**1-3 Activity: Numeric Overflow Coding**

**Overflow Tests**

**A screenshot of a computer program

Description automatically generated**

**Underflow Tests**

**A screenshot of a computer program

Description automatically generated**

**Summary**

This assignment provided us with source code that is aimed at teaching us to write code to detect, prevent, and communicate numeric overflows. The source code contains two functions that operate with and without numeric overflows and underflows, the add\_numbers and subtract\_numbers functions. Writing additional logic was necessary to catch these overflows and underflows, as well as to return a special value to the testing functions if an overflow or underflow was detected to display a console message. To achieve this, modifying the add\_numbers, subtract\_numbers, test\_overflow, and test\_underflow functions was required.

In the add\_numbers function, an if statement was used to check if addition operations resulted in overflow. The if statement: if (increment > 0 && (std::numeric\_limits<T>::max() / increment) < steps) checks if the increment variable is positive and if multiplying by steps exceeds the max value for T. If these conditions are true overflow will occur and a special value, return std::numeric\_limits<T>::max(); is returned to indicate failure. std::numeric\_limits<T>::max() simply represents the maximum value of the datatype T.

Moving on to the subtract\_numbers function, an if statement was used to detect and prevent numeric underflows within the for loop that iterates subtraction operations. The if statement, if ((std::numeric\_limits<T>::is\_signed && result < start) || (!std::numeric\_limits<T>is\_signed && result >= 0)) checks if the result variable value is signed and less than the start variable value or if the result variable value is unsigned and greater than or equal to zero. If either of these conditions are true underflow will occur and the if statement returns std::numeric\_limits<T>::min(); which is a special value to indicate failure much like the add\_numbers() function. std::numeric limits<t>::min() represents the minimum value of the datatype T.

The only other changes to the source code was in the test\_overflow() and test\_underflow() functions. Simple if statements were added to each of these functions that checks if the result variable == std::numeric limits<t>::min() or result variable == std::numeric\_limits<T>::max(). If these were true, print messages of “Overflow detected!” or “Underflow detected!” is output to the console to inform the user that an underflow or overflow has occurred.