```
390 REM SELECT SUBROUTINE
400 REM USING ON...GOSUB
410 ON D GOSUB 530,600,670,740,810,880
470 RETURN
```

Although your version of BASIC probably contains many statements and functions we have not covered, most will be extensions to the 'basic' BASIC designed to take advantage of particular features of your machine. Many of these will relate to graphics features built into the hardware — instructions such as PAINT, PAPER, INK, BEEP and CIRCLE. These tend to be 'machine specific' and so we have not included them in our course, though we will be giving you more details in other articles.

Before ending the basic part of our BASIC course, however, there are some loose ends to tie up — a discussion of the ASCII character set, together with a couple of functions for helping manipulate characters, and a way of defining new functions (or functions not included in your version of BASIC).

Several methods of representing letters of the alphabet and other characters such as numbers and punctuation marks in digital form have been devised over the years. One of the first was Morse code, which uses combinations of dots and dashes to represent characters. From the computer's point of view, Morse code suffers from the disadvantage of using different numbers of bits for different letters — between one and six dots and dashes for each character. Other attempts at making a more regular and systematic character code (e.g. the Baudot code, which uses five bits to represent up to 32 characters) have fallen by the wayside and the almost universal system now in use is the ASCII code (American Standard Code for Information Interchange).

The ASCII code uses one byte to represent the 94 printable characters, the 'space' and a number of control 'characters'. Eight bits could give 256 unique combinations (2^8), but this is far more than is needed to represent the characters of a standard typewriter or computer keyboard, so only seven are used, allowing for 128 unique combinations. (The eighth bit is usually wasted but is sometimes used to specify an alternative set of foreign language or graphics characters.) The binary and decimal ASCII codes for the standard range of characters are given in the table.

As you can see from the table, the ASCII code for the letter A is 65 and for B is 66. The codes for the lower case letters a and b are 97 and 98. Every lower case letter has an ASCII code value larger by 32 than its upper case equivalent. This constant 'offset' makes it easy to convert lower case letters in character strings into upper case letters, and vice-versa. To do this we will need two further functions not used so far in the Basic Programming course — ASC and CHR$.

The ASC function takes a printable character and returns its ASCII code equivalent, so PRINT ASC("A") would print the number 65 on the screen; PRINT ASC("b") would print 98.

The CHR$ function does the opposite; it takes a number, assumes it is an ASCII code and returns the character it represents. Thus PRINT CHR$(65) would print A, while PRINT CHR$(98) would print b. The CHR$ and ASC functions are widely used, along with LEFT$, RIGHT$ and MID$ in programs making heavy use of character strings. Here's a short program that accepts a character from the keyboard, checks to see if it is upper case and converts it to upper case if it is not:

```
10 REM LOWER TO UPPER CASE CONVERTER
20 PRINT "INPUT A CHARACTER"
30 INPUT C$
40 LET C = ASC(C$)
50 IF C > 90 THEN LET C = C -32
60 PRINT CHR$(C)
```

We shall see more of this type of string manipulation in forthcoming parts of the course.

Finally, in this round-up, a look at functions you may not have in your version of BASIC. Almost all versions of the language allow the programmer to create new functions, and these are almost as easy to use as built-in functions. The DEF statement signals to BASIC that a new function is being defined. Here's how to define a function to calculate the volume of a sphere (the formula is $V = \frac{4}{3}\pi r^3$, where r is the radius of the sphere and π (pi) is the constant approximately equal to 3.14159):

```
10 REM FUNCTION TO CALCULATE VOLUME OF A
   SPHERE
20 DEF FNV(X) = 4 * 3.14159 * X * X * X/3
30 PRINT "INPUT RADIUS OF SPHERE"
40 INPUT R
50 PRINT "THE VOLUME OF A SPHERE OF RADIUS
   ";R;" IS"
60 PRINT FNV(R)
70 END
```

This way of defining a function is fairly straightforward, but let's look at the line in detail:

```
DEFines  function identifier
   ↓      ↓
20 DEF FNV(X) = 4* 3.14159 * X * X * X/3
        ↑  ↑
FuNction  dummy variable
```

When the function is defined, the letters FN are followed by an identifying letter — V in the case of the function above — and this must then be followed by a 'dummy variable'. This dummy variable must also be used in the function definition on the right of the equals sign. When the function is used in a program, any numeric variable can be used in place of the dummy variable in the definition.

At a further point in the program above it would be equally possible to use the 'volume of a sphere' function like this:

```
999 LET A = 66
1000 LET B = FNV(A)
1010 PRINT B
```