YISHUN INNOVA JUNIOR COLLEGE
JC2 COMMON TEST 1

CANDIDATE NAME			
CT GROUP		INDEX NUMBER	

H2 COMPUTING **9597/01**

21 Feb 2019

50 Mins

Additional Materials: Removable storage device with the following files :

- EVIDENCE-DOC.doc
- template.py
- module.py

READ THESE INSTRUCTIONS FIRST

Type in the EVIDENCE-DOC document the following:

- Candidate details
- Programming language used

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.

All tasks and required evidence are numbered. The number of marks is given in brackets [ ] at the end of each task.

Copy and paste required evidence of program codes using '`Courier New`' font and screenshots the outputs into the EVIDENCE-DOC document.

At the end of the examination, save all the program codes and submit your EVIDENCE-DOC in the thumb drive provided.

This document consists of **5** printed pages.

Question 1

A stack is a Last-In-First-Out (LIFO) data structure. It can be implemented using a linked list.

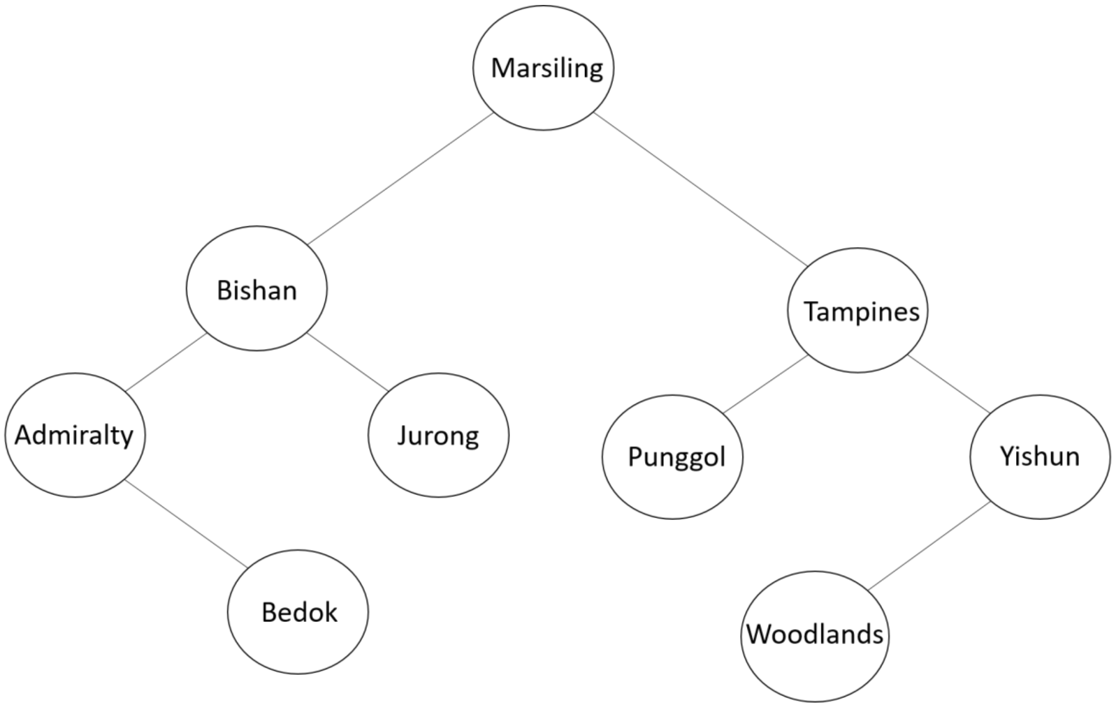
Task 1.1

Write the code for the class `Stack` which inherits all the methods from the class `LinkedList` as provided in the *module.py* file.

The program code must include the following methods:

- `push(value)` appends the parameter value to the stack
- `pop()` removes and returns the next value in the stack **[6]**

A binary tree is used to store the unique MRT station names in Singapore.



The above binary tree is created with the following sequence of commands:

```
CreateNewTree
AddToTree("Marsiling")
AddToTree("Bishan")
AddToTree("Admiralty")
AddToTree("Tampines")
AddToTree("Punggol")
AddToTree("Jurong")
AddToTree("Yishun")
AddToTree("Woodlands")
AddToTree("Bedok")
```

The class `TreeNode`, provided in *template.py*, is defined as follows:

Identifier	Data Type	Description
`LeftPtr`	`INTEGER`	The left pointer for the node.
`Data`	`STRING`	The data value stored in the node.
`RightPtr`	`INTEGER`	The right pointer for the node.

The binary tree can be implemented using variables as specified below:

Identifier	Data Type	Description
`ThisTree`	`ARRAY of TreeNode`	An array used to store the tree nodes.
`Root`	`INTEGER`	Index for the root position of the `ThisTree` array.
`NextFreePosition`	`INTEGER`	Index for the next insertion in the array.

Task 1.2

Using the given *template.py*, write the program code to implement the `add()` method within the `BinaryTree` class, so that it will add a new node to an appropriate position in the tree. **[10]**

Evidence 2:

Your program code for Task 1.2.

[Turn over

Task 1.3

Write a sequence of program statements to:

- create an empty binary tree
- add the MRT station names into the binary tree according to the original sequence of commands.
- use the `display()` method to print the array contents.

Execute your program to test it. **[3]**

Evidence 3:

Your program code for Task 1.3 and the screenshot of the test run.

A method `InOrderTraversal()` is to be added into the `BinaryTree` class, which outputs the data stored in the tree in an alphabetical order. The following pseudocode can be used to perform a **non-recursive in-order traversal** through a binary tree.

```
PROCEDURE InOrderTraversal()
     fringe ← empty Stack
     currIndex ← 0
     currNode ← ThisTree[currIndex]
     REPEAT
          IF currIndex >=0 THEN
               currNode ← ThisTree[currIndex]
               fringe.push(currNode)
               currIndex ← currNode.getLeftPtr()
          ELSE
               currNode ← fringe.pop()
               print(currNode.getData())
               currIndex ← currNode.getRightPtr()
     UNTIL  currIndex = -1 AND fringe = empty Stack
```

Task 1.4

Write program code to:

- use the `Stack` class created in Task 1.1 and implement the `InOrderTraversal()` method within the `BinaryTree` class
- test the program with the data from Task 1.3 **[6]**

Evidence 4:

Your program code for Task 1.4 and screenshot of the test run.

~ End of Paper ~