from this program that the letters A to Z are represented consecutively by the numbers 65 to 90 in the ASCII code. The digits 0 to 9 are represented by the numbers 48 to 57. Although 128 characters, 0 to 127, are theoretically available, only the characters 32 (blank) to 95 (back-arrow) are likely to be useful to the Basic programmer.

**6.3 String functions**

The user may not define functions in Basic with strings as arguments or results, but a number of useful predefined string functions are provided. These are most helpful in manipulating substrings -- i.e. breaking a string into parts, searching for one string within another one, and so on.

The LEN function which gives the number of characters in a string was used in the example in the previous section. Some other string functions are described below. (A complete list appears in Appendix F.)

CHR$(N) gives the character for which the numeric value N is the ASCII code.
Thus CHR$(90)="Z" and CHR$(64)="@". N should be from 14 to 127.
It will be truncated if not integral.

INSTR(A$,B$) searches for B$ within A$ and returns the character position
in A$ where B$ starts, if found; otherwise zero.
Thus INSTR("YESTERDAY","YES")=1, INSTR("YESTERDAY","DAY")=7
and INSTR("YESTERDAY","TODAY")=0.
An optional numeric argument may be used to specify starting position
for the search: thus INSTR(8,"CATS EAT MEAT", "AT")=12
although INSTR("CATS EAT MEAT","AT")=2.
If this argument is present it should be the first one.

LEFT$(A$,N) returns the first N characters of A$. Thus LEFT$("YESTERDAY", 3)="YES".

MID$(A$,I,N) returns the substring starting at position I
containing N characters in A$. If the last argument is omitted,
it returns the whole string starting at I.
Thus MID$("TOMORROW",4,2)="OR" and MID$("TOMORROW",3)="MORROW".

RIGHT$(A$,N) returns the last N characters of the string.
Thus RIGHT$("STRING",4)="RING".

SPACE$(N) returns a string of N spaces. Thus SPACE$(3)="   ".

Although Basic is primarily an introductory language, its string-handling facilities are quite powerful. It is possible to carry out some fairly sophisticated non-numeric computing in Basic, making it a general purpose, not just a mathematical, programming language.

**6.4 Example program**

In order to introduce string handling the following example involves both strings and numbers. The program NUMBER.XXX translates an input number into the words naming that number. For instance 8505 becomes EIGHT THOUSAND FIVE HUNDRED AND FIVE.

The program is written using two subroutines -- one to separate a 3-digit number into 3 digits (hundreds, tens and units), starting at line 1000, which works by repeatedly dividing by 10 and saving the remainder, and another to translate a 3-digit number into words, starting at line 2000 -- not because these routines are used at several places in the program but because this approach makes it easier to decompose the problem into manageable portions.

The number-naming words are stored in DATA statements from lines 101 to 105 (up to a hundred) and on line 110 (over a hundred). A total of only 30 are required. They are read into the string vectors A$ and B$ at the start of the program -- in lines 200 to 240.

The conversion subroutine (lines 2000 to 2222) uses the routine starting on line 1000 to obtain 3 separate digits from the given number N. These are D(3) for the hundreds, D(2) for tens and D(l) for units. They are then used to address the appropriate number-naming words in A$() which are then concatenated with the current text in X$, the output string being assembled. Let us suppose that the input number (N) equals 125, and so that D(3)=1, D(2)=2 and D(l)=5, then the following is an abbreviated trace of its progress through this routine - which is the 'guts' of the program.

| Line | Action | X$ |
|---|---|---|
| 2000 | Enter, skip to 2005. | ? |
| 2005 | Clear X$. | (null-string) |
| 2010 | Skip to 2020 because N is nonzero. | ditto |
| 2020 | Obtain hundreds, tens and units by calling subroutine at line 1000. | ditto |
| 2030 | D(3)=l so proceed to 2035. | ditto |
| 2035 | D(3)=1 and so A$(D(3))=A$(1)="ONE"; append "ONE"+"HUNDRED" to X$. | ONE<br>ONE HUNDRED |
| 2040 | Test whether AND is needed: it is. | ONE HUNDRED AND |
| 2060 | D(2)=2 so jump to 2080. | ditto |
| 2080 | Append A$(18+D(2))=A$(20) which is "TWENTY" to X$ with trailing blank. | ONE HUNDRED AND TWENTY |
| 2085 | D(l)=5 so proceed to 2090. | ditto |
| 2090 | A$(5)="FIVE", so append it to X$. | ONE HUNDRED AND TWENTY FIVE |
| 2095 | Exit | |

The reader who wants to understand this routine and is still not clear from this trace just how it works is recommended to try following through some other 3-digit numbers to explore other paths through it.

