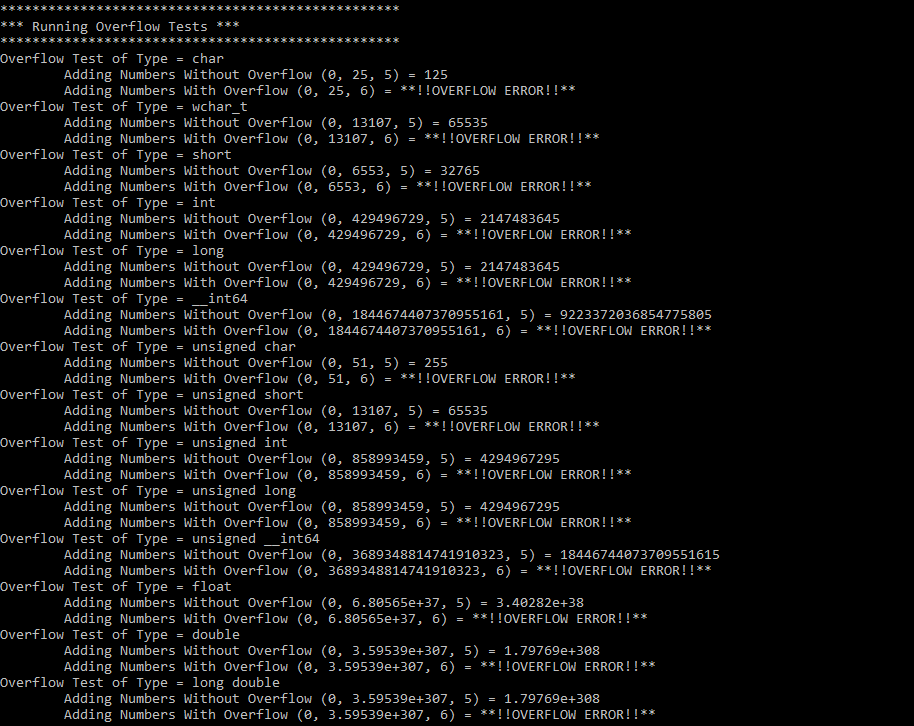
For the overflow testing portion of this project, the process of checking for a result that exceeds the upper limit of the type being tested was simple because we were starting at 0, and incrementing by the maximum value divided by 5. As we looped through the increments, we only had to check to see if the current value subtracted by the maximum value of the type was less than the iterator i. If the iterator i was ever greater than the maximum value minus the result, then we know that the next increment will exceed the bounds of the type.



The underflow portion of the project was a little more difficult. Because we were taking the maximum value of the type and dividing by 5 for our increment, we were potentially choosing a value that was smaller than required to successfully underflow when going through a loop. For example, the maximum value of a signed char is 127. Dividing this by 5 results in a decremental value of 25. But the lower limit of a signed char is -128. If we decrement the starting value by 125 six time as in the example code, then we won’t reach the lower limit because we stop at -23. The observation is that the program correctly underflows when we are dealing with unsigned types and that it does not underflow with signed types. The exceptions were the float, double and long double types. Each of these underflowed with both 5 and 6 iterative steps.

