INTRODUCTION TO COMPUTER PROGRAMMING

Richard Pierse

Class 2:
Introduction to *Liberty BASIC*

Input and Output

Input and output are the cornerstones that enable programs to interact with the outside world. A program gets most of its data and control from the user's input. Through output, the program displays information to the user. In fact, there would be little point in having a program without any output. The most common form of output is text displayed on the screen. However, other forms of output include text or pictures sent to the printer or music files played through the computer's sound card.

Output: the `print` command

In *Liberty BASIC*, output is displayed to the screen through the `print` command.

`print 7 + 5` writes the number 12 to the screen whereas
`print "7 + 5"` writes the character string: `7 + 5`.

The use of quotation marks tells the computer that everything within the quotes (including spaces) should be printed exactly as written.

Each `print` statement starts printing on a new line. To print several items on a single line, separate them with semicolons. For example, the commands

```
print 15
print 20
print 25
```

results in the following output:

```
15
20
25
```

whereas

```
print 15;
print 20;
print 25
```

results in the output:
```
152025
```

Spaces can be inserted between numbers as text strings as in
```
print 15; " "; 20; " "; 25
```
which results in the output:
```
15 20 25
```

The `print` command with no argument as in
```
print
```
results in the printing of a blank line.

It is possible to clear the output window with the
```
cls
```
command which stands for "clear screen".

Sending Output to the Printer

The `print` command can also be used to send output to the printer rather than the screen. In *Liberty BASIC*, this is done by preceding any `print` commands with an `open` command of the form
```
open "Lpt1" for output as #1
```
`Lpt1` is the computer name for the printer. The command defines a ***pointer*** to the printer to be referred to as `#1`. (Any number could be used but `1` is conventional). Then, `print` statements of the form
```
print #1, "This message goes to the printer"
```
will indeed be routed to the printer, whereas `print` statements not referring to `#1` will still be sent to the screen instead.

Before the program ends, the printer must be released from the program by issuing the `close` command
```
close #1
```

Input: the `input` command

Data is received from the keyboard using the `input` command. Consider the command
```
input age
```
When the program reaches this command it displays a question mark and pauses until the user types a value and presses the return key. It then puts that value into the variable `age`.

The question mark that the program displays when it pauses is called a ***prompt***. It is possible to include a prompt message instead of a simple question mark. For example, the command

```
input ”What is your age?”; age
```

causes the computer to display the message

```
What is your age?
```

before pausing until the user enters a value. Note the use of the semicolon after the prompt. The same effect could also be achieved by combining a print command with a simple input command as in:

```
print ”What is your age?”;
input age
```

Variables and Assignment

The computer processes data held in RAM memory. To the computer, data is stored in memory locations, referred to by position e.g. location 4376. Rather than force programmers to refer to memory locations directly, *Liberty BASIC*, along with all other programming languages, supports the use of ***variables***.

A variable stores a data value, such as a number or a character or string of characters. It is like a box, holding the data value. Variables have *names* associated with them, making them easier to remember. You can choose almost any name you like for a variable subject to a few rules. In *Liberty BASIC* these are:

- a variable name must start with an alphabetic character `A-Z`
- after the first character, names can include letters, numbers or decimal points
- variable names can be as long as you like
- upper and lower class letters are treated as different so `MyName` and `MYNAME` are different variables
- variable names cannot be the same as *BASIC* commands.

These names are valid: `Sales04`, `myVAR66xY`, `x.4376`
These names are invalid: `04Sales`, `myVar%$£`, `print`.

Variables that are to hold strings of one or more character, such as “`a`” or “`this is a string`” must have a dollar sign “`$`” appended to their name as in: `name$`, `Company$`, `show$`.

Assigning Values

Assignment statements store values in program variables. The assignment statement takes the form:

vble = expr

where *vble* is a variable name and *expr* is any valid algebraic expression. The meaning of the statement is: 'evaluate the expression and store it in the variable on the left of the equals sign'. It is important to note that the equals sign does not denote algebraic equality but is a shorthand for "*is assigned the value of*". The following are valid assignment statements:

```
age = 67
four = two * three
x = x + 1
```

The last statement looks odd but is actually a very common programming statement. It *means "evaluate the expression x + 1 and assign it to variable x*" or more simply "*add one to variable x*".