From a partial solution, a routine which can handle numbers from 0 to 999, it is a short step to a more general solution, a program which can handle numbers from 0 to 999999999999 (albeit to only 8 significant figures), because numbers in this range can be rendered into words as

```
        '0-999'  BILLION
        '0-999' MILLION
        '0-999' THOUSAND
(AND)   '0-999'
```

where a billion is 10E9. So if we can translate the '0-999', we can easily handle numbers of 12 digits by applying the routine for 3-digit numbers 4 times. (There are other problems, including deciding when 'and' is needed, but this is the essence of the solution.)

[.... program listing omitted ....]

```
RUN

NUMBER.XXX 14:23        28-APR-76

GIVE A NUMBER PLEASE ?1234567.89
ONE MILLION
TWO HUNDRED AND THIRTY FOUR THOUSAND
FIVE HUNDRED AND SIXTY SEVEN
AND A BIT

GIVE A NUMBER PLEASE ?9876543210
NINE BILLION
EIGHT HUNDRED AND SEVENTY SIX MILLION
FIVE HUNDRED AND FORTY THREE THOUSAND
TWO HUNDRED AND FIFTY

GIVE A NUMBER PLEASE ?16094
SIXTEEN THOUSAND
AND NINETY FOUR

GIVE A NUMBER PLEASE ?13

THIRTEEN

GIVE A NUMBER PLEASE ?-1

TIME: 0.56 SECS.
READY
```

The subroutine starting at line 2000 makes extensive use of concatenation (with +) to build up a phrase in X$.

Notice that precision is lost in numbers of over 8 digits, but that the most significant 8 figures are preserved.

**6.5 Exercises**

(1) Write a program to perform a bubble-sort (into alphabetic order) on a vector of strings.

(2) Write a program to produce a table of all the usable character codes (32 to 95) in Basic and their character equivalents. The output should be in two pairs of columns, to fit on a page (rather like Appendix A).

(3) Revise LETTER (shown in Section 6.2) to read several strings from DATA statements and accumulate tallies for them all: use as test data the first 15 lines of this chapter, 1 line to 1 string.

Your revised program should print the letter frequencies for the whole 15 lines. The letters E and T should be very common. Later, extend it to calculate percentages as well.

(4) Extend NUMBER.XXX so that it deals with fractions as well. Thus instead of rendering 77.851 as SEVENTY SEVEN AND A BIT it should print SEVENTY SEVEN POINT EIGHT FIVE ONE. This is not too hard since decimals are spoken one digit at a time.

(5) Write a Basic program to accept a string containing several words, separated from one another by one or more blanks, and to produce as output the number of words in the string and a string composed of the initial letters of each word. Given "BASIC AS SHE IS SPOKE" it should print 5 and BASIS.

*(6) Suppose the vectors A$(l:N) and B$(1:M) both contain words in alphabetic order. Write a program to merge the two into a single alphabetic vector C$.

(7) Write a program to read a length of text from DATA statements and calculate letter-pair frequencies. Treat all non-alphabetic characters as equivalent ('@' is suggested). Accumulate the frequencies in a matrix C of dimension (26,26) -- so that C(l,l) is the frequency of the pair AA, C(26,5) is the frequency of ZE, C(0,2) is the frequency of @B where @ stands for any non-alphabetic character, and so on. It should print only the non-zero entries -- i.e. those letter pairs that actually occur.

(8) Write a subroutine using the CHANGE statement to reverse the order of the characters in a string. Given "BASIC" as input its output should be "CISAB".

*(9) Write a subroutine to reverse a string which does not use the CHANGE statement. Then write a program to use it, as well as the version using CHANGE from the previous exercise, many times to compare the two methods in efficiency.

(10) Write a program that gives the user instruction in Basic. It accepts a Basic keyword (e.g. LET, PRINT etc.) from the keyboard and either prints a standard format and examples of that statement-type or else responds: KEYWORD NOT RECOGNIZED.

(11) Write a program to convert from verbal to numeric representation so that, for example, ONE HUNDRED AND NINETY becomes 190. This is the reverse of NUMBER.XXX. It is not easy.

(12) Write a program to read, and check, Roman numerals between I and M and print out their modern equivalents. Thus CIV as input should cause 104 as output. This is extremely difficult!