The lexical analyzer is a fundamental component in the compilation process of programming languages. The primary purpose of the lexical analyzer is to break down source code into tokens, and to categorize these tokens into different classes or categories. These tokens may include identifiers, keywords, constants, operators, and separators.

In addition to tokenization and categorization, the lexical analyzer constructs a Program Internal Form (PIF), which is a data structure that associates tokens with their positions in the symbol table. The ProgramInternalForm class is designed to maintain a list of pairs, where the first element of the pair represents the token, and the second element of the pair represents the position of the token in the symbol table, if that token is either an identifier or a constant.

Here are the main functions used in the LexicalAnalyzer class and their specification:

- “detect” method
- Parameters:
  line - A string representing a line of source code.
- Returns:
  A list of tokens extracted from the input line.
- Description:
  The detect method parses the input line character by character, identifying and categorizing tokens. It recognizes string and character constants by detecting the opening and closing quotes. It handles multi-character operators by checking thenext character. When it encounters separators, operators, or spaces, it adds the current token to the list and starts constructing a new token. The method ensuresthat the list of tokens does not contain unnecessary whitespace characters.

- “classify” method
- Parameters:
  token - A string representing a token to be categorized.
- Returns:
  An integer representing the category of the token.
- Description:
  The classify method analyzes the provided token based on predefined criteria. Itassigns a category to the token and returns the category code. If the token doesn't match any known category, the method raises a LexicalErrorException to handle unrecognized tokens.

- “codify” method
- Parameters:
  token - A string representing the token to be added to the PIF and ST.
  category - An integer representing the category of the token.
- Description:

The codify method constructs a pair consisting of the token and its category and a pair representing the position of the token in the symbol table. For constants and identifiers, it inserts the token into the symbol table and retrieves its position. Finally, it adds the token, its category, and position to the PIF for further processing.

- “scan” method
- Description:

  The scan method reads the input source code line by line, tokenizes each line, categorizes the tokens using the classify method, and adds them to the PIF and ST using the codify method. It handles any encountered lexical errors by printing error messages and continues scanning the remaining code. After scanning the entire input, it outputs the Symbol Table to the specified file and the Program Internal Form to another file. If no lexical errors are encountered, it displays a message indicating that the program is lexically correct.

- “isIdentifier” method
- Parameters:

  token - a string representing a detected token that must be classified
- Returns:

  true if the token can be classified as an identifier, false if not
- Description:

  The regular expression "([a-zA-Z_][a-zA-Z0-9_]*)" is used to check if a string is a valid identifier.
  It consists of two parts: [a-zA-Z_] and [a-zA-Z0-9_]*.
  [a-zA-Z_] matches the first character, which must be a letter (uppercase or lowercase) or an underscore _.
  [a-zA-Z0-9_]* matches zero or more characters that can be letters, digits, or underscores.

- “isIntegerConstant” method
- Parameters:

  token - a string representing a detected token that must be classified
- Returns:

  true if the token can be classified as an integer constant, false if not
- Description:

  The regular expression "0|([+-]?[1-9][0-9]*)" is used to check if a string is a valid integer constant.
  It consists of two alternatives: 0 and ([+-]?[1-9][0-
  9]*).0 matches the integer literal 0.
  ([+-]?[1-9][0-9]*) matches a number that can be preceded by an optional positiveor negative sign (+ or -).
  [1-9] matches the first digit, which must be a non-zero
  digit.[0-9]* matches zero or more additional digits.

- “isCharConstant” method
- Parameters:
    token - a string representing a detected token that must be classified
- Returns:
    true if the token can be classified as a char constant, false if not
- Description:
    The regular expression "'[a-zA-Z0-9]'" is used to check if a string is a valid character constant enclosed in single quotes.
    '[a-zA-Z0-9]' matches a single character enclosed in single quotes. The character can be a letter (uppercase or lowercase) or a digit.

- “isStringConstant” method
- Parameters:
    token - a string representing a detected token that must be classified
- Returns:
    true if the token can be classified as a string constant, false if not
- Description:
    The regular expression "\"([a-zA-Z0-9 ]+)\"|\"\"" is used to check if a string is a valid string constant enclosed in double quotes.
    \"([a-zA-Z0-9 ]+)\" matches a string of one or more characters (letters, digits, or spaces) enclosed in double quotes.
    \"\" matches an empty string enclosed in double quotes.

- “isBooleanConstant” method
- Parameters:
    token - a string representing a detected token that must be classified
- Returns:
    true if the token can be classified as a boolean constant, false if not
- Description:
    The regular expression "true|false" is used to check if a string is a valid boolean constant.
    It matches either the word "true" or the word "false". The regex was not really needed in this case, but it was used for the sake of consistency.

From the point of view of the ProgramInternalForm class, there are only 2 methods used, and are quite straight forward:

- “add” method
- Parameters:
    token - a string representing a detected and classified token
    symbolTablePosition - a pair of integers representing the position of the token inthe symbol table

- Description:

  This method is used to add a pair to the array. It takes two pairs as arguments: one for the token and its associated code, and another for the positions or indices in a symbol table.

- “toString” method
- Description:

  This method creates a string representation of the array by iterating over each pair and appending them to a StringBuilder, resulting in a formatted string representation of the PIF.

Here is the class diagram of the project, created using LucidChart application: