Manchester Metropolitan University

Department of Computing and Mathematics

ASSIGNMENT COVER SHEET

Unit title:	6G5Z1102: Algorithms and Data Structures
Assignment set by:	Matteo Cavaliere, David McLean
Assignment ID:	3CWK50
Assignment title:	Text Analysis Tool
Assignment weighting:	50% of the total unit assessment
Type: (Group/Individual)	Individual
Hand-in deadline:	As Indicated on Moodle
Hand-in format and mechanism:	Submission on Moodle
Support:	Formative feedback is available in the lab during the timetabled sessions and by the teaching team (see also contact details on the Moodle page of this unit – *Student Support and Information*).

Learning outcomes being assessed:

LO1: Apply problem decomposition to solve programming problems.
LO3: Implementation and appropriate application of a variety of programming techniques including pointers, recursion and algorithms for a variety of problems.
LO4: Apply software development techniques including implementation of appropriate software components to enable the modelling of novel problems.

Note: it is your responsibility to make sure that your work is complete and available for marking by the deadline. Make sure that you have followed the submission instructions carefully, and your work is submitted in the correct format, using the correct hand-in mechanism (e.g. Moodle upload). If submitting via Moodle, you are advised to check your work after upload, to make sure it has uploaded properly. Do not alter your work after the deadline. You should make at least one full backup copy of your work.

Penalties for late hand-in: see Regulations for Undergraduate Programmes of Study (http://www.mmu.ac.uk/academic/casqe/regulations/assessment.php). The timeliness of submissions is strictly monitored and enforced.

All coursework has a late submission window of **5 working days**, but any work submitted within the late window will be capped at 40%, unless you have an agreed extension. Work submitted after the 5-day window will be capped at zero, unless you have an agreed extension.

Please note that individual tutors are unable to grant extensions to coursework. Extensions can only be granted on the basis of a PLP, or approved Exceptional Factors (see below).

Exceptional Factors affecting your performance: see Regulations for Undergraduate Programmes of Study (https://www.mmu.ac.uk/academic/casqe/regulations/assessment/docs/ug-regs.pdf). For advice relating to exceptional factors, please see the following website: https://www2.mmu.ac.uk/student-case-management/guidance-for-students/exceptional-factors/ or visit a Student Hub for more information.

Plagiarism: Plagiarism is the unacknowledged representation of another person's work, or use of their ideas, as one's own. Manchester Metropolitan University takes care to detect plagiarism, employs plagiarism detection software, and imposes severe penalties, as outlined in the Student Handbook (http://www.mmu.ac.uk/academic/casqe/regulations/docs/policies_regulations.pdf and Regulations for Undergraduate Programmes (http://www.mmu.ac.uk/academic/casqe/regulations/assessment.php). Bad referencing or submitting the wrong assignment may still be treated as plagiarism. If in doubt, seek advice from your tutor.

As part of a plagiarism check, you may be asked to attend a meeting with the unit leader, or another member of the unit delivery team, where you will be asked to explain the submitted code. If you are called to one of these meetings, it is very important that you attend.

Assessment Criteria:	Indicated in the attached assignment specification.
Formative Feedback:	Formative feedback is available in the lab during the timetabled sessions and by the teaching team during office hours (see Moodle page of this unit – Student Support and Information).
Summative Feedback Format:	Written feedback in the form of a commented mark grid, plus a general comment on the whole submission.

Introduction

This assignment is coursework based, and has a single main component, worth 50% of the overall unit mark. The tasks you are required to complete for this assessment are detailed in this coursework specification (see below).

Aim

This assignment encourage you to gain practical experience on the use of advanced programming techniques focused on data structures and algorithms that underpin computer science.
In particular, the assignment will assess your knowledge of the main algorithms and data structures used in computer science and your ability to implement and use them in a software application which is a simplified version of a real world problem.

Coursework Overview

To complete this assignment you will need to create a **C# application** that will be used to analyze a text document providing and searching information on the words contained in the document. The application must use the **appropriate algorithms and data structures** (see the specification below). **Text analysis** (a.k.a. text mining) is a very important area of computer science, with many applications and an active area of research (for instance, in natural language processing). It is not uncommon to be asked to produce applications similar to this in industry, normally as a pilot study. On occasions files may differ slightly from the provided specification making error trapping important, however that isn't the case with this file. **Scalability and efficiency** of the solution (to deal with large text files) is paramount.

Specification: Text Analysis Tool (3CWK50)

You are asked to implement a **C# application** which allows to store, manage, and search in an efficient way the information concerning the words present in a text document (the text document is read-in as a text file).

