Assignment 2

All sections: Due by 4/18 (Saturday), 11:59 pm

The second assignment is to write a syntax analyzer. You may use any top-down parser such as a RDP, a predictive recursive descent parser or a table driven predictive parser (100 points).

Hence, your assignment consists of the following tasks:
(10 points for the Documentation and 80-90 points for the code and implementation)

1. Write the Design Documentation and Specifications for your project (1-10 points)
2. Rewrite the grammar provided to remove any left recursion
   (Also, use left factorization if necessary)
   Do the arithmetic expressions first using these rules:

```
<Expression> -> <Expression> + <Term> | <Expression> - <Term> | <Term>
<Term> -> <Term> * <Factor> | <Term> / <Factor> | <Factor>
<Factor> -> ( <Expression> ) | <ID> | <Num>
<ID> -> id
*the <Num> rule is OPTIONAL
```

   Then do the Assignment statement using these rules:

```
<Statement> -> <Assign>
<Assign> -> <ID> = <Expression>;
*using a semicolon ; at the end of the rule is OPTIONAL
```

   Then do the single Declarative statement using these rules:

```
<Statement> -> <Declarative>
<Declarative> -> <Type> <ID>
*using a semicolon ; at the end of the rule is OPTIONAL
```

3. **The parser should print to an output file the <u>tokens, lexemes</u> and the <u>production rules used ;</u>**
   That is, **<u>first, write the token and lexeme found</u>**
   Then, **<u>print out all productions rules</u>** used for analyzing this token
   Note: - a simple way to do it is to have a "print statement" at the beginning of each function that will print the production rule.
   - It would be a good idea to have a "switch" with the "print statement" so that you can turn it on or off.

4. Then do the While and single If statements next for extra points.

5. Integrate your code with the **lexer() or functions** generated in the assignment 1 to get more points.

6. **Error handling:** if a syntax error occurs, your parser should generate a meaningful error message, such as token, lexeme, line number, and error type etc.
Then, your program may exit or you may continue for further analysis.
<u>The bottom line is that your program must be able to parse the entire program if it is syntactically correct.</u>

<u>Use the rules as needed posted on Titanium as a guide line and Turn in your assignment in separate working iterations for maximum credit</u>

Example

Assume we have the following statement

….more ….
a = b + c;
…. more details in class….

***<u>One possible output would be as follows</u>*:**

…. more….
Token: Identifier Lexeme: a
<Statement> -> <Assign>
<Assign> -> <Identifier> = <Expression> ;
Token: Operator Lexeme: =
Token: Identifier Lexeme: b
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> <Identifier>
Token: Operator Lexeme: +
<Term Prime> -> ε
<Expression Prime> -> + <Term> <Expression Prime>
Token: Identifier Lexeme: c
<Term> -> <Factor> <Term Prime>
<Factor> -> <Identifier>
Token: Separator Lexeme: ;
<Term Prime> -> ε
<Expression Prime> -> ε
…. more…..