1E3 Practical 6
24th February 2016

Task A: Gravitational Force Function (1 marks)

Write a function definition that takes arguments for the masses of two bodies (in kilograms) and the distance between them (in metres), and returns the gravitational force between the two bodies. (See notes below).

Use/test your function in a program that computes the gravitational force between objects for which the masses and the distance between them is provided by the user. Allow the user to repeatedly compute gravitational forces as often as desired.

Use the program to compute the approximate gravitational force between you and the person sitting next to you. (It will be very small, less than the weight of a grain of sand, less than 50 micronewtons!)

Notes:

The gravitational attractive force between two bodies with masses m_1 and m_2 separated by a distance d is given by

$$F = Gm_1m_2 / d^2$$

Where G is the universal gravitational constant: $G = 6.674 \times 10^{-11} m^3/kg.s^2$

The gravitational force will be in Newtons (A Newton (N) is a kg . m / sec^2). Use a globally defined constant for G:

```
const double G = 6.674e-11;
```

Task B: Primality Test Function (2 marks)

Write a program to print out the primes between 2 and 1000. Your program must use a boolean function **prime** (which you must write) to test if an integer is prime. In turn, your prime testing function should use a function **isAFactor** (which you must also write) which takes two integers, i and n, and returns true if i is a factor of n, false otherwise. The declaration of isAFactor is

```
bool isAFactor (int i, int n);
```

Notes:

Recall that an integer is prime if it has no factors other than 1 and itself. Thus your prime function should test for factors (using your isAFactor function) from 2 to n-1, and return False if and when it finds a factor, and it should return True if it never finds a factor.

Recall that i is a factor of n iff n%i == 0, so the body of the isAFactor function is simply "return (n%i == 0);"

Task C: Triangle Classification (2 marks)

Write a function **triangleType** that classifies triangles as equilateral, isoceles, or scalene, g iven the (x, y) coordinates of the triangle's vertices. triangletype takes 6 double parameters and returns an integer code indicating the number of equal sides: 3 for an equilateral, 2 for an isoceles and 0 for a scalene,. NB triangleType does not print anything. Nor does it read in anything.

Your solution must use a function **distance** that returns the distance between two points, given by their (x, y) coordinates. (i.e. distance takes 4 parameters of type double).

Test your function (for scalene and isoceles – equilaterals are hard to test – see note below) using a program that asks the user for a triangle's vertices, calls the triangleType function, and prints out an appropriate message based on the value returned by triangleType. For example:

> X and y of first vertex: **0 0**
> X and y of second vertex: **6 0**
> X and y of third vertex: **3 5**
>
> That is an isoceles triangle.

Notes:
Recall that the distance between (x_1, y_1) and (x_2, y_2) is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Remember that the sqrt function is in the standard C++ math library cmath.

Equilateral triangles are hard to test – for (side1==side2) to be true the doubles need to be equal right down to the last bit. With rounding errors this can be problematic. Either

- try **hardcoding** in the following vertices: (0,0), (6,0), (3, sqrt(3.0)*3). E.g. In your program inlcude "y3=sqrt(3.0)*3;".
- Instead of e.g. "(sidex == sidey)" include the **veryClose** function below, and test "veryClose(sidex, sidey)".

```
bool veryClose (double x, double y) {
        //return true iff the size of the difference between the numbers is less than 0.0001
        //fabs gives the absolute value of a flaoting point number

        return (fabs (x-y) < 0.0001);
}
```