Assignment 4

*Submit Question 1 and any **one** of the other questions 2-7 by the end of Tuesday's session.*

For each of the following questions you should use **constants** to store values which will not change in the program, and loops for all repetitive actions.

1. Write a program to calculate the roots of a quadratic equation. A quadratic equation takes the form
   $$ax^2 + bx + c = 0$$
   and its roots can be computed using the mathematical formula:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$
$$r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

A simple program should have the following structure (follow the structure used in the compound interest problem that we did in class):

a) Declare some floating-point numbers to store the input to the problem
   `double a, b, c;`
b) Declare some floating-point numbers to store the output of the problem i.e. the two roots
   `double r1, r2;`
c) Use the `printf` statement to ask the user to input the three required input numbers and use `scanf` to get the values that the user enters into your three input variables
d) Compute each of the roots, using the formula and functions from the math library.
e) Use a `printf` statement to report your answers back to the user.

Note that the program will fail if for some types of input e.g. if `(a==0)` then the program will report a "division by zero" error.

Improve the program by inserting `if` statements to catch these problems and deal with them without the program failing e.g. a simple fix would add a statement such as

```
if (a == 0)
{
   printf("The root cannot be computed when a is
   zero.\n");
}
```

A better solution would output the correct answer for this special case. Similarly, deal with the case when the number under the square root is negative.

For those who find the above steps easy: A less obvious problem with programs such as this is that they sometimes give **inaccurate** results for certain values of a, b and c e.g. a=10e-5, b=10e5,c=1.0. Can you think of any way to get a more accurate answer in these cases?

2. Write a program to print out all the numbers between 0 and 100 inclusive (0 and 100 should be printed too), each number should be followed by a new line character \n.

3. Write a program to print out all the **even** numbers between 0 and 100 inclusive (0 and 100 should be printed too), each number should be followed by a new line character \n. A number is even if it is divisible by 2, or the remainder of the number divided by 2 is 0. The remainder of a division can be calculated using the modulus operator '%'. E.g. 4 % 2 == 0, 5 % 2 == 1.

4. Write a program that allows the user to guess a number between 1 and 10. The program should print the following messages;
   - "You guessed correctly, the answer was <ans>" if they guessed correctly
   - "You guessed incorrectly, the answer is greater\n" if the answer is bigger than their guess.
   - "You guessed incorrectly, the answer is smaller\n" if the answer is smaller than their guess.

5. Modify the program in 4. above, so that it gives the user **three** attempts to guess a number between 1 and 10. Also, If the user enters a number which is not between 1 and 10 inclusive, the program should keep asking the user to enter another number until it is within the correct range.

6. Write a program to read a number from the user and tell them if it is prime. A prime number is a number greater than 1 which is not divisible by any whole number other than itself and 1. The easiest (and least efficient) way to check if a number is prime is to **check if the number is divisible** by all numbers between 1 and itself exclusive (i.e. for 100 check the numbers 2 - 99), but maybe you can think of a better algorithm.

7. Write a program to print out all the **prime** numbers between 1 and 100 inclusive (1 and 100 should be printed too if they are prime), each number should be followed by a new line character \n