Assignment 2

Due Oct 5, 2016 by 7pm **Points** 120

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

- **Don't use the conditional operator (discussed on pages 207-209 of the textbook).** It may seem "efficient" because it's compact, but it's compact to the point of making code less readable (there are those who disagree with me, but for this class don't use it).

Project 2.a

Write a program that asks the user how many integers they would like to enter. You can assume they will enter a number >= 1. The program will then prompt the user to enter that many integers. After all the numbers have been entered, the program should display the largest and smallest of those numbers. When you run your program it should match the following format:

```
How many integers would you like to enter?
4
Please enter 4 integers.
-4
105
2
-7
min: -7
max: 105
```

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

Project 2.b

Write a program that asks the user for the name of a file and then tries to open it. If the input file is there and can be opened, the program should read the list of integers in the file, which will have one integer per line as in the following example:

```
14
9
12
-6
-30
8
109
```

Note: This example is just to demonstrate the format of the input file. Your program would not print these values out to the console or to the output file.

The program will then add together all the integers in the file, open an output file called sum.txt, and write the sum to that file (just that number - no additional text). Remember to close both the input and output files. If the input file is not there (or is there but couldn't be opened for some reason), the program should just print out "could not access file".

Using a string variable as the parameter of the open function is a C++11 feature, so to compile, you'll need the "-std=c++0x" flag as discussed in the section "Note on different C++ standards".

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

Project 2.c

Write a program that prompts the user for an integer that the player (maybe the user, maybe someone else) will try to guess. If the player's guess is higher than the target number, the program should display "too high" If the user's guess is lower than the target number, the program should display "too low" The program should use a loop that repeats until the user correctly guesses the number. Then the program should print how many guesses it took. When you run your program it should match the following format:

```
Enter the number for the player to guess.
-12
Enter your guess.
100
Too high - try again.
50
Too high - try again.
-2000
Too low - try again.
-12
You guessed it in 4 tries.
```

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

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 2.d

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 3.0, then it should return the value 44.1.

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

Project 2.e

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 2.f

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**