Dr. Dapeng Dong

Maynooth University
National University of Ireland Maynooth

CS211FZ Data Structures & Algorithms (II)

Midterm Test (Repeat)
(100 Minutes)

NOTE:

- **Do NOT use "package" in your source code**
- **You must submit the source code files, i.e., the ".java" files.**
- **You are NOT allowed to search information on the Internet.**
- **You are allowed to use class notes during the test.**
- **Sharing your work with others is NOT allowed.**
- **You should NOT change the structure of the code or any other methods that have already been implemented in the source code file.**

Given an incomplete expression tree implementation ("*ExpressionTree.java*"), your tasks are to provide the missing implementations for the methods defined in the source code file.

Task 1:

Complete the implementation for the method "***postfixBuild( )***". This method builds an expression tree from a given algebra expression represented in Postfix Notation. You may refer to the following pseudocode for the implementation.

```
postfixBuild (exp)
   S = ∅
   Node opd1, opd2, currentRoot
   for i to LEN(exp)
      currentRoot = new Node(exp[ i ])
      if not operator
         PUSH(S, currentRoot)
      else
         opd1 = POP(S)
         opd2 = POP(S)
         currentRoot.left = opd2
         currentRoot.right = opd1
         PUSH(S, CurrentRoot)
   POP(S)
```

Task 2:

Complete the implementation for the method "***printInfixNotation( )***" using a recursive method. This method uses the expression tree built from the "***postfixBuild( )***" method to print the algebra expression in Infix Notation.

Task 3:

Complete the implementation for the method "***printPrefixNotation( )***" using an iterative method. This method uses the expression tree built from the "***postfixBuild( )***" method to print the algebra expression in Prefix Notation. You may refer to the following pseudocode for the implementation.

```
printPrefixNotation (currentRoot)
    if currentNode == NIL    return
    S = ∅
    PUSH(S, currentRoot)
    while S ≠ ∅
        visitingNode = POP(S)
        print visitingNode.key
        if visitingNode.rightChild ≠ NIL
            PUSH(S, visitingNode.rightChild)
        if visitingNode.leftChild ≠ NIL
            PUSH(S, visitingNode.leftChild)
```

Sample Output:

Given an algebra expression represented in postfix notation: `ab*cd*+f-a+`

the output may look like the following:

```
Infix:   a*b+c*d-f+a
Prefix:  +-+*ab*cdfa
```