Comp 3220, Fall 2017 – Assignment 1

Due Thursday, October 12, 2017 at 11:59 pm

Building a Lexical Analyzer with Python

The grammar rules for the language "TINY" are listed below. In this assignment we will identify the tokens in this language and build a lexical analyzer (lexer) for recognizing and outputting TINY tokens.

```
# https://www.cs.rochester.edu/~brown/173/readings/05_grammars.txt
#
#  "TINY" Grammar
#
# PGM         -->   STMT+
# STMT        -->   ASSIGN   |   "print"  EXP
# ASSIGN      -->   ID  "="  EXP
# EXP         -->   TERM   ETAIL
# ETAIL       -->   "+" TERM   ETAIL  | "-" TERM   ETAIL | EPSILON
# TERM        -->   FACTOR  TTAIL
# TTAIL       -->   "*" FACTOR TTAIL  | "/" FACTOR TTAIL | EPSILON
# FACTOR      -->   "(" EXP ")" | INT  | ID
#
# ID          -->   ALPHA+
# ALPHA       -->   a  |  b  | … | z  or
#                   A  |  B  | … | Z
# INT         -->   DIGIT+
# DIGIT       -->   0  |  1  | …  |  9
# WHITESPACE  -->   Whitespace
```

You will be reading a text file containing TINY statements character by character. Whenever a token is identified, you need to capture it using the Token class below. Each token has a **type** and **text**. For example, if DOG was identified as a variable in this language, it could have type "id" and text "DOG". The add operator might have type "addOp" and text "+" or type "+" and text "+". These values are somewhat arbitrary – choose values that make sense to you. The Token class is listed below. It has a few constants already defined. You will need to build constants for all the tokens in the grammar. You can modify the given tokens if you like.

```
#  Class Token - Encapsulates the tokens in TINY
#
#   type = the type of token
#   text = the text of the token
class Token:
     # Token Class Variables
     EOF = "eof"
     LPAREN = "("
     RPAREN = ")"
     ADDOP  = "+"
     WS     = "whitespace"
     # ... more needed here

     # Constructor
     def __init__(self, type, text):
          self.type = type
          self.text = text

     # String representation of an instance of Token
     def toString(self):
          return '[Type:{}, Text:{}]'.format(self.type, self.text)
```

Following are the details of the assignment requirements:

1. The purpose of the assignment is to build a Scanner (Lexer) for TINY. A partial implementation of the Scanner class has been provided below.
2. The constructor is passed a file name which contains a TINY program. The constructor opens the file and reads the first character, storing it in class variable c (which acts as a one-character look ahead).
3. Function **nextChar()** updates **c** with the next character
4. Function **nextToken()** returns the next token identified by the scanner.
5. Contiguous whitespace should be combined and emitted as a single token.
6. Consecutive characters are considered a single token.
7. Consecutive numbers are considered a single token.
8. An end of file (EOF) token should be emitted when the file has been completely processed.
9. You will need to modify the constructor so that it fails gracefully if the file doesn't exist.
10. It is important to test all the code you write. Please use the Python unit testing framework
11. You need to upload and submit the following on Canvas:
    - A zip file containing your code for Token, Scanner, and tests that demonstrate that your code works correctly.
    - The zip file should use the following naming convention: **assignment1_username.zip**

```
#  Class Scanner - Reads a TINY program and emits tokens
class Scanner:

     # Constructor - Is passed a file to scan and outputs a token
     # each time nextToken() is invoked.
     def __init__(self, fileName):
           self.position = 0
           self.fileName = fileName
           self.nextChar()

     # Function nextChar() returns the next character in the file
     def nextChar(self):
           self.f = open(self.fileName)
           self.f.seek(self.position)
           self.c = self.f.read(1)

           if not self.c:
                 self.c = 'eof'
           else:
                 self.position = self.f.tell()
           self.f.close()

  # Function nextToken() reads characters in the file and returns
  # the next token – THIS FUNCTION NEEDS TO BE COMPLETED
     def nextToken(self):
           if self.c == 'eof':
                 token = Token(Token.EOF, 'eof')
           elif self.c.isspace():
                 str = ''
                 while self.c.isspace():
                       str += self.c
                       self.nextChar()
                 token = Token(Token.WS, str)
           # more needed here...
           else:
                 # unknown case
                 token = Token('unknown', 'unknown')
           return token
```

Sample test for the Token class:

```
import unittest
from Token import Token

class Test_Token(unittest.TestCase):
     def test_toString(self):
           token = Token('my type', 'my text')
           self.assertEqual('[Type:my type, Text:my text]', token.toString())
           self.assertEqual('my type', token.type)
           self.assertEqual('my text', token.text)

if __name__ == '__main__':
   unittest.main()
```