Number to Words conversion tool design rationale

This document explains the design approach on how the conversion works, and discusses about some other potential solutions that can be applied.

Approach

The main reason to choose this approach instead of other potential solutions are due to the time complexity. Let's breakdown the major functions of the approach.

1. GetAmountToWords method
   - The main purpose of this entry method is to clean up the given input before passing to the private method that process the input.
   - Rounding and truncating the decimal input takes O(1) complexity. This separates the decimal input into 2: The digits before the decimal point, and the digits after the decimal point.
   - Appending string with the StringBuilder takes O(1) complexity since it works as if adding string into a character array.
2. GetUnitAmountToWords method
   - This method converts the integer part of the decimal, that is the digits before the decimal point.
   - It will loop through the supported magnitudes, since C# decimal type supports up to Octillion, that is 10 to the power of 27. Since there are fixed amounts of magnitudes, the operation takes an O(1) complexity as well.
   - Processing the magnitude with division and modules takes O(1) complexity.
   - Due to the nature of the magnitudes, each iteration will takes in 3 digits at a time, thus concatenating the string (with string builder) is technically constant time.
3. Convert3DigitToWords method
   - The method only converts a number between 0 and 999 into words.
   - There are at most 3 conditions checking in the methods (>=100, >= 20, and > 0) for each number.
   - Since we only takes in at most 3 digits, the string builder runs at O(1).
4. GetSubunitAmountToWords method
   - This method converts the fractional parts, that is the digits after the decimal point.
   - Since we have rounded up the fractional parts into at most 2 decimal points, the method calls the Convert3DigitToWords, and technically takes constant time as well O(1).

The total complexity of this approach is O(1) since the number of magnitudes are fixed and the C# decimal type supports up to Octillion. Each iteration can be considered as if it is running within a fixed size array. The limitation is actually on the C# decimal maximum value, that limits the application to support up to 10^27 digits, but I believe the optimized time complexity outlines the limitations.

Potential solutions

There are various approaches to deal with the number to words conversion:

1. Recursive
   - To achieve it, we may recursively breaking down the decimal into smaller components.
   - It will be easier to implement since we can create the conversion logic based on each units to handle the smaller segment by the modules of 10.
   - However, since each segment requires compute power and storage when executing.
   - It may not be the optimized approach as well since the time complexity may become O(log n) since each recursion breaks the input into n magnitude of steps.
2. Hard-coding dictionary
   - To achieve it, we may hard-coding all the potential chunk into a dictionary.
   - The O(1) time complexity can be achieved since look up a key from dictionary takes O(1).
   - This is definitely not a good approach since the permutation may be large, thus, it only suits a fixed small amount of digits.
   - Also, to have a dictionary that consists of all the potential results, it will uses lots of memories.