**Module 4 Assignment**

Zane Milo Deso

Southern New Hampshire University

CS-405: Secure Coding

**Code Reflection**

**Introduction**

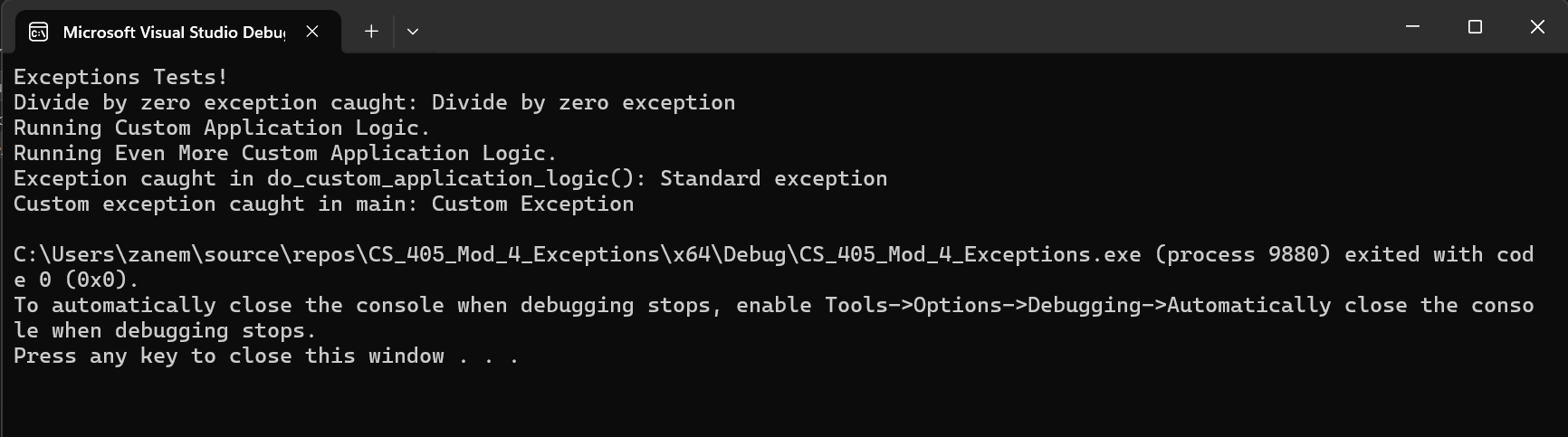
As a Computer Science student attending Southern New Hampshire University, I have been assigned the task of creating an exception try-block wrapping the main function, throwing & catching an exception stemming from std::exception, catching all unhandled exceptions, and create, throw & catch a custom exception. Following the completion of the code portion of this task, I have been asked to provide a summary of my debugging process.

**Custom exception class: name mismatch**

During the coding component of this assignment, I encountered a bug that was confusing to understand where the source was. I was receiving an error message that a std::exception was being caught in the main try-block. Upon further inspcetion of the line that was causing the error, I noticed it was my custom\_exception that was inadvertently catching the std::exception, though this is incompatible as far as I understand, or bad practice. I carefully reread my syntax, which revealed I had typed custom\_exception::exception() instead of simply calling custom\_exception(). Once I adjusted both the throw and catch statement, the error was resolved.  
**Custom exception class: what() inaccessible**

While I was completing the coding portion of this assignment, I encountered an error message specifying that my custom\_exception class what() function was inaccessible. I reviewed my custom class, which extends from the std::exception class, to understand why it was inaccessible in the main function. I noticed that even though the inherited class was public, I had not explicitly made the function public. This was a simple enough fix, which involved inserting “public:” and “override” within the classes syntax to allow for the function to be public, since implicitly the function was private.

**Final Output Screenshot**



## **Conclusion**

In conclusion, it is important to understand exception hierarchy, standard exceptions, try-blocks, and the implementation of custom\_exception classes to gracefully handle exceptions and errors as they may appear in code. C++ is clearly a explicitly statically typed language necessitating the careful observation of both syntax and access modifiers to allow for smooth error and exception handling. I have learned the importance of using e.what() to pass along detailed information to the user or developer attempting to run the software. With clear error messages from the get-go, debugging code can be a breeze, a lesson I will keep close to my chest moving forward as it is invaluable when coding in a complex code base.