YISHUN INNOVA JUNIOR COLLEGE
JC 2 PRELIMINARY EXAMINATION
Higher 2

CANDIDATE NAME			
CG		INDEX NO	

COMPUTING

9569/02

Paper 2 (Lab-based)

30 Aug 2022
3 hours
100 Marks

Additional Materials: Removable storage device with the following files:

- `template.ipynb` (for Question 1, 2 and 3)
- `MRT.TXT`
- `PSEUDOCODE_TASK_3_2.TXT`
- Q4 Web App Folder (with `LOAN.TXT`, `SportsLoans.db`, and templates for `server.py` and `Task_4_1.py`)

READ THESE INSTRUCTIONS FIRST

Answer **all** 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.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

The number of marks is given in brackets [ ] at the end of each task.
The total number of marks for this paper is **100**.

Instruction to candidates:

Your program code and output for Question 1, 2 and 3 should be saved in a single `.ipynb` and downloaded as

`template_<your class>_<your name>.ipynb`

For each of the sub-tasks, add a comment statement, at the beginning of the code using the hash symbol '#', to indicate the sub-task the program code belongs to, for example:

```
In [1]:   #Task 1.1
          Program Code

          Output:

In [2]:   #Task 1.2
          Program Code

          Output:

In [3]:   #Task 1.3
          Program Code

          Output:
```

1 A text file, `MRT.TXT`, contains the locations of the MRT (Red Line) stations where each line contains comma-delimited data that shows the station number, the station name, the latitude and the longitude of a station.

The station data are arranged in ascending order of the station number in the text file.

Task 1.1

Write program code to:

- read the station data from the given text file
- store information of all the stations in a list
- store the values of the longitude and latitude as floats
- return the list [8]

The locations of the stations are indicated by the longitude and latitude coordinates. The stations with larger latitudes are at the northern part of the island and those with smaller latitudes are at the southern part.

The distance between two stations can be found using the *Haversine formula* provided in the `template.ipynb`. You may use the following code to calculate the distance:

```
>>> Haversine(pt1, pt2)
```

where `pt1 = [longitude1, latitude1]` of station 1
and `pt2 = [longitude2, latitude2]` of station 2

[Turn over

Task 1.2

Write program code to:

- iterate through the list of stations to find the northernmost and southernmost stations
- calculate the distance between the northernmost and southernmost stations

```
Output:

Distance between _____________ and _____________ is ___ km.
```

[5]

Task 1.3

Write program code to:

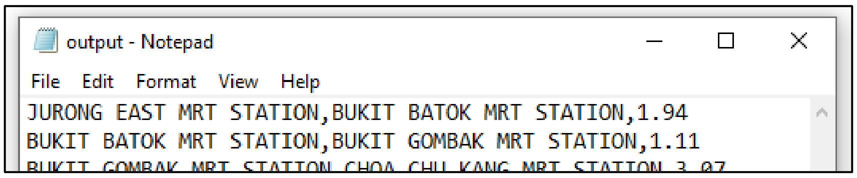
- iterate through the list of stations to find all the distances between two neighbouring stations
- calculate the total distance between all the stations on the Red Line
- print the tabulated data similar to the following:

```
FROM                       TO                         DISTANCE (km)
JURONG EAST MRT STATION    BUKIT BATOK MRT STATION          1.94
BUKIT BATOK MRT STATION    BUKIT GOMBAK MRT STATION         1.11

...
...

 Total Distance:  41.27 km
```

- write the names of the stations and the distance between them to a text file, `output.TXT`.

  Each line contains comma-delimited data that shows the name of station1, the name of station2 and the distance between them

  A sample screenshot of the text file, `output.TXT`, is as shown below:



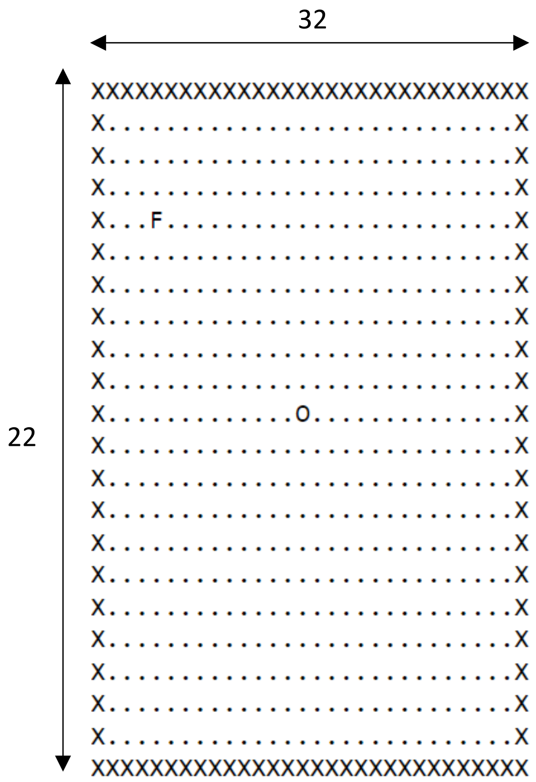
```
JURONG EAST MRT STATION,BUKIT BATOK MRT STATION,1.94
BUKIT BATOK MRT STATION,BUKIT GOMBAK MRT STATION,1.11
BUKIT GOMBAK MRT STATION,CHOA CHU KANG MRT STATION,3.07
```

