CM1601 Programming Fundamentals Assignment Specification (2021)	
Module Leader	Rajitha Jayasinghe
Unit (Group/Individual)	30
Weighing	20%
Qualifying Mark	40% (both parts)
Learning Outcomes Covered in this Assignment:	LO2. Design, code, compile, test and execute fundamental programming concepts using a high-level programming language. LO3. Construct robust, maintainable programs that use object-oriented analysis and design principles
Handed Out Date	20th Feb 2021
Due Date	1st April 2021
Expected Deliverable	Source code and report
Method of Submission	Online
Method of Feedback and Due Date	10th April 2021
BCS Criteria Met by this Assignment	

Assessment Regulations

Refer to the “How you are assessed section” in the Student Handbook for undergraduate students for a clarification of how you are assessed, penalties and late submissions, what constitutes plagiarism etc.

Penalty for Late Submission

Coursework received late without valid reason shall not be accepted and shall receive no grade, but shall count as one of the assessment opportunities prescribed in paragraph 9 of **RGU Academic Regulation A4 section 4.3.**

It is recognized that on occasion, illness, personal crisis or other valid circumstances can mean that you fail to submit and/or attend an assessment on time. In such cases you must inform the School of any extenuating circumstances through a ***Coursework Extension Form*** or a ***Deferral Request Form***, with valid evidence for non-submission of an assessment up to a maximum of five working days after the assessment submission date. This information will be reported to the relevant Assessment Board that will decide whether a student should be allowed to reattempt without penalty (a deferral). For more detailed information regarding University Assessment Regulations and accessing forms, please refer to the following website: www.rgu.ac.uk/academicregulations www.rgu.ac.uk/academicregulations

Grading

Marks will be awarded for the coursework based on the provided Grading Grid. These marks will be mapped onto a grade scale from A-F as determined by the individual module coordinator

Note:

Subject to approval by external examiner

Coursework Specification

Objective

You are instructed to create a command-line program for the Tower of Hanoi game in Python.

Tower of Hanoi

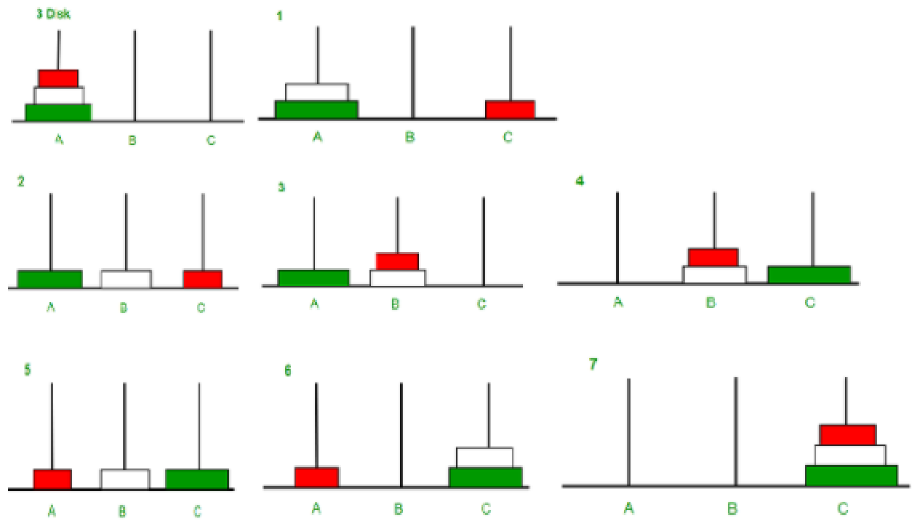
Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. Three rods are named as source, auxiliary and the destination respectively. Rod which bears disks initially is called the source and the objective of the puzzle is to move the entire stack to the destination rod, obeying the following simple rules

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

The rod in between source and the destination is called the auxiliary.

