CSE 224 Programming Assignment #4 Fall 2019

Overview

For this assignment, you're going to read lines of text from stdin, break them into phrases, store them in an array, and record how many times each phrase occurs in the input. Upon reaching the end of the input (signaled by the user hitting CTRL-D alone on a line), your program should print out a list of all phrases it found, along with the number of times each phrase occurred. The list should be sorted in increasing order of length, i.e., print the shortest phrases first, the longest ones last (the order of phrases that are the same length does not matter).

Details

• **convert** everything to uppercase: Hello hello and HELLO are all treated the same;
• **convert** newlines and other whitespace (TAB, etc.) into spaces (so a phrase split on 2 lines doesn't get broken by a newline) Use the isspace() function to detect whitespace
• **ignore** the following characters: ( ) ' " -
• **a phrase:**
- begins with any non-space character other than , . ; : ? !
- is a collection of consecutive letters, characters and spaces, excluding , . ; : ? !
- ends with any of the following characters , . ; : ? !

For your final output, print the occurrence count first, as a 5-digit integer with leading 0's included; followed by the phrase enclosed in <angle brackets>

So here's an example of some pseudocode:

1. read* from stdin until you find the beginning of a phrase
2. read* from stdin until you find the end of a phrase, storing each character in a temporary string
3. if you've never seen this phrase before, save it, and record 1 for the # of occurrences
   if you've already saved this phrase, increment the occurrence counter
4. Repeat steps 1-3 until you hit EOF
5. sort the array of saved phrases by length
6. print the sorted array of phrases, along with the occurrence count.

*you may want to code your own getchar() function, to handle the following:

- ignore ( ) ' " -
- convert all whitespace to a space (' ')
- convert letters to uppercase

You shpuld assume:

- each phrase is 200 or fewer characters (if a phrase is longer, ignore **the remainder** of the phrase);
- there is a maximum of 1,000 phrases (if you encounter more phrases, just ignore them...but keep tallying the first 1,000 phrases)

You should use statically-defined arrays, i.e., declare a char dictionary[1000][201] and int count[1000], where dictionary[i] stores the i^{th} phrase and count[i] stores that phrase's tally. Alternatively, you are free to use structures to simplify your coding :)

All code must be your own, other than routines from stdio.h, string.h, ctype.h and stdlib.h Do not use any pre-built sorting or hash functions.

Here's a sample input file:

> Hello, hahaha: this is a sample file. It contains
> a number of lines, with different punctuations and
> spacings.
> We can analyze this file, breaking the input into phrases,
> and tallying them.
> But be careful: there are a lot of details to worry about!
> So, this is a test file.
> Also, this is a sample file.
> So, it is both! Hahaha.

and here is what it should output:

00002 <SO>
00001 <ALSO>
00001 <HELLO>
00002 <HAHAHA>
00001 <IT IS BOTH>
00001 <BUT BE CAREFUL>
00001 <AND TALLYING THEM>
00001 <THIS IS A TEST FILE>
00002 <THIS IS A SAMPLE FILE>
00001 <WE CAN ANALYZE THIS FILE>
00001 <IT CONTAINS A NUMBER OF LINES>
00001 <BREAKING THE INPUT INTO PHRASES>
00001 <WITH DIFFERENT PUNCTUATIONS AND SPACINGS>
00001 <THERE ARE A LOT OF DETAILS TO WORRY ABOUT>

I strongly recommend experimenting with different input files!

Suggestions

This assignment is an excellent opportunity on which to practice modular coding, and a top-down design methodology. Think carefully about your overall algorithm; break it into a few steps and code your main loop accordingly. Then begin coding the functions to implement each of those steps. Work in pieces, developing, testing and debugging each piece as you go. Some of the pieces you may need include:

- a way to read stdin and pre-process the characters;
- a way to detect the beginning of a phrase;
- a way to detect the end of a phrase;
- a way to store a phrase into a dictionary (either as a new entry or an additional tally on a phrase you've already seen);
- a way to sort your dictionary;
- a way to print your dictionary.

Important/Final Notes

- Submit via Gitlab in a repository named “Index” by the due date/time.
- Your submission **must** include a Makefile
- Document everything (lots of comments!).
- Be careful to handle long phrases, as well as input containing more than 1000 phrases.
- Sample executable is available at /tmp/index-sample.
- /tmp/WarandPeace.txt is a good large testfile!