Katerina Betts and Donald Hutchinson

Professor Charles Zimmer

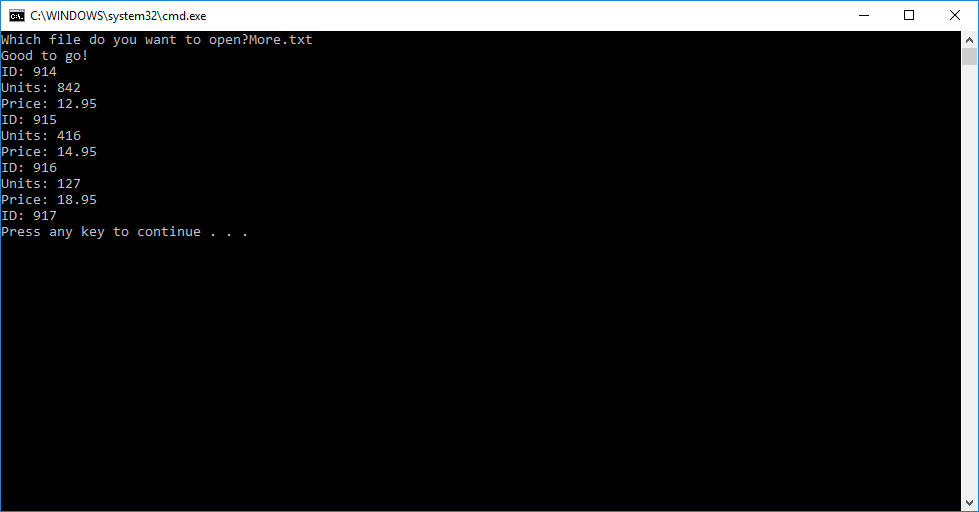
Data Structures

30 January 2018

Lab 2: File I/O, Classes, and Multi-File Programs

In this assignment, the idea of inputting files, utilizing classes, and accessing multiple files within a program were explored. Each different aspect of the lab is important to a future career with a degree in computer science or engineering. When it comes to inputting files, it will be a constant when it comes to working with user interface-based programs and code. To input a file into the code and be able to modify the text file is something that is useful not only to a programmer, but to someone who needs to change a file and cannot access it through anything other than calls. The same can be applied to multiple file programs. It is important to note that most people who utilize code in this way may have more than one file that is necessary to run through. Classes are important because they are utilized to take up less memory and store similar objects. Although this does not come off as useful in the coursework aspect of computer science due to the code being relatively short, but in the actual job aspect of programming, there will be hundreds of lines of code. This makes it easier to store different parts of code and find where the code is at to edit it.

In Task One, the append access flag was used in order to signify that a new file was being created if the file did not originally exist. Another option in this scenario would be to use the in access flag to read the files, and instead of creating a new file, part of the code would check if the file existed and give an error if it did not. Task Two used the append access flag twice because the prerequisites fit perfectly for this task. When the file did not exist, it was created and then appended, and if the file did exist, it was also appended. In Task Three, the read in access flag was used in order to read in a file. This was utilized to get the data from the file to put into arrays in the class that was created.

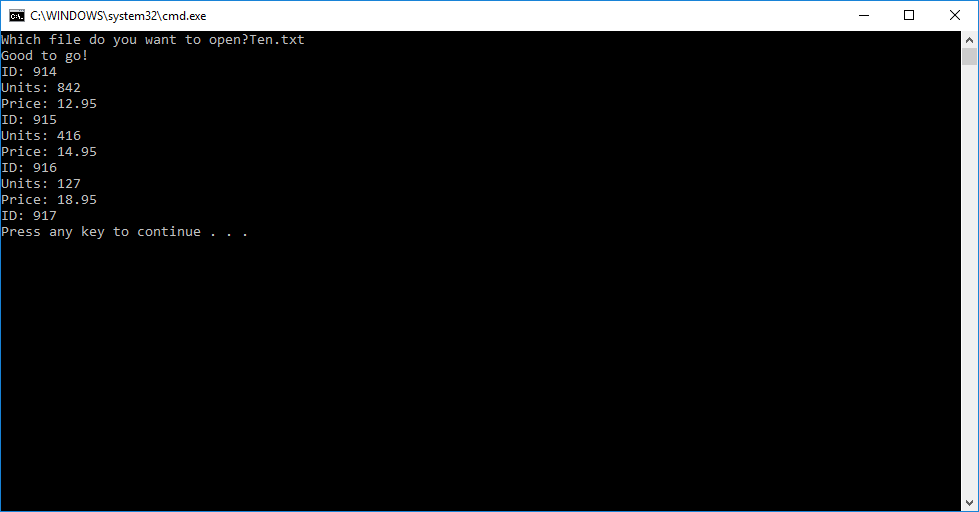
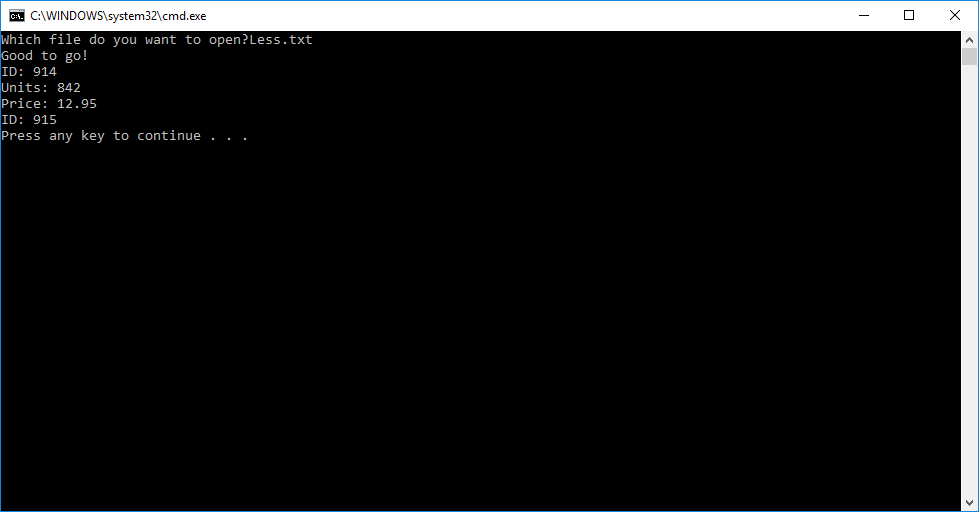


