Formal Languages and Compiler Design – Lab3: Scanner Implementation

Link to Github Repository: https://github.com/912-Crismariu-Ioana/FLCD

Program Scanner class

Fields:

- **symbolTable**: an instance of the SymbolTable class
  (implementation provided in the previous lab)
- **classifier**: an instance of Classifier, a class that stores lists of predefined symbols accepted by the language (separators, operators, reserved keywords)
- **pif**: an instance of the ProgramInternalForm class, a wrapper for the list of pairs of codified tokens and their unique identifier/position in the symbol table
- **programPath**: the path to the file storing the input program
- **PIFPath**: the path to the output file for the representation of the Program Internal Form
- **STPath**: the path to the output file for the representation of the Symbol Table

Methods & general flow of the program:

- **scan()**: method that implements the scanning algorithm
  First the predefined symbol types are loaded from the token input file into the Classifier.
  The program input file is parsed and tokenized, and afterwards lexical analysis is performed.
  Empty lines and lines starting with the '#' character are ignored by the scanner.
  If the input program contains unclassifiable symbols this is considered a lexical error.
  The scanner is stopped and a message containing the line number and the symbol that caused the error is displayed.
  If the program is lexically correct, the scanner also outputs a confirmation message.
  The generated Symbol Table and Program Internal Form are written to the output files.
- **token()**: method that tokenizes a line of the input file
  The input line is parsed character by character.
  Whenever a character representing a possible delimiter of a known symbol type is encountered, the input line is truncated and passed to a method adapted to read that specific type of token. We have 5 such methods: **extractIntegerConst**, **extractStringConst**, **extractCharConst**, **extractOperator** and the more generic **extractToken**. These methods (aside from extractOperator) do not perform a proper classification of the token. They only parse until a different type of delimiter is encountered and extract the result.
- **classify()**: method performing the actual classification
  This method is used to check the previously extracted tokens against known predefined symbols and symbol patterns, by means of the Classifier.
  It is also responsible for building the PIF by adding validated symbols to it and completing the symbol table with newly introduced, valid symbols.