SymbolTable

Representation:

The symbol table contains, in its representation, a list and the size of the list, in which the elements will be place.

At each position of the table there is a list of elements, since if two different constants/variables are hashed to the

same symbol table index, they all can be stored. Thus, an element which is included in the symbol table, is represented

by two indices, one showing the index of the list in which the element is stored, and the second one shows the position

of the element in the specific list.

I chose to implement the symbol table with the data structure hash table. The hash function consists of the sum of the

ASCII codes of the characters of the constant/identifier, modulo the size of the list.

Operations:

- insert(key:string): -

Checks if the element was already stored in the ST or not. If it was, then the element won't be stored again and

it raises an Exception saying that the element already exists in the ST. Otherwise, the element is stored.

- remove(key): None

Checks if the key is present in the ST. If yes, then it removes the key from the data structure. Otherwise, it raises

an error.

- key\_exists(key:string): boolean

Checks if the given key exists in the ST. If it exists, then it returns True, otherwise it returns False.

- hash(key): int

Computes the sum of the ASCII codes of the characters and divides them by the hash table size, then it returns

it. The returned number is the index of the ST list in which the element will be placed.

- \_\_str\_\_ (overridden operation): string

Returns the string representation of the ST.

Program Internal Form

Representation:

The program internal form is a collection based on a data structure array. While the symbol table stores the constants and identifiers, the program internal form stores the operators, separators and reserved words. Moreover, the program internal form keeps a reference to the symbol table, meaning that constants and identifiers/variables are stored in the program internal form and are associated with the index on which they can be found in the symbol table. Any other token has an associated value -1, showing that it is not part of the symbol table.

Operations:

* add (key: string, position: int): None

Adds a (key, position) pair to the PIF, where the key is a token and the position is the position of the token in the symbol table, if present. Otherwise, -1.

Lexic Inserter

Representation:

The LexicInserter class is responsible for scanning the token.in text file and storing the separators, operators, and reserved words in separate lists.

Operations:

* classify (): None

Scans the file which contains the tokens and arranges them in three categories, namely separators, operators, and reserved words.

Token Identifier

Representation:

This class is responsible for splitting the program text into words and identifying whether they belong to a category or not, thus populating the symbol table and program internal form.

Operations:

* is\_identifier(token: string): Boolean

Checks if the given token is an identifier. Returns true, if yes, otherwise it returns false.

* is\_numerical\_constant(token: string): Boolean

Checks if the given token is a numerical constant. Returns true, if yes, otherwise it returns false.

* is\_string\_constant(token: string): Boolean

Checks if the given token is a string constant. Returns true, if yes, otherwise it returns false.

* get\_tokens(token: string): Boolean

Parses the file line by line and separates the text into words, after which it identifies if the words belong to a category or not. If there is a word which does not belong to a category, it prints that a lexical error occured. If there are no lexical errors identified, it prints that the program is lexically correct.

Scanner

Representation:

The scanner class is responsible for opening the input and output files and then storing the returned tokens into the symbol table, respectively the program internal form.

Operations:

* computer\_result(file\_name: string): None

Opens the file in which the text of the program can be found and then stored the returned tokens into the symbol table and program internal form classes.