**Task 1**

The stable class was designed with a counter to keep track of the current amount of horses in the stable. This made it easy to check for potential errors in the add and remove horse functions. In the add horse function, a simple check was created. Just check if the horse count is less than the max horses. If so, add a new horse to the stable array. If not, handle the error. Similarly, in the remove horse function, a check was added to make sure that there are currently 1 or more horses in the stable. If there are zero horses, handle the error.

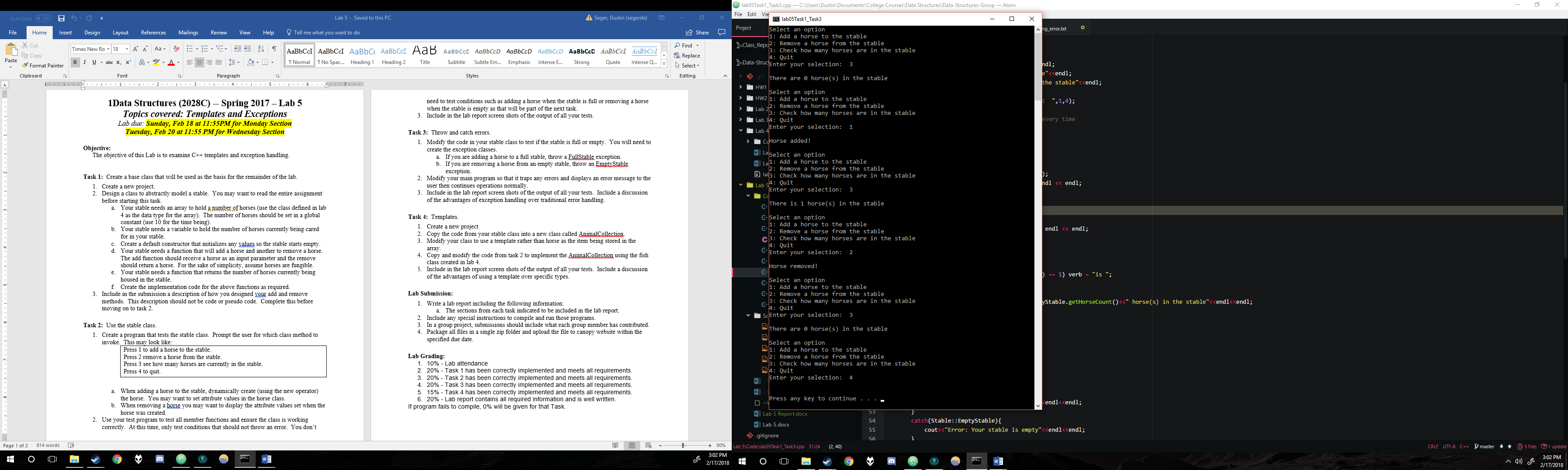
**Task 2**

Figure 1 - Basic add, remove, display, and quit tests

**Task 3**

A screenshot of a computer screen

Description generated with very high confidence

Figure 2 - Empty stable error test (run from beginning of program)

