Github Link: https://github.com/916RasnitaRadu/SEM5-FLCD/tree/main/Lab3

Documentation

Symbol Table

For my Symbol Table I chose to implement 3 separate hash tables, one for identifiers and two for constants (string and integer constants). The Symbol Table takes as parameter a number ‘size’ which will be the size of each hash table.

The hash table class is generic, and the generic type represents the data type of the object that's being inserted into the table (in this case String or Integer). The hash table is represented as an ArrayList in which on every position we have another ArrayList, in order to be able to store the values that have the same hash on the same position.

An element from the hash table has as position a pair of indices, where the first one is the position in the table (the hash value) and the second one is the actual position in the list. The hash function has two implementations, one for when the key is a string and one for integers keys. The hash function for the integers is the given number modulo the size of the list and for string keys is the sum of the ASCII codes of the characters modulo the size of the list.

Operations:

Hash Table: (T denotes the generic type)

- getSize( ) : int – returns the size of the hash table
- add(key : T) : Pair<int, int> - computes the hash value of the key, adds the key to the table and returns the position where the element was added if the operations is successful; otherwise throws an Exception (operation is not successful when the key is already taken)
- getPosition(item : T) : Pair<int, int> - returns the position of the item in the hash table, if it exists; other wise returns a pair of form (-1,-1);
- contains(item : T) : Boolean – returns true if the table contains the given parameter, false otherwise
- hash(key : int) : int – function for computing the hash value for integer constants; computes the position of the list in the table where the int constant will be added
- hash(key : string) : int – function for computing the hash value for string constants and identifiers; computes the position of the list in the table where the element will be added
- getHashValue(key : T) : int – return the corresponding position in the table and calls the corresponding hash function according to the type of parameter ‘key’
- toString( ) : String – returns the string form of the hash table; how the table will look when it will be printed

Symbol Table:

- addIdentifier(key : String) : Pair<int, int> - adds an identifier and returns its position in the Symbol Table; throws exception if the operation fails (same as the add exception)

- addStringConstant(constant : String) : Pair<int, int> - adds a string constant and returns its position in the Symbol Table; throws exception if the operation fails (same as the add exception)
- addIntConstant(constant: int) : Pair<int, int> - adds an integer constant and returns its position in the Symbol Table; throws exception if the operation fails (same as the add exception)
- hasIdentifier(key : String) : boolean – return true if the given parameter is in the ST, false if not
- hasStringConstant(constant : String) : boolean – return true if the given parameter is in the ST, false if not
- hasIntConstant(constant: int) : boolean – return true if the given parameter is in the ST, false if not
- getPosIdentifier(key : String) : Pair<int, int> - returns position of the identifier in the ST
- getPosStringConstant(constant : String) : Pair<int, int> - returns position of the identifier in the ST
- getPostIntConstant(constant : int) : Pair<int, int> - returns position of the identifier in the ST
- toString( ) : String – returns the string form of the symbol table; how the table will look when it will be printed

Scanner

MyScanner class checks if the input program is lexically correct or not. If the program is lexically correct it will write in one file the program internal form and in another one the symbol table, and it will display the message “lexically correct”. Otherwise, it will display the message “lexically incorrect”, along with the line and the index where the mistake is made. The program internal form and symbol table files are also created but they are useless, since the program is not correctly written.

The symbol table is the one implemented in Lab2.

The program internal form is simply a list of pairs composed of a string and the position in the symbol table (i.e. another pair of 2 integers). The string in the pair refers to what token, identifier or constant we store in the program internal form. If we want to store a token we put the string as it is, for an identifier we put “id” and for a constant “const”.

For the tokens (reserved words, operators, separators), the position in the symbol table is considered to be (-1, -1).

Other fields in the MyScanner class:

- reservedWords: the list of reserved words; (eg. “start”, “integer” etc.)
- otherTokens: list of possible other tokens such as operators and separators
- index: the current index (character) in the program
- currentLine: the current line in the program
- 3 regular expressions (regex) for constants (integer and string) and identifiers

Operations:

- setProgram(program: String) : void – setter for the program field
- treatSpaces( ) : void – function that skips spaces in the program string, i.e. the current index is increasing if it encounters a space, also if there are any new lines it increases the current line
- nextToken( ) : void – treats the current program string and identifies the next token, calls one of the functions treatIntegerConstant, treatStringConstant, treatFromTokenList, treatIdentifier; if one returns true, the function nextToken returns; if neither returns, an exception will be thrown stating that there is a lexical error
- treatIntegerConstant( ) : Boolean – method that checks if the next possible token in the program is an int constant with the help of the int regex. It adds it to the symbol table and to the program internal form if it's a valid number and returns true, otherwise returns false
- treatStringConstant( ) : Boolean - method that checks if the next possible token in the program is a string constant with the help of the string regex. It adds it to the symbol table and to the program internal form if it's a valid string const and returns true, otherwise returns false
- treatIdentifier() : Boolean - method that checks if the next possible token in the program is an identifier with the help of the identifier regex. It adds it to the symbol table and to the program internal form if it's a valid string const and returns true, otherwise returns false
- treatFromTokenList() : Boolean - method that checks if the next possible token in the program is a reserved word, operator and separator, and if it is, it adds it to the program interal form and returns true, otherwise it returns false
- scan(filename : String) : void – function that gets as a parameter the name of the program file, it reads the file and sets the "program" field in the class. After that, it reinitializes the index, currentLine, symbol table and program internal form and parses the string, searching for tokens using the nextToken() function. When the end of the file is reached, it writes the in 2 separated files the program internal form and the symbol table;