Lab 2 Report

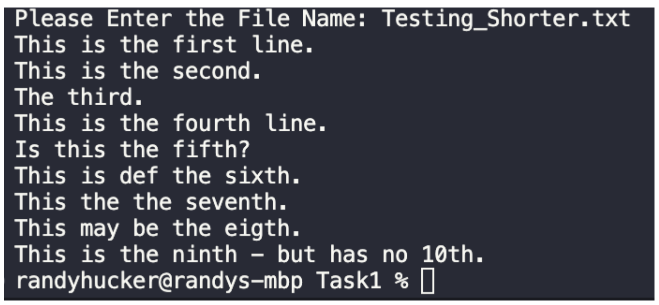
Instructions for Compiling / running code (and locations of input / output files):

- The three file folders in the same directory as this report are all complete visual studio projects (one for each task).
- Open each solution and compile / run the project files like normal
    - All submitted input / output files are located in the general project folders, so they should work correctly because they are in the same folder as the .cpp / .h files

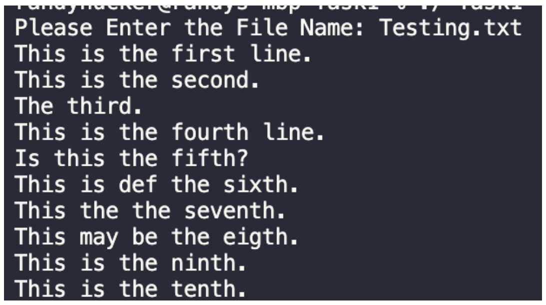
a. *A description of the objectives/concepts explored in this assignment including why you think they are important to this course and a career in CS and/or Engineering. Include screenshot(s) from Task 1 and 2.*

In task 1 and 2 we worked with file access. File access is a necessity in CS due to the number of files that are shared, created, moved, and deleted during the daily workflow. Task 1 required us to read FROM a file, which is important when creating software that relies on transmitted data, and Task 2 required us TO write to a file, which is important for transmitting / storing data for later use. In Task 3 we worked with classes, which is an important way to bind data and functionality together (encapsulation). This concept is useful in many contexts, such as creating libraries to aid in future projects, and making code easier to follow and debug (you can find exactly where a process is going wrong or where a piece of data is incorrect because it is packaged in a logical way).

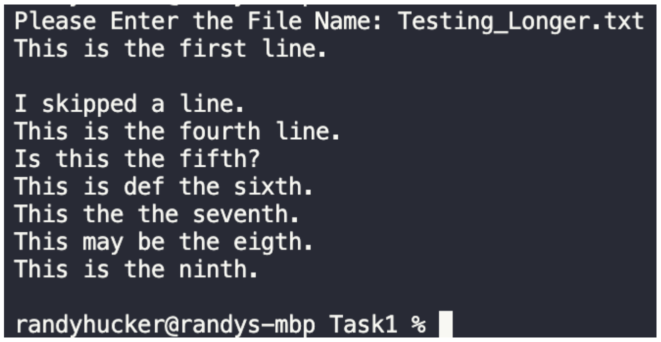
Task 1 Screenshots:



```
Please Enter the File Name: Testing_Shorter.txt
This is the first line.
This is the second.
The third.
This is the fourth line.
Is this the fifth?
This is def the sixth.
This the the seventh.
This may be the eigth.
This is the ninth - but has no 10th.
randyhucker@randys-mbp Task1 % 
```



```
Please Enter the File Name: Testing.txt
This is the first line.
This is the second.
The third.
This is the fourth line.
Is this the fifth?
This is def the sixth.
This the the seventh.
This may be the eigth.
This is the ninth.
This is the tenth.
```

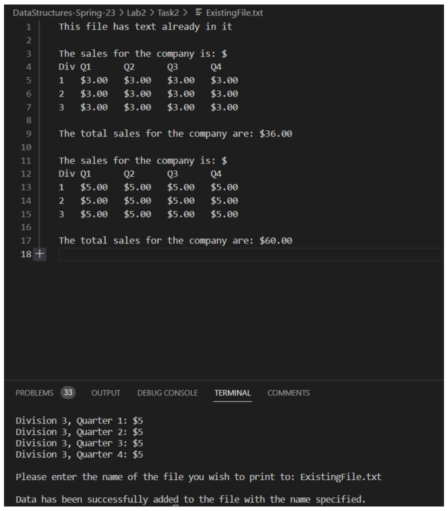


```
Please Enter the File Name: Testing_Longer.txt
This is the first line.

I skipped a line.
This is the fourth line.
Is this the fifth?
This is def the sixth.
This the the seventh.
This may be the eigth.
This is the ninth.

randyhucker@randys-mbp Task1 % 
```

