**CS 405 Module Four Exceptions Activity**

**Division by Zero Handling**

In the divide() function, I handled the divide-by-zero error by throwing an std::invalid\_argument exception whenever the denominator is zero. This helps avoid any undefined behavior and prevents the program from crashing. The exception is caught in the do\_division() function, where a user-friendly message is printed, so instead of crashing, the program just tells the user what went wrong.

**Standard Exception Handling**

For the do\_even\_more\_custom\_application\_logic() function, I simulated a logic error by throwing an std::runtime\_error. This error gets caught in the do\_custom\_application\_logic() function, and the error message is displayed using the what() method. This helped show how standard exceptions can be handled and how the error is passed back to the user in a clear way.

**Custom Exception Handling**

I created a CustomException class by extending std::exception and overriding the what() method. This was important because it allowed me to throw specific custom errors and provide detailed messages when those errors occurred. It made sure that the program didn’t just crash when custom exceptions were thrown, but instead handled them smoothly.

**Catch-All Handler**

I added a catch-all handler (catch(...)) in the main() function. This was useful to make sure that even if something unexpected happens, the program doesn’t just crash but instead catches the error and can respond to it. It gives the program an extra layer of protection in case something really unusual goes wrong.

**Compiler Security Features**

To make the code more secure, I used some compiler flags. For example, in GCC, I used flags like -Wall, -Wpointer-arith, and -Wstrict-overflow=3 to help catch potential issues like buffer overflows and pointer problems before the code even runs. In Visual Studio, I used the /GS and /sdl flags to provide stack protection and security checks, which also helped reduce vulnerabilities. These features were inspired by Seacord’s work on secure coding practices in *Secure Coding in C and C++*.

**Challenges Faced**

One of the main challenges I faced was structuring the exception handling blocks so that they weren’t redundant but still caught all possible exceptions. It was tricky to get both standard and custom exceptions to work together smoothly, especially when using the what() method to handle custom errors. Another issue I ran into was making sure the catch-all handler was in the right place. At first, it wasn’t catching all the errors, but after moving it into the main() function, it worked correctly.

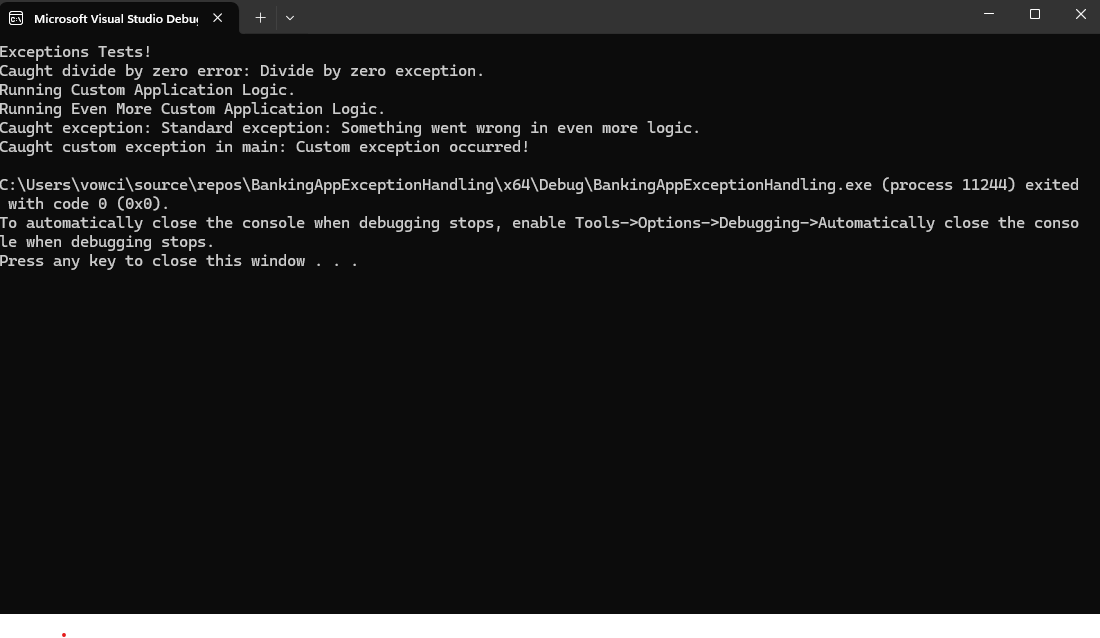
I also had a challenge with the what() method in the custom exception class. It wasn’t displaying the correct message at first because of a small mistake in how I overrode the method. Once I fixed that, the error message was displayed properly.

**Visual Studio Client after Debugger Usage**

A computer screen shot of a program

Description automatically generated

**Output of code**



**References:**

Seacord, Robert C. (2013). *Secure Coding in C and C++ (2nd ed.).* Pearson Technology Group.

GCC Documentation on Compiler Warnings