```
class SymbolTable:
    def __init__(self, size=100):
        """
        Initialize a symbol table with an optional specified size.

        :param size: The size of the hash table. Default is 100.
        """

    def hash_function(self, key):
        """
        A simple hash function that returns the index for a given key.

        :param key: The key to be hashed.
        :return: The hash index for the key.
        """

    def insert(self, value):
        """
        Insert a value into the symbol table using its hash as the key.

        :param value: The value to be inserted.
        """

    def lookup(self, value):
        """
        Look up a value in the symbol table and return the key if found

        :param value: The value to be looked up.
        :return: The key for the value if found, None otherwise.
        """

    def delete(self, value):
        """
        Delete a value from the symbol table.

        :param value: The value to be deleted.
        """

    def display(self):
        """
        Display the contents of the symbol table.
        """

    def __str__(self):
        """
        Return a string representation of the symbol table.

        :return: A string representation of the symbol table.
        """
```

Github: https://github.com/Striletchi-Vlad/FLCD-L1

Lexer Documentation

This documentation provides an explanation of the code for a Python lexer that tokenizes a source code file. The lexer identifies various types of tokens and produces a Program Internal Form (PIF) and Symbol Table (ST). Here's an overview of the code:

The PIF is implemented as a list of tuples.

The regexes are:
    self.identifier_regex = r'^[a-zA-Z][a-zA-Z0-9]?$'
    self.constant_regex = r'^[+-]?[0-9]+$'
The identifier regex allows any letter, lower or upper case, followed by any letter or digit 0 or 1 time.
The constant regex allows a + or – sign, 0 or 1 times, followed by a digit 1 or more times.

`SymbolTable` Class (Not Provided in the Code)

The `SymbolTable` class is imported from an external module `symboltable`. It is used to store identifiers and constants found in the source code. The details of this class are not provided in the code snippet, but it is assumed to be used for storing and managing identifiers and constants.

`Lexer` Class

The `Lexer` class represents the lexer. It reads a source code file, tokenizes it, and generates a PIF and ST. The class contains the following methods and attributes:

`__init__(self, filename)`

- Constructor for the `Lexer` class.
- Initializes instance variables:
  - `pos`: Current position in the source code file.
  - `filename`: The name of the source code file to be processed.
  - `st`: An instance of the `SymbolTable` class for storing identifiers and constants.
  - `lines`: A list to store the lines of the source code.
  - `tokens`: A list of token strings read from "Token.in" file.
  - `separators`: A list of separator characters.
  - `identifier_regex`: A regular expression pattern for matching identifiers.
  - `constant_regex`: A regular expression pattern for matching constants.
  - `PIF`: An empty list for storing tokens and their associated indexes.

`read_file(self)`

- Reads the source code file specified by `filename`.
- Ignores lines starting with `//`.
- Stores the lines in the `lines` list after removing newline characters.

`split_text(self)`

- Splits each line in the `lines` list into tokens using the specified separators and stores them back in the `lines` list.

`lex(self)`

- Tokenizes the source code:
  - For each line in the `lines` list, it iterates through the tokens.
  - If a token is found in the `tokens` list, it is added to the PIF with a value of -1.
  - If a token matches the `identifier_regex`, it is inserted into the symbol table, and the PIF stores the token type as 'id' and the index from the symbol table.
  - If a token matches the `constant_regex`, it is inserted into the symbol table, and the PIF stores the token type as 'const' and the index from the symbol table.
  - If none of the above conditions are met, an error message is printed, and the lexer exits.
- Finally, the PIF is written to a file named "PIF.out," and the Symbol Table is written to a file named "ST.out."

`main(self)`

- Main method that orchestrates the lexer:
  - Calls `read_file` to read the source code file.
  - Calls `split_text` to tokenize the source code.
  - Calls `lex` to perform lexical analysis.

Running the Lexer

- If this code is run as the main script, it creates a `Lexer` instance with the specified filename ('p1err') and calls the `main` method to tokenize the source code and generate the PIF and ST.

Example Usage

```python
if __name__ == '__main__':
    lexer = Lexer('p1err')
    lexer.main()
```

This code processes the 'p1err' source code file, tokenizes it, and stores the results in 'PIF.out' and 'ST.out' files.