

CSE333 Exercise 8

Out: Wednesday, February 9

Due: Wednesday, February 16 by 11 am PST

Rating: 3 (note)

Goals

- Use C++ streams to read files
- Use C++ STL containers (*e.g.*, `map`)
- Write templated C++ code

Problem Description

Write a C++ program that reads a file whose filename is given as a single command-line argument. The program should:

1. Read and count the individual strings in the file, then
2. Print the final list of the strings, sorted alphabetically, and the number of times each word appears in the file.

You **MUST** factor out a helper template function with the following prototype:

```
template <typename T> bool ReadValue(istream& in, T* output);
```

- Each time the client calls `ReadValue` , it should read one token from the input stream `in` , attempt to parse the token into type `T` , and store the value in the location pointed to by `output` .
- Returns `false` if either reading or parsing fails (*e.g.*, conversion errors, EOF conditions), and `true` otherwise.

Additional Details

- In this problem, we define a string as a series of non-whitespace characters.
- Each string should be written on a separate output line with the word first followed by a single space and the number of occurrences.
- Strings must match EXACTLY to be considered the same (including capitalization).

- You may NOT assume that the file ends with a newline character (`'\n'`). Since most text editors automatically add a newline when you save a text file, we are providing you a file called `newline.txt` for testing, which you can download with the command:

```
bash$ wget https://courses.cs.washington.edu/courses/cse333/22wi/exercises/ex8_files/newline.txt
```

Example

If the file `quotes.txt` contains:

```
to be or not to be
to do is to be
to be is to do
do be do be do
```

then running `./ex8 quotes.txt` should output:

```
be 6
do 5
is 2
not 1
or 1
to 6
```

Implementation Notes

File Reading

You do not need to do additional processing of input (e.g., whitespace or punctuation) from a file; using the `>>` operator to read in should suffice. For checking for errors, we recommend you investigate the `good()`, `bad()`, and `eof()` methods for streams. More information on how to read files in C++ can be found here:

<http://www.cplusplus.com/doc/tutorial/files/> (<http://www.cplusplus.com/doc/tutorial/files/>)

Templates

When you write template code, you should make as few assumptions about the template type's functionality as possible (i.e., what it has implemented). For this exercise, you may assume there is a default constructor and `operator>>` with input streams defined. Note that this means that if you were to change `main` and your input file to only contain integers instead of strings, your `ReadValue` function should still work! However, make sure that the code you submit reads strings.

Program Output

The list of strings should be sorted using the ordinary ordering for strings (i.e., `operator<`). There should be exactly one space character between the string and its count, with no whitespace after the count or before the string.

Style Focus

Don't forget that good practices from previous exercises still apply!

Robustness

This exercise reads in user input via command line arguments and reads a file. Both of these can raise errors and need to be checked accordingly. If unrecoverable errors are detected, be sure that you print a useful message to standard error and return `EXIT_FAILURE` from `main`. For this exercise, you can consider a parsing error to be unrecoverable.

Submission


You will submit: `ex8.cc`.

Your code must:

- Compile without errors or warnings on CSE Linux machines (lab workstations, `attu`, or CSE home VM).
- Have no runtime errors, memory leaks, or memory errors (`g++` and `valgrind`).
- Be contained in the file listed above that compiles with the command:

```
bash$ g++ -Wall -g -std=c++17 -o ex8 ex8.cc
```

- Have a comment at the top of your `.cc` file with your name(s) and CSE or UW email address(es).
- Be pretty: the formatting, modularization, variable and function names, commenting, and so on should be consistent with class style guidelines. Additionally, the linter shouldn't have any complaints about your code (`cpplint.py`).
- Be robust: your code should deal with hard-to-handle/edge cases and bogus user input (if there are any) gracefully.

Submit your code on  Gradescope (<https://www.gradescope.com>). Don't forget to add your partner if you have one.