MANCHESTER METROPOLITAN UNIVERSITY

School of Computing, Mathematics & Digital Technology

ASSIGNMENT COVER SHEET

Unit:	6G5Z1102: Algorithms and Data Structures
Assignment set by:	Matteo Cavaliere, David McLean
Verified by:	CDTN Assessment Verification Panel
Moderated by:	
Assignment number:	3CWK50
Assignment title:	Companies Trading Data
Type: (GROUP/INDIVIDUAL)	Individual – see plagiarism statement
Hand-in format and mechanism:	Submission is online, via unit area in Moodle
Deadline:	As Indicated on Moodle

Learning Outcomes 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.

It is your responsibility to ensure that your work is complete and available for assessment by the date given on Moodle. If submitting via Moodle, you are advised to check your work after upload; and that all content is accessible. Do not alter 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.

Exceptional Factors affecting your performance: see Regulations for Undergraduate Programmes of Study: http://www.mmu.ac.uk/academic/casqe/regulations/assessment/docs/ug-regs.pdf

Plagiarism: Plagiarism is the unacknowledged representation of another person's work, or use of their ideas, as one's own. MMU 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.

Assessment Criteria:	Indicated in the assignment specification
Formative Feedback:	Formative feedback is available in the lab during the timetabled sessions.
Summative Feedback format:	Written feedback in the form of a commented mark grid, plus a general comment on the whole submission.
Weighting:	50% of the total unit assessment

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 store and manage, using appropriate algorithms and data structures, a flat data file of different companies (see the specification below). 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 a much larger dataset of companies) is paramount.

Specification: Companies Trading Data (3CWK50)

A large media and software company has acquired a flat data file of economic data for different large companies.

You have been asked to create a **C# application** that will store and structure this data to allow **fast and efficient** searching (and other functionalities) via the company name. You are asked to implement a Binary Search or AVL tree, where each node corresponds to a specific company and its key economic data. The application must be able to read this data from a .csv file document.

An example of *data file* (with fictional but reasonable data) is provided on Moodle (*companies.csv*).

Each line in the data file corresponds to a unique company. For a bare pass the built-in Dictionary collection can be used instead (though obviously much of the search functionality will be sub-optimal). Your trees MUST use the recursive techniques covered in the lectures and labs as a starting point.

The first line of data in the data file consists of the following headings:

Company Name, Net income, Operating Income, Total Assets, Number of Employees, Buyers

The other lines of the data file consist of the data which falls under each heading. For example

AMX, 3033, 4106, 131330, 570000, [MST;WLM]

We assume the company name is a string and that net income, operating Income and total assets are expressed in million $.

For example, the line above denotes a company that is called AMX, with a net income = 3033 million$, an operating income = 4106 million$, total assets of 131330 million$, and 570000 employees.

Note that the last field (buyers) is a list of company names separated by ';' and enclosed in '[ ]'. This field contains an *arbitrary* (varying) number of companies which are the *buyers* of the company. In the above example the *buyers* of the company AMX are the companies called MST and WLM. Notice that this information also says that AMX is a company that *sells* to MST and WLM.

You must implement **an application with a GUI** (may be multiple forms) which allows the user to:

1. Load a text data file (.csv) and store the company information within the tree

2. Manually edit the company information

3. Display the number of companies within the tree and the depth of the tree

4. Remove a company

5. Display all companies names in alphabetic order.

6.

a) Search for a company (and display all company information)

b) *Partial* keyword search by company name: as letters are entered *any* company starting with those letters is shown and can be selected from.

7. Search for and display the *trading partners* of a company. The trading partners of a company are *all* companies who trade (*either* as buyer or seller) with that company.

8. Display the company which has the *biggest potential for trade*. A company's potential for trade can be found by summing the *net income* attribute's for all its trading partners.

If you decide to use a grid-view control on your GUI ensure that you are still using the data structures for storage and efficient retrieval of the company data.

