Starting with the add\_numbers() method, I inserted an if-statement before the increment statement. That if-statement will check to see if the result plus the increment is less than the original result (indicating an underflow, such ass 255 + 1 = -255), or if the result plus the increment is greater than the numerical limit for the given type–making use of the std::numerical\_limits<T>::max() method. If either condition were true, the method would throw an std::overflow\_error stating that “adding would overflow” with the overflow result in parenthesis afterwards. The same logic applies in the subtract\_numbers() method where instead of checking the result plus the increment, we are checking the result minus the decrement. We are also using the std::numerical\_limits<T>::min() method instead, throwing std::underflow\_error when necessary.

Further down, in the test\_overflow() method, we wrap both calls to the add\_numbers() method in a couple of try-catch blocks, both catching std::overflow\_error. As denoted by the comments, the first try-catch block may be unnecessary as testing yielded no overflows here, but it was left this way for consistency and safety. In test\_underflow(), however, the first try-catch block is necessary because of the float type. I am not entirely sure why it underflows even here, but it is better to be safe than sorry.

Screenshots follow on the next two pages.



