Building a Lexical Analyser with Ruby

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 analyser (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 -->   Ruby Whitespace
```

Whenever a token is identified, we encapsulate 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. 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
   attr_accessor :type
   attr_accessor :text

  # Token Class Constants
   EOF = "eof"
   LPAREN = "("
   RPAREN = ")"
   ADDOP  = "+"
   WS     = "whitespace"
  # ... more needed here

  # Constructor
   def initialize(type,text)
      @type = type
      @text = text
   end
  #
   def to_s
      return "[Type: " + @type + " Text: " + @text + " ]"
   end
end
```

It is important to test all the code you write. At the very least, use "puts" to test the class methods. You can build the tests in separate files and load them as needed: load "mytest.txt". The code below tests the Token class methods.

```
# Test the Token class

tok = Token.new("atype","atext")
puts "Token type: #{tok.type}"
puts "Token text: #{tok.text}"
tok.type = "btype"
tok.text = "btext"
puts "Token type: #{tok.type}"
puts "Token text: #{tok.text}"
puts "Token: #{tok}"
```

The first goal is to build a Scanner (or Lexer) for TINY. I have sketched the basic structure in the code below. Here are a few points to consider:

1) 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).
2) Currently, the Scanner abends if it is passed a file that doesn't exist. Modify the code so that it fails gracefully in this circumstance.
3) Method nextCh() updates @c with the next character and it.
4) Method nextToken() returns the next token identified by the scanner. I have begun to define this function, but you will need to complete this function so that it can handle all tokens this grammar defines.
5) You will need to modify the constructor so that it fails gracefully if the file doesn't exist.
6) Contiguous whitespace should be combined and emitted as a single token.
7) An end of file (EOF) token should be emitted when the file has been completely processed.
8) Send me the code for Token, Scanner, and testing.

Here is the Scanner code and helper methods:

```
 #  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.
#   @c        - A one character lookahead
 def initialize(filename)
    @f = File.open(filename,'r:utf-8')
    if (! @f.eof?)
       @c = @f.getc()
    else
       @c = "eof"
       @f.close()
    end
 end

# Method nextCh() returns the next character in the file
 def nextCh()
    if (! @f.eof?)
       @c = @f.getc()
```

```
         else
            @c = "eof"
         end
         return @c
      end
     # Method nextToken() reads characters in the file and returns
     #                    the next token
      def nextToken()

        if @c == "eof"
           tok = Token.new(Token::EOF,"eof")
        elsif (whitespace?(@c))
           str =""
           while whitespace?(@c)
              str += @c
              nextCh()
           end
           tok = Token.new(Token::WS,str)
        elsif
           nextCh()
           # don't want to give back nil token!
           tok = Token.new("unknown","unknown")
        end
     # more needed here...

        return tok
      end
   end
   #
   # Helper methods for Scanner
   #
   def letter?(lookAhead)
     lookAhead =~ /[a-z]|[A-Z]/
   end

   def numeric?(lookAhead)
     lookAhead =~ /^(\d)+$/
   end

   def whitespace?(lookAhead)
     lookAhead =~ /^(\s)+$/
   end
```

Extra Credit (5 points): Modify the TestTinyLexer.rb module so that in addition to printing the tokens out to the console, it also writes all of the tokens to a file (that we can use later to create a parse tree!).