ANGLO-CHINESE JUNIOR COLLEGE
JC1 PROMOTIONAL EXAMINATION

Higher 2

COMPUTING 9569/02

Paper 2 (Lab-based) **22 August 2024**

1.5 hours

Additional Materials: Electronic version of `members.txt` data file
Insert Quick Reference Guide

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.

Note that up to **3** marks out of 50 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 question or part question.
The total number of marks for this paper is 50.

Instruction to candidates:

Your program code and output for each of Tasks 1 and 2 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

Make sure that each of your `.ipynb` files shows the required output in Jupyter Notebook.

1 Name your Jupyter Notebook as:

`TASK1_<your name>_<centre number>_<index number>.ipynb`

Battleship is a simple game where two players attempt to locate and sink their opponent's ships.

Each player has a gameboard, which is a 5 × 5 array of squares, and two ships. Each ship occupies 3 adjacent squares in the gameboard, which must be in either the same row or the same column. No square may be occupied by both ships at the same time. The ships do not move.

Players take turns trying to sink their opponent's ship by calling out a square to hit. A ship sinks when all the squares it occupies have been hit. A player loses when both ships have sunk.

In the diagram below, each `"."` represents an unoccupied square, and `"S"` represents a square occupied by a ship.

5	.	.	.	.	.
4	.	.	.	.	.
3	S	.	S	S	S
2	S	.	.	.	.
1	S	.	.	.	.
	A	B	C	D	E

Once a square has been hit, `"X"` will be indicated on the gameboard. If the square originally contained a ship, the `"S"` is replaced by an `"X"`.

The diagram below shows an example where squares with coordinates C3 and D2 have been hit.

5	.	.	.	.	.
4	.	.	.	.	.
3	S	.	X	S	S
2	S	.	.	X	.
1	S	.	.	.	.
	A	B	C	D	E

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:
```

Task 1.1

The locations of a player's ships are given in a list `ship_lst`. This is a list of six coordinates in the gameboard. The first three coordinates are the squares occupied by one ship, and the next three coordinates are the squares occupied by the other ship.

The list is valid if

- Each ship occupies exactly 3 squares which are adjacent in the same row or the same column;
- No square is occupied by both ships.

For example,

- `["A3", "A1", "A2", "B2", "D2", "C2"]` is a valid `ship_lst`;
- `["A4", "A1", "A2", "B2", "D2", "C2"]` is not a valid `ship_lst`.

Write a function `valid_ships(ship_lst)` to check the validity of `ship_lst`.
The function should return `True` or `False` appropriately. [6]

Task 1.2

The player has a `hit_lst`, which contains a list of squares which have been hit by the opponent.

For example,

- `["A4", "B3"]` is a hit_lst if squares A4 and B3 have been hit by the opponent previously.

Write a function `random_attack(hit_lst)` that randomly returns a coordinate (e.g. `"A2"`) that is not inside `hit_lst`. This represents the next square that is hit. [4]

Task 1.3

Write a function `lose_check(hit_lst, ship_lst)` to check if the player has lost the game. [2]

Task 1.4

Write a function `printboard(hit_lst, ship_lst)` that takes `hit_lst` and `ship_lst` as parameters and displays a gameboard as shown below. A square that has been hit will display `"X"` whether or not a ship was originally on it. [5]

For example,

Values in `hit_lst` : `["C2", "C3", "D3"]`
Values in `ship_lst`: `["A1", "A2", "A3", "C3", "D3", "E3"]`

Outcome of running `printboard(hit_lst, ship_lst)`:

```
. . . . .
. . . . .
S . X X S
S . X . .
S . . . .
```

Task 1.6

Write a program code that makes use of the functions written in **Tasks 1.1 to 1.4** to play a simulated game of battleship against the computer.

In this version of the game, the player places the ships on the gameboard. The computer then attacks the player 15 times. The player does not attack the computer. If the player has at least one ship that has not sunk after 15 attacks, the player wins.

The code should:

1. Prompt the player for 6 coordinates to represent the squares his two ships are on
2. Check that the list of coordinates provided by the player is valid. If invalid, repeatedly prompt the player until a valid list is provided.
3. Print the gameboard
4. Call random_attack() to receive an attack from the computer
5. Display the coordinates of the square hit by the computer
6. Check if the player has lost
7. Repeat steps 3 to 6 for 15 attacks. The attacks should stop if the player has lost before the 15 rounds are up.
8. Display an appropriate message indicating whether the player wins or loses. [8]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as:

`TASK2_<your name>_<centre number>_<index number>.ipynb`

A binary search tree (BST) is a data structure that can be used to store data that comes in order. The table below shows the properties and methods of a `BST` class and a `Node` class.

Node
- data - left - right
+ Constructor(data)

The `left` and `right` attributes are pointers to the left and right children of the `Node`, if any.

BST
- root
+ Constructor() + insert(data) + post_order()

- The `root` attribute points to the `Node` at the root of the `BST`.
- `insert(data)` creates a new `Node`, whose data is `data`, and inserts it into the `BST` at the appropriate location. It may be assumed that no two `Node`s have identical values for `data`.
- `post_order()` returns a list of the `data` in the `Node`s of the `BST`, using a post-order traversal.

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 2.1
        Program code
        Output:
```

Task 2.1

Write program code for the `Node` and `BST` classes. [13]

Task 2.2

The file `members.txt` contains a list of members of an organization and their ages.

Write program code to:

- Create a `BST` called `NameTree` which stores the members' names in alphabetical order.
- Create a `BST` called `AgeTree` which stores their ages in increasing order.
- Create a file called `namelist.txt` which stores their names in the post-order traversal order of `NameTree`.

When the `Nodes` are inserted into `NameTree` and `AgeTree`, they should be inserted in the same order as given in `members.txt`. [9]

Save your Jupyter Notebook for Task 2.