Lab 2 - 4

Link Github: https://github.com/913-groza-vlad/Formal-Languages-and-CompilerDesign/tree/main/Lab2-4

My Symbol Table is implemented using a hash table, which can be used as separate tables for identifiers and constants for MyScanner implementation.

The SymbolTable class is represented as a list (having a specified size) of lists of Strings (these are the symbolic names), the hash table, and it has the following methods:

- the constructor initialize the symbol table with a given size

- private int hash(String key):

This method generates a hash value for a given key/symbolic name by computing the sum of the ASCII codes

of chars of the key in order to place the symbolic name in a specific List in the table and to be easily accessible; this is achieved applying modulo size to the sum of ASCII codes.

- public boolean insertSymbolicName(String key):

It inserts a symbolic name into the symbol table and returns true if the insertion is successful and false if the key already exists in the symbol table.

- public boolean containsSymbolicName(String key):

Checks if a given symbolic name exists in the symbol table and returns true if so, false otherwise.

- public Pair<Integer, Integer> searchPosition(String key):

The method 'searchPosition' searches for the position (index) of a symbolic name in the symbol table.

It returns a Pair containing the position of the list in the table and the index in that list at which the key is found, or (-1, -1) if the name is not found in the table.

- public int getSize():

Getter method for the size field (it returns the size of the symbol table)

- public String toString():

This method provides a string representation of the symbol table for printing

The Pair<K, V> generic class associates two fields of types K and V and we use it in order to retrieve the position of a symbolic table which is a pair of two integers:

the list position in the table and the index in this list where a symbolic name can be found.

Using the toString() method, a pair can be represented as: (key, value);
MyScanner is a class that has the functionality of scanning and processing an input program (given as a text file) in order to identify and classify tokens based on the predefined mini-language rules. It is used to perform lexical analysis.

The MyScanner class has the following properties:

lang : An instance of the Language class, which defines the language-specific rules and characteristics.

identifiersSymbolTable : A symbol table to store and manage identifiers encountered during scanning.

constantsSymbolTable : A symbol table to store and manage constants encountered during scanning.

pif : An instance of the PIF (Program Internal Form) class, used to record the identified tokens.

- the constructor initializes the MyScanner class with file paths for: input source code (program), a file for writing the Program Internal Form (PIFFile), and a file for writing symbol tables (symbolTableFile) and also the language definition, identifiers symbol table, constants symbol table, and the PIF.

- public void scan():

The scan method performs the process of scanning the input source code to tokenize it and write the results to PIF and symbol tables.

- public void scanningAlgorithm(List<Pair<String, Integer>> tokens):

This method processes and classifies tokens based on the language definition, updating the PIF and symbol tables as appropriate. It iterates through the list of token pairs. Checks if each token is a reserved word, operator, or separator according to the language definition and updates the PIF. Identifies and handles identifiers and constants, adding them to the respective symbol tables and updating the PIF. Identifies and handles invalid tokens, marking them as lexical errors. Prints a message if the program is lexically correct or if there are lexical errors.

- public void writeResults():
It writes the content of the PIF to the specified PIF file and the content of the identifiers and constants symbol tables to the specified symbol table file.

- public List<String> splitLine(String line):
The splitLine method tokenizes a line of code based on the language definition and returns a list of tokens.

- public String identifyToken(String line, int position):
It identifies and returns a token (excluding operators and separators) from a given position in a line of code.

- public String identifyOperator(String line, int position):
Identifies and returns an operator token from a given position in a line of code.

- public String identifyStringConstant(String line, int position):
This method identifies and returns a string constant token from a given position.

- public String identifyNegativeNrToken(String line, int position):
It identifies and returns a negative number token from a given position in a line of the program.

PIF class is used to represent the Program Internal Form as a list of pairs of the token (a string) and its position in symbol table (a pair of two integers). It contains the add method in order to add the the program pif all its pairs of token and positions and the toString method for displaying the pif.

Language class defines the rules and characteristics of my specific programming language, providing methods and data structures for identifying and categorizing different language elements such as reserved words, separators, operators, identifiers, and constants.

Methods in class Language:

-public boolean isReservedWord(String word):
Checks if a given word is a reserved word in the language by searching within the reservedWords list.

- public boolean isSeparator(String word):
Checks if a given word is a separator in the language by searching within the separators list. Separators are characters or strings used to separate different language elements.

-public boolean isOperator(String word):
Checks if a given word is an operator in the language by searching within the operators list. Operators are symbols used to perform various operations or comparisons.

-public boolean isIdentifier(String token):
Determines if a given token is an identifier, which is a string adhering to a specific pattern defined in the method. In this pattern, an identifier starts with a letter followed by letters or digits.

-public boolean isConstant(String token):
Determines if a given token is a constant, which can be either a numeric constant (integer or negative integer) or a string constant (enclosed in backticks). It checks the provided token against regular expressions for numeric and string constants.