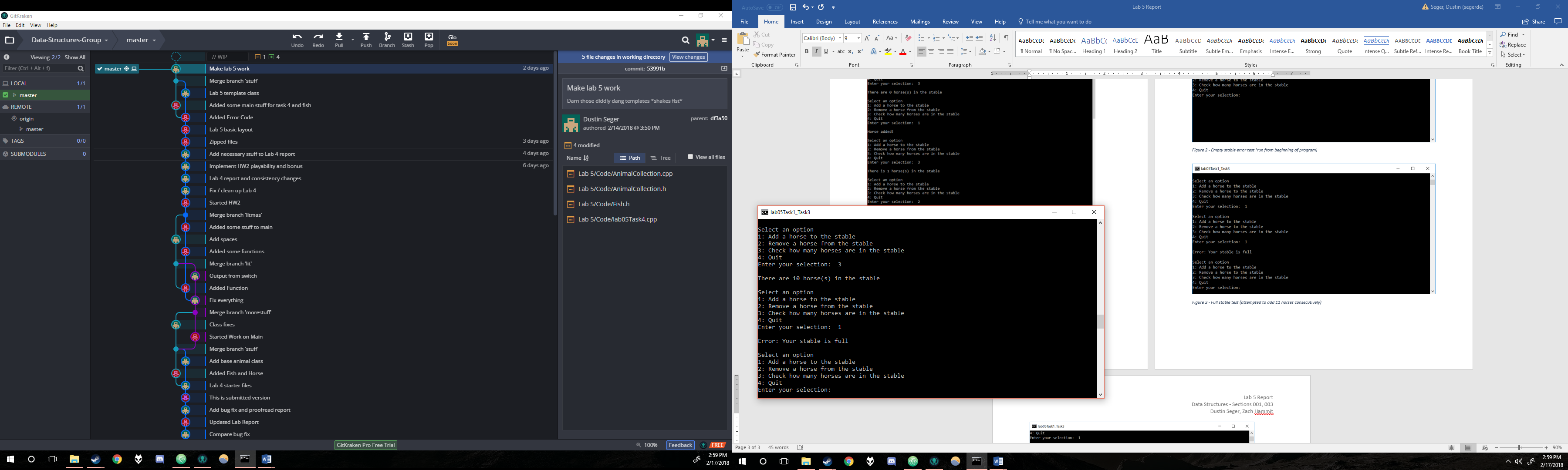


Figure 3 - Full stable error test

A screenshot of a computer screen

Description generated with very high confidence

Figure 4 - Invalid input test

Exception handling is much better practice than typical error handling. Without exceptions, the developer would need to use if statements to catch any errors where they might arise. While this technically works in practice, it does not allow for maintainable code across a large program. If a function has multiple errors that could arise, an if statement at each location would need to be applied to handle it, regardless of how many locations that one error could appear in the same function. With try catch exception handling, you can check an entire block of code for any errors and handle them in an organized manner. This eliminates any duplicate traditional handlings of errors and allows for a much clearer coding environment. With the use of custom exception classes, the work can be organized even more to allow for custom error handling techniques that fit the program’s specific needs.

