Link to Github:
https://github.com/DaridaRazvan/FLCD-Lab2

Symbol Table implemented using a binary search tree

Binary Search Tree is a node-based binary tree data structure which has the following properties:
-The left subtree of a node contains only nodes with keys lesser than the node's key.
-The right subtree of a node contains only nodes with keys greater than the node's key.
-The left and right subtree each must also be a binary search tree.

Add an element
When looking for a place to insert a new key, traverse the tree from root-to-leaf, making comparisons to keys stored in the tree's nodes and deciding based on the comparison to continue searching in the left or right subtrees. In other words, we examine the root and recursively insert the new node to the left subtree if its key is less than that of the root or the right subtree if its key is greater than or equal to the root.

Search an element
What we'll do is we'll start at the root, and then we will compare the value to be searched with the value of the root if it's equal we are done with the search if it's lesser we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are lesser and all the elements in the right subtree are greater. Searching an element in the binary search tree is basically this traversal in which at each step we will go either towards left or right and hence in at each step we discard one of the sub-trees.

Program Internal Form implemented with a List of a pair of 2 integers

At the start of the program identifiers get value 0, constants 1 and each reserved word, separator and operator gets a value starting from 2 forward. They are added in a codification list.

We read a program from a text file line by line and separate it based on the list of separators using a StringTokenizer then check for each separate token if its either a reserved word, separator, operator, identifier or constant. The reserved words, separators and operators are added to the program internal form with a value of -1.

In the case of identifiers and constants, we add them in the program internal form with the position they have in the symbol table after we do an in order traversal of the binary search tree.

EBNF Fa.in

<letter> = "a" | "b" |...| "z" | "A" | ... | "Z"
<digit> = "0" | "1" |...| "9"

<state> = <letter>
<alphabet> = <digit>
initialState = <letter>
<finalState> = <letter>
<transition> = <letter> <digit> <letter>