FileDataBuffer subclass to the DataBuffer

In this assignment we will again create a subclass of the DataBuffer. This time, you want a variant that can save itself to a file, and read itself in again. To do that, you must create a subclass of the DataBuffer called FileDataBuffer that adds two methods: save(string filename) and recover(string filename). Both methods should return a boolean result which is true if it succeeds and false if it does not succeed.

The data from the buffer should be stored in a text file. In other words, when saving the data the program must write the data as human readable text, and not in binary form, and when recovering the data it reads the values in from a text file with human readable numbers. To read and write text from a file, use the ofstream and ifstream classes. These classes are quite easy to use. But, true to the reputation of C++, the syntax is sometimes less intuitive than it could be (i.e. compared to that of the corresponding operations in Java).

One additional requirement is that, as in the original DataBuffer::print() method, the FileDataBuffer::save() method must write the data in rows of 10 values in each row. Each column of the output should be 7 characters wide. The restore method must also work with the same data format that is found in the file that it writes.

Remember that the DataBuffer class knows how much data is in the buffer. So in addition to reading in the data, it must set the value of the variable for how much data is actually in the buffer. (If you open an empty file, the amount of data should show 0.) It must also check to make sure that if the data file has more data than space in the buffer, that it does not go over the size of the buffer.

This assignment should be done in two phases. In the first phase, make the file open and close and the reading and writing work. In the second phase, add the code needed to handle situations where the file does not exist, or you are trying to open a file for writing where you do not have write permission. Use C++ exception handling for this feature.

In order to test your solution, you should write the file using one FileDataBuffer object and then read it from the file with a second FileDataBuffer object. Finally you should compare the one to the other to verify that the one that was written out has the same data as the one that was read in. To do that, you must add one additional method, called sameAs() that takes another DataBuffer object as its argument (using a pointer to an object rather than copying the object) and return a Boolean. The return value should be true if they match in every data element with the same index, and false if they differ in any corresponding element.

The output file can go in the project folder in the same location as the source code files. That way you don't need to worry about a path when you run it from your IDE. For one extra credit point, you can add a second argument to the save and restore methods for the path and allow the file to be found anywhere in your file system. In this case, the path and filename variables must be separate. (It doesn't count to just pass a file with full pathname as the file.)

One additional extra credit point is available if you add a line to the top of the file that numbers the columns (0 1 2 3 4 5 6 7 8 9) and starts each line with its offset followed by a colon (0: 10: 20: 30: 40: …). The values in the columns should still line up. Restore should recover the values while ignoring the labels.