METROPOLITAN STATE UNIVERSITY℠

Mathematical & Computer Science Department
Principles of Programming Languages - Spring 2020

Homework 03

Deadline[1]: Monday Feb. 17th

Consider the grammar below written in extended BNF (Backus-Naur Form) that generates declarations for a single numerical identifier involving four different independent properties of numbers.

```
<stmt>      → declare <ws>+ <ident> (<ws>+ <option>)*
<ident>     → $ <letter>+
<letter>    → 'a' | 'b' | … | 'z' | 'A' | 'B' | … | 'Z'
<ws>        → ' '
<option>    → <mode> | <scale> | <precision> | <base>
<mode>      → real | complex
<scale>     → fixed | floating
<precision> → single | double
<base>      → binary | decimal
```

The table identifies the lexical unit types (tokens) of the grammar.

Token #	Description
1	declare
2	$
3	identifier
4	real
5	complex
6	fixed
7	floating
8	single
9	double
10	binary
11	decimal

[1] Homework is due right before class.

METROPOLITAN STATE UNIVERSITY℠

Mathematical & Computer Science Department
Principles of Programming Languages - Spring 2020

Write a lexical analyzer for the language described by the grammar. Feel free to use the Python code discussed in class and available at [https://github.com/thyagomota/20SCS3210]. You are also allowed to use any other PL. However, if you are using a PL different than Python or Java you will need to give me specific instructions on how to run your code (which version of the PL did you use, what specific compiler, IDE, etc.). If I cannot run your code I cannot grade your work! Note that you are only required to submit the source code (e.g.: `lex.py` or `lex.java` for example).

The output of your lexical analyzer should be a list of pairs containing a lexical unit followed by its token number, in the order of their appearance. Below are some source codes (with expected outputs) for you to try. You can download them from here.

source1.dec

```
declare $speed real fixed double decimal
```

output:

```
declare Token.DECLARE
$speed Token.IDENTIFIER
real Token.REAL
fixed Token.FIXED
double Token.DOUBLE
decimal Token.DECIMAL
```

source2.dec

```
declare $acceleration complex floating
```

output:

```
declare Token.DECLARE
$acceleration Token.IDENTIFIER
complex Token.COMPLEX
floating Token.FLOATING
```

source3.dec (one tab after “$altitude”, new line after “double”)

```
declare $altitude	double

binary floating
```

output:

```
declare Token.DECLARE
$altitude Token.IDENTIFIER
double Token.DOUBLE
binary Token.BINARY
floating Token.FLOATING
```

source4.dec (lexical analyzer doesn't care about the order of tokens)

```
real declare fixed $speed double decimal
```

output:

```
real Token.REAL
declare Token.DECLARE
fixed Token.FIXED
$speed Token.IDENTIFIER
double Token.DOUBLE
decimal Token.DECIMAL
```

source5.dec (unrecognizable symbol)

```
declare speed real fixed double decimal
```

output:

```
Exception: Lexical Analyzer Error: unrecognized symbol found!
```