Edwin Jones

CS-405: Secure Coding

May 11, 2025

Process Summary

For this activity, I updated the add\_numbers() and subtract\_numbers() functions to prevent numeric overflow and underflow for all standard C++ arithmetic types. I used std::numeric\_limits to check whether the result of each operation would exceed the allowable range of the type before performing it.

For integer types, I checked whether the current value—prior to applying the increment or decrement—was already close enough to the type’s maximum or minimum such that applying the operation would exceed the range. For floating-point types, the arithmetic operation is performed, and the result is then checked using std::isinf() to determine if it has become infinite. This leverages the IEEE-754 standard, where overflowing a floating-point value results in positive or negative infinity rather than wrapping. If the result is found to be infinite, an exception is thrown to indicate an overflow.

I also used if constexpr to differentiate behavior based on whether the type is integral or floating-point, ensuring compile-time specialization and efficiency.

In the test functions (test\_overflow() and test\_underflow()), each arithmetic function call is wrapped in a try-catch block. This allows the program to continue executing even when an exception is thrown. If no exception occurs, the computed result is printed. However, if an exception is caught, a placeholder such as "N/A" is printed instead to clearly indicate that the operation failed due to overflow or underflow to prevent misleading numerical output.

A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.