You are asked to implement a Binary Search Tree or an AVL Tree where each *node* corresponds to a *unique word* presents in the text document along with the *word information*: the number of times that unique word is present in the text (number of occurrences), and the locations in the text document (line number and position) in which the unique word occurs.
Instead of a tree, for a bare pass the built-in Dictionary collection can be used instead.

Your trees must use the recursive techniques covered in the lectures and labs as a starting point.

To store the unique words in the tree (or dictionary) you should use a ***Word class*** and a ***Location class***.

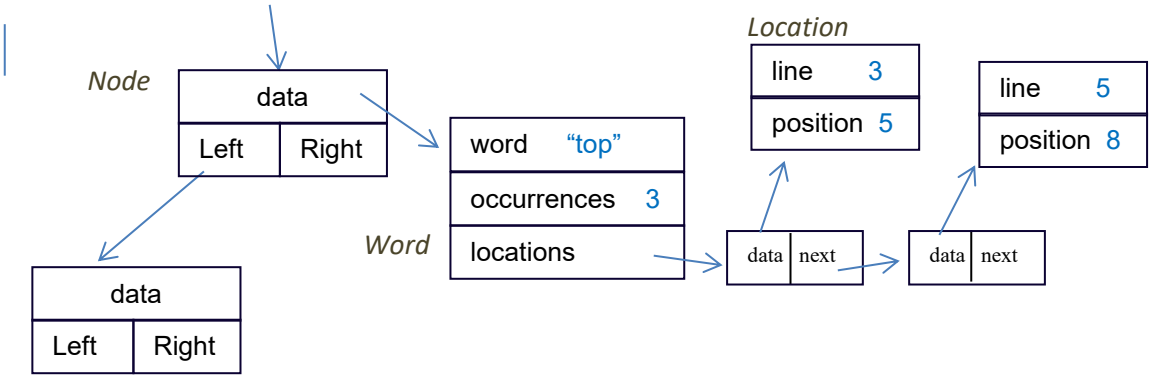
The ***Word class*** should contain the following information:

- word
- number of occurrences of the word in the text document
- list of *locations* where the word is present (to store multiple locations use the built in LinkedList<> generic collection)

The locations of the words should be stored using an appropriate ***Location class*** which should contain the following information:

- line number of the text file where the word occurs
- position within line where the word occurs

Store the objects in the tree (or in the dictionary) using the word (in *lowercase*) as the *key for comparison* (use alphabetical order). If you use a tree, the diagram below shows the organization of the classes.



You must implement a **GUI** (may be one or multiple forms) which allows the user to:

1. **Load** a text file and store the unique words and the corresponding information in the chosen data structure (tree or dictionary)
2. Manually **edit** (and save in the data structure) the information of a unique word (allow the user to modify the number of occurrences and the locations)
3. **Display** the number of unique words present in the text
4. **Remove** a unique word from the data structure
5. Display the **concordance** of the text – i.e. show all the unique words present in the text in alphabetic order and the corresponding number of times in which they occur in the text
6. **Search** for a specific unique word (and display the word information) by either typing the A) full unique word or a B) prefix of the unique word (if a prefix is typed by the user, then all unique words starting with that prefix should be displayed)
7. **Search** for
   - the most common unique word present in the text
   - unique words which occur more than a specified number of times
8. **Display** all the unique words present in the text in decreasing order of occurrence
9. Display **Collocation** – for a specified pair of unique words display the number of times they appear next to each other (for example "pound note") from **searching the implemented data structure**. Assume they must exist on the same line.

Marking Scheme

To pass (40-50) a reasonable attempt at 1,2,3,4 using a built-in Dictionary Collection, rather than a tree.

50-60 as above but using a BSTree, may have some minor bugs, partially working

60-70 as above including 5, 6 but using an AVLTree – may have minor bugs

70-80 All 1-8 attempted – must use an AVL Tree and evidence of bullet points below

80+ As above but all 1-9 attempted and evidence of bullet points below

Credits will be given for:

- Efficiency of the Implemented Algorithms
- OO design - Complete and Working Classes/Methods
- Usability of the GUI
- Tested code – you can include a file of tests for different tree methods (Blackbox testing) or use NUnit
- Validation of the code and of the GUI
- Reusability and maintainability of the code

Feedback will be given in the form of a commented mark scheme (see below).

