The purpose of Module Four assignment was to add comprehensive exception handling to a C++ application. The goal was to make the application more robust by improving its capabilities to properly handle errors & exceptional situations. The process involved creating custom exception classification that would inherit from the standard C++ exception class. It provided specific error messages for our end needs. Implementation progressed through each function, adding appropriate exception handling as outlined by the TODO comments.

In the “*do\_even\_more\_custom\_application\_logic*” function, a standard “*runtime\_error*” exception was added to demonstrate basic exception throwing. The “*do\_custom\_application\_logic*” function was wrapped with exception handling that catches standard exceptions, displays error messages, and allows the program to continue running & throwing custom exceptions that were made earlier.

For mathematical safety, the divide function was modified to check for division by zero errors and throw an *“invalid\_argument”* exception when detected. The *“do\_division”* function then handles this specific exception type, showing how to target particular error conditions.

The main function received a more robust exception handling structure. It was wrapped with multiple catch blocks that handle exceptions in order from most specific to most general. It was made to catch custom exceptions, then standard exceptions, and finally any unexpected exceptions. This layered approach ensures all possible errors are properly managed.

When the program is executed (See screenshot attached below), it demonstrates proper handling in action. We can see division errors, being caught & handled, standard exceptions being processed, & custom exceptions being properly managed. The flow moves from exception handling, specific function-level catches to broader application-level management. The intent is to mirror what professionals demonstrate with some firsthand experience.

