Name:______________________________ **Class:___________**

JURONG JUNIOR COLLEGE

JC2 Preliminary Examination 2018

COMPUTING Higher 2

9597 / 1

23 August 2018

3 hours 15 minutes

Additional materials:	EVIDENCE-DOC WEBLOG.txt COUNTRIES.txt QUICKSORT.txt CUSTOMERDATA.txt

READ THESE INSTRUCTIONS FIRST

Type in the EVIDENCE-DOC document the following:

- Candidate details
- Programming language used

Answer **all** the questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

All tasks and required evidence are numbered.

Copy and paste required evidence into the EVIDENCE-DOC document.

At the end of the examination, fasten all your work securely together, with the cover page in front.

- You should have printed a hardcopy of your EVIDENCE-DOC.
- You should have saved the final softcopy of EVIDENCE-DOC in your computer desktop.
- Do not change the original file name.

The number of marks is given in brackets [ ] at the end of each question or part question.

This document consists of **12** printed pages.

1.

A web log, `WEBLOG.txt`, keeps track of the date and time the server is being accessed, as well as the client that accessed it. The client is identified by either the host name or internet address. The format of `WEBLOG.txt` is as follows:

```
<host name>|<DD/MMM/YYYY:HH:MM:SS>
```

or

```
<internet address>|<DD/MMM/YYYY:HH:MM:SS>
```

Below is a sample of `WEBLOG.txt`:

```
199.72.81.55|01/Jul/2018:08:00:01
jurongjc.moe.edu.sg|01/Jul/2018:09:08:06
199.120.110.21|01/Jul/2018:11:30:09
trioon.com|01/Jul/2018:11:45:58
```

A client address (host name or internet address) can appear in `WEBLOG.txt` multiple times. Similarly, the date it accessed the server can be duplicated because it can access the server more than once in a day.

A summary report, `SUMMARYREPORT.txt`, is to be generated to list the unique clients (hosts name/internet address) and their corresponding dates.

Below is a sample of `SUMMARYREPORT.txt`:

```
199.72.81.55          01/Jul/2018,03/Jul/2018,04/Jul/2018
jurongjc.moe.edu.sg   01/Jul/2018,02/Jul/2018,03/Jul/2018
trioon.com            01/Jul/2018,05/Jul/2018
```

Task 1.1

Write a procedure `readLog()` to read `WEBLOG.txt`. Use suitable data structure(s), and prepare the log information to create `SUMMARYREPORT.txt`.

Evidence 1: Your `readLog()` procedure code. **[6]**

Task 1.2

Write a procedure `processLog()` to generate `SUMMARYREPORT.txt`.

Evidence 2: Your `processLog()` procedure code. **[4]**

Evidence 3: Screenshot of `SUMMARYREPORT.txt` after running the program. **[1]**

Task 1.3

Add code to your Task 1.2 program to display the highest number of days the server was accessed by any client, and the corresponding client(s). Below is a sample of screen output:

```
>>> Highest frequency (days): 3

>>> Accessed by:

>>> 199.72.81.55

>>> jurongjc.moe.edu.sg
```

Evidence 4: Your program code for **Task 1.3**. **[4]**

Evidence 5: Screenshot of the program output. **[1]**

2.

The data file `COUNTRIES.txt` contains a list of country names.

Task 2.1

Write program code to store country names in `COUNTRIES.txt` into an array, sort them in **descending** order using **insertion sort** and output the sorted sequence.

Evidence 6: Program code for **Task 2.1** **[7]**

Evidence 7: Screenshot of the program output. **[1]**

Task 2.2

Write program code to store country names in `COUNTRIES.txt` into an array, sort them in **ascending** order using **quicksort** and output the sorted sequence. You may use the pseudocode in `QUICKSORT.txt`. Do note the pseudocode is incomplete. You will have to fill in the missing codes `A, B, C, D, E, F, G, H`.

Evidence 8: Program code for **Task 2.2** **[8]**

Task 2.3

Write additional code to count and display the total number of comparisons made in completing the quicksort process.

Evidence 9: Program code for **Task 2.3** to highlight the additional code by using **bold** and ***italics***. **[3]**

Evidence 10: Screenshot of running **Task 2.3**. **[1]**

3.