[12]

2 In a computer game, a player moves a snake (`"O"`) within a 30m by 20m open field marked with `"."` that is surrounded by an electric fence marked with `"X"`.

The field is represented on the screen by a rectangular grid. Each square metre of the region is represented by an x-coordinate and a y-coordinate. The top left corner of the electric fence has `x=0` and `y=0` and the bottom right corner of the fence has `x=31` and `y=21`.

The snake starts at the centre of the field `(15,10)` and moves left, right, up or down according to the direction entered by the player. The player will move the snake towards the food item (`"F"`) appearing in a random position in the field. The game is turn-based; the player enters the direction and the snake moves one position in that direction. The field is displayed after each move.



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X..............................X
X..............................X
X..............................X
X..F...........................X
X..............................X
X..............................X
X..............................X
X..............................X
X..............................X
X..............O...............X
X..............................X
X..............................X
X..............................X
X..............................X
X..............................X
X..............................X
X..............................X
X..............................X
X..............................X
X..............................X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Task 2.1

Write program code to display the field as shown above.

- Use a suitable data structure to store the coordinates of the field `(".")` and electric fence `("X")`.
- The starting position for the snake `("O")` should be at the centre of the field `(15,10)`.
- The food item (`"F"`) is placed randomly within the field. It cannot appear at the same position as the fence `("X")` or the snake `("O")`. [8]

To move the snake, the player is prompted to input a direction. The following are the valid inputs:

Input character	Action
"w"	Snake moves up
"s"	Snake moves down
"a"	Snake moves left
"d"	Snake moves right
Empty string	Continue with previous move. If no previous move, do nothing

When the player enters a move, an `"O"` is placed at the snake's new position, the previous position will be replaced with a `"."` and the updated field is displayed. The player will continue to move the snake until it reaches the food item `("F")`.

If the player moves the snake `("O")` into the electric fence `("X")`, it will be returned to the starting position at `(15,10).`

Task 2.2

Add to your program code to:

- prompt the player for an input and validate the direction.
- calculate the new position of the snake.
  If the new position is on the electric fence, return it to the starting position at `(15,10)`; Otherwise, update the field with the `"O"` at the new position and `"."` at the previous position.
- continue to prompt the player for an input until the snake reaches the food item.

[8]

When the snake reaches the food item, the player will gain one point for his score. A new food item `("F")` will appear at another random position and the player will continue to direct the snake towards it.

The game ends when the snake touches the electric fence for the third time and the player's score will be displayed.

Task 2.3

Add to your program code to end the game when the condition is met and display the message "**End Game**" with the player's score. [6]

