For the following projects (and all future projects in this course):

- **Do not include a main method in the files you submit** - just the definitions of the assigned functions. I will compile your code with my own main method for testing, and there can only be one main method in a program. You will of course need to have a main method for testing purposes - just make sure you comment it out (or delete it) before submitting your file.
- **Do not use any global variables.** Global variables and their drawbacks are discussed on pages 357-9 of the textbook.
- If the assignment description doesn't specify a return value for a function, then it should have a return type of *void*.

Project 4.a

The following formula can be used to determine the distance an object falls due to gravity in a specific time period:

$$d = \frac{1}{2}gt^2$$

where d is the distance in meters, g is 9.8, and t is the time in seconds that the object has been falling. Write a function named *fallDistance* that takes the falling time as an argument. The function should return the distance in meters that the object has fallen in that time. For example if the function is passed the value 3.2, then it should return the value 50.176.

The file must be named: **fallDistance.cpp**

Project 4.b

Write a void function named *smallSort* that takes three int parameters **by reference** and sorts their values into ascending order, so that the first parameter now has the lowest value, the second parameter the middle value, and the third parameter has the highest value. For example if the main method has:

```
int a = 14;
int b = -90;
int c = 2;
smallSort(a, b, c);
cout << a << ", " << b << ", " << c << endl;
```

Then the output should be:

```
-90, 2, 14
```

The file must be named **smallSort.cpp**.

Project 4.c

A hailstone sequence starts with some positive integer. If that integer is even, then you divide it by two to get the next integer in the sequence, but if it is odd, then you multiply it by three and add one to get the next integer in the sequence. Then you use the value you just generated to find out the next value, according to the same rules. Write a **function** named *hailstone* that takes the starting integer as a parameter and **returns** how many steps it takes to reach 1 (technically you could keep going 1, 4, 2, 1, 4, 2, etc. but you will stop when you first reach 1). If the starting integer is 1, the return value should be 0, since it takes no steps to reach one (we're already there). If the starting integer is 3, then the sequence would go: 3, 10, 5, 16, 8, 4, 2, 1, and the return value should be 7.

The function should just return the value, not display it (though of course you can use print statements during testing and debugging).

The file must be named: **hailstone.cpp**