Marking Scheme

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

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

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

70-80 all 1-7 attempted – must use an AVL Tree. Evidence of points below

80+ As above but all 1-8 attempted and evidence of bullet points below

Credit will be given for:

- OO design - Complete and Working Classes
- Working Implementation
- Usability of the GUI
- Efficiency of your search algorithms – use AVLTree / BSTree ideas properly
- Tested code – you can include a file of tests for different tree methods (blackbox testing)
- Validation
- Refactored 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. 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, under Assessments, right bar on Moodle).

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

Plagiarism

It is NOT OK to copy whole or parts of online code examples and pass them off as your own. Similarly it is NOT OK to borrow code in whole or part from your peers, both of you would be guilty of plagiarism. We will use an AUTOMATED PLAGIARISM CHECKER to compare submissions against other students work and online examples.

Additional Notes and Help

Getting Started: Notice that 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 may 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.

Any Class stored in a tree or collection should be defined as *public* to prevent any accessibility compilation warnings, e.g. *public class Company : IComparable*

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

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

As a company has a varying number of buyers you should use a *LinkedList* of type *string* to store these. You may use the built-in *LinkedList<string>*.

There are a variety of techniques for dealing with *multiple forms* and transferring information between them (for an example, see Moodle - Handling Multiple forms, that you can found in the materials of the first teaching week).

In Visual Studio, 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. *Events* documentation can be found at MSDN (via a web browser); also see the materials on Moodle for the first teaching week.

Remember that a class should be *as general as possible* in order for future re-use. You can make more specific class types via *inheritance*, just as we did with AVLTree inheriting from BSTree which inherits from BinTree.

Reading Files The code below shows a (possible) technique to read a .csv data file and display each string on the console. This includes breaking up the list of buyers.

```
// This code shows a way to read a .csv data file and display each string on the console.
// This includes breaking up the list of buyers.
using System;
using System.IO;
namespace fileReadExample
{
    class Program
    {
        static string[] headers = new string[6]; //column headers

        static void Main(string[] args)
        {
            const int MAX_LINES_FILE = 50000;
            string[] AllLines = new string[MAX_LINES_FILE];

            //set a string variable with the location of the data file. Use ReadAllLines
            to read the data
            string path = @"H:\companies.csv";

            AllLines = File.ReadAllLines(path);
            foreach (string line in AllLines)
            {
                if (line.StartsWith("Company")) //found first line - headers
                {
                    headers = line.Split(',');
                }
                else
                {
                    //split data using commas
                    string[] columns = line.Split(',');

                    Console.Write(columns[0] + ","); //first string in line;
                    Console.Write(columns[1] + ","); //second string in line;
                     //add similar instructions to read the other strings

                    // read the full set of buyers
                    string[] buyers = columns[5].Split(';', '[', ']');
                    foreach (string partner in buyers)
                    {
                        if (partner != "")
                        {
                            Console.Write(":" + partner);
                        }
                    }
                }
                Console.ReadKey();
                Console.WriteLine("");
            }
        }
    }
}
```

Example of Marking Grid

		Criteria – run solution	Dictionary 40-50	BSTree 50-60	AVLTree 60+
1		Load file into DataStruct			
2		Edit a company			
3		Display company count and height (tree)			
4		Remove a company			
5		Display All (alphabetic)			
6		Search & Display (A: whole name, B: Partial name – multi companies)			
7		Search&Display ALL companies trading with X			
8		Search & Display biggest trade potential (must involve a search)			

Credit for

Classes	Complete	CompanyTree – all reusable
Working		
GUI Usability – intuitive?	Menu	Validation
Search – efficiency (tree based)		
Testing	BlackBox/Strategy	NUnit
Refactored		
Company / Buyers	Attrib, Types (private)	Buyers - string
	Properties	Iterator (Buyers)
IComparable	Substring matching	
Tree – Generic	Public/private	Remove balance
All Recursive		
FindCompanyByName		
FindCompanyTradingPartners	Substring matching	
Maintainable code	Presentation (Indentation ect)	Var Names (Meaningful)
Extras (Beyond spec – in scope)	Addition of an Help Function	

Support for the Assessment

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).