School of Computing, Edinburgh Napier University

Assessment Brief Pro Forma

1. Module number	SET07109
2. Module title	Programming Fundamentals
3. Module leader	Simon Powers
4. Tutor with responsibility for this Assessment Student's first point of contact	Simon Powers S.Powers@napier.ac.uk
5. Assessment	Practical Skills Assessment 1
6. Weighting	24% of module assessment
7. Size and/or time limits for assessment	You should spend approximately 24 hours working on this assessment.
8. Deadline of submission	01/03/2020 at 15:00 Your attention is drawn to the penalties for late submissions.
9. Arrangements for submission	Submit to the Moodle Dropbox by 15:00 on Sunday 1st March. You are advised to keep your own copy of the assessment.
10. Assessment Regulations	All assessments are subject to the University Regulations.
11. The requirements for the assessment	See attached specification.
12. Special instructions	See attached specification.
13. Return of work	Feedback will be provided at the demonstration session. Final marks will be returned via Moodle. Additional one-to-one feedback will be given to any student that requests it in a subsequent practical session.
14. Assessment criteria	See attached specification.

SET07109 Programming Fundamentals Coursework 1: Creating a command line application to find and replace strings in text files

1 Overview

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

In this coursework you are to build a command line application, called `find`, that can find occurrences of a string in a text file, and optionally replace those occurrences with another string that the user specifies. The application will take a number of command line arguments that tell it how to work. For example, to find all occurrences of *hello* in a file called `input.txt`, and replace them with *hi*, saving the result in a file called `output.txt`, we would run the application as follows on the command line:

```
find hello -i input.txt -r hi
```

Note that the input file (`input.txt` in this case) should remain unmodified, while the output file should contain all of the text from the input file, except for every occurrence of *hello* being replaced with *hi*. The next section gives the full specification of the application in detail.

Your application must be written entirely in C, and may only use prewritten functions from the C standard library. Applications written in C++ or any other language will not be marked. If you are in any doubt about what is allowed then you should consult the module leader.

2 Specification

The application is required to search a text file for all occurrences of a string that the user specifies. This string will be given as the first command line argument. The application should output each word that contains the string to the console (a word is defined as a continuous sequence of alphabetic characters). For example, if *el* is searched for in a file that contains the line of text: `Hello, and welcome to C.`
the console output should be:

```
Hello
welcome
```

The `-r` *replacement* command line argument is optional. Where given, an output file called `output.txt` should be created by your application. This should contain all of the text of the input file (including punctuation and white space), with the exception that every occurrence of the found string should be replaced with *replacement*. For example, if *el* is searched for in a file that contains the line of text:
`Hello, and welcome to C.`
and the command line argument `-r EL` is given, then the output file should contain the text:
`HELlo, and wELcome to C.`

The console output should then contain the original words in which the string was found, followed by the words after replacement, e.g.

```
Hello HELlo
welcome wELcome
```

Where the input file contains multiple lines of text, then the find and replace should occur throughout the whole text file.

The application should understand the command line arguments shown in Table 1, which should be accepted in any order.

3 Testing and advice

You are supplied with two test text files on Moodle. The first file, `single_words_test.txt`, contains a single word on each line. This file will be easier for you to process. The second file, `sentences_test.txt`, contains full sentences including punctuation and white space, which your

Argument	Description
`-i` *filename*	Mandatory. Gives the name of the input text file.
`-r` *replacement*	Optional. Indicates that an output file called `output.txt` should be produced that contains all of the text of the input file (including punctuation and white space), with the exception that every occurrence of the found string should be replaced with the string *replacement*.
`-c`	Optional. Indicates that the case of the letters in the input file should be ignored when searching for the string. For example, when searching for *hello*, both *Hello* and *hello* should match. Without this option, only *hello* should match.

Table 1: Command line flags for the `find` application.

application should handle correctly as per the above specification. You can achieve some marks even if your application only works on the easier `single_words_test.txt` file. If you are unable to get the command line arguments working then you may hardcode the string to search for, the replacement string, and the input filename into your program. Although you will lose some marks for this (see the mark scheme at the end of this document), it will allow you to demonstrate the functionality of your program. During your demonstration we will test your application on the `single_words_test.txt` and `sentences_test.txt` files, by searching for and replacing various strings.

You should examine the functions available in the C standard library to see those that may assist you. Documentation for the functions in the C standard library is available here: `http://www.cplusplus.com/reference/clibrary/`. For a reminder about command line arguments you should consult Section 1.8 of the Workbook.

You should make use of your timetabled practical sessions before the hand-in date, where the teaching team will provide you with feedback on your solution as you develop it.

4 Code quality

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 where appropriate.

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 remember to close any files that you open when your application has finished with them.

You must include a *makefile* to build the application. The *makefile* must also contain a clean configuration to delete object and executable files.

Your code will be assessed on all of these criteria.

5 Submission and Demonstration

Your code must be submitted to the Moodle Assignment Dropbox by 15:00 on Sunday 1st March. If you submit late without prior authorisation from your Personal Development Tutor your grade will be capped at 40%.

The submission must be your own work, written by you for this module. Collusion is not permitted. The university regulations define collusion as

> "Conspiring or working together with (an)other(s) on a piece of work that you are expected to produce independently."

Please acknowledge all sources of help and material through comments in the source code giving a link to the material that you have consulted, e.g. include a URL to any Stack Overflow code segments that you have incorporated in your work. If you use *Git* or other version control then please make sure that your coursework is stored in a private repository to prevent access by others. **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 demonstrated to a member of the teaching team. If the coursework is not demonstrated then this will result in a grade of 0. The demonstration session is the point where the teaching team will give you summative (marked) feedback on your work. Final marks will be returned within three working weeks via Moodle. Further one-on-one feedback will be made available in the subsequent practical sessions to any student who requests it.

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

1. The code file(s) required to build your application.

2. A *makefile* to allow building of your application.

3. A *readme* file explaining how to use the `find` application, as well as how to build it from the source, and the tool chain used for building (e.g. Microsoft Compiler, Clang, etc.).

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.

6 Marking Scheme

The marking scheme for this coursework is shown in Table 2 at the end of this document.

BE WARNED! This coursework is intended to be challenging and will take time to think through and implement a solution. You will need 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!

We hope you enjoy the process of designing and implementing a complete application from scratch. If you have any queries please contact the module leader as a matter of urgency.

Description	**Marks**
Finds strings in the `single_words_test.txt` file	3
Finds strings in the `sentences_test.txt` file	3
Replaces strings from the `single_words_test.txt` file	3
Replaces strings from the `sentences_test.txt` file	3
Output file created correctly with `-r` option	2
`-i` command line argument to specify input file name works	1
`-c` command line argument to ignore case works	2
Commenting standard in specification met	1
Code quality: Variable names, indentation, file handling	3
Makefile builds application successfully	1
Makefile written to specification, including a clean target	1
Submission collated correctly, including readme	1
Total	24

Table 2: Marking scheme.