Documentation

The task is to create a basic calculator that can evaluate mathematical expressions represented as strings. This problem requires careful handling of various arithmetic operators, including addition (+), subtraction (-), multiplication (*), and division (/). The input string can contain non-negative integers and may also include spaces, which should be ignored during processing. The expression is guaranteed to be valid, meaning that operators and numbers are correctly sequenced. A key requirement of the calculator is to adhere to the standard operator precedence rules: multiplication and division must be performed before addition and subtraction. Furthermore, when performing division, the result must be truncated towards zero, ensuring that expressions like 3/2 yield 1 instead of 1.5.

The solution employs a stack data structure to effectively evaluate the expression, which is instrumental in managing intermediate results. As the calculator iterates through the string character by character, it builds numbers from consecutive digits until an operator is encountered. When an operator is found, the calculator performs the operation dictated by the last operator encountered, using the current number and any previously stored values in the stack. For multiplication and division, the calculator pops the last number from the stack, executes the operation, and pushes the result back onto the stack. In contrast, addition and subtraction simply involve adding or subtracting the current number directly to the values in the stack. This approach allows the calculator to maintain an accurate representation of the ongoing evaluation of the expression.

Once the entire expression has been processed, the final result is computed by summing all the numbers left in the stack. This method is efficient, with a time complexity of O(n), where n is the length of the input string. This efficiency is crucial, particularly given the constraints that allow for strings of up to 300,000 characters. Additionally, the calculator must handle all integers within the range of 32-bit signed integers, ensuring that both input and output conform to these limits. The design of the calculator, therefore, must account for potential edge cases, such as long sequences of operations or varying numbers of spaces between elements, while maintaining the integrity of the arithmetic evaluations.

In summary, the implementation of this basic calculator involves parsing a string representation of a mathematical expression, managing operator precedence, and ensuring accurate calculations through a structured approach using a stack. The challenges posed by varying operator types, the need for truncation in division, and the presence of spaces necessitate a well-thought-out solution that adheres to both mathematical rules and programming best practices. By ensuring the expression is processed efficiently and correctly, the calculator can provide reliable results for a wide range of valid mathematical inputs.