Report Document – Daniel Dixon

Contents:

Implemented Features:

Option Choice: The user is given the choice of choosing either option 1, keyboard entry, or option 2, text file entry, either choice takes goes to different methods and allows the user to input their data before analysis is

conducted. There is also a third option to exit the application when done, this was added so several inputs can be entered by the user without having to start up the application again.

Keyboard input: The user may input whatever they wish, separating sentences with either a full stop or by entering the enter key. Analysis will be carried out when the * key is entered. Every letter is stored in a char array and looked over individually before the analysis is done.

Text file input: As with the Keyboard input, text analysis is carried out and details on the text file are gathered. There is an option to analyse a file of your own choosing by giving the full location. When a word from the text file is more than seven letters long it is stored as a text file named “longwordoutput.txt” created in the same directory that the initial file is taken from.

Data Structures Applied:

Char and String Arrays: Before being taught lists and other data structures, several char arrays were put in place that would take the user input from either the keyboard or text file and run through the basic analysis, this was deemed the easiest route as the array only had to be read through once and separate counters could be used to calculate each character’s features (vowel, consonant upper case etc.). String Arrays were used later on when for storing all the words in the two sentiment text files. This was done for efficiency as each word in the list was separated by a distinct characteristic (\r\n in this case) and so could be easily sorted and stored this way.

Lists: Lists were used mainly for the sentiment function, they were seen as more useful than linked lists as each word in either the positive or negative text file did not need to be in a specific order or linked to each other in any way apart from the fact they were all positive. It was also chosen instead of Arrays as the list had to be traversed several times and it is much simpler to do it as a list with short commands than an array with a number of different large loops.

Dictionary: A Dictionary was researched thoroughly (C# Kicks) and then used to create an alphabet, which was then used to compare to the user input. Because A key could be used (which was the alphabet in this case) and that key links to a specific value, it was perfect for linking all the letters to the number 0 initially and then adding 1 to it every time the character is seen in the code.

Additional Features Added:

Extra letter frequency functionality: When searching for the frequency of letters in the input, the application can look for either the number of specific letter or for every letter in the input (see dictionary) and then output how many of those characters exist in the user’s input.

Exception Handling: Several efforts were made to fix possible exceptions that could break the code, generally these exceptions were found by using different inputs myself from different locations or getting a different viewpoint from bug finding conducted by other users. This handling included not adding to the sentences variable when only a space or star is entered into the keyboard input. Allowing people to search for specific letters multiple times without their totals being added together.

Sentiment Analysis: To get the user's sentiment, a list of words was used that were seen as either positive or negative, they were taken from twitter's own sentiment analysis (Breen, 2010) and the text files created used to compare with the user input. They were initially going to be taken from Yahoo's Language Detection (Several Authors , 2016) but the report was not as comprehensive in terms of words used as the one used was.

Possible Future Improvements:

If this task was to be completed again, I would do several things to create better code. Improved Exception Handling and code efficiency would be the main goal initially as there were several areas I believed could be made more efficient with more time. I would also have liked to implement grammatical mistakes analysis, looking at each word of the user input and testing it against the oxford dictionary, checking for errors in spelling.

Video URL:

YouTube: https://youtu.be/jUcoueEyzzA

References:

Jeffrey Breen. (2010). *Twitter Sentiment Analysis.* Available: https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107/tree/master/data/opinion-lexicon-English. Last accessed 22nd Dec 2016.

Sam Allen. (2007-2016). *DotNetPerls C# Guides.* Available: https://www.dotnetperls.com/. Last accessed 21st Dec 2016.

Several Authors. (2016). *Abusive Language Detection in Online User Content.* Available: http://www2016.net/proceedings/proceedings/p145.pdf. Last accessed 15th Dec 2016.

Visual C# Kicks. (Unkown). *C# Data Structures.* Available: http://www.vcskicks.com/csharp_data_structures.php. Last accessed 20th Dec 2016.