HOMEWORK #2:

Lexical Analyser using Flex

Due Date: **Friday, October 30th, 11:59.59pm**

Description:

For this assigment, you will write a lexical analyser using FLEX, in order to recognize a variety of tokens. Your program should output information about each lexeme it encounters.

Tokens:

Your lexical analyser should recognize the following tokens:

- Integers (INTCONST) non-empty sequences of digits optionally preceded with either a '+' or '–' sign.
- Decimal (DECCONST) numbers are Integers followed by a '.', followed by a non-empty sequence of digits. (e.g. `3.14`, `00.01`, `123.0`).
- Scientific (SCICONST) numbers are Decimal numbers followed by character '`E`', followed by a non-zero integer. (e.g. `12.0E4`, `1.23E-6`).
- Hexadecimal (HEXCONST) are non-empty sequences of digits or the characters '`A`', '`B`', '`C`', '`D`', '`E`' or '`F`' followed by the suffix '`H`'. (e.g. `12AD0H`, `123H`, `1A2B3CH`,).
- Keywords, (KEYWORD) specific strings that form the language. For this homework we will consider the the following keywords: '`if`', '`else`', '`func`', '`let`', and '`while`'.
- Identifiers (IDENT) are strings that consists of a letter followed by zero or more letters or digits; and that are not hexadecimal numbers (e.g. `x`, `size`, `name`, `p3`, `rval`).
- String Constants (STRCONST) are strings that consists of a double quote ' " ' followed by zero or more letters or digits or spaces, followed by another double quote ' " ' (e.g. `"hello"`, `"size"`, `"The Quick Brown Fox"`).
- Operators, (OPERATOR) the symbols '+', '–', '*', and '/'.

Your lexical analyzer should also identify and ignore comment, which start with the character '`%`' and run to the end of the line. Your lexical analyser should also keep track of the number of lines processed.

Submission:

Submit through the UNIX systems using the command ‘`cssubmit 3500 a 2`’. Submit a single file ‘`mylexer.l`’. Your file will be compiled, run and tested using the following chain of commands:

```
flex mylexer.l
g++ lex.yy.c -o lexer.ex
lexer.ex < inputFileName
```

Output:

The output of your lexical analyzer should match the sample output.

Sample Input and Output:

Input

```
while some func input + -1234 %what about this?
*/- 0123 -99 + x camelCase &&^
%%% yet another comment
print if flex func 203.978 -22.4 + "30x2" ' !
ABCH FFF  123.456   %% Here be dragons.
1+2 3+4>t "a
bc"
5 #@ 12.53E231 2B or not toBE1 78E / -42.. "another str constant"
```

Output

```
TOKEN: KEYWORD    LEXEME: while
TOKEN: IDENT      LEXEME: some
TOKEN: KEYWORD    LEXEME: func
TOKEN: IDENT      LEXEME: input
TOKEN: OPERATOR   LEXEME: +
TOKEN: INTCONST   LEXEME: -1234
TOKEN: OPERATOR   LEXEME: *
TOKEN: OPERATOR   LEXEME: /
TOKEN: OPERATOR   LEXEME: -
TOKEN: INTCONST   LEXEME: 0123
TOKEN: INTCONST   LEXEME: -99
TOKEN: OPERATOR   LEXEME: +
TOKEN: IDENT      LEXEME: x
TOKEN: IDENT      LEXEME: camelCase
```

```
TOKEN: ?           LEXEME: &
TOKEN: ?           LEXEME: &
TOKEN: ?           LEXEME: ^
TOKEN: IDENT       LEXEME: print
TOKEN: KEYWORD     LEXEME: if
TOKEN: IDENT       LEXEME: flex
TOKEN: KEYWORD     LEXEME: func
TOKEN: DECCONST    LEXEME: 203.978
TOKEN: DECCONST    LEXEME: -22.4
TOKEN: OPERATOR    LEXEME: +
TOKEN: STRCONST    LEXEME: "30x2"
TOKEN: ?           LEXEME: '
TOKEN: ?           LEXEME: !
TOKEN: HEXCONST    LEXEME: ABCH
TOKEN: IDENT       LEXEME: FFF
TOKEN: DECCONST    LEXEME: 123.456
TOKEN: INTCONST    LEXEME: 1
TOKEN: INTCONST    LEXEME: +2
TOKEN: INTCONST    LEXEME: 3
TOKEN: INTCONST    LEXEME: +4
TOKEN: ?           LEXEME: >
TOKEN: IDENT       LEXEME: t
TOKEN: ?           LEXEME: "
TOKEN: IDENT       LEXEME: a
TOKEN: IDENT       LEXEME: bc
TOKEN: ?           LEXEME: "
TOKEN: INTCONST    LEXEME: 5
TOKEN: ?           LEXEME: #
TOKEN: ?           LEXEME: @
TOKEN: SCICONST    LEXEME: 12.53E231
TOKEN: INTCONST    LEXEME: 2
TOKEN: IDENT       LEXEME: B
TOKEN: IDENT       LEXEME: or
TOKEN: IDENT       LEXEME: not
TOKEN: IDENT       LEXEME: toBE1
TOKEN: INTCONST    LEXEME: 78
TOKEN: IDENT       LEXEME: E
TOKEN: OPERATOR    LEXEME: /
TOKEN: INTCONST    LEXEME: -42
TOKEN: ?           LEXEME: .
TOKEN: ?           LEXEME: .
TOKEN: STRCONST    LEXEME: "another str constant"
8 lines processed.
```

-

Hint.l

```
                          /* ---- PROLOGUE ---- */
%{
#include <iostream>
using namespace std;

int no_lines = 0;
%}
                         /* ---- DEFINITIONS ---- */
%option noyywrap
DIGIT     [0-9]

%%                       /* ---- REGULAR EXPRESSIONS ---- */

[ \t]           ;
\n              { no_lines++; }
{DIGIT}+        { cout << "Found an number: " << yytext << endl; }
[a-zA-Z0-9]+    { cout << "Found a string: " << yytext << endl; }

%%                       /* ---- EPILOGUE ---- */

int main()
{
    cout << "Hello FLEX" << endl;
    yylex();
    cout << "Done!" << endl;
    return 0;
}
```