A screenshot of a computer

Description automatically generated

The program uses ‘std::runtime\_error’ and ‘std::invalid\_argument’ to handle specific runtime issues while also defining a custom exception class, ‘CustomException’. By wrapping potentially problematic code within try-catch blocks, the program ensures that errors are captured and displayed to the user instead of causing unexpected crashes.

Each function is responsible for its own exception handling. ‘do\_division()’ catches only division-related exceptions, preventing divide-by-zero errors. ‘do\_custom\_application\_logic()’ manages exceptions thrown by ‘do\_even\_more\_custom\_application\_logic()’, which ensures errors don’t propagate uncontrollably. Finally, main() acts as a final safeguard by catching any remaining exceptions, making the program stable and secure.

Using a ‘catch (...)’ handler in C++ can be both beneficial and problematic. On the positive side, it prevents the application from crashing unexpectedly by catching all unhandled exceptions. This ensures stability, especially in production environments where unpredictable errors may arise. However, it has a downside—since catch (...) does not provide details about the exception, debugging can be difficult. Without access to meaningful error messages, developers may struggle to diagnose and resolve the root cause of an issue. Therefore, while a catch-all handler is useful for ensuring the application continues running, it should be used sparingly and complemented with more specific exception handling.

Issues Encountered and Resolutions

One major issue was the program crashing when attempting to divide by zero. This was resolved by modifying the divide() function to throw a ‘std::invalid\_argument’ exception and wrapping ‘do\_division()’ in a try-catch block to handle this specific error. Another challenge was ensuring meaningful error messages were displayed to the user. Initially, exceptions were caught without using what(), resulting in vague error outputs. By implementing ‘e.what()’, the program now provides clear, user-friendly error messages.

Additionally, there was an issue with custom exceptions not being caught properly in main(). Initially, ‘std::exception’ was caught before ‘CustomException’, causing all exceptions to be handled by the standard exception block. To resolve this, ‘CustomException’ was explicitly caught first, ensuring it was handled correctly before more general exceptions.