Exercise 11 – Other Useful Functions

Objective

The major objectives of this session are to practice using variadic functions, random number functions, and the time and date functions from the standard C library.

Reference Material

This session is based entirely on material in the Other Useful Functions in The Standard Library chapter. This practical session is located in the following directory:

	Microsoft Windows	*Linux*
Directory:	c:\qacadv\misclib	**~/qacadv/misclib**
Solution directory:	c:\qacadv\misclib\Solution	**~/qacadv/misclib/Solution**

Overview

The first two questions involve the standard macros found in `stdarg.h`. In question 2 you will be asked to implement a straightforward version of `printf()`. This will give useful insight into why the "real" `printf()` behaves as it does. Question 3 involves generating 6 numbers for the lottery. The final question gives you some practice using the standard time and date support in the C library.

Practical Outline

1. **On Microsoft Windows,** open **sumints.sln, on Linux** change your current working directory.

   Examine the code in the file **sumints.c**. You are required to implement the `SumList` function, which has the following prototype:

   ```
   int SumList ( int, ... );
   ```

   This function is called with at least one `int` value and sums all its arguments. The end of the argument list is given by zero so that the call (already present in the file):

   ```
   a = SumList ( 7, 1, 4, 0 );
   ```

   would yield a value of 12. Why do the last two calls to `Sumlist` in the program go so wrong? Would you have expected a compiler warning for the very last call?

2. **On Microsoft Windows,** open **printf.sln**, and examine the code in the file **printf.c**.

   On Linux, examine the code in the file **myprintf.c** (printf is a Linux command).

   There is a test harness for a function called `pf`, which will be a much-simplified version of `printf`. This function will understand the following format specifiers:

```
$i prints an int
$I prints a long
$f prints a float
$d prints a double
$c prints a char
$s prints a string
```

   There are several calls to the routine in the test harness. Make sure these print what you expect. Remember to be careful handling `float`s and `char`s as mentioned in the chapter.

3. **On Microsoft Windows,** open **lottery.sln, on Linux** stay in the same working directory.

   Using the (empty!) file called **lottery.c, w**rite a program to generate 6 lottery numbers. This is not as straightforward as you might at first think, since once you've generated a particular number you cannot generate it again. There are two alternative strategies you might consider:

   a) Fill an array with all the valid numbers 1 through 49. Generate a random index into the array. Print out and *remove* this number from the array. This is analogous to having all the balls whizzing round inside Arthur and one falling out every so often.

   b) Generate a random number in the range 1 through 49 and save it in an array of 6 integers. Generate a second random number and scan the array to ensure you have not printed this number before.

4. **On Microsoft Windows,** open **time.sln, on Linux** stay in the same working directory.

   Linux note: the program generated is called **times**, which the same name as a shell built-in. When you run the program, prefix it with its path, for example:

```
./times
```

   Using the empty file called **times.c**, write a program to calculate when time began and when time will end. The chapter discussed how the number of seconds past a particular date and time is used in **all** time calculations. However, different implementers choose different start dates and by definition must have different end dates too. Also this number of seconds, if implemented as a `long`, will have a fixed maximum value (which *if* it were a long would be given by `LONG_MAX` in

`limits.h`). Or maybe an `unsigned long` is used, in which case you would need a different constant, `ULONG_MAX`.

You will have to do some detective work, looking in `time.h` to see how `time_t` is typedef'd and choosing an appropriate value from `limits.h`.

The "beginning of time" value is 0. That will fit into a `time_t` no matter how it is implemented. It is then up to you to convert it into a suitable format such that it can be displayed.