Functions with Pointers in C

Your previous work with functions showed you how to pass data **by value** to a function to have it perform some operations and return a single result. What can be done if you want to return two results, or more from a function? The answer lies in the use of something called a pointer in C, which allows you to pass data **by reference** rather than by value These we will call our Level Four functions. In it's simplest form, this means that rather than sending the information that a variable contains you are sending the address in memory where the data is stored. This requires a bit of planning and care, as this means that you are allowing the function to have access to the data that is local to main. This can be very powerful, but can also cause trouble if you don't ensure that your data is properly saved or protected, as the function can, and will, alter those values.

An example of a Level Four function.

```
#include <stdio.h>

//Function Prototypes
void biggerNum(int *, int *);                    //A function to determine the larger of two integers

int main (void)
{
        int small=5, big=3;                      //Two integers to use

        biggerNum(&small, &big);

        printf("The small number is %i and the large number is %i",small, big);

        system("PAUSE");
        return 0;
}

/*************************************************
Function:       biggerNum
Does:           Determines the larger of two integers
Receives:       Pointers to two integers
Uses:           Nothing
Returns:        Nothing
*************************************************/
void biggerNum (int *x, int *y)
{
        int temp;                                //In case the numbers need to be swapped
        if(*x > *y)                              //If x is bigger, swap them
        {
                temp = *x;                       //An * "dereferences" the pointer
                *x = *y;
                *y = temp;
        }                                        //No need for an else or a return
}
```

Assignment

As with the Level Three functions, some of the ideas below are adaptations of tasks you have previously worked on. Use your previous code, recognizing that the objective here is to learn how to use Level Four Functions.

1. Create a properly structured C program which allows the user to enter 2 integers **num1** and **num2** in main, and also defines the integer variables LCM and GCF in main. Create the function LCMandGCF called, from main, which will be passed the two integers and pointers to LCM and GCF. The function will determine the results of the LCM and GCF which will then be displayed in main. NOTE: you will NOT be returning values!
   Prototype: LCMandGCF(int, int, int *, int *)

2. Create a properly structured C program which allows the user to enter a **courseName** and **courseMark** as a string and a float respectively, in main. Create the function progressReport which will be passed the courseName and courseMark along with pointers for a letterGrade character and string message. The letter grade and appropriate message will be determined in the function and then displayed in main.
   Prototype: progressReport(char *, float, char *, char *)

3. Create a properly structured C program which allows the user to enter three integers as the values A, B and C for a quadratic equation of the form $Ax^2 + Bx + C = 0$. The function solveQuad will be sent the integers A, B and C and a pointer to a string that outputs a message on the roots of the quadratic. This could be a good opportunity to use the sprintf() function so that a message of the form "The equation $x^2 - 2x - 3 = 0$ has two real roots of 3 and -1". Similar messages could be returned for two equal roots or for no real roots. If you are really interested you could deal with imaginary roots too.
   Prototype: solveQuad(int, int, int, char *)

4. Create a properly structured C program which allows the user to enter two integers for a **lowNum** and a **highNum** for a range. The function numTypes will be called from main and will be sent the two integers for the range along with pointers for **numPerf**, **numDef** and **numAbund** which represent the number of perfect, deficient and abundant numbers within that range. The function should error trap to ensure that the high is larger than the low.
   Prototype: numTypes(int, int, int *, int *, int *)

5.