**Faculty of Engineering, Environment and Computing**

# 210CT Programming, Algorithms and Data Structures

## Assignment Brief

| | | | |
|---|---|---|---|
| Module Title<br>**Programming, Algorithms and Data Structures** | **Individual** | **September** | Module Code<br>**210CT** |
| Coursework Title<br>**Individual Programming Assessment** | | | Hand out date:<br>**14th October 2020** |
| Lecturer<br>**Dr Ian Cornelius** | | | Due date and time:<br>Date: **11th November 2020**<br>Online: **18:00:00** |
| Estimated Time (hrs):<br>**60**<br>Word Limit*: **Not Applicable** | Coursework type:<br>**Individual Programming Assessment** | | **30%** of Module Mark |

Submission arrangement online via Aula:
**Submit the Assignment to Aula**
File types and method of recording:
**GitHub Repository URL**
Mark and Feedback date (DD/MM/YY):
**Two weeks after VIVA**
Mark and Feedback method (e.g. in lecture, electronic via Aula):
**Electronic VIVA marking scheme and feedback sheet via AULA**

Module Learning Outcomes Assessed:

1. Understand and select appropriate algorithms for solving a range of problems.
2. Design and implement algorithms and data structures for novel problems.
3. Reason about the complexity and efficiency of algorithms.
4. Demonstrate the use of object-oriented programming features and advanced language features such as events, unit testing and GUIs.

Task and Mark distribution:

**Note that Games Technology students must undertake this piece of coursework using the C++ language, as per the expectation of your BSc. programme specification.**

**Tasks are to be undertaken in either the C++ or Python programming language. You are not permitted to make use of any algorithm libraries or middleware in the coding of your work. You will include comments in your code which explain the code's behaviour, and which justify your algorithm selection. You are strongly advised to test your code for compilation on a system other than your own prior to submission, as non-compiling code will not pass (see Marking Rubric).**

1. The file Shakespeare.txt is present in your Moodle page. Write a piece of software which reads in the text file and parses from it the entirety of Shakespeare's Sonnets (beginning on line 253,

ending on line 2867). Parse only the words of the Sonnets, not numbers. Build a Binary Search Tree from the parsed words, allowing duplicate nodes where a word appears more than once.

2. Having constructed your Binary Search Tree, you shall write two functions:
    - A function which will search the Tree for a user-specified word and returns the number of times the word appears in the Tree. Efficiency of this search will affect mark awarded.
    - A function which searches the Tree to determine the 66 most commonly used words in Shakespeare's Sonnets, excluding 'the' and 'a'. It will proceed to print these 66 words to screen in alphabetical order

3. You will implement a function which permits the deletion of a word (and all instances of the word) from the Binary Search Tree. Rebalancing of the Tree should be handled by the function.

4. You will implement the structure for a weighted, undirected graph, where the vertices represent the words computed in Task 2. The vertex data structure shall include an array of the numbers of the Sonnets in which the word appears. An edge will exist between any words which appear in the same Sonnet, and the weighting of an edge between those words will be inversely proportional to the number of Sonnets in which both words appear. For example, if 'his' and 'heart' appear at any point in the same Sonnet, the nodes representing them will have a connecting edge; if they only appear together in Sonnets 4 and 18, the edge shall have a weighting of '1/2' or '0.5'.

5. Implement a function which determines whether or not the graph is strongly connected. It shall return a Boolean value of 'true' or 'false'.

6. You shall implement Dijkstra's algorithm which shall accept two words – a start vertex and a goal vertex. The search shall traverse your graph to compute the shortest route between the two words, and output the route in terms of the words traversed. In addition, it shall generate an integer score for each Sonnet in which the traversed words feature, incrementing the score by each appearance, and output the number of the highest scoring Sonnet.

**IMPORTANT: You will create a Coventry University GitHub Repo to store your source code and manage version control of your work. Evidence of version control must include regular pushes to the Repo over the period between the Handout Date and the Due Date. Your eventual submission to Moodle will be a link to that Repo which must include all your source code. You will ensure that you add ab6459@coventry.ac.uk, csx239@coventry.ac.uk, ac0745@coventry.ac.uk as Collaborators to enable marking**

Notes:

