4-1 Activity: Exceptions

**Screenshot**

A screen shot of a computer

Description automatically generated

**Summary**

In this assignment we are tasked with the implementation of exceptions to pinpoint suspect code and prevent the application from crashing to the desktop. In accomplishing this, several key tasks were required to be completed. This included the creation of a custom exception class; throwing custom and standard exceptions; creating custom, standard, and catch-all handlers; and implementing console messages to inform the user when an exception was caught and the what() method text.

Starting with the implementation of the custom exception class, this class was named custom\_exception and implements a what() method that returns a custom error message when custom\_exception.what() is called. This message simply outputs “Custom error message” to the console. The next task was within the do\_even\_more\_custom\_application\_logic() function and simply required throwing any standard exception. This was accomplished by adding the line, throw std::exception(“Standard Exception”);.

The following task was within the do\_custom\_application\_logic() function and required wrapping the call to the do\_even\_more\_custom\_application\_logic() function with an exception handler that catches std::exception. This task also required displaying a message to the user and the exception.what() and to continue processing. This was implemented by utilizing a try-catch block, the try block wrapping the if-statement within the block and the catch block catching the std::exception and printing an error message and .what() to the console. Following the try-catch block, the requirement to throw a custom exception derived from std::exception was accomplished with the line, throw custom\_exception();

The next task was in the divide() function and required throwing an exception to deal with divide by zero errors. This was achieved by implementing an if statement that checks if the denominator used for division is zero and throws a std::logic\_error that prints, “Divide by zero exception.” The logic\_error was used because it is a standard C++ defined exception class used for logical errors within the program code. The following task required creating an exception handler within the do\_division() function that only captured exceptions thrown by the divide() function. This task was addressed by wrapping the do\_division() function in a try-catch block. The try block wrapping the divide() logic, while catching exceptions thrown by divide and printing an error message and e.what() to the console.

The final task required creating exception handlers that caught exceptions in a specific order that wrapped the entire main function and displayed messages to the console. In accomplishing this task, the main function was wrapped in a try-catch block with the try block containing the logic of the main function. The catch blocks were utilized in accordance with the specified order of custom\_exception, std::exception, and uncaught exception. Each handler displayed an error message unique to the exception and an e.what(), with the exception of the catch-all handler.

As far as addressing if I believe that the catch-all handler is a good or bad idea, I think that it is a good idea if used appropriately. That is, if it is used in conjunction with other more descriptive handlers. This can help to further narrow down suspect code by further implementing exception handlers until the issue is pinpointed. On its own, a catch-all handler is not a good idea to implement as it is not descriptive enough to assist with addressing errors, especially in larger code bases.