Lexical Analyzer

<u>Requirement</u>

Considering a small programming language (that we shall call mini-language), you have to write a scanner (lexical analyzer).
The mini-language should be a restricted form of a known programming language, and should contain the following:
- 2 simple data types and a user-defined type
- statements:
    - assignment
    - input/output
    - conditional
    - loop

The scanner input will be a text file containing the source program, and will produce as output the following:
- PIF - Program Internal Form
- ST - Symbol Table

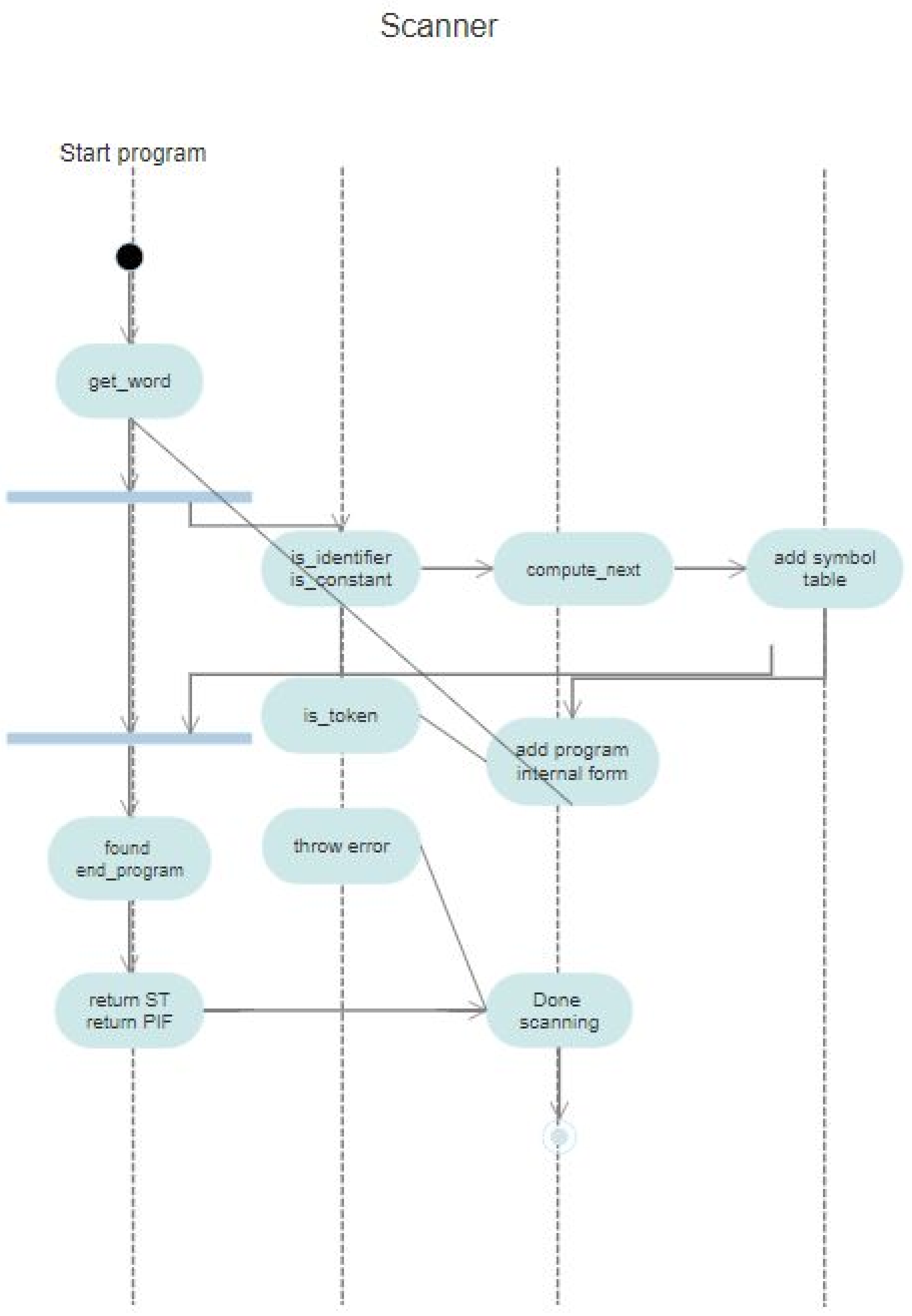
In addition, the program should be able to determine the lexical errors, specifying the location, and, if possible, the type of the error.

The scanner assignment will be differentiated based on:
1. Identifiers:
    - length at most 8 characters
2. Symbol Table:
    - unique for identifiers and constants
3. Symbol Table Organization:
    - lexicographically sorted table

<u>Solution</u>

We have two data structures that help us keeping track of all the variables or commands to be done in our mini-language.
The first one is the symbol table which has together the identifiers and the constants. If it is something else than an identifier or a constant, we check if it is in the codification table, table that has all the possible commands to be executed or the operators.
If it is not an identifier, constant and it is not in the codification table, there is a lexical error.
If there is an identifier with the length greater than 8, then there is a lexical error.
If the identifier is not created as stated in the specifications, or some strings do not end with apostrophes or a char has more that one character, there is a lexical error.
When encountering an identifier or a constant, we add it in the symbol table with the its id as the length of the symbol table at that time. Then, being a lexicographically sorted table, we have to add it where it should be, thus we have a function that looks for the smallest

lexicographically word greater that the word we want to add. We set the ‘next’ value of the current word as the id of the word that we found, and we iterate over the entries from the symbol table to see what word had the ‘next’ value as the id that we found at the previous step, and if found, change its ‘next’ value as the id of the last inserted word.
When we add a word into the program internal form table, if it is an identifier or a constant, we add its id from the symbol table, otherwise we set its id as ‘-1’.


Scanner
Start program
get_word
is_identifier
is_constant
compute_next
add symbol table
is_token
add program internal form
found end_program
throw error
return ST
return PIF
Done scanning