DOCUMENTATION

Github link -> https://github.com/Moldovan-Andrei-Bogdan/flcd/tree/main/lab3

--- Symbol Table ---

Composed from 3 separate hash tables, one for identifiers, one for integer constants and one for string co nstants.

Hash tables are represented each by a list where its elements are also lists in order to be able to store ele ments that hash to the same value. Each hash table has a size.

An element from the symbol table has as its position a pair of indices, the first one being the index of the li st in which the element is stored and the second one being the actual position in the list.

In the case of integer values, the hash function is the value % n, where n is the size.

In the case of string constants/identifiers, the hash function is the sum of ASCII codes % n, where n is the size.

Note -> the hash table is generic

-- Operations --

** Hash Table **

- hash(key: int): int - computes the position in the ST where the integer constant will be added

- hash(key: string): int - computes the position in the HT where the constant / identifier will be added

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

- getHashValue(key: T): int - return the corresponding position in the ST based on the type of the key

- add(key: T): (int, int) - add the key to the hash table and return its position if the operation has been succ essful, otherwise throw an exception

- contains(key: T): boolean - return true if the key is in the hash table, false otherwise

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

- toString(): string - return the string representation of the hashTable

** Symbol Table **

- has 3 hash tables -> one for identifiers, one for string constants and one 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 a string constant and return its position in the ST

- hasIdentifier(name: string): boolean - return true if the given identifier is in the ST, otherwise return false

- hasIntConstant(constant: int): boolean - return true if the given integer constant is in the ST, otherwise re turn false

- hasStringConstant(constant: string): boolean - return if the given string constant is in the ST, otherwise r eturn false

- 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() - get the string representation of the symbol table

** Scanner **

- The Scanner class / service is responsible for building the program internal form (PIF) and also to check whether the input program is valid or not from a lexical point of view

- The program internal form (PIF) is a list composed of pairs of the form string and position in the symbol table, where the string is the actual token, string const, int const or identifier.

- When it comes to tokens, the position in the symbol table will be (-1, -1) because they do not belong to the symbol table

- The Scanner also has fields for the program, symbol table, index and current line of the program

-- Operations --
- setProgram(program: string): void - sets the current active program

- readTokens(): void - read the tokens from "token.in" file and classify them as reserved words or other to kens

- discardSpaces(): void - skips the spaces from the program and update the current line and the index

- discardComments(): void - skips the comments from the program and update the index

- treatStringConstant(): boolean - treats the case in which we have a string constant in the program;
it checks whether is lexically correct (no invalid chars and quotes closed correctly)
and if it is, then it adds it to the symbol table (if it does not exist)
and to the program internal form;
it updates the index and returns true;
return false if the string const is invalid

- treatIntConstant(): boolean - treats the case in which we have an integer constant in the program;
it checks whether the number is valid (only numbers)
and if it is, it adds it to the symbol table (if it does not exist)
and to the program internal form;
it updates the index and returns true;
if the integer is invalid, the method returns false

- checkIfValid(possibleIdentifier: string, programSubstring: string): boolean
- checks if a possible identifier is valid by checking if it is part

of a declaration or if it is in the symbol table
- If none of the conditions are met, then the possibleIdentifier is an invalid token and thus return false

- treatIdentifier(): boolean - treats the case in which we have an identifier in the program;
it checks whether we have a valid identifier (
starts with _ or a letter, contains only letters, digits and _ ,
is part of a declaration or it was previously defined
);
if everything is fine, it adds it to the symbol table (if it does not exist)
and to the program internal form, updates the index and returns true;
return false otherwise

- treatFromTokenList(): boolean - checks if the current element of the program is a reserved word or another token and if yes, adds it to the
program internal form and returns true;
return false otherwise

- nextToken(): void - treats the current case and if no match then throw lexical error at current line

- scan(programFileName: string): void - reads the program from the given file and checks for the next token until the end of the program is reached;
finally, it writes the PIF and ST to an output file
and also shows if the program is lexically correct;
if an exception is caught, it displays the message

FA

The FA class is a finite automation which implements operations on an automaton read from a file. The file for the automaton contains the states, alphabet, transitions, initial state and final states (out states). The states, alphabet, transitions and final states are kept in separate lists.

Operations:
- init() – reads the file and identifies the states, alphabet, final states, initial state, transitions; throws Exception if the file is not in the correct form

- printListOfString(listname: String, list: String[]) – prints a list in a specific format

- printStates(), printAlphabet(), printOutputStates() – wrapper methods for printing the corresponding lists

- printInitialState() – display the initial state

- printTransitions() – prints the transitions of the finite automaton

- checkAccepted(word: String): Boolean – checks if the given string is accepted by the FA (starting from the initial state we reach a final state)

- getNextAccepted(word: String): String – get the substring of the input word that is accepted by the FA