CSE340 Fall 2017 Project 1: Lexical Analysis

Due: **Friday, September 1, 2017 by 11:59 pm MST**

The goal of this project is to give you hands-on experience with lexical analysis. You will extend the provided lexical analyzer to support more token types. The next section lists all new token types that you need to implement.

1. Description

Modify the lexer to support the following 3 token types:

```
REALNUM   = (pdigit digit*) DOT digit digit* + 0 DOT digit* pdigit digit*
BASE08NUM = ((pdigit8 digit8*) + 0) (x) (08)
BASE16NUM = ((pdigit16 digit16*) + 0) (x) (16)
```

Where

```
pdigit   = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9
digit    = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9
pdigit8  = 1 + 2 + 3 + 4 + 5 + 6 + 7
digit8   = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7
pdigit16 = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + A + B + C + D + E + F
digit16  = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + A + B + C + D + E + F
```

Note that `NUM` and `DOT` are already defined in the lexer, but here are the regular expressions for the sake of completeness (`DOT` is a single dot character, the quotes are used to avoid ambiguity):

```
NUM = (pdigit digit*) + 0
DOT = '.'
```

The list of valid tokens including the existing tokens in the code would be as follows. This list should be used to determine the token, if the input matches more than one regular expression.

```
1. IF          8. DIV          15. RBRAC         22. GTEQ
2. WHILE       9. MULT         16. LPAREN        23. DOT
3. DO         10. EQUAL        17. RPAREN        24. NUM
4. THEN       11. COLON        18. NOTEQUAL      25. ID
5. PRINT      12. COMMA        19. GREATER       26. REALNUM
6. PLUS       13. SEMICOLON    20. LESS          27. BASE08NUM
7. MINUS      14. LBRAC        21. LTEQ          28. BASE16NUM
```

2. Instructions

Follow these steps:

- Download the `lexer.cc`, `lexer.h`, `inputbuf.cc` and `inputbuf.h` files accompanying this project description. Note that these files might be a little different from the code you've seen in class or elsewhere.
- Add your code to the files to support the token types listed in the previous section.
- Compile your code using GCC compiler on `CentOS 7`. You will need to use the `g++` command to compile your code in a terminal window. See section 4 for more details on how to compile using GCC.

Note that you are required to compile and test your code in CentOS 7 using the GCC compiler. You are free to use any IDE or text editor on any platform, however, using tools available in CentOS (or tools that you could install on CentOS) could save time in the development/compile/test cycle.

- Test your code to see if it passes the provided test cases. You will need to extract the test cases from the zip file and run the test script `test1.sh`. See section 5 for more details.
- Submit your code on the course submission website before the deadline. You can submit as many times as you need. Make sure your code is compiled correctly on the website, if you get a compiler error, fix the problem and submit again.

Keep in mind that

- You should use C/C++, no other programming languages are allowed.
- All programming assignments in this course are individual assignments. Students must complete the assignments on their own.
- You should submit your code on the course submission website, no other submission forms will be accepted.
- You should familiarize yourself with the CentOS environment and the GCC compiler. Programming assignments in this course might be very different from what you are used to in other classes.

3. Evaluation

The submissions are evaluated based on the automated test cases on the submission website. Your grade will be proportional to the number of test cases passing. If your code does not compile on the submission website, you will not receive any points.