Hand-in

Submit to Moodle a zipped electronic version of your solution directory including the executable and all solution files needed. Your zip file should use the convention: SurnameForenameStudentIDNumber.zip, e.g. BloggsFred1505678.zip

The zip file must be submitted before this **assignment deadline** (see Moodle).

The submitted C# code must run for the Visual Studio configuration installed in the lab.

Additional Notes and Help

Unique Words

The *unique words* of a text are all the words present in the text minus the repetitions.

For example, the text below (in italic) has 121 words and 88 unique words:

It was one January morning, very early, a pinching, frosty morning. The cove all grey with hoar frost, the ripple lapping softly on the stones, the sun still low and only touching the hilltops and shining far to seaward.
The captain had risen earlier than usual and set out down the beach, his cutlass swinging under the broad skirts of the old blue coat, his brass telescope under his arm, his hat tilted back upon his head. I remember his breath hanging like smoke in his wake as he strode off, and the last sound I heard of him as he turned the big rock was a loud snort of indignation, as though his mind was still running upon Dr Livesey.

These are the unique words of the text with at least 3 occurrences (in parenthesis the number of occurrences):

the (11), his (8) , and (4), was (3), of (3), as (3)

The most common unique word is “the” with 11 occurrences.

The number of times the unique words “as” and “he” appear next to each other is 2.

There are many online tools which allow to visualize the statistics on the words and unique words in a text file[1].

[1] E.g., https://wordcounttools.com/ , https://wordcounter.net/ , https://planetcalc.com/3205/

Getting Started:

The data structures necessary to implement this application have been already addressed in the portfolio exercises.
For this application, you may choose to use either a *Dictionary*, a *BSTree* or an *AVL Tree* (see marking scheme). You can start this project by re-using and adapting (in the appropriate way) the code that you have already developed in the portfolio exercises concerning these data structures. GUIs have been used in Weeks 1 and 7.
Any Class should be defined as *public* to prevent any accessibility compilation warnings,
e.g. *public class Word : IComparable*

To store complex class instances (e.g. Word class) you will need to implement the IComparable interface, for example:

```
public int CompareTo(object other)
{
  Company temp = (Company)other;
  return name.CompareTo(temp.name);
}
```

Remember that a class should be as general as possible in order for future re-use (code reusability is an important issue in software design). You can make more specific class types using inheritance, just as we did with AVLTree inheriting from BSTree which inherits from BinTree.

You can add events to controls in design time by selecting the control and clicking on the lightening strike in the properties box. Then select the event you want to add.
Event documentation can be found at https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/

If you choose to have a GUI with multiple forms, there are a variety of techniques for dealing with multiple forms and transferring information between them (see the materials of the first teaching week "handling multiple forms in C#" where we have seen Windows Forms Applications).

Reading File

There are many different ways to read a text file in C#. The code below shows a possible way to read a text file and display each word on the console

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.IO;

namespace FileRead
{
    class Program
    {
        static void Main(string[] args)
        {

            const int MAX_FILE_LINES = 50000;
            string[] AllLines = new string[MAX_FILE_LINES];

            // set the path where the file to be read is saved
            string path = @"H:\myText.txt";

            AllLines = File.ReadAllLines(path);

            foreach (string line in AllLines)
            {

                //split words using space , . ? and ;
                string[] words = line.Split(' ', ',', '.', '?', ';');

                foreach (string word in words)
                    if (word != "")
                    { Console.WriteLine(word);}
            }
            Console.ReadKey();

        }

      }

   }
```

Example of Marking Grid

		Criteria – run solution	Dictionary 40-50	BSTree 50-60	AVLTree 60+
1		Load file into Data Structure			
2		Edit unique word information			
3		Display number of unique words			
4		Remove a unique word			
5		Display text concordance			
6		Search & Display the Info of a unique word (A: using full word, B: using prefix of a word)			
7		Search most common unique word and all the unique words that occur more than a specified number of times			
8		Display the unique words in decreasing number of occurrences			
9		Display Collocation			

Credit for

Classes	Complete	WordTree – all reusable
Working		
GUI Usability – intuitive?	Menu	Validation
Search – Algorithm Efficiency		
Testing	BlackBox / Strategy	NUnit
Refactored		

Classes Word / Location	Attrib, Types (private/public)	Locations – list / string
	Properties	Iterator
IComparable	Substring matching	
Tree – Generic	Public/private	Remove balance
All Recursive		
FindWord		
FindMostCommonWord	Substring matching	
Maintainable code	Presentation (Indentation ect)	Var Names (Meaningful)
Extras (Beyond spec – in scope)	Addition of an Help Function	