Edinburgh Napier UNIVERSITY

Assessment Brief Proforma

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*
5. Assessment	*Practical Skills Assessment 2*
6. Weighting	*36% of module assessment*
7. Size and/or time limits for assessment	*You should spend approximately 36 hours working on this assessment.*
8. Deadline of submission	Your attention is drawn to the penalties for late submissions *17/04/2018*
9. Arrangements for submission	*Submit to the Moodle Dropbox by 23:55 on Tuesday 17th April.* *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 and feedback	*Initial feedback will be provided at the demonstration session. Additional feedback and marks will be returned via Moodle.*
14. Assessment criteria	*See attached specification.*

SET07109 Programming Fundamentals
Coursework 2

1 Overview

This is the second coursework for SET07109 Programming Fundamentals. This coursework is worth **60%** of the coursework total. The coursework is worth **60%** of the module total. Therefore this coursework is worth **36%** of the total marks available for this module.

This coursework contains two parts. In Part A you are to build a library class that implements a binary search tree data structure for storing `string` data for efficient searching. This is an extension of the binary search tree case study presented at the end of Chapter 6 in the Workbook. You should carefully study pages 151-157 of the Workbook for the definition of a binary search tree and necessary background information. In Part B you are to use your binary search tree class to implement an efficient spell checker.

2 Part A: The binary search tree library

You will be provided with three files – a header file and supporting C++ file for the `BinarySearchTree` class, and a test file to test the class. Your task is to complete the implementation of the methods and operator overloads in the `BinarySearchTree` class, as specified in Tables 1 and 2. **You must not modify the header file**. Note that strings in C++ can be compared using equality and inequality operators (e.g. < and >) to obtain their order, e.g. "abc" < "def" returns true.

All of the printing functions should return the words in the tree separated by a space (e.g. "for he she").

You will need to ensure that your compiler is operating to the C++ 2011 standard for the coursework files to compile. On the Microsoft compiler this should be automatic. With other compilers you may need to pass a command line argument. For example, in the GNU C++ compiler you would use `g++ -std=c++11 filename.cpp`.

The `BinarySearchTree` class should be built as a library file using a makefile.

3 Part B: Using your library to implement a spell checker

For this part of the coursework you are supplied with 3 files: `dictionary.txt`, `single_words_test.txt`, and `sentences_test.txt`. Your task is to write a program that inserts all of the words from the dictionary file into your binary search tree from Part A. Your program should then read in the `single_words_test.txt` and `sentences_test.txt` files and check for each word in those files whether or not the word occurs in the dictionary. If it does not then the word is misspelt, in which case your program should output the misspelt word to the console. When processing the `sentences_test.txt` file punctuation should be handled correctly, e.g. "hello." should be treated as "hello". You may, if you wish, ignore capitalisation by converting all words in both the dictionary and the input to lowercase.

4 Restrictions and Resources

Your application must be written entirely in C++ using the C and C++ standard library. Your application must build in a standard compiler such as Microsoft's `cl`, `g++` or `clang`.

5 Submission

Your code must be submitted to the Moodle Assignment Dropbox by 23:55 on Tuesday 17th of April. If your submit late without prior authorisation from your personal tutor your grade will be capped at 40%.

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 during the week beginning 16th April. If the coursework is not demoed to a member of the teaching team this will result in a grade of 0. The demo session is the point where the teaching team will give you verbal feedback on your work. Final grades, and additional feedback, will be returned via Moodle.

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

- The code file(s) required to build your binary search tree library and spell checker.
- The supplied test file.
- A code file to demonstrate your spell checker.
- A `makefile` to allow building of your application. The `makefile` should build the binary search tree as a separate library, and also allow building of the supplied test file and your spell checker against this library. It should contain targets to run the supplied test file, and targets to run your spell checker separately on the `single_words_text.txt` and `sentences_test.txt` files.
- A *read me* file indicating how the `makefile` is used to build your library and spell checker, and the toolchain used to build it (i.e. 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 coursework marks will be divided as follows:

Description	**Marks**
Basic constructors	2
Copy constructor	2
Insert method	2
Remove method	4
Exists method	2
Different printing methods	2
Equality operators	2
Input / output operators	4
Part B: Spelling checked in single words test file using tree	4
Part B: Spelling checked in sentences test file using tree	4
Resources free and cleared up correctly	2
Code quality	2
Makefile works correctly	2
Submission zip folder complete and correctly collated	2
Total	36

BE WARNED! This coursework requires an understanding of the material covered in units 5 to 11 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 the module leader as a matter of urgency. This coursework is designed to be challenging and will require you to spend time developing the solution.

Method	**Description**
`BinarySearchTree()`	Creates an empty binary search tree
`BinarySearchTree(string word)`	Creates a binary search tree with an initial word to store
`BinarySearchTree(const BinarySearchTree &rhs)`	Creates a binary search tree by copying an existing binary search tree. This must be a deep copy, not a reference.
`~BinarySearchTree()`	Destroys the binary search tree, cleaning up used resources.
`void insert(string word)`	Adds a word to the binary search tree. If the word already exists in the tree nothing happens.
`void remove(string word)`	Removes a word from the binary tree if it exists.
`bool exists(string word)`	Returns true if the word is in the tree, false otherwise
`string inorder()`	Generates a string containing the words in the tree in alphabetic order
`string preorder()`	Generates a string containing the words in the tree in *pre-order* fashion
`string postorder()`	Generates a string containing the words in the tree in *post-order* fashion

Table 1: List of Methods for the `BinarySearchTree` Class

Operator	**Example**	**Description**
`==`	`bool res = tree1 == tree2`	Checks if two trees are equal. Two trees are equal when they both contain the exact same words in the same positions in the tree.
`!=`	`bool res = tree1 != tree2`	Checks if two trees are not equal. Two trees are not equal if they do not both contain the exact same words in the same positions in the tree.
`>>`	`cin >> tree`	Reads in words from an input stream into the tree
`<<`	`cout << tree`	Writes the words, *in alphabetical order*, to an output stream

Table 2: List of Operator Overloads for the `BinarySearchTree` Class