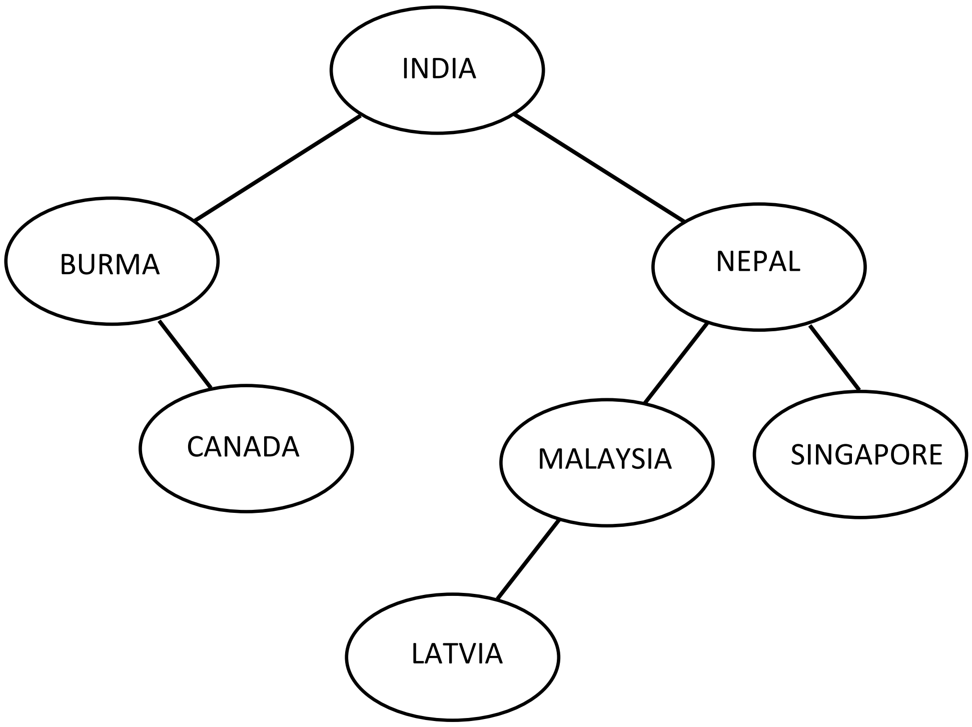
[Turn over

3 A binary tree Abstract Data Type (ADT) has commands to create a new tree, add unique data items to the tree and print the tree.

The sequence of commands:

```
CreateNewTree
AddItem('INDIA')
AddItem('NEPAL')
AddItem('MALAYSIA')
AddItem('SINGAPORE')
AddItem('BURMA')
AddItem('CANADA')
AddItem('LATVIA')
```

would create the following binary tree:



The program to implement this ADT will use the classes `BSTree` and `Node` designed as follows:

Class: `Node`	
Properties	
Identifier	**Description**
`Data : STRING`	The node's data value
`Left : INTEGER`	The left pointer for the node. (Default: `None`)
`Right : INTEGER`	The right pointer for the node. (Default: `None`)
Methods	
Identifier	**Description**
`Constructor(NewTreeItem)`	Constructs a `Node` object with `NewTreeItem` as the data value
`SetLeft(x : INTEGER)`	Set the left pointer to the index position of the `Node` in the list `ThisTree`
`SetRight(y : INTEGER)`	Set the right pointer to the index position of the `Node` in the list `ThisTree`
`GetData() : STRING`	Returns the node's `Data` value
`GetLeft() : INTEGER`	Returns the node's left pointer
`GetRight() : INTEGER`	Returns the node's right pointer

Class: `BSTree`	
Properties	
Identifier	**Description**
`ThisTree : LIST`	The tree data stored as `Nodes` in a `LIST`
`Root : INTEGER`	Index of the tree's root `Node` in the `ThisTree` list. (Default: `None`)
`NextFree : INTEGER`	Index for the tree's next new `Node` in the `ThisTree` list. (Default: `0`)
Methods	
Identifier	**Description**
`Constructor()`	Constructs a `BSTree` object
`AddItem(NewTreeItem)`	Add a new `Node` with `NewTreeItem` into the binary tree
`InOrderTraversal()`	Display the data items in order

Task 3.1

Write program code to define the classes `BSTree` and `Node`. The program code must create a new binary tree which has no node and the pointer `Root` is set to `None`.

Do not attempt to write the methods `AddItem` and `InOrderTraversal` at this stage.

[4]

The following pseudocode inserts a data value into the binary tree structure.

The `LastMove` variable holds the direction of the previous traversal move as follows:

`X` – no move yet made

`L` – move was to the left

`R` – move was to the right

```
PROCEDURE AddItem(NewTreeItem)

  IF Root = None  THEN
    ThisTree.append(Node(NewTreeItem))
    Root ← NextFree
    NextFree ← NextFree + 1
  ELSE
    //traverse the tree to find the position for the new item
    Current ← Root
    LastMove ← 'X'

    REPEAT
      Previous ← Current
      IF NewTreeItem < ThisTree[Current].Data  THEN
        //move left
        LastMove ← 'L'
        Current ← ThisTree[Current].Left
      ELSE
        // move right
        LastMove ← 'R'
        Current ← ThisTree[Current].Right
      ENDIF
    UNTIL Current = None

    IF LastMove = 'R'  THEN
      ThisTree[Previous].Right ← NextFree
    ELSE
      ThisTree[Previous].Left ← NextFree
    ENDIF

    ThisTree.append(Node(NewTreeItem))
    NextFree ← NextFree + 1

  ENDIF

ENDPROCEDURE
```

Note: The above text is available in the text file `PSEUDOCODE_TASK_3_2.TXT`

Task 3.2

Write non-recursive program code to implement the `AddItem(NewTreeItem)` method to add a new `Node` with the `NewTreeItem` data value into the binary tree.

