Sample Documentation

Cheque Writing Exercise

digisoln.com

Table of Contents

Executive Summary

digisoln.com require cheque writing software that can convert a numerical value into the words representing this value. This documentation outlines potential and used approaches to solve this problem, and uses test case scenarios to examine the effectiveness of this solution presented.

Through analysis of the methods available to complete this problem, the single intuitive parsing of the sequence of digits from left to right was found to be the most efficient, scalable and intuitive solution. This was reflected in the test results, which highlighted problems to do with the semantics of how words on a cheque are actually read to the client as being a necessary focus point for future development.

Finally, it is determined that the software is not yet ready to go live, as in its current state it is neither viable nor profitable for any financial institution. Further work must be done to integrate this software with other tools, to expand its features, and to improve its user interface and help facilities.

Overview

digisoln.com require a cheque writing program, which accepts a number as input and coverts this into words. These words must be output to the user as a string. The following documentation includes a discussion of the solution presented, as well as a test plan with results.

Solution Discussion

There were multiple approaches to this problem, which have been detailed in the following sections.

Approach 1: Multiple parsing of the numeric value

A possible solution to this problem would be to parse the numeric value a number of times, and on each parse perform a different task as shown:

Parse	Direction	Activity
1	Forwards	Convert each singular digit into words
2	Backwards	Having discovered the end, parse in reverse, starting from the final cent, counting back and joining the correct words together to form conjunctions – e.g. “Fifty-five”.
3	Backwards	Again parsing in reverse, starting from the end, counting back to convert words in the “teens” – e.g. “Fifteen”, “Fourteen”, “Eleven”.
4	Backwards	Finally counting backwards, add all remaining words – e.g. “THOUSAND”, “HUNDRED”, “DOLLARS” etc.

The use of this *multiple parsing* was **not deemed ideal** – as the cost of time / efficiency of this algorithm is far greater than the solution needs to be.

Approach 2: Avoid multiple parsing, working backwards from the end of the sequence

It appears that Approach 1 primarily parses from the end – this is ideal in working backwards from the cents to determine the size of each denomination presented.

However, knowing the capabilities of String processing in .Net in determining length, size and positioning of elements within a string, it was determined that parsing from the front of the sequence – *as the number would be read by a human* – will provide a more intuitive abstraction of the programming presented as a human would expect.

Approach 3: Intuitive hybrid of the above approaches

By modularising the activities listed in Approach 1 into separate utility functions within the main class, as well as single parsing of the numeric sequence from left to right as a human would read it, the final approach will present the most scalable, efficient and intuitive solution possible.

This solution will follow the following steps, which can be found within the main conversion subroutine presented within the code:

1. Initialise variables (including the sequence itself, and its length)

==> START ITERATION LOOP FOR EACH DIGIT IN SEQUENCE (LEFT TO RIGHT)

2. Read numbers and convert the currently processed digit to words:
    a. If required, process the numbers into “teen” or conjunction words;
    b. Otherwise, handle the numbers as normal / singular digits
3. Check for punctuation, denominations and joining words for the current digit

==> END ITERATION LOOP FOR EACH DIGIT IN SEQUENCE (LEFT TO RIGHT)

4. Return the converted words to the user.

It was decided approach three was the most eloquent and scalable solution.

Test Plan

The following results were taken from the live release of version # of the software on ##/##/##:

Value	Result	Benchmark Met	Comment
123.45	ONE HUNDRED AND TWENTY-THREE DOLLARS AND FOURTY-FIVE CENTS	✓	
0001.01	ONE DOLLAR AND ONE CENT	✓	
-123456789123	Incorrect currency value.	✓	
19*4156	Incorrect currency value.	✓	
1001	ONE THOUSAND ONE DOLLARS AND ZERO CENTS	**Possibly:** Depending on the semantics of the client's preference on how this number should be read, this will require more attention.	
099000456.	NINETY-NINE MILLION FOUR HUNDRED AND FIFTY-SIX DOLLARS AND ZERO CENTS	✓	
twenty-two dollars	Incorrect currency value.	✓	Prefer more useful help prompt.
123456789123456789	ONE HUNDRED AND TWENTY-THREE QUADRILLION FOUR HUNDRED AND FIFTY-SIX TRILLION SEVEN HUNDRED AND EIGHTY-NINE BILLION ONE HUNDRED AND TWENTY-THREE MILLION FOUR HUNDRED AND FIFTY-SIX THOUSAND SEVEN HUNDRED AND EIGHTY-NINE DOLLARS AND ZERO CENTS	✓	
123456789123456789 123456789	There was an error processing your request.	✓	Prefer more useful help prompt.
.1	ZERO DOLLARS AND TEN CENTS	✓	
0000.01	ZERO DOLLARS AND ONE CENT	✓	
1	ONE DOLLAR AND ZERO CENTS	✓	
	Incorrect currency value.	✓	As above.

Future Directions

As highlighted in the testing, it is recommended that more useful help prompts be given to users in order to assist them using the software. Further semantics of correct money annunciation based primarily on client preference is key to resolving the highlighted test issue of word ordering. Finally, it must be noted that more time must be spent creating a commercially viable piece of software; in its current state, the Cheque Writing program is not ready to go live. Further to this, in its current state, it is not forecast to be successful as a financial use tool.

Integrating this with a full suite of financial applications, in which cheques could be printed using attached peripheral devices would best further the potential profitability of this software.