HOMEWORK #2:

Lexical Analyser using Flex

Due Date: **Friday, March 25th, 11:59.59pm**

Description:

For this assignment, you will write a lexical analyser using `flex`, in order to recognize a variety of token types. 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`,).
- Binary (BINCONST) are non-empty sequences of digits '0', and '1' followed by the suffix 'B'. (e.g. `10110B`, `101B`, `001100B`,).
- Keywords, (KEYWORD) specific strings that form the language. For this homework we will consider the the following keywords: '`if`', '`else`', '`func`', '`let`', '`print`' 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 comments, 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`'.
Put your **name** in your source file.
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 -lfl -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 00B 1010101 B "a
bc"
5 #@ 12.53E231 2B or not toBE1 111B
78E / -42.. "another str constant"
```

Output

```
#0: TOKEN: KEYWORD    LEXEME: while
#1: TOKEN: IDENT      LEXEME: some
#2: TOKEN: KEYWORD    LEXEME: func
#3: TOKEN: IDENT      LEXEME: input
#4: TOKEN: OPERATOR   LEXEME: +
#5: TOKEN: INTCONST   LEXEME: -1234
#6: TOKEN: OPERATOR   LEXEME: *
#7: TOKEN: OPERATOR   LEXEME: /
#8: TOKEN: OPERATOR   LEXEME: -
```

```
#9: TOKEN: INTCONST   LEXEME: 0123
#10: TOKEN: INTCONST   LEXEME: -99
#11: TOKEN: OPERATOR   LEXEME: +
#12: TOKEN: IDENT      LEXEME: x
#13: TOKEN: IDENT      LEXEME: camelCase
#14: TOKEN: OPERATOR   LEXEME: &
#15: TOKEN: OPERATOR   LEXEME: &
#16: TOKEN: ?          LEXEME: ^
#17: TOKEN: KEYWORD    LEXEME: print
#18: TOKEN: KEYWORD    LEXEME: if
#19: TOKEN: IDENT      LEXEME: flex
#20: TOKEN: KEYWORD    LEXEME: func
#21: TOKEN: DECCONST   LEXEME: 203.978
#22: TOKEN: DECCONST   LEXEME: -22.4
#23: TOKEN: OPERATOR   LEXEME: +
#24: TOKEN: STRCONST   LEXEME: "30x2"
#25: TOKEN: ?          LEXEME: '
#26: TOKEN: ?          LEXEME: !
#27: TOKEN: HEXCONST   LEXEME: ABCH
#28: TOKEN: IDENT      LEXEME: FFF
#29: TOKEN: DECCONST   LEXEME: 123.456
#30: TOKEN: INTCONST   LEXEME: 1
#31: TOKEN: INTCONST   LEXEME: +2
#32: TOKEN: INTCONST   LEXEME: 3
#33: TOKEN: INTCONST   LEXEME: +4
#34: TOKEN: OPERATOR   LEXEME: >
#35: TOKEN: IDENT      LEXEME: t
#36: TOKEN: BINCONST   LEXEME: 00B
#37: TOKEN: INTCONST   LEXEME: 1010101
#38: TOKEN: IDENT      LEXEME: B
#39: TOKEN: ?          LEXEME: "
#40: TOKEN: IDENT      LEXEME: a
#41: TOKEN: IDENT      LEXEME: bc
#42: TOKEN: ?          LEXEME: "
#43: TOKEN: INTCONST   LEXEME: 5
#44: TOKEN: ?          LEXEME: #
#45: TOKEN: ?          LEXEME: @
#46: TOKEN: SCICONST   LEXEME: 12.53E231
#47: TOKEN: INTCONST   LEXEME: 2
#48: TOKEN: IDENT      LEXEME: B
#49: TOKEN: IDENT      LEXEME: or
#50: TOKEN: IDENT      LEXEME: not
#51: TOKEN: IDENT      LEXEME: toBE1
#52: TOKEN: BINCONST   LEXEME: 111B
#53: TOKEN: INTCONST   LEXEME: 78
#54: TOKEN: IDENT      LEXEME: E
#55: TOKEN: OPERATOR   LEXEME: /
#56: TOKEN: INTCONST   LEXEME: -42
#57: TOKEN: ?          LEXEME: .
#58: TOKEN: ?          LEXEME: .
#59: TOKEN: STRCONST   LEXEME: "another str constant"
9 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;
}
```