Name:_________________________________ Class:____________

JURONG PIONEER JUNIOR COLLEGE

JC2 Preliminary Examination 2024

COMPUTING
Higher 2 **9569/02**

14 August 2024

Paper 2 (Practical) **3 hours**

Additional materials:	Cover Page
	Electronic version of `TASK1.txt` data file
	Electronic version of `TASK3.txt` data file
	Electronic version of `TASK4.txt` data file
	Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

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.

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.

Note that up to **6** marks out of 100 will be awarded for the use of common coding standards for programming style.

The number of marks is given in brackets [ ] at the end of each task.

The total number of marks for this paper is 100.

This document consists of **13** printed pages.

[Turn over

Instructions to candidates:
Your program code and output for each of Task 1 to 4 should be downloaded in a single`.ipynb` file. For example, your program code and output for Task 1 should be downloaded as `TASK1_<your class>_<your name>.ipynb`.

1 Name your **Jupyter Notebook** as:
`TASK1_<your class>_<your name>.ipynb`

The task is to implement a Vigenère cipher encryption algorithm.

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution. A key is repeated to match the length of the plaintext, and each letter in the plaintext is shifted according to the corresponding letter in the key.

For example:

The plain text is `“HELLO WORLD”` and the key is `“ABLE”.`

The key, repeated to match the 11 characters in the text including spaces, is `“ABLEABLEABL”.`

Each letter in both the plain text and key is first converted to a number (A=0, B=1, C=2, D=3, E=4, F=5, ..., Z=25). This is to be done with the help of ASCII values.

The two numbers are then added up to give the encrypted letter.

First letter `‘H’` in the plain text is number 7 and first letter `‘A’` in the key is number 0. Calculate 7 + 0 = 7. Hence, the first encrypted letter is `‘H’`.

Second letter `‘E’` in the plain text is number 4 and second letter `‘B’` in the key is number 1. Calculate 4 + 1 = 5. Hence, the second encrypted letter is `‘F’`.

When a letter goes beyond `‘Z’`, it returns to `‘A’`. Use modular arithmetic with modulus 26 to wrap around if necessary.

Space is replaced with the character `‘!’`.

Plaintext character	H	E	L	L	O		W	O	R	L	D
Key character	A	B	L	E	A	B	L	E	A	B	L
Encrypted character	H	F	W	P	O	!	H	S	R	M	O

The resulting encrypted text is `“HFWPO!HSRMO”.`

Task 1.1

Write a function `CharToNum()` that takes an uppercase letter as parameter and returns an integer corresponding to its position in the alphabet, where `‘A’`=0, `‘B’`=1, `‘C’`=2, ..., `‘Z’`=25. You are required to use ASCII encoding in your implementation. [2]

Task 1.2

Write a function `encrypt()` that takes a string `text` and a string `key` as input parameters.

The `encrypt()` function returns an encrypted string where each letter in `text` is shifted forward by the corresponding letter in `key`. You are required to use the function from **Task 1.1** in your implementation.

Assume both the `text` and `key` contain only uppercase letters, and spaces are allowed in the text. No other characters are in the `text` and `key`. [7]

Task 1.3

The text file `TASK1.txt` contains messages that needs to be encrypted. Thereafter, the encrypted messages are stored in a text file named `ENCRYPTED.txt`.

Write the program code to:

- Read the data from the text file `TASK1.txt`
- The key is `“JPJC”`
- Use your function from **Task 1.2** to encrypt the content
- Write the encrypted messages to the text file `ENCRYPTED.txt`

[3]

Test your program with the plain text in the file `TASK1.txt`.

Display the content of `ENCRYPTED.txt` after you have run the program. [1]

Save your Jupyter Notebook for Task 1.

[Turn over

2 Name your **Jupyter Notebook** as:
`TASK2_<your class>_<your name>.ipynb`

A programmer is writing Coconut Island Game to be played on the computer.

