Module Four Exceptions Activity

In this activity, we are given a .cpp source code file for a large banking web application. Developers have narrowed down some issues with the source code that causes abrupt termination, and we are tasked with capturing the errors in the code and displaying errors to a user rather than crashing to the desktop. The best way to do this is to use exceptions, which is the goal of this assignment.

The source code includes a few “TODOs” for us developers. The first TODO requires us to throw any standard exception. Using the standard namespace (I think this is the correct word), we want to use something such as “throw std::bad\_typeid();”. There are numerous exception types that we can use, and this wanted “any” exception thrown.

Another TODO requires us to wrap the call to a function with an exception handler to catch “std::”exception”, display a message and exception.what(). We can wrap the call with a try/catch/throw block. In the try part, we want to keep an “if” statement to run a function. We then use the “Catch” to catch “std::exception”, and in the catch block, we print a message and the exception.what(). We also have another TODO in this section, to throw a custom exception. A custom exception does require us to create our own exception using the std::exception. We can create our own class by using something such as “struct customException : std::exception”. This means that our new customException is derived from std::exception (we can name “customException” to anything). Back to our TODO, we can now use “throw customException();” to throw that exception, which we will catch later in the main section.

In “float divide (float num, float den)”, we are asked to throw an exception for errors when dividing by zero. Doing this requires an “if” statement to test if the denominator is 0. If the denominator is 0, we throw a logic error exception with a message explaining the issue.

The “void do\_division() noexcept” section also requires an exception handler to capture only the exception thrown by divide. Using another try/catch block, we “try” “auto result = divide(numerator, denominator);”, and “catch” this exception.

The final TODO requires us to create exception handlers to catch the custom exception, std::exception, uncaught exception, wrap the entire function and display a message to the console. Using a try/catch/catch block, we can wrap the two calls to functions (“do\_divison();” and “do\_custom\_application\_logic();”) in a “try” statement. Using a “catch” statement, we can catch our custom exception and display an error using std::cerr. This can print a message and then the except.what(). This is the same with our second “catch”, but catching the std::exception. I’m not sure if this needs to print any more messages other than what I have included so far.

A screenshot of a computer

Description automatically generated