GitHub Link: https://github.com/roll22/FLTC

Lab 3 – Documentation

# Purpose

The purpose of this lab was to implement a scanning algorithm for the language specified during Lab1, using the Symbol Table from Lab 2

Implementation

The algorithm goes through the program code line by line and does the following steps:

1. Tokenize

split = re.split('([^A-Za-z\_0-9.,-])', line)

split = list(filter(lambda x: x is not None and x != '', map(lambda x: x.strip(), split)))

1. Classify

if token in reservedWords:  
 self.pif.add(reservedWords.index(token), None)  
if token in reservedOperatorsSeparators:  
 self.pif.add(reservedOperatorsSeparators.index(token) + 100, None)

1. Codify

elif is\_identifier(token):  
 self.st.add(token)  
 self.pif.add('ID', self.st[token])  
elif is\_constant(token):  
 bla = is\_constant(token)  
 self.st.add(token)  
 self.pif.add(bla, self.st[token])

Each line is split by spaces, separators and operators. We have to take into consideration the distinction between +/- as binary operators and unary operators.

During runtime, I split each line in tokens.

For each token, I check if it is either an:

-separator

-operator

-keyword

-constant

-identifier

Otherwise, it is a Lexical Error and the error is printed in the output file pointing the line and the error.

Then, if the token is a constant or identifier, we add it to the symbol table (if it is not already there) and then add the corresponding code of the token + its position in the symbol table to the PIF.

## Instructions

To analyze a program, my script has to be run from the terminal, like this:

**python main.py <input\_file\_name>**

If the user fails to provide a file name, the user will be prompted with an Error Message:  
 **if** len(sys.argv) != 2:  
 **raise** Exception(**"analyze <input\_file\_name>"**)

Following the execution, the output can be found inside the program\_ultra\_smecher.out.

In the case the input program is correct, the user will find the PIF and Symbol Table.

Otherwise, inside the file will be the lexical error he has to fix.

# Interface

**class** Scanner

The Scanner class only has one method, the constructor one, which does all the magic. Following is the source code for the Scanner class:

class Scanner:  
 pif = PIF()  
 st = HashST()  
  
 def \_\_init\_\_(self):  
 if len(sys.argv) != 2:  
 raise Exception("analyze <input\_file\_name>")  
 else:  
 output = open(sys.argv[1][:-3] + ".out", "w")  
 try:  
 with open(sys.argv[1]) as f:  
 line\_number = 1  
 line = f.readline()  
 while line:  
 print(line)  
 split = re.split('([^A-Za-z\_0-9.,=<>-])', line)  
 split = list(filter(lambda x: x is not None and x != '', map(lambda x: x.strip(), split)))  
 buffer = ""  
  
 for token in split:  
 if token == "\'":  
 if buffer == "":  
 buffer += token  
 continue  
 else:  
 buffer += token  
 token = buffer  
 buffer = ""  
 elif buffer != "":  
 buffer += token  
 continue  
  
 if token in reservedWords:  
 self.pif.add(reservedWords.index(token), None)  
 elif token in reservedOperatorsSeparators:  
 self.pif.add(reservedOperatorsSeparators.index(token) + 100, None)  
 elif is\_identifier(token):  
 self.st.add(token)  
 self.pif.add('ID', self.st[token])  
 elif is\_constant(token):  
 bla = is\_constant(token)  
 self.st.add(token)  
 self.pif.add(bla, self.st[token])  
 else:  
 raise Exception(  
 "Lexical error. Invalid token: '{}' on line {}".format(token, line\_number))  
 if buffer != "":  
 raise Exception(  
 "Lexical error. Unclosed string: '{}' ".format(buffer))  
 line = f.readline()  
 line\_number += 1  
  
 output.write(str(self.pif))  
 output.write("\n")  
 output.write(str(self.st))  
 except Exception as e:  
 output.write(str(e))

We just need to instantiate it and it will take care of everything itself.