Statement of Problem

Use Visual Studio 2019 (Community Edition is free!) or newer version to craft a C# Windows Forms App (.NET Framework 4.6.1 or newer version) with a button and a text field. The button brings up a file selector that allows the user to select a CSV file. Once the CSV file is selected, the following process is to be performed on it and the results are to be displayed in the text box

Each row in the CSV file represents a different, large number. Find the final ten digits of the sum of all rows of the CSV file (that is, the ones place, the tens, etc.). This process should be robust in that it should work with no regard to the size of the numbers (it should work with small numbers as well!)

My added methods are contained in CSVSumForm.cs as follows:

-browseButton_Click(sender: object, e: EventArgs)

-readCSV(filePath: string)

-parseLine(line: string, lineCounter: int): ulong

-updateOutputBox(currLong: ulong)

Thoughts During and Afterwards

Does this program allow for many, many large numbers to be processed?

The program I made uses ulong to store the numbers, if each field from the input file is at least 10 digits then ulong will overflow as you approach field number 2 billion. If more than 2 billion fields would be present I would swap from ulong to BigInteger, Decimal, or some custom type depending on what is needed.

Is it easy to use/adjust it to (or does it simply work out of the box with) a pure comma separated values file? (aka no single quotes)

I parsed out any non-digit characters, so the single quotes are ignored but so are decimals and negative symbols. The program should work with any single column csv file such as the one provided.

How could this be made faster?

Using a MemoryMappedFile, using pointers for my variables, using pre-made libraries for the file reading/parsing, waiting to update the TextBox until the entire file has been read and therefore converting types less often, a different language, using something other than a Windows Form App, etc. would all likely be a bit faster. However, given the instructions and the simplicity of the project I didn't think the above changes were necessary.

Does it work with negatives? Should it?

It does not currently work with negatives. I couldn't think of a real-world application that would need both negatives and positives for the 10 least significant digits so I ignore negatives.

If I needed to account for negatives I would change my parseLine method to not remove negative symbols on its initial pass over. I would then check if the string has a negative symbol on the front before a second pass over the string to remove negative symbols. I would then get the substring and "add" the negative symbol back in if it initially had one. I also wouldn't use ulong as the data type.

Is the entire CSV being loaded and thus held immediately into active memory? How might this cause problems for very large number loads?

I'm using StreamReader to read the CSV line by line so the whole file is not loaded at once.

Does the memory usage of your application grow over time? Could this be prevented?

There appears to be a 2-3 MB increase every time an OpenFileDialog is opened, upon closing the OpenFileDialog the Process Memory goes back down by 1-2 MB. So, there is an increase in memory usage as the program is used. I dispose of the OpenFileDialog and the StreamReader and am under the impression that the garbage collector gets the rest so I'm not sure if the memory increase could be prevented. I would greatly appreciate you telling me if/how I could prevent the increase after looking at the program.

Are we processing unnecessary information?

I don't think I am processing much unnecessary information. I initially read a full line/row from the csv, the first column is then sent to be modified and the rest are discarded.

Saving perhaps the most important for last: What kinds of input would destroy your program? Should it not crash? Does it crash gracefully?

A corrupted csv or some other file type that has its extension changed to csv will not work correctly but will likely not crash. For each line in the corrupted file a MessageBox will consecutively appear stating "Conversion of string to long failed." as well as the line number, and the program will count that line as 0. For example, I converted a .png file to .csv and tried my program and was given a MessageBox stating failed conversion on line number n, around 43,000 times.

Additional question.

I put the few methods I needed (browseButton_Click, readCSV, parseLine, and updateOutputBox) in the CSVSumForm.cs file. If there is a preferred practice, such as creating a new class with the logic in it (even for projects this small) I'd appreciate the feedback. Thank you.