CS 3202 – Spring 2024
Assignment 2, 20 pts.
Due: Mon, Mar 11th at 9:00a

Objectives

- Learn and apply the basics of C++ programming
  - Creating and using classes
  - Iteration, control constructs, etc.
  - Using STL library classes and algorithms.
- Write a command-line program
- Apply best practices in OO design, e.g. model-view separation
- Apply clean coding practices

Getting started

1. Read through the entire assignment description before beginning.

2. Create a new C++ console application using Visual Studio (NOT Visual Studio Code). In Visual Studio create a C++ Windows Console application. I suggest using the one that prints "Hello World" be default. Name the project TextAnalyzer.

3. Add a notes.txt file that will be used to note any issues that remain in the application upon submission.

Specifications

The program must be written in C++ must be able to be loaded as a Visual Studio solution and be executed on the CS Windows Advanced Lab from vlab.westga.edu. Make sure the project can load, compile and run from Visual Studio.

Any project that does not compile will receive a zero.

Requirements

1. This C++ application will be a command-line application that will implement functionality for the following usage statement:

usage: TextAnalyzer infile [outfile] [options]
by *Firstname Lastname*
Analyzes and outputs the number of words in the infile.

infile	The input file to analyze the words.
outfile	The output file to write the output to.

options:

/?	Displays this usage statement.
/a <word count>	Add the specified count of the word to the output. E.g., /a apple 3 would add three occurrences of apple.
/c<number>	Changes the number of output columns to the number specified. The default number of columns is 4.
/d <word count>	Deletes the specified count of the word from the output. E.g., /d apple 3 would delete 3 occurrences of apple.
/da <word>	Delete the specified word completely from the output. E.g., /da banana would eliminate banana from the output.
/o	Automatically overwrites the outfile without prompting, if the outfile already exists. If this option is not specified and the outfile exists the user will be prompted on whether to overwrite the outfile.
/sa	Displays the output in alphabetic order instead of by frequency which is the default output format.
/w<number>	Changes the column width for the output columns. The default column width is 18.

- Note for the /c and /w options there is no space between the letter and the number. The option would be specified as -w15 **NOT** -w 15 or -c3 **NOT** -c 3. The other options have a space between the option and the input into that option, for example /d apple 3.
- For more information on usage statements read the following: How to write usage statements.
    - Note: The format of the above usage statements is slightly different than the format given at the above website, but the same principles apply.
- Information on processing command-line arguments can be found at some of the C++ resources websites given in Moodle.

2. Words - The input file will consist of text, which may contain punctuation.

    a. A word is defined as a string of one or more letters.
    b. Apostrophes and hyphens are allowed, if they are surrounded by letters.
    c. Other characters may separate words (e.g. commas, semi-colons), but are not part of a word.
    d. A word could have multiple symbols at the beginning of the word, but these are not part of the word. For example, "#Hello there!" would be just the words `hello` and `there`.

e. The program should be case insensitive, e.g., This and this would be considered the same word.
f. All displayed words should be in lower case.
g. Duplicate words should not appear in the list of words twice, but instead should add to the frequency count for that word.

3. Infile – if the input file is not found the program should indicate so.

4. If the command entered to run the program at the prompt is incorrect, i.e., the command-line arguments are not correct. the usage statement must be displayed.

5. Output requirements

    a. The frequency output must be in decreasing order. The frequency heading should state "*Words occurring ? times*", where ? represents the frequency of the words. If there are no words of a particular frequency then that frequency should not be displayed. Within the frequency group the words should be output in alphabetic order.
    b. The output for alphabetic order should state "*The words starting with ?*" where ? is the letter the words start with. If there are no words that start with a particular letter then the nothing should be output at all for that letter.
    c. All words output should be in lower-case with no surrounding punctuation.
    d. The output for all the words for a particular letter should be in the specified number of columns with the default value being four (4) columns. Each column should be left justified.
    e. The default output column width will be 18, but can be adjusted at the command-line.

6. Implementation requirements

    a. You may store the words in the data structure of your choice.
    b. Use C++ constructs and not C constructs.
        i. For example, formatting of output should NOT be with `printf` statements, but instead use the C++ ostream library to format the output.
    c. The implementation needs to follow best program design and implementation practices such as separation of concerns, etc.
    d. Format the code using the ReSharper Code Cleanup, but only perform the *Built-in: Reformat Code*.
    e. Make sure every public function has a function specification. This includes the main function and all stand-alone functions.

Example output

Given an example text file called test.txt below:

Answer: The cat in the hat

ate green eggs and ham, said “Sam I am.” Another
great story and another great adventure.

Example 1

>TextAnalyzer test.txt

```
Words occurring 2 times:
and               another           great             the

Words occurring 1 time:
adventure         am                answer            ate
cat               eggs              green             ham
hat               i                 in                said
sam               story
```

Example 2

> TextAnalyzer /d the 1 test.txt /a and 4 /da great /w15

```
Words occurring 6 times:
and

Words occurring 2 times:
another

Words occurring 1 time:
adventure      am             answer         ate
cat            eggs           green          ham
hat            i              in             said
sam            story          the
```

Example 3

> ***TextAnalyzer /w15 /c3 –sa test.txt output.txt –o***

The following output would be displayed to the screen and written to output.txt and overwrite that file if it exists.

```
Words starting with a:
adventure      am             and
another        answer         ate

Words starting with c:
cat

Words starting with e:
eggs

Words starting with g:
great          green

Words starting with h:
ham            hat

Words starting with i:
i              in

Words starting with s:
said           sam            story

Words starting with t:
the
```

Submission

To submit the project do the following:

1. Update the notes.txt file to list any known bugs. *If there are no known issues, please indicate so.*
2. Clean the project to remove all files that can be regenerated upon building the project.
3. Close Visual Studio.
4. Navigate to the root folder for the project and zip the entire solution and name the zip file *FirstnameLastname*A2.zip and submit the zip file to Moodle by the due date.

Grading rubric

Any program that does not compile will receive a 0. Partial credit is not possible for any program that does not compile.
If a program is only partially complete and a category cannot be accurately assessed you will not receive full credit for that category.

	Exceptional	Acceptable	Amateur	Unsatisfactory
Functionality	12 pts. – grading on continuous scale.			
Readability/formatting organization	3 pts.	2 pts.	1 pt.	0 pts.
Reusability	3 pts.	2 pts.	1 pt.	0 pts.
Documentation	2 pts.	NA	1 pt.	0 pts.

Grading description

	Exceptional	Acceptable	Amateur	Unsatisfactory
Readability/ formatting/ organization	The program is exceptionally well organized, very easy to follow, and there are not any compiler warnings.	The code is fairly easy to read and there are a few compiler warnings that were not legitimately explained as to why they still exist.	The code is readable only by someone who knows what the code is supposed to be doing and/or there are many unresolved compiler warnings.	The code is poorly organized and very difficult to read.
Reusability	The code could be reused as a whole or each routine could be reused.	Most of the code could be reused in other programs.	Some parts of the code could be reused in other programs.	The code is not organized for reusability.
Documentation	The required documentation is well written and clearly explains what the code is accomplishing. There is not any redundant inline commenting.		The required documentation is there, but not complete or is only somewhat useful in understanding the code.	The documentation is poor and very incomplete.