Laboratory no. 1

Problem statement:

Considering a small programming language (that we shall call mini-langage), you have to write a scanner (lexical analyzer).

The scanner input will be a text file contain the source program, and will produce as output the following:

- PIF - Program Internal Form
- ST - Symbol Table

In addition, the program should be able to determine the lexical errors, specifying the location, and, if possible, the type of the error.

1. Identifiers:
   a. length at most 8 characters
2. Symbol Table:
   b. separate tables for identifiers, respectively constants
3. Symbol Table Organization:
   b. lexicographically binary tree

Implementation details:

- The atoms(identifiers and constants) have been memorized in two distinct binary trees.
- Codification table in loaded from file and keep in a dictionary
- The Program internal form is kept in memory as a list of tuples containing the code from codification table and the value of the atom or -1 if is a reserved word

Flow of execution

- The codiffcation table is red from a file and kept in memory as a dictionary
- The source code is red line by line to identify the atom
- Depending on the type of atom they are saved either in a constant table or identifier tabel
- The atom is save in the PIF
- The content of PIF,identifiers table and constant table is write to a files

Specification

```
class Scanner:


        def parse(self):
            '''
            Read source code from a file and parse the file
            to identify the atoms and save the atoms in
            the appropriate table
            :return: None
            :except: FormatException
                if the atom is not a well formatted(e.g 1ac for a variable ) or
        the
                length of an atom is > 8 the exception is thrown
            '''
        def write_to_file(self):
            '''
            Write the content of the PIF,identifiers table
            and constant table to files.
            :return:
            '''

        def __add_constant(self, const):
            '''
            Save the atom into the constant table and PIF
            :param const: The atom representing a constant
            :return:
            '''
        def __add_token(self, token):
            '''
            Save the indentifier into the PIF and identifiers table or
            if is a reserved word only in PIF
            :param token: atom representing an identifier or reserved word
            :return:
            '''
        def __get_char(self, line):
            '''
            Return a character at a time from a given line
            :param line: a line from source code
            :return: a character from the line
            '''
        def __populate_cod_table(self):
            '''
            Helper method to load from file the codification table
            and store it into a hash table
            '''

class Node:
   '''
   The implementation of a binary tree
   '''
       def insert(self, data):
          '''
          Add a new node in the binary tree in the
          appropriate positon(left or right of the current leaf)
          :param data:
          :return:
       def print_tree(self):
          '''
              Returns a string representation of the
              Binary tree content
          :return:
          '''
```