Example

If n=3, then the minimum number of attempts to move all 3 disks to the destination is 7. In the example source, auxiliary an destinations are marks as A,B and C respectively.

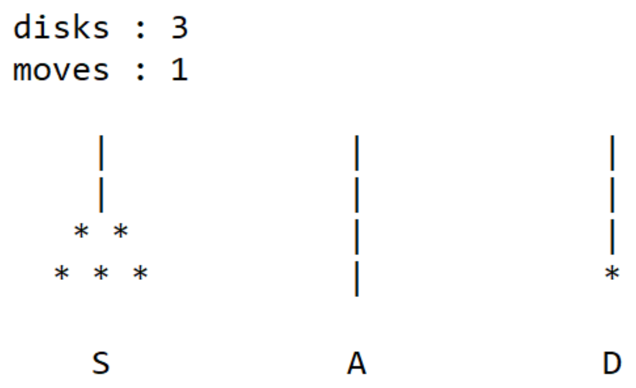


Play

Following features are developed and the player has the freedom to use.

1. First, the game needs to be introduced (mainly the rules). The player can start playing the game by entering the difficulty level.

   - Expert : the player needs to be able to move all disks to the destination rod within minimum number of moves
   - Intermediate : the player can do the above movement within minimum number of movements + 3
   - Novice : the player has the freedom to move disks without any conditions (no limits for the moves)
2. After the difficulty level selection, the game starts. Each time the player will be given a chance to enter the disk he/she wants to move along with the rod.
3. Each time the player execute a movement, output needs to be displayed. This needs to be continued until the player quit or the number of moves for the difficulty level exceeds. One sample output was given as follows



```
disks : 3
moves : 1

    |            |            |
    |            |            |
   * *           |            |
  * * *          |            *

    S            A            D
```

4. At a given time stamp, the player should have the ability to go to the previous step or fast forward given number of steps. There after output needs to be displayed and the player will actively participate thereafter.
5. If the player is using the “expert” difficulty level, then for each move, the game needs to give a feedback which states the relevant move is right or wrong.
6. Game ends, if the player is able to complete the task before run out of moves. If the player uses “novice” difficulty level, disregard number of moves and the game ends when he/she completes the task. Of course any given the player and exit the game too.

Tactics

If you want to move all the disks to the destination rod in minimum number of moves, first you need to understand the pattern and then you need to follow the pattern. Otherwise, you will have to move the disks without a certain goal and end up with unplanned number of moves.

Tasks and important hints

1. All the steps mentioned above needs to be fully implemented.
   a. You can have your own way to display each step. But it should not be just strings and number and it is mandatory you display it visually.
   b. After the program receives initial input such as difficulty and number of disks, the program needs to calculate, minimum number of moves, all the best moves which causes minimum number of moves and store it appropriately. This will be important when you fast forward steps and compare the player’s move. However, you can have your own implementation for implementing the fast forward feature and the comparison. You need to justify the implementation and explain it in your report.

c. You need to write the logic for moving the disks, following the rules. Do not use recursions when you implement the solution.
d. Further, you needs to record user's moves and that will be helpful when you want to go back to the previous step.

2. Draw flowcharts for the following phase
   a. Algorithm for moving disks from source to destination when the number of disks are both odd and even.
3. Write a test plan to test the rules that you have written (go to the section Tower of Hanoi). Each test case should have a name, inputs, expected output and the actual output. You do not need to add the status of the test case as it is not required to implement and execute.

Marking scheme

Criteria	**Marks**
Flow charts • Core tower of Hanoi algorithm (If it covers all the rules, address both odd and even number of disks)	5
Implementation • Introduction and initialization • Implementation of Tower of Hanoi algorithm • Display steps • Fast forward + previous step feature • Feedback in expert level	 5 5 5 5 5
Validations • Following rules and error messages when necessary	5
Additional improvement • Improvements **only** after the completion of mandatory requirements	2.5
Test plan (follow the requested format)	2.5
Viva	10