Task 2 Screenshot (screenshot is just for terminal):



DataStructures-Spring-23 > Lab2 > Task2 > ExistingFile.txt

```
This file has text already in it

The sales for the company is: $
Div Q1      Q2      Q3      Q4
1   $3.00   $3.00   $3.00   $3.00
2   $3.00   $3.00   $3.00   $3.00
3   $3.00   $3.00   $3.00   $3.00

The total sales for the company are: $36.00

The sales for the company is: $
Div Q1      Q2      Q3      Q4
1   $5.00   $5.00   $5.00   $5.00
2   $5.00   $5.00   $5.00   $5.00
3   $5.00   $5.00   $5.00   $5.00

The total sales for the company are: $60.00

```

```
Division 3, Quarter 1: $5
Division 3, Quarter 2: $5
Division 3, Quarter 3: $5
Division 3, Quarter 4: $5

Please enter the name of the file you wish to print to: ExistingFile.txt

Data has been successfully added to the file with the name specified.
```

b. *Why you selected the file access flag(s) you selected including what access flag(s) considered but didn't use for each of the Tasks.*

For task 1, the only logical flag was ::in, so that's what we chose.
For task 2, we needed to handle either a new file or an existing file, so again, the flags were obvious: ::app & ::out.
For task 3, we just needed to bring in data from a file. That prevented us from needing to decide on anything except for the ::in flag.
Overall, the necessary flags seemed very blatantly obvious so we didn't need to consider any other options except for the ones that we knew were obviously needed.

c. *An explanation of why you designed the class and input file the way you did in Task 3.*

For the class design, we simply copied the data members from the structure created in lab 1, and then added the new ones requested. We decided to make the data of the class private, to prevent the programmer from accessing them freely. This is a safer way to work with the class, and prevents accidental data manipulation. As far as member functions, we included getters and setters for all of the private data. We also included a clacSales() function that setSales() calls, rather than allowing the programmer to set sales to a value of their choice. This is because sales depend on units and price, so it shouldn't be changed unless units or price is. All member functions are public with the exception of calcSales() (it is simply a helper method for setSales() so it doesn't need to be public.

We designed the input file to be human readable, so before each piece of data is a label with a colon and a space after (e.g. ("ID: ")). This makes it easy to read in the data using >>, while still making it easy to interpret.

d. *What you learned about testing in the 3 Tasks. Did you consider the test cases before you stated, during or after you completed coding?*

For the first 2 tasks, the testing was relatively straight forward. We considered the different cases beforehand, which saved us time because we were able to code around the problems we would've faced had we gone in blind.

Testing Task 3 was more challenging. We used VSCode liveshare on a mac for this task. When we went to debug, we needed to develop a Makefile that allowed us to generate the required g++ output so that a .out file would be generated. To debug, we needed to make changes to the code in Visual Studio so that we could debug intelligently. We considered the test cases before we started which took a bit of time initially, but saved us more time overall as we didn't need to troubleshoot or problem solve as often. The class structure went together well and didn't take too much time to create. The longest part of the testing process was continually ensuring that we had the proper getter/setter functions with the required libraries accessed.

Additionally, we had some trouble with setg on the visual studio compiler while trying to move the cursor back to the beginning of the file, but we resolved this by closing the file and opening it again to reset the cursor. Visual Studio also did not like the way we were initializing the array, because we weren't using a compile time constant. We solved this by dynamically allocating the array, which also makes our program more robust. It is able to handle files with any number of products, without further modification. We used a set of while loops at the beginning of task 3 to handle a non-existent or empty file, so it should be able to handle several types of user error. All of these errors in Task 3 really helped cement the importance of breakpoints for debugging. Stepping through the program helped us figure out what exactly was breaking and in what cases. This was invaluable information for resolving the problems.