Github repo: https://github.com/915-Petruta-Razvan/LFTC_Labs

Documentation

The Symbol Table is composed of 3 separate hash tables:
- for identifiers
- for int constants
- for string constants

Hash Table

Details: the implementation of this data structure is generic. A hash table has a predefined size (capacity) and is built as a list of lists, such that an element in the list has 2 indices, first is the index of the list in which the element is stored and the second index represents the index in that list.
In order to determine the first index, we use a hash function for the elements depending on the type of the element:
- for integers the hash is the size % element
- for strings (that includes identifiers as well) the hash is (the sum of ASCII code of every character) % size

After we determine the first index, we push the element in the list corresponding to that list

Methods:
- Hash(key: int): computes the hash index for an int element (as specified above)
- Hash(key: string): computes the hash index for an string element (as specified above)
- Add(key: T): adds an element to the hash table and returns its position if the operation is successful, otherwise it throws an exception
- Contains(key: T): checks if the element is present in the hash table and returns true, otherwise it returns false
- GetPosition(key: T): checks if the element is present in the hash table and returns its position, otherwise it returns (-1, -1)
- ToString(): overrides the ToString method and adapts it to a nicer format

Symbol Table

Details: contains 3 hash tables that will be initialised with a predefined capacity: for identifiers, int constants and string constants.

Methods:
- AddIdentifier(name: string): adds an identifier to the identifiers hash

table and returns its position or throws an exception if the operation could not be completed
- AddIntConstant(constant: int): adds an int constant to the int constants hash table and returns its position or throws an exception if the operation could not be completed
- AddStringConstant(constant: string): adds an string constant to the string constants hash table and returns its position or throws an exception if the operation could not be completed
- HasIdentifier(name: string): checks if an identifier is present in the identifiers hash table
- HasIntConstant(constant: int): checks if an int constant is present in the int constants hash table
- HasStringConstant(constant: string): checks if a string constant is present in the string constants hash table
- GetIdentifierPosition(name: string): returns the position of an identifier in the identifiers hash table
- GetIntConstantPosition(constant: int): returns the position of an int constant in the int constants hash table
- GetStringConstantPosition(constant: string): returns the position of a string constant in the string constants hash table
- ToString(): overrides the ToString function and adapts it to a nicer format, in which we can see each hash table

Program Internal Form

Details: contains a list of pairs of the form <string, position_in_symbol_table>, where the string is represented by a keyword, separator, operator, identifier, int constant or string constant, and the position is also a pair of 2 integers (as a hash table is implemented as a list of lists).

Methods:
- AddElement(element: pair<string, pair<int, int>>): adds an element to the PIF list.

Scanner Exception

Details: a custom exception build for the scanner in the case of a lexical error.

Scanner

Details: it is responsible for lexically analysing a program. It builds the program internal form and the symbol table, or, if there is an error, it signals the line and index of the error.

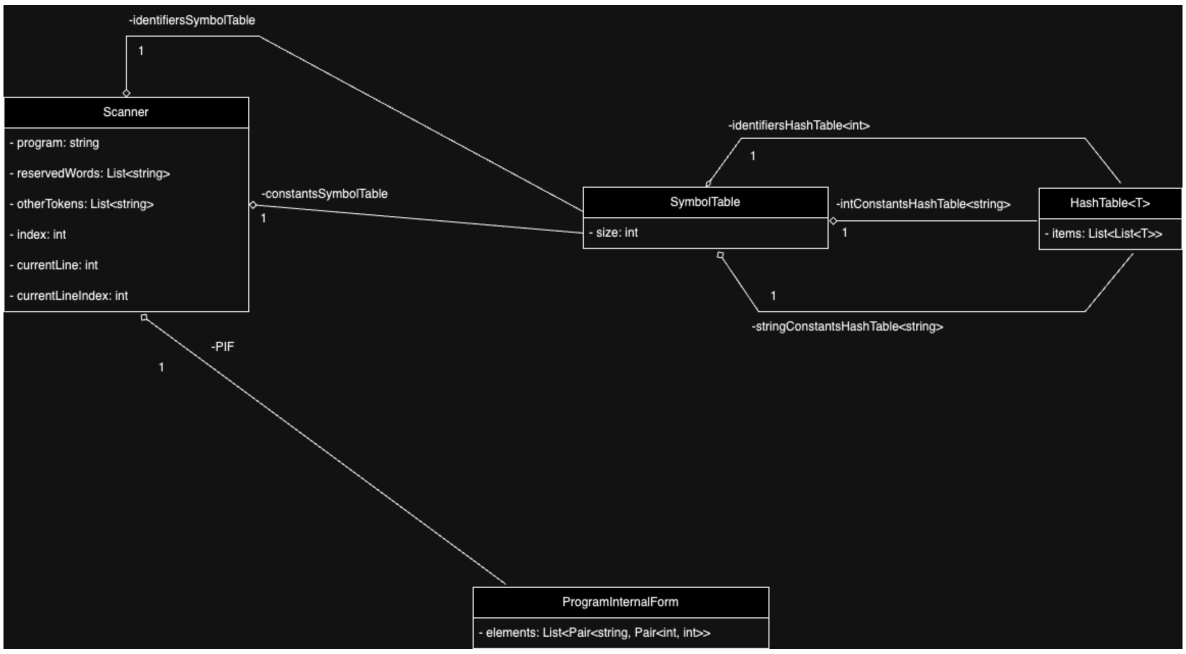
Fields:

- program: the whole program as a string
- reserverWords: includes the reserved words of our language
- otherTokens: includes the separators and operators
- identifiersSymbolTable: symbol table used for storing the identifiers
- constantsSymbolTable: symbol table used for storing int constants and string constants
- index: the current index in the program
- currentLine: the current line in the program, used to specify on which line an error might occur
- currentLineIndex: the current index on the current line, used the starting position on the line with an error

Methods:

- Scan(pathToProgram): stores the program in memory from the file specified by its path, takes token by token (from index 0 until the end of the program), classifies it and codifies it in program internal form.
- ResetTokens(): resets the symbol tables, the indexes and the current line
- ReadTokens(): reads the tokens from the file "tokens.in" and classifies them as keywords or other tokens (separators and operators)
- NextToken(): skips the whitespaces and then checks the token for possible classes. If there is no matching class for our token, it means we have a lexical error
- SkipWhitespaces(): skips the whitespaces and updates the currentLine and currentIndexLine accordingly
- CheckTokens(): checks if the current token is a keyword, separator or operator
- CheckIdentifiers(): checks if the token is an identifier. The regex follows the rules specified in our mini language specification, it needs to start with a letter or '_', followed by any number of letters, digits or '_' characters
- CheckStringConstant(): checks if the current token is a string constant. The regex follows the rule specified in the mini language, which is any number of characters enclosed between " and " characters. We also check if we encountered an " we also need the enclosing one, otherwise we have a lexical error
- CheckIntConstant(): checks if the current token is an int constant. The regex respects the rule specified in the mini language, which is 0, or we can have the sign present or not followed by a non zero digit and any number of digits afterwards. We can also check that it's not a trial for an identifier (it cannot start with a digit)

Class Diagram


-identifiersSymbolTable
1
Scanner
- program: string
- reservedWords: List<string>
- otherTokens: List<string>
- index: int
- currentLine: int
- currentLineIndex: int
-constantsSymbolTable
1
-PIF
1
-identifiersHashTable<int>
1
SymbolTable
- size: int
-intConstantsHashTable<string>
1
HashTable<T>
- items: List<List<T>>
1
-stringConstantsHashTable<string>
ProgramInternalForm
- elements: List<Pair<string, Pair<int, int>>