CS 474: Object Oriented Programming Languages and Environments

Fall 2022

First Ruby project

Due time: 11:59 pm on Sunday 10/2/2022

In this project you will define a simple command-line Ruby application that implements a simple *Set Calculator*. Sets must be implemented as Binary Search Trees (BST). Here are the key features of the project:

1. You must maintain up to 3 sets denoted by *X, Y* and *Z*. Each of the 3 sets must be printed in ascending order using an in-order traversal of the corresponding BST.
2. All sets hold exclusively numerical values. No duplicate elements are allowed under any circumstances.
3. You must implement simple set operations such as union, intersection, deep copying and element insertion using the BSTs.
4. You are not required to keep the BSTs balanced.
5. Your program executes a continuous command loop according to the command list below.

Recall that every node *n* in a BST has two children, subject to the property that the left child holds a value less than *n*'s value, and the right child holds a value greater than *n*'s value. In-order traversal recursively explores nodes in this order, starting from the root: (1) the left subtree of a node *n*, (2) node *n*, and (3) the right subtree of *n*.

Your submission will be graded according to the following criteria: (1) compliance with this specification, (2) the presence of code comments, and (3) conciseness (e.g., avoiding definition of unnecessary data structures).

When your project is started, all 3 sets are initially empty. Subsequently, the project will run a command loop implementing the commands below. Make sure to output the contents of sets *X, Y* and *Z* on the console after each command is executed.

- `X` *values* — This command takes a comma-separated list of numeric arguments denoted by *values*. Now sets *X* is initialized to the values specified on the command line. The previous content of *X* is lost.
- `Y` *values* — This command is similar to command *X* above, except that it resets the content of set *Y*.
- `Z` *values* — This command is similar to command *X* above, except that it resets the content of set *Z*.
- `a` *i* — Numeric argument *i* is inserted in an appropriate location in set *X*. The other two sets are not modified. Of course, insertion must preserve the BST properties of set *X*.
- `r` — The content of the 3 sets is rotated. Set *Y* takes the content of *X*; set *Z* takes the content of *Y*; and set *X* takes the previous content of *Z*. The BSTs are not modified or copied in this operation.
- `s` — The contents of sets *X* and *Y* are switched. The BSTs are not modified or copied in this operation.
- `u` — This command takes the union of sets *X* and *Y*. The result set is stored as *X*.
- `i` — This command takes the intersection of sets *X* and *Y*. The result set is stored as *X*.
- `c` — Set *X* is recursively deep copied into set *Y*. The previous content of *Y* is lost. After this command is executed, sets *X* and *Y* may not share any data structures.
- `l` *aString* — This command takes a string argument defining a one-argument lambda experssion. This lambda is applied to every element of set *X*. The result of each lambda execution is output on your console. Set *X* is not modified. You may assume that the syntax of the lambda in the string is correct.
- `p` — The content of the 3 sets is output on your console.
- `q` — This command quits your project.

You must work alone on this project. Your project code should be in a special package called CS474. Save all your code by filing out that package in the file xxx_yyy.rb, where xxx and yyy denote your first and last names. Submit the file using the submit link in the assignment page of the Blackboard course web site. No late submissions will be accepted.

Hints. Use method *eval()* to evaluate a string. For instance the following code creates a lambda from a string and evaluates the lambda, returning 30:

```
aString = "-> {a, b = 10, 20 ; c = a + b ;}"
aBlock = eval aString
aBlock.call
```