You may use the text file `PSEUDOCODE_TASK_3_2.TXT` as a basis for the writing of your program code. [6]

Task 3.3

Write a sequence of program codes to:

- create an empty binary tree
- add the data items in the order shown:
  `INDIA, NEPAL, MALAYSIA, SINGAPORE, BURMA, CANADA, LATVIA`
- display the contents of `ThisTree` in index order.
  For each node, display the index, data value, left pointer and right pointer.

[6]

Task 3.4

Write program code for the additional method `InOrderTraversal()` to output the data stored in the tree in alphabetical order. [7]

[Turn over

4 The college's Physical Education (PE) department launched the *ActiveDay* programme to encourage students to lead an active lifestyle. The students can borrow any sports equipment for a day to play with their friends. The students are awarded different points for the equipment they loan depending on the physical intensity of the sport. The points will be accumulated for an award at the end of the semester.

A relational database is used to track the loan of the sports equipment, and the students' details. The students can browse and reserve the available equipment on a web page at least a day before the loan date.

The students may loan more than one piece of the same equipment and will receive the number of points for each piece of the loaned equipment.

The table descriptions for the database are as follows:

- `Equipment (ID, Name, Points)`
- `Student(ID, Name)`
- `Loan (LoanID, EquipmentID, StudentID, Qty, LoanDate, ReturnDate)`

The primary and foreign keys are indicated with underline and dashed underline respectively.
The values for the `LoanID` field in the `Loan` table are auto-generated integers.
The `LoanDate` and `ReturnDate` in the `Loan` table are the `YYYY-MM-DD` text format.

The database `SportsLoans.db` provided contains all the three tables, with the `Equipment` and `Student` tables populated with data.

Task 4.1
Write Python code to populate the `Loan` table with the data from the `LOAN.TXT` file.
Save your program code as `Task4_1_<your name>.py`. [5]

Task 4.2

A student, Andy, wishes to check his total accumulated points and compare it against the top three students' accumulated points.

Write SQL code to:

- compute and show the total points accumulated by Andy
- compute and display the names and the accumulated points of the top three students in descending order of the points.

Save the two SQL codes as a text file and name it as

`Task_4_2_<your name>.sql` [5]

Task 4.3

Write a Python programme and the necessary files to create a web application. The web application offers the following menu options:

```
1. Loan sports equipment
2. Check Leaderboard
```

The options will display the loan request form in the `loan.html` and the top three students' accumulated points in the `leaderboard.html` as described in Task 4.4 and 4.5 respectively.

Save your program as

`server_<your name>.py` and `index_<your name>.html`

Run the web application and save the output of the program as

`Task4_3_output_<your name>.html` [4]

Task 4.4

When option 1 is selected in the menu, an equipment loan request form will be displayed. The form allows the students to:

- select their name from a drop-down list
- select ~~multiple items~~ one item and indicate the loan quantity ~~for each item~~
- input the start date for the loan.

Modify the Python code `server.py` in Task 4.3 to receive the inputs from the loan request form and add the loan details into the database `SportsLoans.db`. The return date `ReturnDate` should be auto-generated as two days after the `LoanDate`, regardless of whether the return date falls on a weekend or not.

For example, if the loan date is on `2022-08-25`, the return date will be generated as `2022-08-27`.

Save your program as `server_<your name>.py` and `loan_<your name>.html`.

Run the web application and save the output of the displayed loan request form as `Task4_4_output_<your name>.html` [5]

Task 4.5

After adding the loan details into the database, the web server will render a web page `success.html` to be displayed on the client's browser showing the following:

- student's name,
- return date
- number of points to be awarded for this loan.

Save your program as

`server_<your name>.py` and `success_<your name>.html`

Run the web application and save the output of the program as

`Task4_5_output_<your name>.html` [8]

Task 4.6

When option 2 is selected in the menu, the program will prompt the student to input a name.

Modify the Python code `server.py` in Task 4.5 to compute the accumulated points and render a web page `leaderboard.html` to display the followings:

- the student's total accumulated points
- the top three students' names and their accumulated points in descending order of the points.

Save your program as

`server_<your name>.py` and `leaderboard_<your name>.html`

Run the web application and save the output of the displayed leaderboard as

`Task4_6_output_<your name>.html` [3]

BLANK PAGE