Figures One Through Three From Task Three:

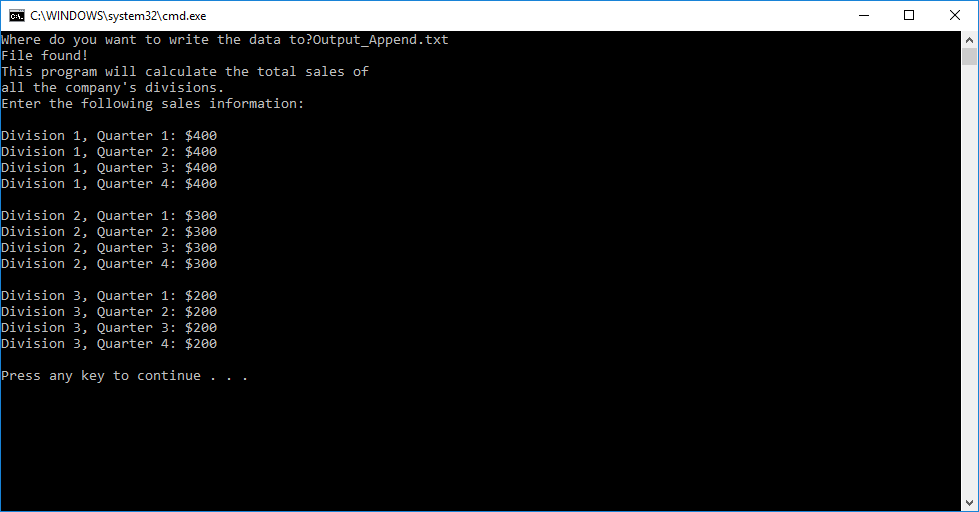
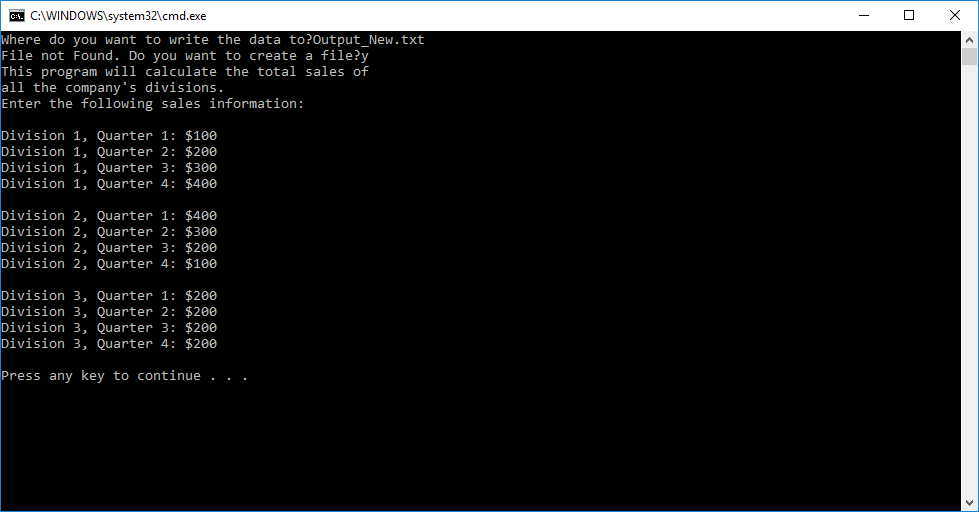
*Figure One shows the output when the file was exactly ten lines.*

*Figure Two shows the output when the file was more than ten lines.*

*Figure Three shows the output when the file was less than ten lines.*

Figures Four and Five from Task Two:



*Figure Four is the appended file from Task Two.*

*Figure Five is the new files created in Task Two.*

For the class in Task Three, the design was taken from the structure and arrays in the previous lab. The functions in the last lab were also put into the class in order for it to be more all-encompassing. There was also a function added to complete the process of reading the file and getting all the data in the file. In the text file, the format provided by the lab document was utilized because it was a logical way to arrange the data. This made it easier to know when and where to pick up certain data as well as which lines were important for each array. Also, having the different variables, or what corresponded to variables, on each line made getting the information simple and line-by-line.

The testing in Task Three was simple, but a lot of factors had to be accounted for before testing the text files with more, less, or no lines. The first test, the one with the nine products already in it, was the one that was considered when first writing the code. After that portion of the code was written, it was fixed in order to accommodate the other three tests in this task. In this case, modifying existing code seemed like the easiest solution; however, if code were longer, it would be better to consider the other tests as the code is being written.