The island is represented as a rectangular grid, 10 rows by 15 columns. Each square of land on the island is represented by a pair of coordinates in a 2D array. The top left square of the island has coordinates where row = 0 and column = 0. There are 10 squares of land down and 15 squares of land across.

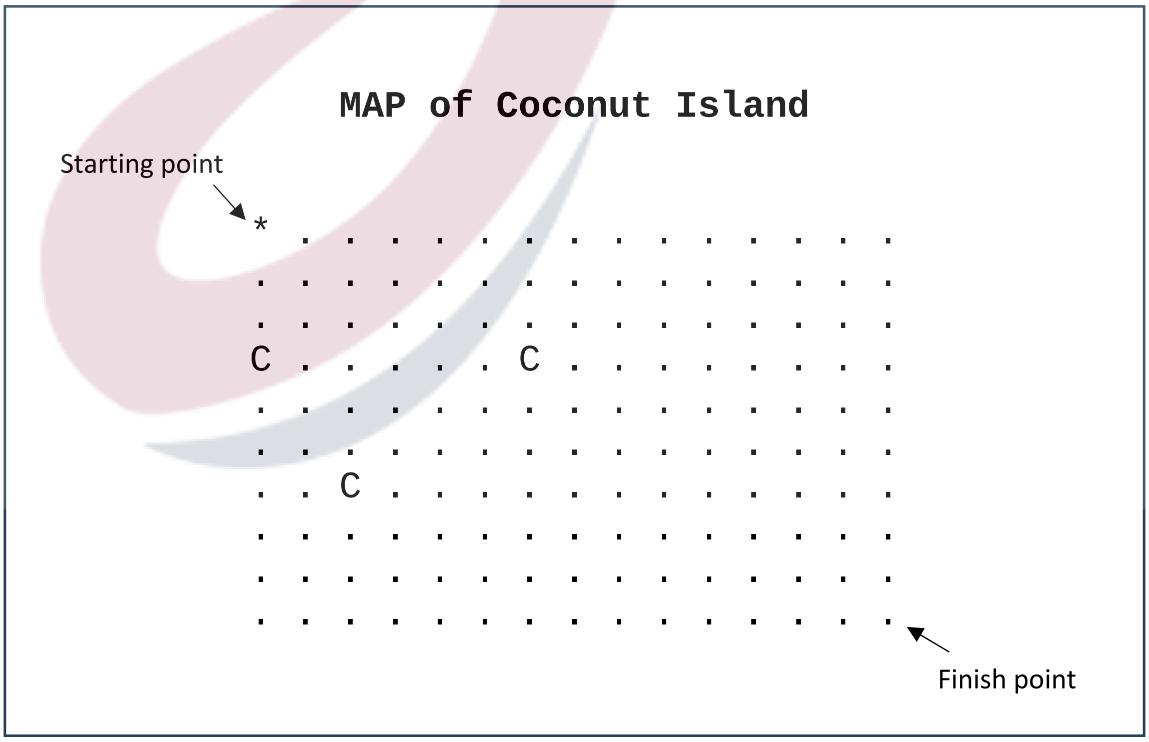
The computer will:

- generate three random locations where coconut trees will be located
- start player at the top left corner of the grid (row = 0, column = 0)
- prompt the player for the move, in the format direction left (L), right (R), up (U), down (D), followed by a space and the distance representing the number of grids moved in the chosen direction (e.g.: `R 6` would move the player six grids to the right from the current location)
- plot the path the player has moved
- display the contents of the array by outputting for each square of land:
    - `'.'` for unexplored land
    - `'C'` for coconut tree
    - `'E'` for eaten coconut when player's move ends on a coconut tree
    - `'*'` for explored land i.e. path walked on by player

The rule of the game is:

- player has maximum eight moves
- to win the game, the player needs to
    - o visit the three squares with coconut trees to eat them, and
    - o reach the finish point at bottom right corner (row = 9, column = 14)

A sample grid of the island, with three coconut trees, is displayed:



```
MAP of Coconut Island

* . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
C . . . . . C . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . C . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
```