The class **Node** has the following properties and methods:

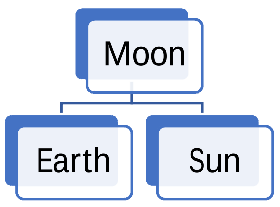
Class: `Node`	
Identifier	**Description**
`data : String`	Data stored at the node.
`left : INTEGER`	Points to the left child of the node. Implement 0 as NULL value.
`right : INTEGER`	Points to the right child of the node. Implement 0 as NULL value.
Methods	
`Constructor()`	Initialises an instance of `Node`.
`set_data(data : String)`	Modifier method for `data`.
`get_data() : String`	Accessor method for `data`.
`set_left(index : INTEGER)`	Modifier method for `left`.
`get_left() : INTEGER`	Accessor method for `left`.
`set_right(index : INTEGER)`	Modifier method for `right`.
`get_right() : INTEGER`	Accessor method for `right`.

Task 3.1

Write program code for the class `Node`.

Evidence 11: Program code for **Task 3.1**. **[4]**

The class **`BinarySearchTree`** has the following properties and methods:

Class: `BinarySearchTree`	
Identifier	**Description**
`root : INTEGER`	Points to the root of binary search tree. Implement 0 as NULL value.
`nodes : ARRAY[31] OF Node`	The array index starts at **1** and the dataset (i.e.: binary search tree) has a maximum of 31 Node objects.
`nextFree : INTEGER`	Index for the next unused node.
Methods	
`Constructor()`	Initialises the `root`, `nextFree` and `nodes` of `BinarySearchTree`.
`addNode(data : String)`	Inserts new node into binary search tree. The `data` of each parent node is larger than the `data` of its left child.
`set_root(index : INTEGER)`	Modifier method for `root`.
`get_root() : INTEGER`	Accessor method for `root`.
`searchNode(item : String) : BOOLEAN`	Searches if `item` is stored in one of the nodes of the binary search tree. Returns TRUE if found and FALSE otherwise.
`inOrderTraversal(root : INTEGER)`	Displays the `data` of each node stored in the binary search tree when traversed using an in-order algorithm. For example: **Binary Search Tree**:  **Display Output**: **`Earth`** **`Moon`** **`Sun`**
`balance() : BinarySearchTree`	Traverses the binary search tree to return a new balanced binary search tree. A **recursive** algorithm is required.

<u>Task 3.2</u>

Write program code for the class `BinarySearchTree`. You may assume space will not be reused should any node be deleted and the binary search tree will not be full.

Evidence 12: Program code for **Task 3.2**. **[18]**

The class **HashTable** has the following properties and methods:

Class: HashTable	
Identifier	**Description**
`size : INTEGER`	Number of items in the hash table array.
`hashTableArray : ARRAY[size] OF BinarySearchTree`	An array storing `BinarySearchTree` objects from indices **0** to **size -1**.
Methods	
`Constructor(size : INTEGER)`	Initialises the `size` and `hashTableArray` of `HashTable`.

`hash(key : String) : INTEGER`	A hashing function that calculates the address of the hash table.
	
	Takes `key`, a String, as an argument. The last six characters in `key` are digits.
	
	Sums each of the first three digits, divides the total by `size` and returns the remainder.
	
	For example, where `size` is 5:
	
	`hash("111JalanTenteram`**`322111`**`")`
	`>>> 2`
	
	`hash("23PasirRisAve4`**`520021`**`")`
	`>>> 2`

Task 3.3

Write program code for the class `HashTable`.

Evidence 13: Program code for **Task 3.3**. **[4]**

ACE OF CODERS PTE LTD would like to conduct a lucky draw. Each customer is entitled to submit one entry. All customer data entered is stored in the text file `CUSTOMERDATA.txt`. The format of the text file is as follows:

```
<CUSTOMER NAME> <CONTACT NUMBER>|<ADDRESS>
```

Below is a sample of `CUSTOMERDATA.txt`:

```
Felicia Lee Si Ying 98635610|19JalanTenteram322019

Yap Chee How 67767515|23TohTuckDrive608023

Christy Lopez 92233123|507WestCoastRoad120507
```

There will be five lucky draw winners. The company has identified five regions in Singapore and would like to pick one winner from each region. The hashing function from Task 3.3 will categorise an address in one of the five regions.

