Minilanguage Scanner

1. Project requirements

   The minilanguage scanner will take as input a file containing code written in the language that I've specified for the first part of the laboratory. The program will produce an output containing the symbol table and the program internal form.

   Inside the program, the operators, separators, reserved keywords and the codification table are also specified.

   Restrictions imposed at the laboratory:

   - Identifiers can have a length of at most 8 characters
   - The symbol table has to be unique for both identifiers and constants
   - The symbol table has to be lexicographically sorted

2. Used technologies

   I have implemented the scanner in Python 3, using PyCharm as my IDE.

3. Used data structures

   The codification table is declared as a dictionary and contains the IDs of each reserved keyword, separator and operator used in the language. It also contains the IDs of constants and identifiers.

   The symbol table is a list that contains tuples consisting of the identifier/constant that needs to be added to the list and a globally unique identifier that represents that identifier/constant. The list gets sorted every time after adding a new element to the list, and that is why adding the GUID was necessary, so we don't lose the location of the first element of the pair.

   The program internal form is a list that consists of tuples in which the first element is the ID of the element that we've just parsed in the codification table, and the second element is the GUID returned by adding the element to the symbol table.

4. Algorithm

   The scanning algorithm is as follows: for all tokens in the file, check if they are an operator, separator or reserved keyword. If so, it adds a tuple to the program internal form consisting of the codificationID of the token and the value -1, since we do not add that token to the symbol table.

   If the token is an identifier, add the identifier to the symbol table and insert a tuple in the PIF containing the codification ID of an identifier and the ID of the identifier in the symbol table.

   If the token is a constant, add the constant to the symbol table and insert a tuple in the PIF containing the codification ID of a constant and the ID of the constant in the symbol table.

   Last but not least, if the token is none of the above, it means there's an error in the source code.

5. Regular expressions
    a. ^(0|[+-]?[1-9][0-9]*) - checks if the token is 0, or begins with +/- any number from 1-9 (first character) and then (0-9), since we cannot have signed numbers with trailing 0s

    b. ^\".*\"$ - token begins with “ and ends with “ (this checks if the token is a string) Note: strings cannot have quotes inside them (not handled by this regular expression)

    c. ^[a-zA-Z]([a-zA-Z]|[0-9]|_){,7}$ - this checks if the given token is an identifier. Rules: first character is a-z/A-Z, then there can be any combination of letters, numbers, or _, and the total length of the identifier cannot be more than 8