Mathematical Expressions

The last two assignment statements above evaluate expressions using the operators "`+`" (addition) and "`*`" (multiplication). A complete list of mathematical operators is given in the table:

Operator	*Description*
`()`	groups expressions together
`^`	exponentiation
`*` , `/`	Multiplication and division
`+` , `-`	Addition and subtraction

The order of the operators in the table is important, it indicates **operator precedence** which is the order of evaluation when more than one operator appears in an expression. For example, the value of the expression

```
5 + 2 * 3
```

is `11` because the multiplication operator `*` has higher precedence than the addition operator `+` and so is evaluated first. Parentheses may be used to change the order of evaluation as in

```
(5 + 2) * 3
```

which equals `21`.

The exponentiation operator raises a number to a power as in `10^3` (=1000) and `81^0.25` (=3).

Mathematical Functions

Liberty BASIC provides functions for the standard mathematical operations such as *logarithm*, *sine*, *cosine* etc. The format is

fname(*arg*)

where *fname* is the function name and *arg* is the function ***argument*** which can be any valid expression.

The most commonly used functions are listed in the table:

Function	*Description*
`exp(x)`	*e* raised to the power `x`
`log(x)`	natural logarithm of `x`
`cos(x)`	cosine of `x` (`x` in radians)
`sin(x)`	sine of `x` (`x` in radians)
`tan(x)`	tangent of `x` (`x` in radians)
`int(x)`	integer part of `x` (rounded down)
`abs(x)`	absolute value of `x`

Note that *Liberty BASIC* has no *square root* function. However, square roots (and other roots) can be easily calculated using the `^` operator.

String Expressions

We have seen that variables with names ending in *$* hold character strings. Strings can be assigned to such variables using assignment statements such as

```
yrname$ = ”John Doe”
yraddress$ = ”21 Acacia Avenue”
```

Expressions with strings are also possible using the operator `+` which concatenates strings together. For example, the expression

```
yrname$ + ” lives at ” + yraddress$
```

evaluates to the string

```
”John Doe lives at 21 Acacia Avenue”.
```

String Functions

Liberty BASIC also provides a number of functions for manipulating characters and strings.

Characters are represented in a computer by the numbers *0-255* using a code called *ASCII*. The *Liberty BASIC* functions `chr$()` and `asc()` allow you to convert from the *ASCII* code value to the character that it represents and vice versa. For example, the ASCII code *32* represents a space character so that `chr$(32)` is equal to the character `" "` and `asc(" ")` equals the number 32. Generally, you don't need to work with ASCII codes but they can be useful for representing non-standard characters such as ä (ASCII 132) or ® (ASCII 169). The complete ASCII code table is listed in Appendix C of Perry (2002).

The functions `val()` and `str$()` convert numeric strings to the number they represent and vice versa. For example, if variable `mystr$` holds the string `"123.456"` then `val("123.456")` is equal to the *number* 123.456.

The function `len()` counts the number of characters in a string whereas `instr(s1,s2)` gives the starting position of string `s2` within string `s1`. For example, if `mystr$` holds the string `"the quick brown fox"`, then

`instr(mystr$,"the")` = 1
`instr(mystr$,"fox")` = 17 and
`instr(mystr$,"dog")` = 0 since `"dog"` isn't in `mystr$`.

Finally, the three functions `left$()`, `right$()` and `mid$` extract sub-strings from the left, right or middle of a string. For example

`left$(mystr$,6)` = `"the qu"`
`right$(mystr$,6)` = `"wn fox"`
`mid$(mystr$,11,5)` = `"brown"`.

The following table lists the *Liberty BASIC* string functions:

Function	*Description*
`chr$(n)`	*character `n` from the ASCII table*
`asc(c)`	ASCII number of character `c`
`val(s)`	numeric value of string `s`
`str$(n)`	string value of number `n`
`len(s)`	the length of string `s`
`instr(s1,s2)`	starting position of string `s2` in string `s1`
`left$(s,n)`	first `n` characters of string `s`
`right$(s,n)`	last `n` characters of string `s`
`mid$(s,n1,n2)`	`n2` characters of string `s` starting from `n1`

Some Exercises

Try out the following exercises:

1. Write a program to calculate a firm's profits given the following user input:
   a) fixed costs
   b) unit variable costs
   c) product price
   d) product quantity
2. A lump sum *X* is invested at annual interest rate *r* compounded annually. Write a program to input *X, r,* and *n* and compute the value of the investment after *n* years.
3. Repeat exercise 2 assuming that interest is compounded continuously. Hint: you will need to use the `exp()` function.
4. Write a program to read in a sentence as a string and write out the first three words (one per line). Hint: use the `instr()` function to search for spaces in the string.
5. Write a program to compute the area of a circle of given radius. Hint: the formula is area = Πr^2, where $\Pi \cong 3.1415926$.