Task 3.4

Using the classes programmed previously, code a main program that repeatedly:

- displays the following menu:

```
1. Prepare hash table
2. Generate winners
3. Search for entry
```

- calls an appropriate procedure depending on the user's choice

When implementing the choices, take note of the following:

1. **Prepare hash table**
   User inputs suitable size of hash table to be initialised.
   Reads customer data from `CUSTOMERDATA.txt`.
   Uses address as key and customer name, contact number as value.
   Stores the customer names, contact numbers into the hash table.
2. **Generate winners**
   Randomly select one winner from each binary search tree in the hash table.
   Display their names and contact numbers.
3. **Search for entry**
   Staff of the company may not enter the lucky draw.
   Displays whether the staff name and contact number entered by user exists in the hash table.

Evidence 14: Program code for **Task 3.4**. **[10]**

Task 3.5

Run your program. Complete choice 1 before searching the following.

```
>>> Sandra Chelvan 92233123

>>> Haz Awang 87767888

>>> Seah Hon Hui 91144000
```

Evidence 15: Screenshot of running **Task 3.5**. **[2]**

4.

Rock Paper Scissors is a well-known hand game played between two players, whom we shall call P1 and P2. For each round, each player simultaneously forms one of three signs with an outstretched hand. Rock (represented by "0"), scissors (represented by "2") and paper (represented by "5").

Program `get_round_winner(p1_hand, p2_hand)` that decides who wins from the hand signs of P1 and P2. If P1 wins, the function returns 1. If P2 wins, the function returns 2. If the game is a draw, the function returns 0.

Sample Execution:

```
get_round_winner ("2", "5") # scissors vs paper - P1 wins

>>> 1

get_round_winner ("2", "0") # scissors vs rock - P2 wins

>>> 2

get_round_winner ("2", "2") # scissors vs scissors - Draw

>>> 0
```

Task 4.1

Write program code for the function `get_round_winner`.

Evidence 16: Program code for **Task 4.1**. **[4]**

P1 and P2 played a game of Rock Paper Scissors over a few rounds. The winner is the player who wins more rounds than he losses. A draw occurs if there is no winner. The hand signs of each player for the game are represented as a string of "0", "2" and "5". For example, "205" represents playing scissors in the first round, rock in the second, and paper in the third.

Program `get_game_winner(p1_hands, p2_hands)` that decides who wins the game from a sequence of hand signs from P1 and P2. If P1 wins, return 1, if P2 wins, return 2, if the game is a draw, return 0. You must use `get_round_winner`. You may assume that `p1_hands` and `p2_hands` are strings of the same length containing only "0", "2" and "5".

Sample Execution:

```
get_game_winner ("", "") #no hand played, game ends in a draw

>>> 0

get_game_winner ("205", "520") #P1 wins all 3 rounds, P1 wins

>>> 1

get_game_winner ("22222", "50505") #P1 wins

>>> 1

get_game_winner ("2050", "0255") #P2 wins

>>> 2

get_game_winner ("52025", "52025") #Draw

>>> 0
```

Task 4.2

Write program code for the function `get_game_winner`.

Evidence 17: Program code for **Task 4.2**. **[6]**

P1 and P2 would like a more interesting interface where rows and columns display the game played. **Each game has four rounds** and the **overall result** (“P1 wins”, “P2 wins” or “Draw”) **will be displayed when the game is over**.

Here is a sample screenshot of the first turns taken by player.

```
Round           P1              P2              Round Winner
1               -               -               -
2               -               -               -
3               -               -               -
4               -               -               -

Player's 1 turn.
Enter Rock (0), Paper (5) or Scissors (2):2

Round           P1              P2              Round Winner
1               S               -               -
2               -               -               -
3               -               -               -
4               -               -               -

Player's 2 turn.
Enter Rock (0), Paper (5) or Scissors (2):0

Round           P1              P2              Round Winner
1               S               R               P2
2               -               -               -
3               -               -               -
4               -               -               -
```

Task 4.3

Write program code for the game described. Implement using **modularity** and **2D array**. Include function `validate()` to ensure the player chooses `0`, `2` or `5`. Prompt the player to re-enter if input is invalid.

Evidence 18: Program code for **Task 4.3**. **[10]**

Evidence 19: Design four appropriate test cases for the program. Present them in a table format. **[4]**

Evidence 20: Run your program and produce screenshots for a game which ends in a draw and another game which P1 wins. **[2]**

End of Paper