Scanner documentation

1. Presentation of the project requirements

Based on the specification of the mini-language chosen I implemented a scanner that will take as input a text file containing the source program and will produce as output the following:

* PIF -> Program Internal Form
* ST -> Symbol Table

In addition, the program should be able to determine the lexical errors, specifying the location, and if possible, the type of the error.

Some additional restrictions:

1. length at most 8 characters
2. unique for identifiers and constants
3. lexicographically sorted binary search tree
4. Used technologies & Data structures

I chose to use Python as my programming language of choice for this laboratory and PyCharm as IDE.

Regarding the data structures used in the application I had to use a lexicographically sorted binary search tree for my ST (Symbol Table) representation. I chose to implement the binary search tree over an array such that the children of the element found at position i are 2\*i and 2\*i + 1. This implementation helped me achieve an unchanging position for every element in the tree, which was necessary for the construction of the PIF (Program Internal Form).

I declared my PIF (Program Internal Form) as a list of tuples, each element of the list is a tuple of (code, id) form. The code for separators/operators/reserved words is the value from its corresponding key in the codification table and for identifiers and constants it is the position in the symbol table. The id is either -1 for separators/operators/reserved words, 0 for identifier or 1 for constant.

The special symbols such as separators, operators and reserved words are represented in a list.

The codification table is a dictionary in which the key is either a separator / operator / reserved word or an identifier/constant and the value is a natural number greater than 1 for the first category, 0 for a identifier and 1 for a constant .

1. Implementation description

I started reading my language instructions from the file line by line then I continue by splitting the line in tokens using the function tokenGenerator which will give constantly one token until the line ends.

If the given token is an operator / separator / reserved word I added to the PIF with code -1. Next I checked if the token is an identifier or a constant and in both cases we add it in the corresponding ST (or search for it). Then, with the position known, I added in the PIF and continued with the next token.