Name:_____ **Class:**_____

JURONG PIONEER JUNIOR COLLEGE

JC2 Test 1 2025

COMPUTING **9569**

Higher 2

24 February 2025

Hybrid (Lab-based and written)

1 hour 15 minutes

READ THESE INSTRUCTIONS FIRST

Answer **all** the questions.

Section A – Lab-based

Section B – Written

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 **3** marks out of 40 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 **40**.

This document consists of **4** printed pages.

Section A – Lab-based

Instructions to candidates:

Your program code and output should be saved in a single **`.ipynb`** file using Jupyter Notebook. For example, your program code and output for Task 1 should be saved as:

`TASK1_<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:
```

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

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

A binary tree is a hierarchical structure where each node has at most two children. Your task is to implement a **dynamically linked binary tree**.

Once your tree is implemented, use it to build a postfix expression tree using the given `Stack` class. The code for the `Stack` class is saved in `Task1.txt`.

The class `TreeNode` contains three attributes:

- `value` the string value
- `left` the left child node
- `right` the right child node.

The class `TreeNode` contains the following method:

- a constructor that initialises `value`, `left`, `right` to appropriate values for an empty node.

The class `BinaryTree` contains one attribute:

- `root` the root node.

The class `BinaryTree` contains the following methods:

- a constructor that initialises `root` to appropriate value for an empty binary tree
- `build_expression_tree` that takes a string as a parameter and builds a binary tree from the postfix expression using a stack
- `postorder_traversal` that takes a node as a parameter, uses a **recursive** algorithm and prints the values in post-order traversal.

Task 1.1

Write program code to declare the class `TreeNode` and its methods. **[3]**

Task 1.2

Write program code to declare the class `BinaryTree` and its methods.

Implement the following algorithm for `build_expression_tree` method:

1. Instantiate an empty stack for storing tree nodes.

2. For each character `char` in `postfix_expr`:
   - If `char` is an operand (`0` to `9`):
     i. Instantiate a new `TreeNode` with `char` as its `value`.
     ii. Push this node onto the stack.
   - If `char` is an operator (`+`, `-`, `*`, `/`):
     i. Instantiate a new `TreeNode` with `char` as its `value`.
     ii. Pop two nodes from the stack:
     iii. The first popped node is the right child of the new node.
     iv. The second popped node is the left child of the new node.
     v. Push this new node back onto the stack.

3. After processing all characters, the stack contains only one node (root of the tree). Pop the remaining node from the stack and assign it to the `root` of the tree. **[16]**

Task 1.3

Write the main program to:
1. Instantiate a new `BinaryTree`.
2. Get user to input the postfix expression string.
3. Call `build_expression_tree` method to build the binary tree.
4. Call `postorder_traversal` method to output the values. It takes the `root` of the tree as its parameter.

Test your program using `"34+2*7-"` as the user input. **[5]**

Save your Jupyter notebook for Task 1.

Section B - Written

Submit your answers on A-4 sized writing papers.

2 The following recursive algorithm for the function `evaluate_expression_tree` takes in a root node, `node` of a tree or sub-tree as its input parameter and returns the evaluated value of the postfix expression stored in a binary tree.

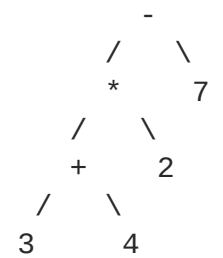
```
01  FUNCTION evaluate_expression_tree(node) RETURNS INTEGER
02      IF node = NULL THEN
03          RETURN 0
04      ENDIF
05
06      IF node.left = NULL AND node.right = NULL THEN
07          RETURN node.value
08      ENDIF
09
10      left_value ← evaluate_expression_tree(node.left)
11      right_value ← evaluate_expression_tree(node.right)
12
13      IF node.value = '+' THEN
14          RETURN left_value + right_value
15      ELSE IF node.value = '-' THEN
16          RETURN left_value - right_value
17      ELSE IF node.value = '*' THEN
18          RETURN left_value * right_value
19      ELSE IF node.value = '/' THEN
20          RETURN left_value / right_value
21      ENDIF
22  ENDFUNCTION
```

(a) State the following:

i. The definition for a recursive function. **[2]**

ii. The line number(s) of the general case(s). **[2]**

(b) The following binary expression tree stores the postfix expression `"34+2*7-"`.



i. Draw a trace table to show the step-by-step evaluation of `evaluate_expression_tree` on the given expression tree. Use the column headers `node.value`, `node.left`, `node.right`, `left_value`, `right_value`, and `RETURN`. **[6]**

ii. State the number of times `evaluate_expression_tree` called itself. **[1]**

(c) Explain why a binary search tree cannot be used to construct a postfix expression tree. **[2]**