https://github.com/911-Calota-Ovidiu/FLCD

Documentation

The Symbol Table consists of three hash tables: one for identifiers, one for integer constants, and one for string constants. Each hash table is represented as a list of lists to handle collisions. The tables have a fix ed size. Elements in the symbol table are identified by pairs of indices: the first index points to the list, and the second index is the position within that list.

For integer values, the hash function is value modulo the list size. For string constants and identifiers, it's t he sum of ASCII codes of characters modulo the list size. The hash table implementation is generic.

Operations

Hash table:

- hash(key: int): int – computes the position in the ST of the list in which the integer constant will be added

- hash(key: string): int – computes the position in the ST of the list in which the string constant/identifier wi ll be added, based on the sum of the ASCII codes of their characters

- getSize(): int – return the size of the hash table

- getHashValue(key: T): int – return the corresponding position in the ST according to the type of the para meter 'key'

- add(key: T): (int, int) – add the key to the hash table and return its position if the operation is successful; otherwise, throw an exception

- contains(key: T): boolean – return if the given key is in the hash table or not

- getPosition(key: T): (int, int) – return the position in the ST of the given key, if it exists; otherwise, return (-1, -1)

- toString() (overridden method) – return the string representation of the hash table

Symbol table

- has 3 hash tables: for identifiers, for string constants and for integer constants

- addIdentifier(name: string): (int, int) – add an identifier and return its position in the ST

- addIntConstant(constant: int): (int, int) – add an integer constant and return its position in the ST

- addStringConstant(constant: string): (int, int) – add an string constant and return its position in the ST

- hasIdentifier(name: string): boolean – return if the given identifier is in the ST or not

- hasIntConstant(constant: int): boolean – return if the given integer constant is in the ST or not

- hasStringConstant(constant: string): boolean – return if the given string constant is in the ST or not

- getPositionIdentifier(name: string): (int, int) – get the position of the identifier in the ST

- getPositionIntConstant(constant: int): (int, int) – get the position of the integer constant in the ST

- getPositionStringConstant(constant: string): (int, int) – get the position of the string constant in the ST

- toString() (overridden method) – get the string representation of the whole symbol table

ScannerException
-Exception class, it extends RuntimeException and overrides one method.

Scanner
-class responsible with building the PIF (Program Internal Form) and ST (Symbol Table), and with checkin g for lexical errors. The PIF is a list composed from pairs of the form (String, PositionInST) where String is the token, string const, int const or identifier. Tokens in ST are considered to be on position (-1,-1). The S canner has fields for program (String), ST, index and current line.

-setProgram(String program): setter for program

-readTokends(): read token.in and separate them into reserved words and other tokens

-skipSpaces(): method to skip spaces and update current line and index

-skipComments(): method to skip comments and update current line and index

-treatStringConstant() Boolean : method to treat the case in which there is a string in the program. It chec ks if the string is lexically correct (no invalid characters). If it is correct, then the string is added to ST and PIF, the index is updated and the method returns true. Otherwise, it returns false.

-treatIntConstant() Boolean : method to treat the case in which there is an integer in the program. It check s if the number is valid (no characters other than digits). If it is correct, then the number is added to ST an d PIF, the index is updated and the method returns true. Otherwise, it returns false.

-checkIfValid(String possibleIdentifier, String programSubstring) Boolean: checks if the possibleIdentifier i s valid (it is part of the declaration or it is in ST), and if it is, the method returns true, otherwise false.

-treatIdentifier() Boolean: method to treat the case in which we have an identifier in the program. The met hod checks if it is valid (starts with either a character or _ and contains only characters, digits and _, it is p art of a previous declaration or a previously defined identifier), and if it is, then the identifier is added to ST and to the PIF (if it does not exist);
the method updates the index and returns true. If the identifier is not valid, the method returns false.

-treatFromTokenList() Boolean: checks if the current element of the program is a reserved word or a toke n. If it is, then it is added to PIF and the program returns true, otherwise false.

-nextToken(): treats the current case, and if there is no corresponding case, throw and exception that ther e is a lexical error at current line and index

-scan(String programFileName): reads the program from the file programFileName and checks for nextTo ken() until the and. At the end it writes the PIF and ST of the program and prints a message stating that th e program is lexically correct. If an exception is caught, the program prints the message of that exception.