https://github.com/Matei-Daria/LFTC

For the scanner implementation the class `MyScanner` was used. This class utilizes the classes `SymbolTable` and `ProgramInternalForm`.

SymbolTable

The class `HashTable` implements the hash table data structure, with all the basic operations. The wrapper class `SymbolTable` uses the `HashTable` class and calls its methods, those needed for the implementation of a Symbol Table.

The `HashTable` class is implemented as follows. The keys for a hash table are integers, so the hash table is represented as a list of lists, or a list of "buckets", where the index of the bucket is the key from the hash function. The class has the following methods:

- `private int hash(String value)` – Private hash function where the formula is h(k) = sum_of_ascii_chars(k) % nr_of_buckets.
- `public boolean add(String value)` – Calculates the hash function for the value (the bucket index) and adds it to the correct bucket; returns true if value was added, false if not (if the value is already in the table)
- `public void remove(String value)` – Searches for the value in the hash table and, if present, removes it.
- `public boolean contains(String value)` – Returns true if the value is in the hash table, false otherwise
- `public int getIndexInBucket(String value)` – Searches for the value in the hash table and, if present, returns its index in the bucket it is stored, otherwise -1
- `public int getAsciiSumOfChars(String value)` – Calculates the sum of ASCII characters contained in a string value

The `SymbolTable` class has a `HashTable` class field and functions `add`, `contains` and `toString`, which call the corresponding functions in the `HashTable` class, through the instance of the class field.

ProgramInternalForm

`ProgramInternalForm` has as a field an array of `ProgramInternalFormPairs`.

Class `ProgramInternalFormPair` is a simple record with parameters `String token`, `HashTablePosition position`. `token` represents the token stored in the PIF and `position` represents the 2-coordinate position of an element in a normal hash table. The hash table was chosen as the representation for the Symbol Table for this problem, so the `HashTablePosition` instance points to the token's position in the Symbol Table.

`ProgramInternalForm` has methods `add`, which adds a new ProgramInternalFormPair to the array of pairs, and `toString`.

MyScanner

For class `MyScanner`, we have as fields instances of `SymbolTable` and `ProgramInternalForm`, as well as lists of `Strings`, which represent the language tokens (operators, separators, keywords, alphabet), as specified and subsequently read in this class from the `token.in` file. Other fields are used, such as ones used to differentiate between alphabet elements allowed as the first element of an identifier and those not allowed, and the file name of the file which contains the currently scanned program.

The class `MyScanner` has the following methods:

- `private String readProgramFromFile()`
  - reads the program to be scanned from a file whose name is given in the class constructor, line by line
- `private void readLanguageTokens()`
  - reads the language tokens from the `token.in` file
  - the `token.in` file contains extra words, such as "Separators", "Keywords", etc., used in order to differentiate between different types of tokens which are stored in a `token.in` file; this differentiation is taken into account when distributing the tokens read line by line to the class fields previously mentioned
  - in this method, the alphabet is transformed into variables with all alphabet elements concatenated, to be later used in the regex
- `private List<String> getListOfSeparatedStrings()`
  - transforms the string that contains the program read from file into a list of strings obtained by separating the text using the separators read from `token.in`
- `private List<TokenLinePair> getListOfTokenLinePairs()`
  - uses the list of separated strings obtained from the previous method to create a list of `TokenLinePairs`; `TokenLinePair` is a record class with parameters `String token, Integer line` and it contains the token from the program and the line of the program that it is on
  - a special case for determining the tokens from the list of separated strings is the fact that our language can have string constants which can have the character space in them, meaning that the `getListOfSeparatedStrings()` method separated them, when they are actually the same token; this means that we have to keep track of when a string const begins (char "), when we are currently parsing a string const, and when a string cost ends (char ").
  - another special case is when encountering a "\n" string and we need to increase the line of the program that we are currently on
  - in the end, if none of the cases above apply and the string is not the space character, we have found a new token that can be added to the list of `TokenLinePairs`
- `public void scan()`

- the defining method of this class
- uses the list of `TokenLinePairs` obtained from the previous method
- adds, one by one, the tokens from this list to the `ProgramInternalForm` class field and validates and further categorizes them based on the type of token they belong to (operator, separator, keyword, constant, identifier); to validate the operators, separators, keywords and identifiers, we use the language tokens specified in the `token.in` file; for the constants, however, we need to use a regex which specifies more complex rules for the validation of constants defined in our language
- if the token does not belong to any of the categories of tokens, it is categorized as a lexical error and an error message is shown to the user
- in the end, if all tokens are successfully validated, a success message is shown to the user

Main

In class `Main`, a new scanner is instantiated for each of the input files p1.txt, p2.txt, p3.txt, p1err.txt. Then, the corresponding files with the termination -ST.out and -PIF.out are generated and the contents of the Symbol Table and, respectively, Program Internal Form are written to those files.