Link to GitHub repository: https://github.com/octaviaah/flcd

I chose to use Python as my programming language of choice for this lab and PyCharm as IDE.

I decided to use the HashTable as the data structure for the Symbol Table, due to the fact that the HashTable's operations run in O(1). The SymbolTable class has 2 instances, an Identifier Table and a Constant Table.

The HashTable is kept as a Pyhton list with Deque(double ended queue), on each position. It ensures that there are no conflicts, because 2 elements hash to the same position, they will both be added to the same deque.

Hash function:

The hash function is a modular function(h(k) = k mod m) where:

- k represents the sum of all ASCII(American Standard Code for Information Interchange) values of the key's characters
- m represents the size of the HashTable

Operations of the HashTable:

- contains(key): returns a boolean value which indicates whether the given value exists in the Symbol Table
- get(key): return a pair of indexes: the index of the list where the key hashed and the index of the position in the respective deque
- add(key): adds the key into the Symbol Table
- str(): prints the table

ReadFile.py:

This file contains a function(read_file()) which reads the token.in file and parts the special tokens into 3 lists: the first one represents the operators of the language, the second one the separators, and the third one the reserved words.

Program Internal Form:

The Program Internal Form(PIF) is a list of pairs of type (token, position), where token represents an identifier, a constant, an operator, a separator or a reserved word, while position is a tuple which has, as first element, the hash key of the corresponding symbol table and, as second element, the actual position of the token in the deque. The operators, separators and reserved words will have, as default position, (-1, -1), because they do not appear in any of the 2 instances of the symbol table.

Tokenizing algorithm:

The tokenizing algorithm loops all the characters on each line and checks if the current character is an operator, separator, or if it's part of a reserved word, constant or identifier. The tokens are appended to a list of tokens and the list is returned.

Scanning algorithm:

The scanning algorithm uses the tokenizing algorithm to split the program into tokens and builds the instances of the symbol table and the PIF. If the token is an operator, separator or a reserved word, it is added to the PIF and its associated position will be (-1, -1)(as explained in the Program Internal Form part). If the token is an identifier, it will be added to the instance of the symbol table which handles the identifiers, and it will also be written to the PIF, alongside its position. The same approach is used when a token is a constant. If the token couldn't be categorized in one of above categories, it means there is a lexical error, which will be printed in the console.