Exercise 7 – Functions

Objective

The major objective is to design and use C functions.

Reference Material

This is based on the *Functions* chapter. The questions also refer to questions in the *Looping Constructs* and *Making Decisions* practical exercises.

This practical session is located in the following directory:

Windows Directory:	**c:\qacprogex\function**
Windows Solution Directory:	**c:\qacprogex\function\solution**
Linux Directory:	**/home/user1/qacprg/FUNCTION**
Linux Solution directory:	**/home/user1/qacprg/FUNCTION/Solution**

Overview

Question 1 is a standalone question. Question 2 refers to question 3 of the Looping practical exercise. Question 3 is an exercise on changing a working functions interface. The optional question is based on questions 3 and 4 of the Decisions practical exercise.

Practical Outline

1. Open the Visual Studio Solution **wrt_nums.sln**, and take a look at the code template provided in **wrt_nums.c**. Write a function called `write_nums` that takes two `int` parameters. It prints out all the numbers falling *between* these two parameters. The prototype for this function will be:

   ```
   void write_nums(int, int);
   ```

   Extend the `main` function so that it calls `write_nums` with 2 arguments supplied by the user. Assume the first is always less than the second.

   Thus, if the user types in -3 and 5, the program would print:

   ```
   -2, -1, 0, 1, 2, 3, 4
   ```

2. Open the Visual Studio Solution **countdwn.sln**, and take a look at the code template provided in **countdwn.c**. This program is based on question 3 of the Looping practical, but uses a pair of functions called `getposint` and `downprint` to improve modularity:

```
int getposint(void);
void downprint(int, int);

int main(void)
{
     int number = 0, step = 2;

     number = getposint();          /* Get valid positive integer */
     downprint(number, step);       /* Display in steps of 2     */

     return 0;
}
```

Your task is to write these `getposint` and `downprint` functions.

3. Open the Visual Studio Solution **power.sln**, and take a look at the code template provided in **power.c**. This is a working program, and contains a `main` function and a `power` function. Build and run the program to see how it behaves. Change the interface to the `power` function so that it takes a `double` as the first parameter. Test this function by changing your `main`.

4. This question is based on the questions 3 and 4 of the Decisions practical. If you completed this question, open the Visual Studio Solution **c:\qacprogex\decision\weekday.sln** (if you didn't complete the question, you can open the supplied solution instead, in the Visual Studio Solution **c:\qacprogex\decision\solution\val_date.sln**). Rewrite the program so that the source code contains five functions:

   `int is_leap(int y)` returns true if `y` is a leap year, else it returns false.

   `int is_valid_date(int d, int m, int y)` returns true if the date specified by `d/m/y` is valid, otherwise it returns false (you'll need to call `is_leap`.)

   If `int zellers(int d, int m, int y)` is given the three parts of a date as arguments, the function returns a value from 0 to 6, which is the index of the day of the week the given date falls on. This function will also need to call `is_leap`.

   `void print_day(int z)` will display the conventional days of the week associated with Zeller's index number in `z`.

   `main` should look like this (in pseudo C). Naturally, we expect 2000 Compliance!

```
do
     /*ask the user for a date */
while (!is_valid_date(...))
```

```
z = zellers(...)
print_day(z)
```

When you have tested your program and are satisfied with the output, create a global variable to hold the number of failed attempts, i.e..= the number of invalid dates entered. Increment this counter within the `is_valid_date()` function as appropriate and display a message at the end of your `main` informing your end user of this value.

A solution for this question may be found in **function\solution\val_date.sln**.