**Module 4 – Exceptions**

**Justin Hancock**

**CS-405 Secure Coding**

**09.24.2024**

In the code, I first created a custom exception class called CustomException that inherits from std::exception. This custom class overrides the what() method to provide a specific error message when it is thrown. For handling standard exceptions, I modified the do\_even\_more\_custom\_application\_logic() function to throw a standard std::runtime\_error exception. In do\_custom\_application\_logic(), I wrapped the call to this function inside a try/catch block that catches any exceptions derived from std::exception. When an exception occurs, it prints the error message provided by what(), then continues execution. After handling this, the function also throws the custom exception, which is explicitly caught in the main function.

In the divide() function, I added a check for division by zero. If the denominator is zero then the function throws a std::invalid\_argument exception. The do\_division() function wraps the call to divide() in a try/catch block to handle this specific division error and prevent the program from crashing.

In main(), I wrapped the entire function in a try/catch block that first catches the custom exception, then catches any standard exceptions, and finally, it includes a catch-all handler (catch(…)) for any other unhandled exceptions. This structure makes sure that any potential issues will be caught and handled, preventing the program from terminating unexpectedly.

One of the first bugs I encountered was a division by zero issue in the divide() function that originally caused the program to crash. I fixed this by adding a check to see if the denominator was zero, and then throwing an appropriate exception to handle it. Another issue was the custom exception being thrown in do\_custom\_application\_logic() but not being caught explicitly in main(). To resolve this, I added a specific handler in main() to catch the custom exception. Lastly, I added a catch-all handler in main() to handle any uncaught or unknown exceptions, promising the program would not fail in unexpected situations. This structure makes the program more robust by handling a wide range of errors systematically.

The catch-all handler ensures that no exceptions go unhandled, which can be helpful in keeping the program from crashing in unexpected situations. However, because it doesn’t provide detailed information about what caused the error, it can make debugging more difficult. The handler is useful when stability is the main concern, but it is better to rely on specific exception handlers whenever possible to get more helpful error messages for debugging.

A screenshot of a computer

Description automatically generated