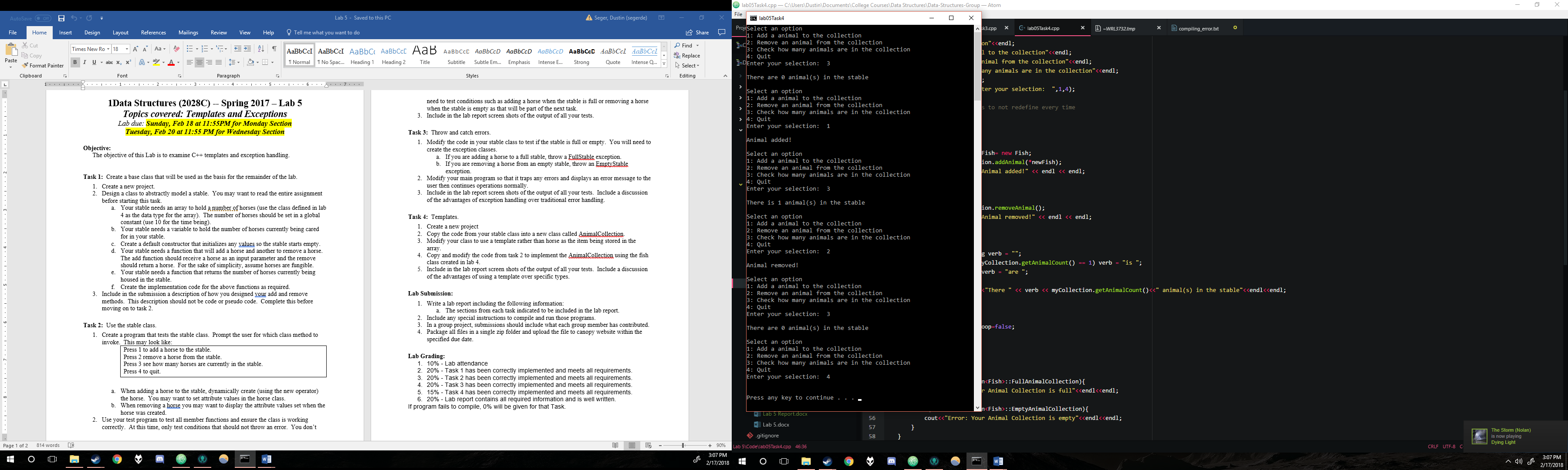
**Task 4**

Figure 5 – Animal collection basic operations test

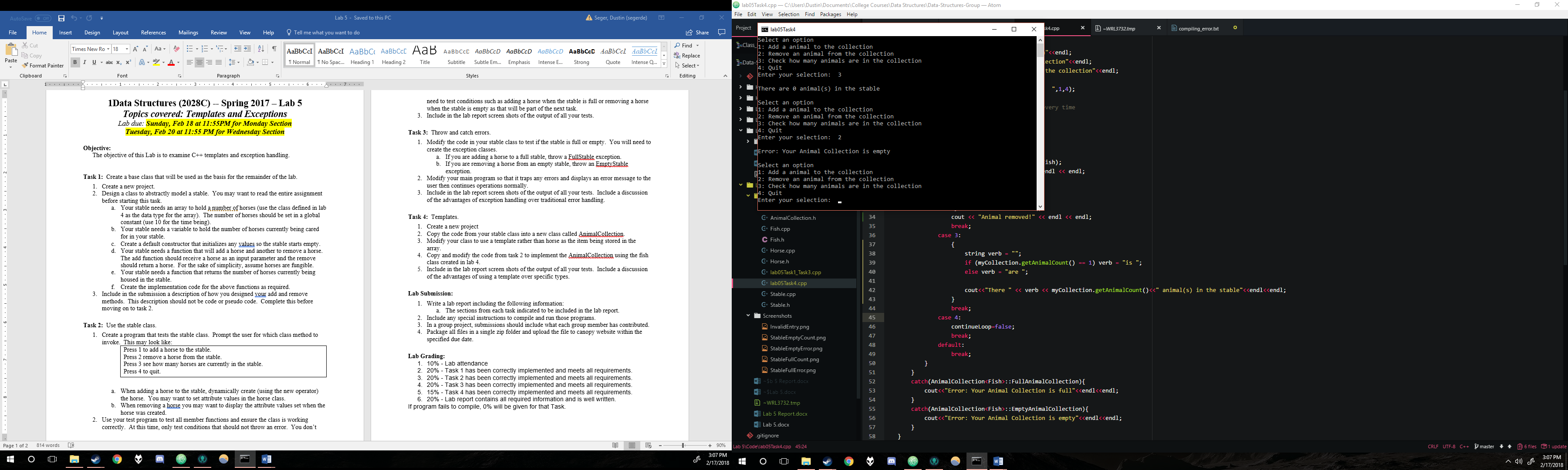


Figure 6 - Empty animal collection test

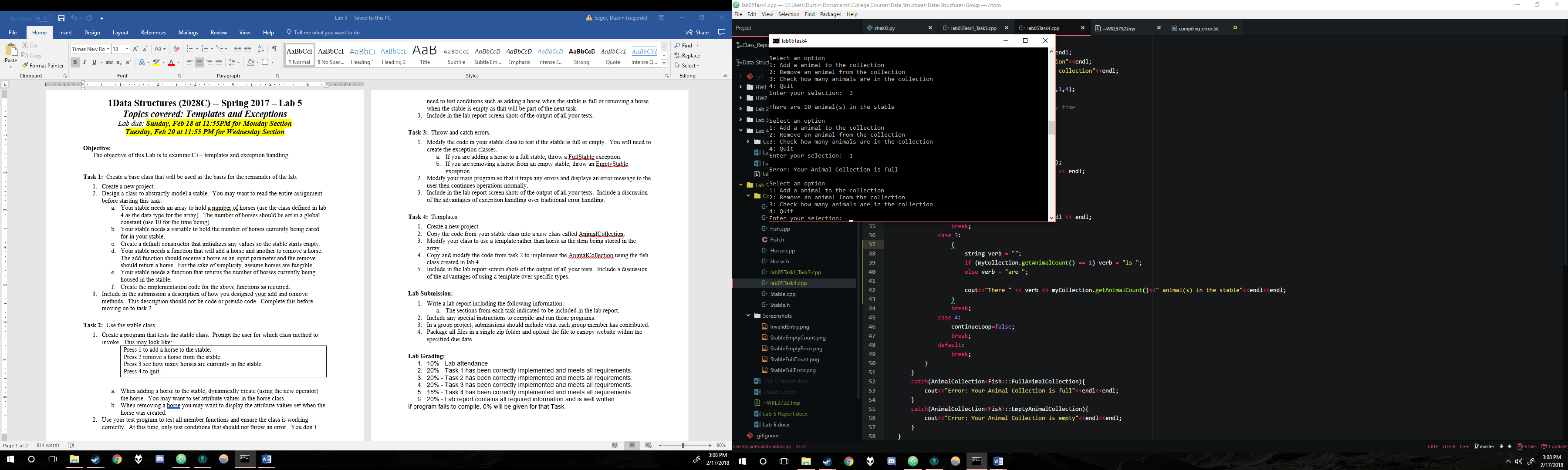


Figure 7 - Full animal collection test

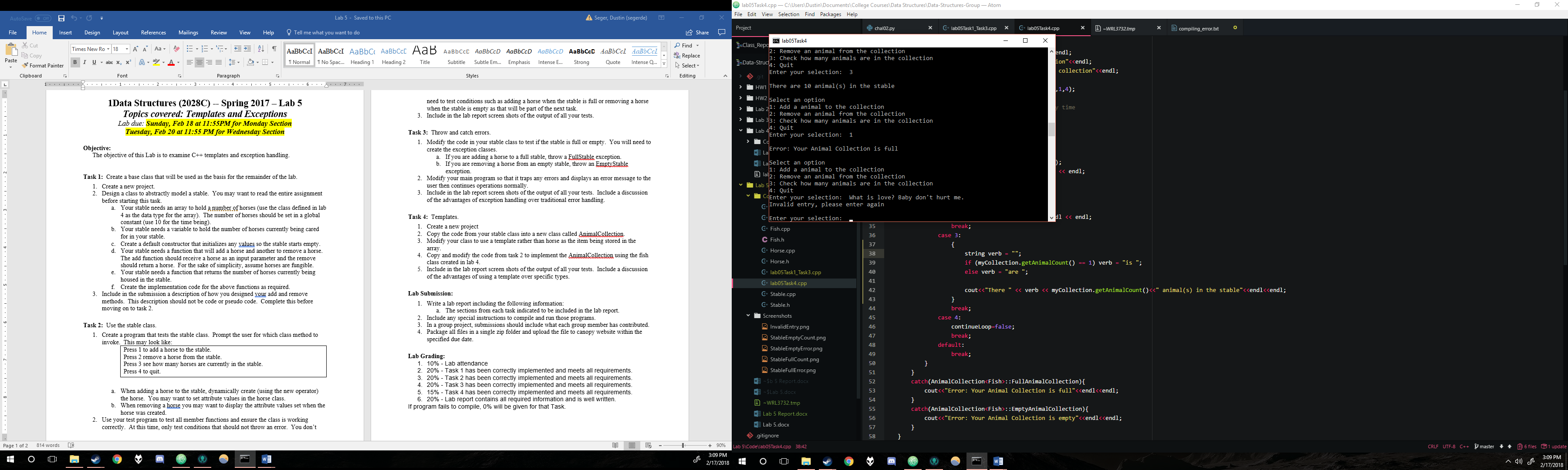


Figure 8 – Animal collection invalid input test

Templates allow for significant advantages over creating specific types of the same thing. When creating something as a template, it allows for multiple types to use the same function or class and simply have it work with a different type. Since only one function is needed, it promotes the concept of abstraction in that the user or other developers do not need to worry about what type that they put in. They simply need to know what it will do to or how it will handle the type. This one function benefit also allows for much more concise code. To accomplish the same concept without templates, a developer would simply have to copy and paste the function with overloaded parameters. While this would work relatively the same, it creates a much more obscure and less maintainable developing environment, especially when dealing with a large amount of code.

**Submission**

Tasks 1, 2, and 3 were coded by Zach with logical assistance from Dustin. Task 4 was implemented by Dustin. The work is equitable.

**How to Compile**

lab05Task1\_Task3 contains the main for Tasks 1 to 3

lab05Task4 contains the main for task 4.

Both programs should be compiled with G++ and the default Windows settigns