1. You are expected to use the Coventry University APA or CU Harvard style for referencing. For support and advice on this students can contact Centre for Academic Writing (CAW).
2. Please notify your registry course support team and module leader for disability support.
3. Any student requiring an extension or deferral should follow the university process as outlined here.
4. The University cannot take responsibility for any coursework lost or corrupted on disks, laptops or personal computer. Students should therefore regularly back-up any work and are advised to save it on the University system.
5. If there are technical or performance issues that prevent students submitting coursework through the online coursework submission system on the day of a coursework deadline, an appropriate extension to the coursework submission deadline will be agreed. This extension will normally be 24 hours or the next working day if the deadline falls on a Friday or over the weekend period. This will be communicated via your Module Leader.
6. You are encouraged to check the originality of your work by using the draft Turnitin links on Aula.
7. Collusion between students (where sections of your work are similar to the work submitted by other students in this or previous module cohorts) is taken extremely seriously and will be reported to the academic conduct panel. This applies to both courseworks and exam answers.
8. A marked difference between your writing style, knowledge and skill level demonstrated in class discussion, any test conditions and that demonstrated in a coursework assignment may result in you having to undertake a Viva Voce in order to prove the coursework assignment is entirely your own work.
9. If you make use of the services of a proof reader in your work you must keep your original version and make it available as a demonstration of your written efforts.
10. You must not submit work for assessment that you have already submitted (partially or in full), either for your current course or for another qualification of this university, with the exception of resits, where for the coursework, you maybe asked to rework and improve a previous attempt. This requirement will be specifically detailed in your assignment brief or specific course or module information. Where earlier work by you is citable, i.e. it has already been published/submitted, you must reference it clearly. Identical pieces of work submitted concurrently may also be considered to be self-plagiarism.

**Mark allocation guidelines to students (to be edited by staff per assessment)**

| 0-39 | 40-49 | 50-59 | 60-69 | 70+ | 80+ |
|---|---|---|---|---|---|
| Work mainly incomplete | Most elements completed; | Most elements are strong, | Strengths in all elements | Most work exceeds the | All work substantially |

| | | | | | |
|---|---|---|---|---|---|
| and /or weaknesses in most areas | weaknesses outweigh strengths | minor weaknesses | | standard expected | exceeds the standard expected |

**Marking Rubric**

| GRADE | Tasks 1 to 4 (15%) | | Tasks 5 and 6 (15%) | |
|---|---|---|---|---|
| **First**<br><br>**≥70** | • Code uploaded to GitHub repository, compiles and executes.<br>• All **four** tasks have been implemented successfully.<br>• Code is appropriately commented and structured using object orientation.<br>• For the upper end of this bracket, intelligent decisions have been undertaken in the structuring of the graph to better suit the needs of **Tasks 5 and 6**.<br>• Student has justified their implementation/algorithm selection decisions in the code comments.<br>• Version control in evidence. | | • Code uploaded to GitHub repository, compiles and executes.<br>• Complete implementation for both tasks.<br>• Code is appropriate commented and structured, using object orientation<br>• Student is able to explain their code and justify their implementation/algorithm selection decisions within exhaustive comments.<br>• Version control in evidence. | |
| **Upper Second**<br><br>**60-69** | • Code uploaded to GitHub repository, compiles and executes.<br>• **Three** tasks have been implemented successfully.<br>• Code is appropriately commented and structured using object-orientation.<br>• The student has justified their implementation/algorithm selection decisions within the comments.<br>• Version control in evidence. | | • Code uploaded to GitHub repository, compiles and executes.<br>• Complete implementation of **Task 5** and partial functional implementation of **Task 6** (e.g. search implemented, but not Sonnet scoring).<br>• The student has explained their code and justified their implementation decisions in the comments, object-orientation used.<br>• Version control in evidence. | |
| **Lower Second**<br><br>**50-59** | • Code uploaded to GitHub repository, compiles and executes.<br>• **Two** tasks have been implemented successfully.<br>• Code is appropriately structured using object-orientation.<br>• The student has explained their code and justified their implementation/algorithm decisions in the comments. | | • Code uploaded to GitHub repository, compiles and executes.<br>• Complete implementation of **Task 5** with a meaningful attempt for **Task 6.**<br>• Student has explained their code and justified their design decisions in the comments.<br>• Object-orientation in evidence. | |
| **Third**<br><br>**40-49** | • Code uploaded to GitHub repository, compiles and executes.<br>• **Task 1** has been implemented successfully (e.g. parsing of the text file works, generating the required tree)<br>• Student has commented their code. | | • Code uploaded to GitHub repository, compiles and executes.<br>• Functional implementation of **Task 5** with no attempt made at **Task 6.**<br>• Student has explained their code and justified design decisions with their comments.<br>• Object-orientation in evidence. | |
| **Fail**<br><br>**<40** | • Code uploaded to a GitHub repository but does not compile or execute.<br>• None of the relevant tasks have been completed. | | • Code uploaded to a GitHub repository but does not compile or execute.<br>• None of the relevant tasks have been completed. | |
| **Late submission** | 0 | | 0 | |