Building a Lexical Analyser with Ruby

The grammar rules for the language "TINY" are listed in a text file called "original_grammar.txt". In this assignment we will identify the tokens in this language and build a lexical analyser (lexer) for recognizing and outputting TINY tokens.

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.

Token Class

I have given you a partially complete Token class that you should modify for this assignment called "**TinyToken.rb**". It has a few constants already defined. The VALUES of these variables represent the name of the token. You will need to build more constants to store all of the tokens that exist in this language. For this assignment, you get to decide what categories of lexemes there are and what you want to name those categories.

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.rb". 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 "Lexer"

The first goal is to build a Scanner (or Lexer) for TINY. I have sketched the basic structure in file named "**TinyLexer.rb**". Here are a few points to consider:

1) The constructor is passed a file name which contains the source code for 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). **I already did this.**
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 returns it, unless it has reached the end of the file, in which case it will return "**eof**". **I already did this.**
4) Method nextToken() returns the next token identified by the scanner. It is not complete, as it does not identify all Tokens in your grammar yet. In addition to identifying and returning a token, this method should print what it finds, each time it identifies a token (just like the example of the lexer in the PowerPoint for chapter 4). **You must complete this method.**
5) Contiguous whitespace should be combined and emitted as a single token. **I already did this for you**.
6) An end of file (EOF) token should be emitted when the file has been completely processed. **I already did this for you**.
7) There are several helper methods defined at the bottom of "TinyScanner.rb" that you can use to help you to identify different types of characters (like numbers, letters, and whitespace).

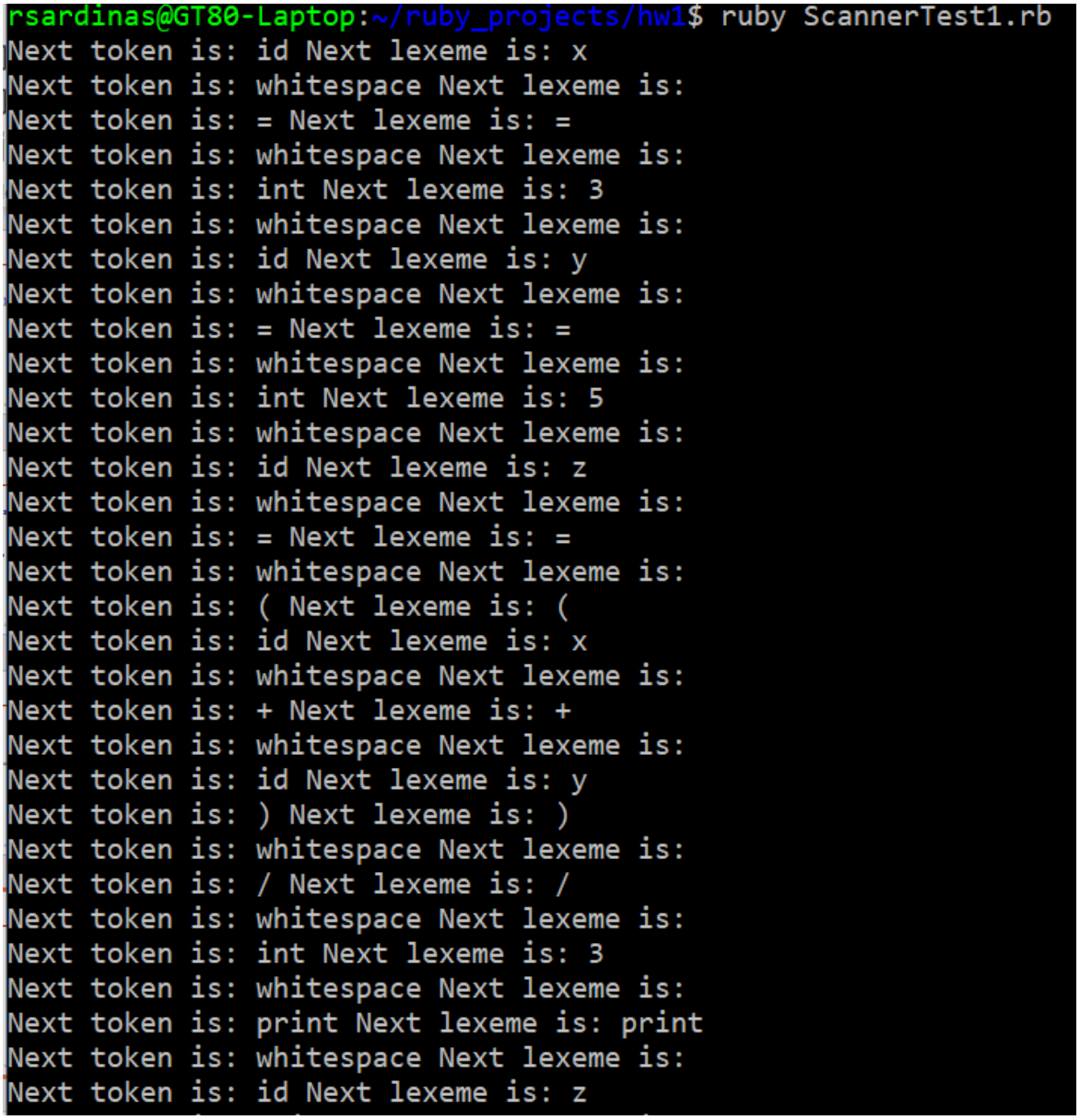
Display and Print Tokens

I've included a file called "**TestTinyLexer.rb**" that you can use to test your lexer (previous section). For the time being, it simply scans a file and stores all of the tokens in a text file called "tokens". Feel free to modify this file to test different things about your lexer, or to open the tokens file to see what was actually saved.

I've also included a sample input file (input.txt) that adheres to our TINY programming language. You should experiment with different inputs that both adhere to and don't adhere to the grammar to verify the correctness of your lexer.

Sample Program Output

Below is a screenshot of sample output that would result from using the included sample input file.



Bonus

You can earn up to 20 bonus points for this assignment. Submitting the assignment before the unofficial due date will earn you 5 points (provided you do not modify after that date), plus you can earn up to an additional 15 points by creating a lexer for a modified version of the grammar that includes limited support for Boolean expressions, Single branch if statements, and while loops.

The modified grammar can be found in the file called "grammar_w_bool.txt".