Formal Languages and Compiler Design

Lab 3 – Documentation

Ariadna Vilasó Escarp

Link to the repository:

https://github.com/UO270119/Formal-Languages-and-Compiler-Design

Problem statement:

Implement a scanner (lexical analyzer): Implement the scanning algorithm and use ST from lab 2 for the symbol table.

Details:

• ST.out should give information about the data structure used in representation

• If there exists an error the program should give a description and the location (line and token)

For this project we use two classes: *LexicalScanner.java* and *HashTable.java.*

LexicalScanner: It will process a given file, line by line, and it will read the tokens.

If the token is either a reserved word (empieza, acaba), an operator (<, <=, >, >=) or a separator ((,), [,],), it will be added as a new row to the PIF table along its position.

If the token is an identifier or a constant, it will be added as a new row to the Symbol Table (ST) along its position.

If the scanner detects anything else it will be considered a lexical error.

Its most important methods are:

- *public void scan():* It reads the given file line by line, separating first the words with separators. Then, if it doesn't have a separator and the string or char is not lexically correct, it will be created as a stringConstant or charConstant. If it doesn't have a separator and the string or char is lexically correct it will be added to the list of detected tokens.
- *private void readTokens()*: Its reads the file line by line and adds the tokens to the tokenList.
- *private void readSeparators():* It reads the file line by line and adds the separators to the separatorList.
- *public void classifyTokens():* It will generate the PIF.out, and ST.out, classifying the detected tokens in two groups. If it detects reserved words, operators or separators, it will add them to the PIF table. If it detects identifiers and constants, it will insert them in the HashTable (ST). Anything else will be considered a lexical error. If there are some string or chars containing lexical errors due to some missing quotes (simple or double) it will also indicate those.
- *public void writeToSymbolTable():* It will read the contents of the HashTable and convert it to the output SymbolTable.

- *private void splitWordWithSeparator (String word, String separator, Integer line):* It will create an array of strings with the separated list of words. It reads word by word looking for separators (which will be considered special cases) and converts them to string or char constants.
  It will be considered a special case if it detects a separator or if strings or chars are not lexically correct, and in that case, it will be created as a stringConstant or charConstant. If it's not an special case, it will split the previously created array and add everything to the list of detected tokens depending on its length.

HashTable: It's the same class used for Lab 2. To represent the Symbol Table I chose to use the HashTable structure. The possible collisions will be solved using a linked list. The hash function we will use will be computed calculating the modulus of the ASCII code of the character from the corresponding token, divided by the capacity, which is the size of the token list.

Its methods are:

- *private int hashingFunction (String identifier):* Calculates the hashing function (explained above).
- *public boolean insert (String identifier):* Inserts a token in the symbol table while solving possible collisions.