A computer screen with white text

AI-generated content may be incorrect.

In this assignment, I updated a C++ program to handle different types of exceptions. I created a custom exception by making a new class called MyCustomException that inherits from std::exception. This class overrides the what() function to give a clear message when the exception is thrown. I also used standard exceptions like std::runtime\_error and std::invalid\_argument to handle common errors, such as trying to divide by zero. These exceptions help the program react properly when something goes wrong instead of just crashing.

To handle the exceptions, I used try and catch blocks in the right places. For example, the divide() function checks if the denominator is zero and throws an exception if it is. In the do\_division() function, I added a try block to catch that specific error and print out a message. In the main function, I used multiple catch blocks to handle different exception types. I caught my custom exception first, then standard exceptions, and finally added a catch-all block to make sure nothing unexpected could crash the program.

One thing I learned was that using a catch-all handler (just catch (...)) can be useful to make sure the program doesn't crash if something totally unexpected happens. But it’s not always the best solution because it doesn’t give you any information about what went wrong. It’s better to catch specific types of exceptions first so you can respond to them properly. I didn’t run into major issues, but I did have to remember to throw the exceptions in the right places and to catch them in the correct order. This helped the program handle errors more safely and clearly.