The following is a sample run of the game:

```
Welcome to Coconut Island Game!

* . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
C . . . . . C . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . C . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .

You are at position: (0, 0)  Coconut eaten: 0
Enter your move (L/R/U/D distance): D 3
```

```
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
E . . . . . C . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . C . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .

You are at position: (3, 0)  Coconut eaten: 1
Enter your move (L/R/U/D distance): R 6
```

```
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
E * * * * * E . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . C . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .

You are at position: (3, 6)  Coconut eaten: 2
Enter your move (L/R/U/D distance): D 3
```

```
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
E * * * * * E . . . . . . . .
            *
. . . . . .   . . . . . . . .
            *
. . . . . .   . . . . . . . .
. . C . . . * . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
You are at position: (6, 6)  Coconut eaten: 2
Enter your move (L/R/U/D distance): L 4
```

```
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
E * * * * * E . . . . . . . .
. . . . . . * . . . . . . . .
. . . . . . * . . . . . . . .
. . E * * * * . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .

You are at position: (6, 2)  Coconut eaten: 3
Enter your move (L/R/U/D distance): D 3
```

```
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
E * * * * * E . . . . . . . .
. . . . . . * . . . . . . . .
. . . . . . * . . . . . . . .
. . E * * * * . . . . . . . .
. . * . . . . . . . . . . . .
. . * . . . . . . . . . . . .
. . * . . . . . . . . . . . .

You are at position: (9, 2)  Coconut eaten: 3
Enter your move (L/R/U/D distance): R 12
```

```
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
* . . . . . . . . . . . . . .
E * * * * * E . . . . . . . .
. . . . . . * . . . . . . . .
. . . . . . * . . . . . . . .
. . E * * * * . . . . . . . .
. . * . . . . . . . . . . . .
. . * . . . . . . . . . . . .
. . * * * * * * * * * * * * *

You are at position: (9, 14)  Coconut eaten: 3

You have won!
You have reached finish point and eaten 3 coconuts.
```

The Coconut Island Game is to be implemented using procedural programming.

Task 2.1

Write the following functions:

1. `createIsland(rowsize, colsize)`
   - creates a `grid` of size `rowsize` x `colsize`, its elements initialized to `‘.’`, and with the starting position marked by `'*'` at the top-left corner (row = 0, col = 0)
   - returns the grid

2. `plantCoconuts(grid)`
   - plants 3 coconut trees `'C',` randomly generated, on the `grid`
   - ensure the square is empty before planting
   - returns the `grid`

3. `display(grid)`
   - displays the current state of the `grid`

4. `move(grid, current, direction, distance, coconut)`
   - moves the player in the specified `direction` by a certain `distance` from the `current` position. The directions can be left (L), right (R), up (U), or down (D)
   - mark in the `grid` every square of the player’s path with `‘*’`, if the square does not have a coconut tree on it. If there is a coconut tree on the square, leave the marking `‘C’` unchanged
   - if the player’s move ends on a coconut tree, update the `grid` to mark the square as eaten `'E'` and increment `coconut` eaten by 1
   - returns a tuple containing the `grid`, `current` and `coconut`

Assume all moves the player input are valid which will not result in out-of-range coordinates.

[16]

Task 2.2

Write the function `main()` that uses the subroutines written in **Task 2.1** to perform the following:

- Create the island, called `grid`.
- Plant three coconut trees on the island.
- Initialize the player starting position as (row = 0, col = 0), coconut eaten as 0, and other necessary variables.
- Display the `grid.`
- Use a loop to do the following (maximum 8 moves):
  - Print player's current location and coconut eaten
  - Prompt player to input the move in the format `<direction> <space> <distance>`, where `direction` is a character `L/R/U/D` and `distance` is an integer. Refer to sample run.
  - Update the `grid` with the player's move.
  - Display the `grid.`
  - Check if player has won (reached finish point and eaten 3 coconuts). If yes, exit the loop.
- If player has won, display:
  ```
  You have won!
  You have reached finish point and eaten 3 coconuts.
  ```
- Otherwise, display the message:

  `Sorry you did not win.`
  `You did not reach the finish point.`
  and/or: `You have not eaten enough coconut.`
- Exit game. [10]

Run the `main()` function and show a game with a win or lose result. [1]

Save your Jupyter Notebook for Task 2.

