Joseph Les

CS-405 Professor Hodde

Numeric Overflow Coding

In addressing the numeric overflow and underflow issues in the banking application, I undertook a systematic approach to ensure the functions add\_numbers and subtract\_numbers could reliably detect, prevent, and communicate these conditions. Initially, the task involved identifying where in the code the overflow and underflow could occur.

The first step was to modify these functions to detect potential overflow or underflow conditions before performing arithmetic operations. For overflow detection in add\_numbers, I introduced a check to ensure that the current result plus the increment would not exceed the maximum value representable by the data type. If this condition was violated, the function would return false, signaling an overflow. Similarly, for underflow detection in subtract\_numbers, I added conditions to check if the result minus the decrement would fall below the minimum value for signed types or below zero for unsigned types. If an underflow was detected, the function would also return false.

A computer screen with text on it

Description automatically generated

A computer screen with text on it

Description automatically generated

Next, I updated the test functions (test\_overflow and test\_underflow) to handle the return values from add\_numbers and subtract\_numbers. These test functions were designed to print appropriate messages to the console, indicating whether the operation succeeded or if an overflow or underflow was detected.

(underflow example)

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

In summary, by implementing detailed checks for numeric limits and enhancing the test functions to report these conditions, I ensured that the add\_numbers and subtract\_numbers functions could effectively manage overflow and underflow scenarios. This process underscored the importance of meticulous validation and error handling in developing robust software, particularly in critical applications such as banking systems.