CSC488 Week 2 Tutorial

Introduction

This exercise is intended to help you get some practice with the lex portion of PLY, as well as familiarize you with scanning and tokenizing input string. You will be working with a programming language which looks like mini Java (aka, stripped down version of Java).

Requirements

The starter code is provided in `miniJavaLexer.py` file. Within this file, you should notice the `tokens` list, which has lists of token names we will be using for lexical analysis and also to parse the input language later on. Each of the tokens are tied to a single regular expression to define the said token (eg., `t_PLUS` variable). These regular expressions can also be tied to a function if additional work needs to be done (eg., `t_DECIMAL` function). You will also notice `reserved` dictionary, where the key represents a rule to match a specific token, and value represents the said token.

If you want to read more about any of the above concept, refer to PLY documentation. I recommend you to keep this link handy, since you would need to familiarize yourself with PLY for future tutorial and your project.

Language Requirements

MiniJava should support following symbols on top of symbols which are supported from the starter code:

- Operators: greater, greater than, less, less than, equals, not equals, not, and
- Delimiters: period, comma, brackets, braces
- Some Java keywords: class, public, static, extends, void, int, boolean, return, null, true, false, this, new, String, main

This should allow your lexer to tokenize basic miniJava programs.

Testing Your Code

Starter code has useful functions implemented for testing. `miniJavaLexer.py` takes in a single command-line argument, which is a path to the file with miniJava source code you want to test your lexical analysis with. So, you can run the following to test your program:

```
python3 miniJavaLexer.py testfile
```

For example, consider a file `file.java` with following content:

```
3 + 4 * 5432 / 6 + asdf;
```

Running `miniJavaLexer.py` with `file.java` as its command-line argument should result in following output:

```
LexToken(DECIMAL,3,1,0)
LexToken(PLUS,'+',1,2)
LexToken(DECIMAL,4,1,4)
LexToken(TIMES,'*',1,6)
LexToken(DECIMAL,5432,1,8)
LexToken(DIVIDE,'/',1,10)
LexToken(DECIMAL,6,1,12)
LexToken(PLUS,'+',1,17)
LexToken(ID,'asdf',1,19)
LexToken(SEMICOL,';',1,13)
```