3 Name your **Jupyter Notebook** as:
`TASK3_<your class>_<your name>.ipynb`

Binary tree is one of the fundamental data structures in Computer Science that represents hierarchical relationships. They are efficient for various operations, including searching, sorting, and traversing.

In certain scenarios, transforming a binary tree into a linear structure, such as a linked list, can be beneficial. For example, flattening a hierarchical data structure like a binary tree to a non - hierarchical one like a linked list allows for sequential access to the elements, simplifying operations that require iterative processing of nodes. This transformation retains both the original structure and the values of the nodes, arranged in an order determined by the **pre-order traversal** of the nodes in the binary tree.

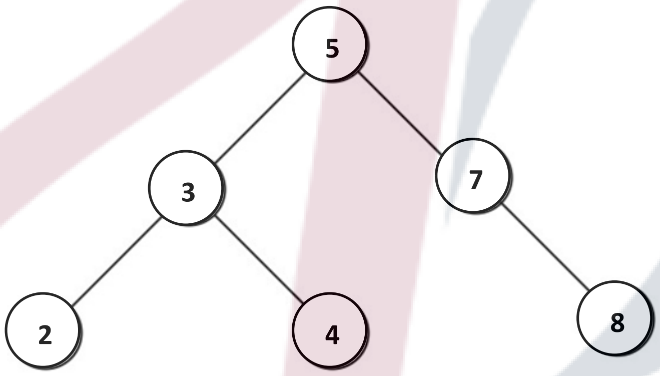


Figure 1: A binary search tree with six nodes

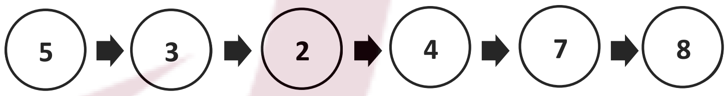


Figure 2: The flattened linked list with six nodes

In this task, you will implement a method to flatten a binary search tree BST into a linked list using the `right` pointers to create the sequence, making all `left` pointers `None`.

The root of BST is the head node of the linked list. The linked list should use the `right` pointers to point to the next node, and the `left` pointers should all be `None`.

The transformation from BST to a linked list retains both the original structure and the values of the nodes, arranged in an order given by its **pre-order traversal** of BST.

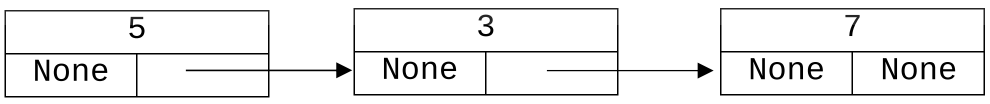


Figure 3: A flattened linked list with three nodes

A binary search tree BST is implemented using Object-Oriented Programming (OOP).

The class `Node` contains three attributes:

- `value` is the data in the node,
- `left` points to the left node, and
- `right` points to the right node.

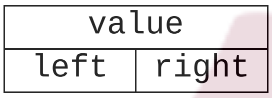


Figure 4: A `Node` object

The class `Node` contains the following method:

- a constructor to set the `left` and `right` pointers to `None`, and `value` to an integer parameter.

The class `Tree` contains the following attribute:

- `root` points to its root node.

The class `Tree` contains the following methods:

- a constructor to set the `root` to a `Node` object parameter,
- an iterative method `insert` to take the `Node` object parameter and store it in the correct position in the tree
- a recursive method `in_order_traversal` to perform in-order traversal to output the data in the tree
- a recursive method `flatten` to flatten the tree into a linked list.

