Summary

In this assignment, I modified the add\_numbers() and subtract\_numbers() functions to detect and prevent numeric overflow and underflow in a banking application. The original functions performed repeated addition and subtraction operations without checking whether the result exceeded the allowable range of the data type, leading to incorrect calculations (such as negative numbers).

Changes Implemented:

Added Overflow and Underflow Detection:

* Before adding a number, the function checks whether the result would exceed std::numeric\_limits<T>::max(), which represents the maximum value that the data type can store.
* Before subtracting a number, the function checks whether the result would fall below std::numeric\_limits<T>::min(), which represents the minimum value the data type can store.

Prevented Overflow and Underflow:

* If an overflow or underflow was detected, the function returned false to indicate failure, and the result was not modified.
* If no overflow or underflow occurred, the function returned true and stored the correct computed value.

Modified Test Functions (test\_overflow() and test\_underflow()):

* The test functions now check whether add\_numbers() or subtract\_numbers() returned false and print "Overflow Detected!" or "Underflow Detected!" accordingly.
* If no issues occurred, the result was printed normally.

Challenges and Solutions:

* One challenge was ensuring the solution worked for all supported data types, including integers and floating-point numbers. By using std::numeric\_limits<T> and std::is\_integral<T>::value, I ensured that checks were only applied when necessary.
* Another challenge was maintaining the integrity of the function templates while implementing error handling. Using boolean return values allowed the caller functions to detect and handle overflow and underflow conditions gracefully.

Results:

After implementing these changes, I ran the program and confirmed that it correctly detects and prevents overflow and underflow for various data types. The test cases now correctly display whether operations succeeded or failed due to numeric limits.

Basically, the add and subtract functions now do a check before the calculation to see if will result in an overflow or underflow situation and then print the warning to the user.

Below is a screenshot of the console output demonstrating the successful detection of numeric overflow and underflow.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated