# Lab 5 – Strings – Editor Functions

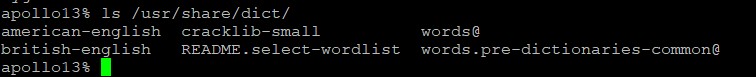
In this lab, each student is to write a program called **prog5.c** that performs various string operations on a text file such as searching for text, replacing words, checking spelling, and formatting lines. The student should exhibit his understanding of:

* **Operations on Strings**
* **Dynamic Memory Allocation**
* **File Operations**
* **Using Command Line Arguments**
* **Calculating and Storing Statistics**

**Background**

**Linux** provides one or more dictionary files which contains common English words. This program will allow a user to input a dictionary filename to check the spelling of a file.

On the **Apollo** machines, several dictionaries are present:



**Input**

The program should accept three arguments as

**prog5 *input\_file* *output\_file dictionary***

where ***input\_file*** is the name of an **ASCII** input text file to be operated on, ***output\_file*** is the name of the output file to create, and ***dictionary*** is the name of the dictionary file to use for checking spelling.

**Menu**

The program should prompt a user for the following options.

* Check the spelling of the file using ***dictionary***.
* Search for and replace a given string in the file.
* Save the modified file to the output file, ***output\_file***
* Exit the program

**Operation**

Upon execution, the program is to open the input file, ***input\_file***, and read in each line of the document into separate dynamically allocated strings contained in a string array. *Note: All operations should be performed on this file data stored in RAM.*

**Spell Checking**

To check the spelling of the document, the program should start at the beginning of the file and read in sequential words checking their spelling—*regardless of case*—against the given dictionary, ***dictionary***.

When a word is found that is not present in the given dictionary, the program should prompt the user asking if they wish to replace the word or ignore it.

If they wish to replace it, then the program should prompt the user for a new word with which to replace it.

If they wish to ignore the word, then the program should add that word to a personal dictionary file. *Note: The first time a word is added to the personal dictionary, the user should be prompted for the user dictionary filename.*  Once a user dictionary has been created, it should also be checked when checking spelling.

**Search-and-Replace**

When performing a search-and-replace, the program should prompt for a text string to be searched for. *Note; The text strings can be any sequence of printable* ***ASCII*** *characters including spaces. Furthermore, the program does not have to search across lines of the file, that is, across a carriage return-line feed boundary.*

When the phrase searched for is located, the program should display the location of the phrase found in terms of the line of the file (1 to n) and the character position in that line.

The program should then prompt for a new text string to replace the string searched for. *Note: This* *phrase can be any sequence of printable ASCII characters including spaces.*

For bonus points, the program can prompt the user whether they wish to ignore the case, or not, when performing a search and replace.

For bonus points, the program can prompt the user whether they wish to search for “whole words only”, or not.

For bonus points, the program can prompt the user to search across separate lines of the file.

**Storing the Modified File**

When the user selects to store the modified file contained in RAM, the program should write the lines stored in memory to the output file, ***output\_file***.

For bonus points, the program can write a maximum of eighty characters to each line in the output file, ***output\_file***.

**Test Documents**

I will provide several text files to analyze.

**Further Considerations**

The program should be structured neatly, easily readable, and well commented. Code should be modularized with functions, logically structured, and written to perform efficiently as possible. Furthermore, variable and function names should be such that the software is as “self-commenting” as possible.

The **main** function should mainly contain only functions called to complete the separate tasks given.

**Creation and Submission**

***Each individual student must complete their own program. Copying other students’ code will be tested for and will not be tolerated.***

The following line should be used to compile the program

**gcc -Wall -g prog5.c -o prog5**

The code you submit must compile using the **–Wall** flag and no compiler errors or warnings should be printed. To receive credit for this assignment the code must compile and at a minimum perform some required function.

Code that does not compile or crashes before performing some required function will not be accepted or graded. **All students must do a final check on one of the CES Ubuntu machines to verify that gcc using Ubuntu shows no warning messages before submitting the project.**

The project must be submitted to Canvas before midnight on **Wednesday, April 3rd**.