GitHub link: https://github.com/914-Mihalescu-Valentina/LFTC/tree/main/Assignment4

Analysis: The Scanner should be able to read a given program line by line, split the lines into correct tokens, classify them into the five categories (operators, separators, reserved words, identifier, constants), and if a token is not part of those five categories, print that there is a lexical error on that line number. After that, there should be the codify part, where the tokens get placed into the program internal form (PIF) and symbol table (ST).

Implementation: the algorithm implements the splitting into tokens (detection), classifying and codifying. Splitting is done by using java split function on space as delimiter, with special care for the cases where there are more tokens not delimited by space (ex: x=3;) and for the case of string constants, to not split the actual content of the strings by space. The tokens for classifying are read from the file token.in, with the format “token_category:token1,token2,…,” (ex: “separator:[,{,(,}” “operator:+,-,==,<=”). For identifiers and constants, instead of actual tokens, there are regex rules for identifying if a token is an identifier or constant. Constants include regex for numerical constants as well as for string constants. Codifying makes use of the symbol table done previously. The PIF is a list of pairs of pairs, where first pair contains the token and the pair of positions, where position is -1,-1 if the token is not a symbol and the position in the hash table and linked list from the symbol table, if it is a symbol.

The error in my program perr consists of the appearance of $ and || which are not accepted by my language

I have a separate class for pif which is in fact based on a pif field which is a list of tuples and I have some important functions in my Scanner class:

Def scan(self):

Def codify(self,token):

Def classify(self,token):

Def detect(self,input_string):

Def is_constant(self,token)

Def is_identifier(self,token):

My program is also regex based

The testing code has 4 main cases: 3 successful cases each corresponding to one correct input program and 1 corresponding to the error program