SET07109 Programming Fundamentals

1 Overview

This is the first coursework for SET07109 Programming Fundamentals. This coursework is worth **40%** of the coursework total. The coursework is worth **60%** of the module total. Therefore this coursework is worth **24%** of the total marks available for this module.

In this coursework you are to build a command line tool, called `spell`, that can check a text document for spelling errors by checking that every word in the document is contained in a dictionary file. The tool outputs the words that are not in the dictionary, along with the corresponding line numbers on which they occur.

The tool will take a number of command line arguments that tell it how to work. For example, to spell check a file called "input.txt", and output the misspelt words and their line numbers to the file "output.txt", the tool will be run as follows on the command line.

```
spell -i input.txt -o output.txt
```

The specification will go into detail about how the application should behave and the minimum requirements.

2 Specification

The `spell` tool is required to output words that are not included in the supplied "dictionary.txt" file, along with the line numbers on which these words occur. The dictionary contains a sorted list of acceptable words, with one word per line. This is supplied to you in the file "dictionary.txt" (taken from the open source Fedora Linux distribution). The tool should understand the command line arguments shown in Table 1.

Therefore we can spell check the file *mydoc.txt* using the command:

```
spell -i mydoc.txt -o output.txt
```

Argument	**Description**
`-i` *filename*	The name of the file to spell check.
`-o` *filename*	The name of the file to write the misspelt words and their line numbers to.
`-c`	Indicates that the case of the words to be checked can be ignored.

Table 1: Minimum command line flags for the `spell` application.

We can also spell check *mydoc.txt*, ignoring case, as follows:

```
spell -i mydoc.txt -o output.txt -c
```

Some default functionality is required by the application:

1. If no input filename is provided then the application should read from `stdin`
2. If no output filename is provided then the application should write to `stdout`

Recall from the Workbook that `stdin` and `stdout` refer to the console.

Every time the tool finds a word not in the dictionary it should output both that word, and the line number in the input document in which that word occurs. The tool must check the entire input file. You may assume that, apart from upper and lower case letters, the only characters the input will contain are commas, full stops, new lines, question marks, spaces, and tabs. A word is defined as a continuous sequence of upper and lower case letters only. Where the case is not ignored (i.e. the tool is run without the -c option) then it should account for the fact that the first character of the first word in a sentence will be uppercase. These words should not be classed as a misspelling if there is a corresponding word in the dictionary that begins with a lower case letter. Note that your application does not need to enforce that there is actually a capital letter in the first word of a sentence.

You are supplied with two input files that we will use to test your application. The first contains a single word on each line, the second contains complete sentences (the text in these files is based on the C Programming Wikibook located here: `https://en.wikibooks.org/wiki/C_Programming/Why_learn_C%3F`).

As an extension for additional marks, the tool should exploit the fact that the words in the dictionary file are sorted in order to increase performance. You should do this by searching the dictionary using the standard *binary search algorithm.* Note that the "dictionary.txt" file is sorted in ASCII order. You should research this algorithm, and implement it yourself in your program code (the use of prewritten library routines for this will not be accepted for the extension marks).

3 Code quality and best practise

Code quality includes meaningful variable names, use of a consistent style for variable names, correct indentation, and suitable comments. Variable names must begin with a lower case letter. Constants should be written in uppercase, and used instead of literal values wherever possible.

The source code should have a comment at the top detailing the name of the author, the date of last modification, and the purpose of the program. Every function should be preceded by a comment describing its purpose, and the meaning of any parameters that it accepts. Any complex sections of code should be commented.

You must include a `makefile` to compile the application. The `makefile` must also contain configurations to execute the application on each of the two test files, with and without the -c option. It must also contain a clean configuration to delete .obj and .exe files.

Finally, because C is a low level language, remember to free any memory that you dynamically allocate, and close any files that you open.

4 Submission

Your code must be submitted to the Moodle Assignment Dropbox by 23:59 on Sunday the 26th of February. If your submit late without prior authorisation from your personal tutor your grade will be capped at 40%.

The tool must be coded entirely in C. Submissions in C++ or any other language will not be marked. Your program code may only make use of standard C libraries. Ask a member of the teaching team if you are unsure.

The submission must be your own work. **If it is suspected that your submission is not your own work then you will be referred to the Academic Conduct Officer for investigation**.

All coursework must be demoed in your timetabled practical session in Week 8. If the coursework is not demoed to a member of the teaching team this will result in a grade of 0. If the demo is not done in Week 8 then your mark may be capped at 40%, unless you have authorisation from your personal tutor. You must print

Description	**Marks**
File input-output functionality	2
Command line arguments work	2
Application searches the dictionary for each word in the input	4
Application handles capitalisation correctly	2
Application outputs misspelt words and their line numbers	2
Default application behaviour works correctly	2
Resources freed and cleared up correctly	2
Code quality	2
Extension: implementation of the binary search algorithm	2
Makefile complete and works correctly	2
Submission zip folder complete and correctly collated	2
Total	24

Table 2: Marking scheme.

the mark sheet from Moodle, type your name and matriculation number, and have it available on paper at the start of the demo. The demo session is the point where the teaching team will give you initial feedback on your work. Further one-on-one feedback will be made available in the subsequent practical sessions to those students who request it.

The submission to Moodle must take the form of the following:

- The code file(s) required to build your application.
- The supplied test files and dictionary.
- A `makefile` to allow building of your application.
- A *read me* file explaining how to use the `spell` tool, as well as how to build it from the source, and the tool chain used for building (e.g. Microsoft Compiler, GCC, etc.) including version numbers.

These files must be bundled together into a single archive (a zip file) using your matriculation number as a filename. For example, if your matriculation number is *1234* then your file should be called *1234.zip*. All submissions must be uploaded to Moodle by the time indicated.

5 Marking Scheme

The marking scheme for this coursework is shown in Table 2. To pass this coursework you will require a minimum of 10 marks, although you should be aiming to achieve a much higher mark to illustrate your understanding of the material.

BE WARNED! This coursework requires a thorough understanding of the material covered in the first 4 units of the module. If you do not complete these you will struggle. DO NOT LEAVE THIS WORK TO THE LAST MINUTE!

If you have any queries please contact a member of the teaching team as a matter of urgency. This coursework is designed to be challenging and will require you to spend time developing the solution.