Data Structures Lab 2

9/17/2019

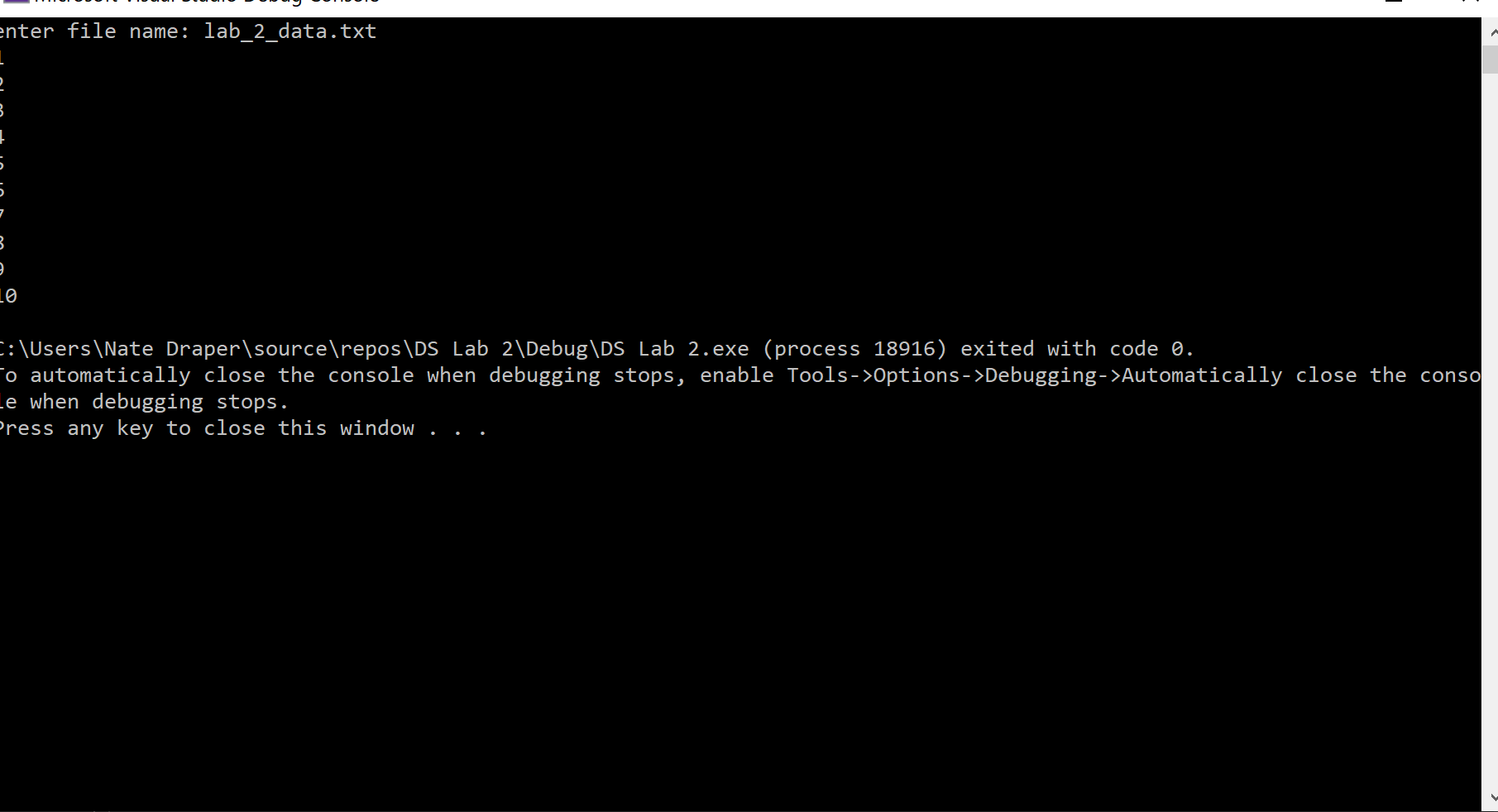
Nathan Draper

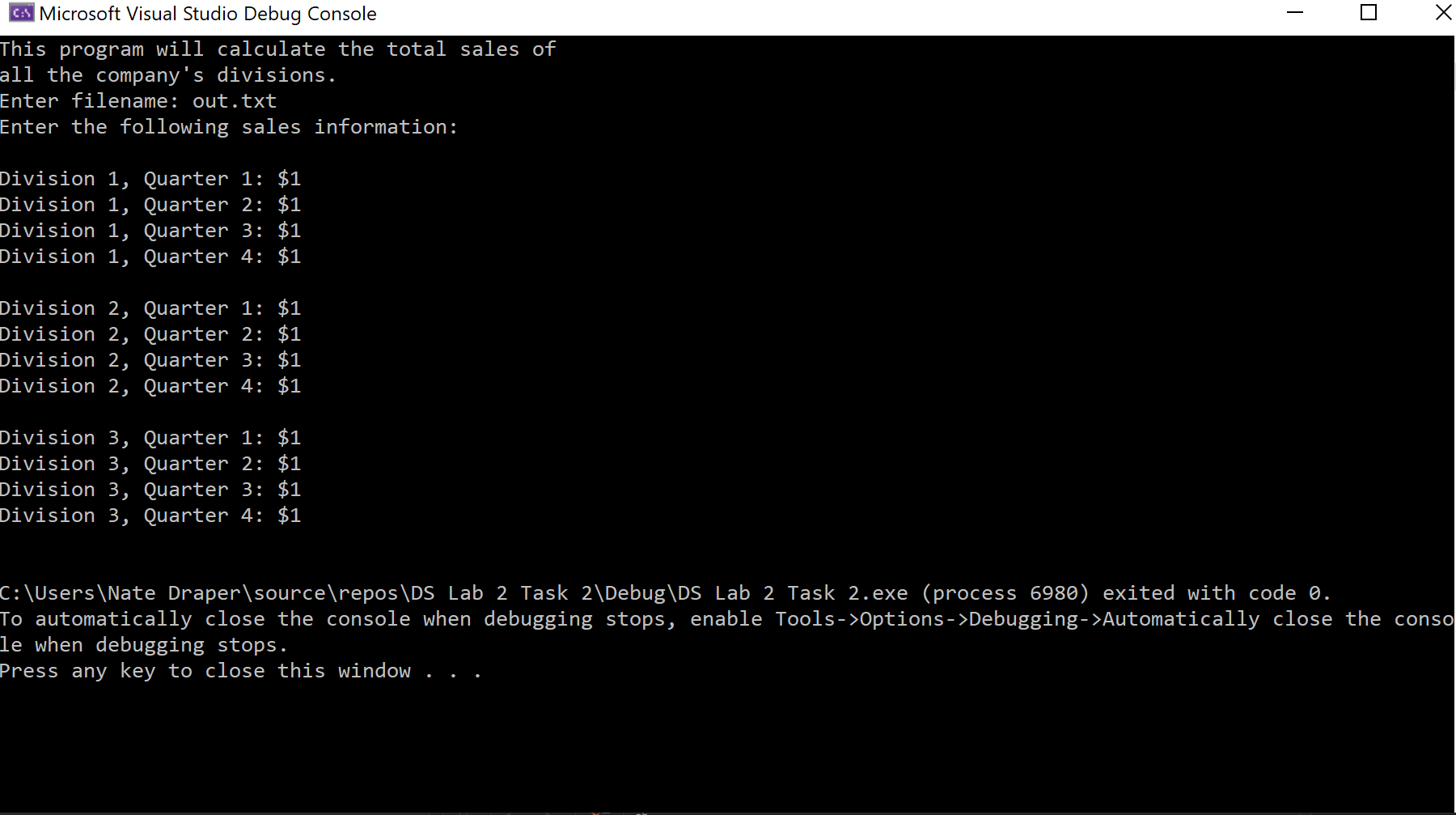
Spencer Jones

Michael Reagan

Group #10

**Discussion**

1. The first objective of this lab was to use the standard C++ file input and output libraries to allow the program to read input from a text file and to output text to a text file. The second objective for this lab to use classes in C++ which used a header file and implementation file. These concepts are important to Computer Science and Engineering because using input and output files is very common for testing and debugging programs that have a large amount of inputs. Classes are also very important because it helps the readability of the program and allows for problems to be identified easier.  
     
   Task 1:  
   

Task 2:  


1. For the first task, the only functionality that was needed was to read input from a text file. This meant that the only file access flag needed was “in” in order to read the file. No other flags were considered because the file only needs to be read. For the second task, the output file needs to be created if it doesn’t exist and if it does exist, append to it. This means that the file access flag “app” was needed to allow for the append functionality. This was the only flag that was considered. In the third task, there was an input and output file that we needed to consider. The input file only needed to be read from, so “in” was used. For the output file, there were two options that were considered. The two options were “app” and “trunc”. Our group considered “trunc” which would overwrite the file if it existed. The advantage of this would be that the most recent version of the input file is the only version in the output file. Our group went with “app” instead because it allowed the text file to operate like a log because it kept previous totals.

C) The input file was designed to have the file be human readable, but also be easily parsed by the program. Each property of the product is on its own line and contains the property name and information separated by a delimiter. We chose this delimiter to be a colon because it makes the file easily read and is easy for the program to search for. This delimiter is what the program searches for in order to separate out the product names from the values. There is also an empty line between each product so that it is easily read by a human to determine where each new product begins and ends.

The class was designed very simply. The class was made to have fields for each of the inputs from the text file which were made private. When a new object was created, all of the relevant fields were needed. There was also another field called “sale” which was calculated inside of the constructor to make the code a bit simpler. Because the fields were made private, each field needed a get and set method to allow the values to be changed or read if they needed to be. The header file was also updated to contain the correct function declarations.

D) For the most part, we didn’t consider the test cases until after the code was written. We learned that it might be valuable to consider test cases before (in a test-driven style) because there were parts of the code that needed to be changed to fit the test cases. We almost overlooked that the program needed to account for the possibility of an empty input file in task 3. In our case, it wasn’t difficult to add a simple check. However, it could have been worse if the test case required a major refactoring of the code. It’s good practice to consider test cases before and during the code writing process.