COSC 4785
Compiler Construction I
Programming Project 1, Flex

1. This is a *get-your-feet-wet* assignment using flex. The assignment is worth 50 points. Flex generates a C or C++ code file named (by default) either "lex.yy.c" or "lex.yy.cc," because that is the name. I do not know why. We will change that behavior and and OURS will be named "program1_lex.cpp." The C++ lexer we create will require that you include <FlexLexer.h> in the main program.

2. When using *flex* for C++ input files and desiring C++ compatible output, use *flex++*. Additional use the *–warn* option. That means that to generate "program1_lex.cpp" from the command line we do

   ```
   $> flex++ --warn program1.lpp
   ```

   And if there are any warnings, FIX THEM!!!!!

3. In order to change the output file name you need to include an option in the declaration section. Make sure that it is NO enclosed in a %{ %} block.

   ```
   %option outfile=program1_lex.cpp
   ```

   This needs to start at the left margin.

4. You will be writing a very simple program, all contained in just a few files. It will do a simple lexical analysis of a text input file. This will read from *standard input* and write to *standard output*. You MUST test this and make sure that it compiles and works. Assuming that your program works, I can compile your source program. You do not need to use *make* but *make* will simplify your life.

5. You will use C++ code for the scanners. The flex input file will be named "program1.lpp." This is to differentiate it from C code input files which usually use the ".l" file extension. This will have NO user code section.

6. You will create a *main* program in the file "program1.cpp." In it, you must declare a variable of type *yyFlexLexer*. This will be your "scanner" and one of its members is the function *yylex()*.

```
// program1.cpp

#include<FlexLexer.h>

int main()
{
  yyFlexLexer myScanner;
  int rtn;

  //initialization code

  while((rtn=myScanner.yylex()) != 0) {
    // loop code goes here
  }

  //more code

  return 0;

}
```

7. The program will output information in the following manner.

```
Line    Column  Type    Length  Value
1       1       7       2       It
1       3       1       1
1       4       7       3       was
1       7       1       1
1       8       7       3       the
```

The first column heading starts at column 1 (left margin). Each of the following headings are separated by a tab. There will be one row for every lexeme (token) recognized. Each column will be positioned just like the headers. Please do not print blank lines, extra characters, garbage, fancy things, anything but what I have listed.The two solid lines are for clarity and not to be included in your output.

8. The "type" (one of ten) in column 3 will be one of the following:

   (a) 0 = space ( )

   (b) 1 = tab (\t)

   (c) 2 = punctuation (.,?;"'()[]) This is the set of characters: period, comma, question mark, semicolon, double quote, single quote, left paren, right paren, left bracket, right bracket, left brace, right brace.

   (d) 3 = operator (+ − ∗/% =~) That last one is the tilde.

   (e) 4 = integer (one or more digits)

   (f) 5 = floating point number (digits followed by a period followed by digits)

   (g) 6 = scientific number (floating point number followed by e/E, an optional +/-, and digits). Examples:

   - 42E01
   - 1e17
   - 42.7e09
   - 9e+08
   - 2.3E-45

   (h) 7 = word (starts with an _ or letter and followed by any number of _ or digits or letters)

   (i) 8 = newline (\n)

   (j) 9 = An error, something NOT recognized because it is not listed above.

9. Note that the return value from the *yylex()* function that indicates it has reached the end of the input is **0** so you might have to do something to make sure you do not return 0 accidentally.

10. The length is the number of characters that were matched for this lexeme. The line and column are where, in the input, the match started.

11. A *value* (the characters actually matched) is printed for everything BUT space, tab, and newline. In the case of space or tab you will match more than one character at a time if there are multiple spaces or tabs. Do not match multiple newlines at once. You have to be careful because the *flex* matching is always greedy, that is it matches as much as possible when using ∗ or + operators.

12. You really must ask questions. Do not come to me day it is due to tell me that you could not figure how to start the assignment, that you could not log into a Linux machine, that you could not figure out how to compile the program, or that *stack**overflow*** or Stack**Exchange** did not have anything like this.