MANCHESTER METROPOLITAN UNIVERSITY

School of Computing, Mathematics & Digital Technology

ASSIGNMENT COVER SHEET

Unit:	6G5Z1102 Algorithms & Data Structures
Assignment set by:	Dr David McLean
Verified by:	Dr Fawaz Ghali
Moderated by:	Dr Fawaz Ghali
Assignment number:	1CWK50
Assignment title:	International Trading Data
Type: INDIVIDUAL	INDIVIDUAL – see plagiarism statement
Hand-in format and mechanism:	via Unit area on Moodle
Deadline:	04/05/2018 (see moodle)

Learning Outcomes Assessed:

1. Apply problem decomposition to solve programming problems.
3. Implementation and appropriate application of a variety of programming techniques including pointers recursion and algorithms for a variety of problems. *(pointers is covered in 2CWK50)*
4. apply software development techniques including implementation of appropriate software components to enable the modelling of novel problems.

Assessment Criteria:	Indicated in the attached assignment specification.
Formative Feedback:	Checkpoint: formative submission week 18
Summative Feedback format:	Example marking grid provided in attachment
Weighting:	This Assignment is weighted at 50% of the total unit assessment.

Learning Outcomes Assessed:

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.

Employability statement:

This assignment is based on a simplified version of a real world problem. 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. Bare in mind that scalability of the solution (to deal with a much larger dataset) is paramount.

International Trading Data

A large multinational trading company has acquired a flat data file of economic data for different countries. You have been asked to create a **C#** application that will store and structure this data to allow **fast and efficient searching** (and other functionality) via the **country name**. You are to implement a Binary Search or AVL tree, where each node corresponds to a specific country and its economic data. The application must be able to read this data from csv file documents (an example is provided on Moodle) where each line in the file corresponds to a unique country. 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 the file consists of the following headings:

```
Country,GDP growth,Inflation,Trade Balance,HDI Ranking,Main Trade Partners
```

Other lines consist of the data which falls under each heading, for example:

```
USA,1.8,2,-3.1,4,[Canada;UK;Brazil]
```

Note that the last field (Trade Partners) is a list of country names separated by ';' and enclosed in '[ ]'. This field will contain an arbitrary (varying) number of countries.

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

1. Load a text file and store country information within the tree
2. Manually edit country information

3. Display the number of unique countries within the tree and the depth of the tree
4. Remove a country
5. Display all country names in alphabetic order.
6. a) Search for a country (display all country information)

   b) Partial keyword search by country name, as letters are entered any countries starting with those letters are shown and can be selected from.

7. Search for and display ALL countries who trade with a particular country
8. Display the country which has the biggest potential for trade. A country's potential for trade can be found by summing the GDP growth 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 trees for storage and efficient retrieval of the country 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 **Country** 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 be AVL Tree. Evidence of bullet 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

Hand-in :

For both **summative** (final hand-in) and **formative** 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

You should hand-in your work to date on the **formative submission date**. Your work will be checked at this point and you will be provided with **written feedback** to enable you to improve your submission. You can then resubmit your improved submission on the final hand-in date, but your work must be accompanied with a record of the changes you have made in light of the feedback you were given. This record should consist of a written description and highlighted sections within your code (e.g. by comments).

Notes and help

Any Classes that are stored in a tree or collection should be defined as public to prevent any accessibility compilation warnings, e.g. public class Country : IComparable

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

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

As a country has a varying number of trading partners we should use a **LinkedList** of type string to store these. You can use the built-in `LinkedList<string>`.

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 MSDN (via a web browser).

There are a variety of techniques for dealing with multiple forms and transferring information between them (see Moodle - unit resources- Handling Multiple forms… for one example of this).

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.

Files

The code below shows a simple technique to read a csv file and display each string on the console. This includes breaking up the list of trading partners.

```
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];

            //reads from bin/DEBUG subdirectory of project directory
            AllLines = File.ReadAllLines("countries.csv");

            foreach (string line in AllLines)
            {

                if (line.StartsWith("Country")) //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] + ”,”);  //2nd string in line;

                    string[] partners = columns[5].Split(';', '[', ']');
                    foreach (string tradePartner in partners)
                    {
                        if (tradePartner != "")
                        {
                            Console.Write(":" + tradePartner);

                        }

                    }
            }
            Console.ReadKey();

       }
}
```

Example marking grid

	Criteria – run solution	Dictionary 40-50	BSTree 50-60	AVLTree 60+
1	Load file into DataStruct			
2	Edit a country			
3	Display Country Count and Height			
4	Remove a Country			
5	Display All (alphabetic)			
6	Search & Display (whole name, Partial name – multi countries)			
7	Search&Display ALL countries trade with X			
8	Search & Display big trade potential (must involve a search)			

Credit for

Classes	Complete	CountryTree – all reusable
Working		
GUI Usability – intuitive?	Validation	
Search – efficiency (tree based)		
Testing	BlackBox/Strategy	NUnit
Refactored		

Class files		
Country / Partners		Partner - string
Attrib, Types (private)		
Properties	Iterator (Partners)	
IComparable (Name/Album/string)	Substring matching	
BSTree (only)		**AVL**
Generic		Remove balance
All Recursive	Public/private	
Insert		
Remove		
FindCountryName		
FindCountryByPartner	Substring matching	
Maintainable code	Refactored - concise	Var Names (Meaningful)
presentation		
Extras (Beyond spec – in scope)		