CSCI 235 Spring 2020

Lab 10 **Apr. 14**

Connecting to a VNC display

You will use a lab account for a VNC display. The lab account already started a VNC session, so you don't need to do Step 3 or 4 of Lab 7's Task #1. Two steps that you need to do are:

1. Create a tunnel from your machine to a VNC display (Step 5 of Lab 7's Task #1).
   a. A cslab machine number and a display number will be provided to you.
   b. For example, if the given machine is **cslab10**, and the display number is **12**, the command for creating a tunnel to the display will be:
      - Mac Terminal:
        `ssh –L 5901:localhost:`**`5912`** *`your_loginID`*`@`**`cslab10`**`.wheaton.edu`
      - Windows PuTTY:
        Host Name: **`cslab10`**`.wheaton.edu`
        For Connection, SSH, Tunnels: `5901` for the "Source port" and `localhost:`**`5912`** for the "Destination".
   c. The CS system will prompt for your (ID and) password as usual.

2. Start a VNC viewer on your machine (Step 6 of Lab 7's Task #1).
   a. The given display's VNC password will be provided. You need to use this password to connect to the display.

Tips:

- If you don't see the menu bar at the top of the CS desktop window, right-click the desktop screen to get Applications menu.
- When you are done with the desktop, close the desktop and connection by doing:
  - For TightVNC, use the X button at the top corner of the window.
  - For Mac, in Screen Sharing menu at the top, select Connection, and select Close.

===

Recursive methods for tree traversals
Parse trees

Documentation and styling of your code will be part of your grade.

Create a new directory and cd to it. Then copy starting files from the course directory to your current directory.

```
$ cp /cslab/class/csci235/labs/lab10/* .
```

You should have four Java files and one class file (ExprStringSlicer.class) in the directory.

Task #1: Writing recursive `TreeNode` methods
`TreeNode` is a class for a binary tree, and `TreeDriver` tests the `TreeNode` class.

Open the two files, and inspect the code. Draw a diagram for the tree that will be built by the driver. Compile and run the driver.

Switch to `TreeNode.java`, and inspect the instance variables and its methods. Trees are naturally traversed in a recursive manner, and thus you will write recursive methods for TreeNode. Analyze the code of `count()`. It returns the number of nodes in the subtree rooted at the node on which it is called. Note that the number of nodes is 1 (for the current node, the calling object) plus the number of nodes in its left subtree, plus the number of nodes in its right subtree.

- It is a recursive method, but no explicit base case has been coded. What base case has been implied in the code? What value will be returned when the base case is executed?

Recall the *preorder*, *inorder*, and *postorder* traversals shown in class. Write the three, recursive display methods. Refer to the code of `count()` for different cases on a node. Update the `System.out.println()` statements in `TreeNode` so that the integers at the nodes are **printed on a single line**.

Test your recursive methods with the original driver. You may need to edit the original code.
Test the methods again with several, new trees: trees with nodes that have only a left child or only a right child.

Task #2: Parse trees
For this task, you will use the three files: ExprNode.java, ExprStringSlicer.class, and Interpreter.java. `Interpreter` is the driver, and `ExprNode` is the interface that we discussed in class. The main method of the driver calls `parse`, and its first line is as follows:

```
String nodes[] = ExprStringSlicer.slice(expr);
```

What would be the heading of the slice method? Is this a static method, or an instance method? The source code of `ExprStringSlicer` is not given, but you can still use the program because its class file is provided.

`ExprStringSlicer.slice()` is the method to get a string array for building an expression tree. This method works as follows:

- It takes a string of either an integer or (*expression op expression*) as seen in class.
- It returns an array of strings.
  - If the input string contains just an integer, the string will be referred by the first element of the string array. The array size will be 1.
  - If the input string consists of *expression*, *op*, and *expression*, the first expression will be referred by the first element, the *op* will be referred by the second element, and the second expression will be referred by the third element of the array. The array size will be 3.

For example, when `expr` is `"(42-17)"`, `nodes[0]` refers to `"42"`; `nodes[1]` refers to `"-"`; and `nodes[2]` refers to `"17"`.

- If `expr` is `"10"`, `nodes[0]` refers to `"10"`.
- If `expr` is `"((18*10)/18)"`, `nodes[0]` refers to `"(18*10)"`; `nodes[1]` refers to `"/"`; and `nodes[2]` refers to `"18"`.

Step #1: Write two classes that implement the `ExprNode` interface: one for `ConstantNode` and the other for `OperatorNode`. No starting file is provided for these classes. Refer to the example code in Exercises #7. Compile the two Java files.

Step #2: Complete the `parse()` method in `Interpreter`. The main method is complete.

- Read the block comment carefully. The returning ExprNode will be the root of the expression tree.
  - What expression? Where will it be returned? What is the variable that holds this returned value?
  - In the main, what does the statement, `expr.evaluate()` do?
- See the condition of the base case.
  - Why does it become the base case?
  - What should be assigned to `out`?

- Complete the recursive case.
  - What will be assigned to `nodes[]`? How to use the array's string objects to build the tree?
  - What should be assigned to `out`?

Hand-test your code for `parse()` by drawing a diagram for the input `"((2*5)/3)"`. If your code builds a proper tree for the input, run Interpreter.java with the input. The output should be `3`. Test your program with several, different expressions.

Important: After you email your files to the TA, delete your lab folder for the next team. Your display will be available during the scheduled lab hours only.

What to submit

- Email TreeNode, OperatorNode, ConstantNode, and Interpreter Java files (as attachments) to your TA:
  - 8:30 am session: to Drew at drew.smith@my.wheaton.edu
  - 1:15 pm session: to Brian at brian.drown@my.wheaton.edu
  - **In the subject line**, put "CSCI 235 Lab 10 (*the first initial of your first name and your last name*; *the first initial of the partner's first name and the last name*)"
    - For example, CSCI 235 Lab 10 (HKim; JSmith)