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	03/03/2019 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 3rd 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	Initial feedback will be provided at the demonstration session. Additional feedback and 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 Coursework 1: Creating a C Language Preprocessor

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 application, called `preprocess`, that contains basic functionality of a C language preprocessor. Your application will read in a `.c` file containing source code written in the C programming language, and output a preprocessed version of that file according to the specification below.

Your application must be written entirely in C, using only the standard libraries supplied with a default installation of Visual Studio, GCC or Clang. Applications written in C++ or any other programming language, or those that use any other libraries, will not be marked. You should ask the module leader if you have any doubt as to what is permitted.

2 Specification

The features specified below are listed in increasing order of how difficult they are to implement.

The `preprocess` application is required to read in a `.c` file and output the following *to the console*:

1. The number of non-empty lines in the `.c` file. Non-empty lines are defined as those that contain at least one character that is not a newline, space, or tab.

2. The number of comments in the `.c` file. For this coursework, a comment is defined as beginning with `//` and ending at the end of that line, i.e. you do not need to consider multiline comments beginning with `/*`.

The application is also required to produce an output file which has the same name as the `.c` input file but with a `.o` file extension. This output file should contain a preprocessed version of the input file *containing the same lines of program code but modified according to the following specification*:

3. All comments should be removed. As above, a comment is defined as beginning with `//` and ending at the end of that line, i.e. you do not need to consider multiline comments beginning with `/*`.

4. A line that begins with `#include` *"filename.h"* should be replaced with the entire contents of the file called *filename.h*. Your application should remove the quotes when processing the filename. You may assume that *filename.h* is located in the same directory as the input file. The name *filename.h* should be considered as a placeholder, i.e. your application should work for any filename following a `#include`.

5. A line that begins with `#define` *constant_name value* should be processed as follows. The line should be removed. Throughout the rest of the input file, wherever *constant_name* appears it should be removed and replaced with *value* in the output file. You may assume that *constant_name* will always be proceeded and followed by a space in the input file. *constant_name* and *value* should be considered as placeholders, i.e. your application should work for any sequence of characters in place of *constant_name* and *value*.

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

Argument	**Description**
`-i` *filename*	The name of the `.c` input file to preprocess.
`-c`	Indicates that comments should be left in the output file instead of being removed.

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

Therefore we can preprocess the file `myprog.c` to produce the output file `myprog.o` using the command:

```
preprocess -i myprog.c
```

We can also leave comments in the output file for *myprog.c* as follows:

```
preprocess -i myprog.c -c
```

3 Testing

You are supplied with two test `.c` files along with the correct output in the corresponding `.o` files. During your demonstration you will be asked to first delete the `.o` files that we have given you, and then run your program on the `.c` files, with and without the `-c` option. Your *makefile* should include targets to perform these operations.

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 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 build 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, you must remember to free any memory that you dynamically allocate using `malloc` (if used), and to close any files that you open.

Your code will be subject to code review after the demonstration and 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 3rd March. If your 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 in your timetabled practical session on Tuesday 5th, Wednesday 6th, or Thursday 7th March. If the coursework is not demonstrated to a member of the teaching team this will result in a grade of 0. You must print the mark sheet at the end of this document and have it available on paper at the start of your demonstration. The demonstration session is the point where the teaching team will give you initial 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 `preprocess` application, as well as how to build it from the source, and the tool chain used for building (e.g. Microsoft Compiler, GCC, 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 at the end of this document. Features that are more difficult to implement attract higher marks. You do not need to successfully implement all features to pass. To pass this coursework you will require a minimum of 10 marks. However, you should be aiming to achieve a much higher mark to illustrate your understanding of the material.

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.

MARKING SHEET (TO BE FILLED IN DURING THE DEMONSTRATION)

NAME__________________________ **MATRIC** _____________

DEMONSTRATOR__________________________

Application functionality	**Mark awarded / max mark**
Makefile builds application	/ 1
Number of non-empty lines in source file output to console	/ 1
Number of comments in source file output to console	/ 2 (1 mark: some counting but incorrect total; 2 marks: fully working)
Comments removed when producing output file	/ 2 (1 mark: comments partially removed; 2 marks: comments fully removed)
#include works	/ 3 (1 mark: #include line removed from output but file not included; 2 marks: partially working; 3 marks: fully working)
#define works	/ 5 (1-2 marks: major errors; 3-4 marks: minor errors; 5 marks: fully working)
Output file written to disk	/ 1
-i command line argument to specify input file name works	/ 1
-c command line argument to leave comments in output file works	/ 1

OVERALL

_____/17

CODE REVIEW SHEET
(TO BE FILLED IN BY THE MARKER AFTER THE DEMONSTRATION)

NAME____________________________ **MATRIC** ______________

Item	**Mark awarded / max mark**
Commenting standard in specification met	/ 1
Variable names / indentation	/ 1
Dynamically allocated memory freed (if used)	/ 1
Files closed before program terminates	/ 1
Makefile contains all sections in specification	/ 1
Readme file present to specification	/ 1
Zip file submitted to specification	/ 1

OVERALL

_____**/7**

Total marks

From demonstration________ /17

From code review ________/7

Total________ /24