https://github.com/AndreiG23/FLCD

FLCD Scanner Documenation

Ghiurcuta Andrei-Bogdan 938pr

Scanner

Symbol Table

Symbol table is implemented as a hash table, in Python. The elements are represented as a main list with given size, where the positions represent the hashed values of the keys. Collisions are solved using separate chaining, so on each position, elements are placed using the deque from Python.

The class attributes are:

- Table size (given as parameter)
- Number of elements (updated with each addition, the numbers of symbols)
- The elements

The implemented operations are:

1. **add(key)** – adds a new symbol in the table
2. **getPosition(key)** – for given symbol returns its position as a tuple (indexInList, positionInDeque)
3. **exists(key)** – checks if a symbol is in the table or not
4. **size()** – returns the number of symbols currently in the symbol table

When adding a new symbol, its hash value is computed and the element is added at the computed position in the list; if there are more elements on that position, the new element is added at the end of the list(deque). If the same symbol already exists in the symbol table, it is not added again.

The hash value is computed as follows:

- the symbol is converted to a string (if it is a number)
- compute the sum of ascii codes from the string

- return the sum modulo the size of the hash table

Program internal form (PIF)

PIF is implemented as a list of tuples. Each tuple contains token and location.

• if token is operator/separator/keyword it appears as it is, with the location (-1,-1)

• if token is id/constant, “id”/“constant” appear together with the actual location in the symbol table

Methods:

- **add(token, index)**: adds the tuple to the list of elements

Utils.py file contains 2 functions:

- **readTokens(file)**: reads operators, separators, keywords from token.in
- **readProgram(file)**: reads the given program and returns a list of the lines

Scanner

Class that takes care of tokenizing, scanning and providing the output.

Attributes:

- operators, separators, keywords: read from the tokens file
- identifiersTable: symbol table containing only identifiers
- constantsTable: symbol table containing only constants
- pif: instance of Program internal form class
- errors: keeps track of lexical errors

Methods:

- **scan()**: main method that scans the program using other methods and classify the tokens, deciding if the program is lexically correct or not
- **generateOutput()**: contructs the PIF.out and ST.out
- tokenize(line): splits a line into tokens (considering operators and strings)
- **search_for_operators(line)**: searches for simple or compound operators in aline and surrounds them with whitespace so they can be split later from the rest

- **search_for_strings(line)**: searches for constant strings (text surrounded by “”), adds them into a list and replaces them in the line with their index from that list

- * other Boolean functions that check if a given token is operator/separator/keyword/identifier/constant

Regex:

identifier: ^[a-z][\w]$*

- begins with lowercase letter and has any number of letter/digit/underscore after

constant number: ^-?[1-9][0-9]$*

- optional ‘-‘ sign, must start with nonzero digit than has any number of digits

constant string: ^\"[A-Za-z0-9\.\?\!,]\"$*

- must start and end with “”, has any number of letters/digits U { . , ? ! space}

Class Diagram