Task 3.1
Write the program code to declare the class `Node` and its constructor. [3]

Task 3.2
Write the program code to declare the class `Tree` and its methods. [18]

Task 3.3
A BST with six nodes is defined in the text file TASK3.TXT.

Write the main program to:

- use the code in the text file TASK3.TXT to instantiate BST,
- print the values output by `in_order_traversal` method, and
- flatten the BST into a linked list `LL` using the `flatten` method. [4]

Task 3.4
Test your program and display the data of the nodes in `LL` during traversal. [2]

Save your Jupyter Notebook for Task 3.

4 Name your **Jupyter Notebook** as:
`TASK4_<your class>_<your name>.ipynb`

The Airport Flight Information System plans to develop a web application to allow visitors to view information of flights arriving at or departing from Singapore Changi Airport. The database will have one table to store this information.

The fields in the `Flight` table are as follows

`Flight:`

- `flightNum`: a unique string assigned to the flight
- `departure`: the location of the airport the plane departs from
- `destination`: the location of the airport the plane arrives at
- `departureTime`: the time the plane departs from the airport
- `arrivalTime`: the time the plane arrives at the airport

Task 4.1

Write a Python program that uses SQL code to create the database `Airport` with the table given. Define the primary key for the table. [3]

Task 4.2

The text files `TASK4.txt` store the comma-separated values for the table in the database.

Write a Python program to read in the data from the file and store in the `Flight` table in the database. [5]

Run your program to test the data has been entered into the database correctly. [1]

Save your Jupyter Notebook for Task 4.

Task 4.3

Write a Python program and the necessary files to create a web application. The web application has the following menu options.

View all Arrivals

View all Departures

Query Flight

Save your Python program as:
`TASK_4_3_<your class>_<your name>.py`

with any additional files/ subfolders in a folder named:
`TASK_4_3_<your class>_<your name>` [3]

Task 4.4

Write a Python program and the necessary files to create a web application that will display the information for all flight arrivals in Singapore.

The flights should be displayed on the web page labelled **Arrival**. It is arranged in chronological order of the arrival time, in a table with the following table header:

- Time (that is, arrival time)
- From (that is, departure location)
- Flight (that is, flight number)

The resulting web page should be accessed from the "View all Arrivals" menu option from **Task 4.3**.

Save your Python program as:
`TASK_4_4_<your class>_<your name>.py`

with any additional files/ subfolders in a folder named:
`TASK_4_4_<your class>_<your name>` [6]

Run the web application. Save the arrival web page output as:
`TASK_4_4_<your class>_<your name>.html` [1]

Task 4.5

Write a Python program and the necessary files to create a web application that will display the information for all flight departures from Singapore.

The flights should be displayed on the web page labelled **Departure**. It is arranged in ascending order of the departure time, in a table with the following table header:

- Time (that is, departure time)
- To (that is, destination location)
- Flight (that is, flight number)

The resulting web page should be accessed from the "View all Departures" menu option from **Task 4.3**.

Save your Python program as:
`TASK_4_5_<your class>_<your name>.py`

with any additional files/ subfolders in a folder named:
`TASK_4_5_<your class>_<your name>` [2]

Run the web application. Save the departure web page output as:
`TASK_4_5_<your class>_<your name>.html` [1]

Task 4.6

Write a Python program and the necessary files to create a web application that:

- contains a HTML form for user to input `flightNum`
- receives a `flightNum` string from the HTML form, then
- creates and returns a HTML document that enables the web browser to display in a table with heading, the flight details of that `flightNum`.

The flight details are as follows

- `flightNum`
- `departure`
- `destination`
- `departureTime`
- `arrivalTime`

The web page with the HTML form should be accessed from the "Query Flight" menu option from **Task 4.3**.

Save your Python program as:
`TASK_4_6_<your class>_<your name>.py`

with any additional files/ subfolders in a folder named:
`TASK_4_6_<your class>_<your name>` [4]

Run the web application. **Query for flight SQ111**.

Save the query result web page output as:
`TASK_4_6_<your class>_<your name>.html` [1]

END OF PAPER