CS 256 – Programming Languages and Translators
Assignment 1

- This assignment is due by 1 p.m. on Friday, January 31, 2014
- This assignment will be worth 2% of your grade
- You are to work on this assignment by yourself

Basic Instructions

For this assignment, you are to use **flex** to create a C++ program that will perform lexical analysis for a small programming language called MFPL (described below). If you flex file is named `mfpl.l`, you should be able to compile and execute it on one of the campus Linux machines (such as rc##ucs213.managed.mst.edu where ## is 01-24) using the following commands:

```
flex mfpl.l
g++ lex.yy.c -o mfpl_lexer
mfpl_lexer < inputFileName
```

where inputFile name is the name of some input file to be lexically analyzed.

Your program should output information about each token that it encounters in the input source program. You will need a token type `UNKNOWN` for any tokens that cannot be properly categorized as an operator, keyword, identifier, etc. Sample input and output are given at the end of this document.

Your program should continue processing tokens from the input file until the end of the file is detected. Note that you program should not do anything other than recognize tokens (e.g., no syntax checking, etc.), as that is the only purpose of lexical analysis.

MFPL Programming Language

For now, you only need to be concerned with the tokens in the MFPL programming language.

- Identifier (`IDENT`)
  - Starts with a letter or underscore, followed by any number of letters, digis, and/or underscores.
- Integer Constant (`INTCONST`)
  - A sequence of one or more digits, optionally preceded by a + or a `-`. Don't worry about the size limit of integer constants.
- String Constant (`STRCONST`)
  - A string constant is a series of characters that is wrapped in **either** single or double quotes (matching, of course). You do not need to worry about the distinctions for escaped characters (such as `\n, \", \t`, you will not encounter strings such as: `"\""`). A string that starts on a line must end on the same line.

- Keywords
  - The keywords are: **let*, if, lambda, print, input, and, or, not, t, nil**. The language is case sensitive; the keywords must be lower case (otherwise, they should be recognized as identifiers)
- Operators
  - the operators are: `+, -, *, /, <, >, <=, >=, =, /=`
- Parentheses
  - Your code needs to recognize parenthesis. The lexical analyzer should not check for matching parentheses, just distinguish between the two kinds (left and right).
- Comments
  - Similar to the C++ style comments (//), except a semicolon (;) is used instead of //. Comments should be scanned over and ignored (not included in your output).

In summary, your program should report the following types of tokens (and lexemes): **LETSTAR, LAMBDA, INPUT, PRINT, IF, LPAREN, RPAREN, ADD, MULT, DIV, SUB, AND, OR, NOT, LT, GT, LE, GE, EQ, NE, IDENT, INTCONST, STRCONST, T, NIL, UNKNOWN**

We will be using an automated script to grade your submissions; you **must** use exactly these token names; otherwise you will lose points.

Sample Input and Output

You should output the token and lexeme information for every token processed in the input file even if the lexeme is not unique for the token (for example, the lexem for ever LAMBDA token will be lambda).

Given below is some sample input and output. With the exception of whitespace and capitalization, the output produced by your program should be identical for this input!

Input

```
let* ( some lambda input + -1234 ;what about this?
*/- 0123 99 + x _underscore_this) &&^
;;; yet another comment
print if flex  let 203978 -2 + "30x^2" % !
1+2
3 + 4 > t
-5+ 6
"a
bc"
7
5 >= nil and 4 or not toBe1 <<==78
/= -42 'another str constant'
```

Output

```
TOKEN: LETSTAR    LEXEME: let*
TOKEN: LPAREN     LEXEME: (
TOKEN: IDENT      LEXEME: some
TOKEN: LAMBDA     LEXEME: lambda
TOKEN: INPUT      LEXEME: input
TOKEN: ADD        LEXEME: +
TOKEN: INTCONST   LEXEME: -1234
TOKEN: MULT       LEXEME: *
TOKEN: DIV        LEXEME: /
TOKEN: SUB        LEXEME: -
TOKEN: INTCONST   LEXEME: 0123
TOKEN: INTCONST   LEXEME: 99
TOKEN: ADD        LEXEME: +
TOKEN: IDENT      LEXEME: x
TOKEN: IDENT      LEXEME: _underscore_this
TOKEN: RPAREN     LEXEME: )
TOKEN: UNKNOWN    LEXEME: &
TOKEN: UNKNOWN    LEXEME: &
TOKEN: UNKNOWN    LEXEME: ^
TOKEN: PRINT      LEXEME: print
TOKEN: IF         LEXEME: if
TOKEN: IDENT      LEXEME: flex
TOKEN: IDENT      LEXEME: let
TOKEN: INTCONST   LEXEME: 203978
TOKEN: INTCONST   LEXEME: -2
TOKEN: ADD        LEXEME: +
TOKEN: STRCONST   LEXEME: "30x^2"
TOKEN: UNKNOWN    LEXEME: %
TOKEN: UNKNOWN    LEXEME: !
TOKEN: INTCONST   LEXEME: 1
TOKEN: INTCONST   LEXEME: +2
TOKEN: INTCONST   LEXEME: 3
TOKEN: ADD        LEXEME: +
TOKEN: INTCONST   LEXEME: 4
TOKEN: GT         LEXEME: >
TOKEN: T          LEXEME: t
TOKEN: INTCONST   LEXEME: -5
TOKEN: ADD        LEXEME: +
TOKEN: INTCONST   LEXEME: 6
TOKEN: UNKNOWN    LEXEME: "
TOKEN: IDENT      LEXEME: a
TOKEN: IDENT      LEXEME: bc
TOKEN: UNKNOWN    LEXEME: "
TOKEN: INTCONST   LEXEME: 7
TOKEN: INTCONST   LEXEME: 5
TOKEN: GE         LEXEME: >=
TOKEN: NIL        LEXEME: nil
TOKEN: AND        LEXEME: and
TOKEN: INTCONST   LEXEME: 4
TOKEN: OR         LEXEME: or
TOKEN: NOT        LEXEME: not
TOKEN: IDENT      LEXEME: toBe1
TOKEN: LT         LEXEME: <
TOKEN: LE         LEXEME: <=
TOKEN: EQ         LEXEME: =
TOKEN: INTCONST   LEXEME: 78
```

```
TOKEN: NE          LEXEME: /=
TOKEN: INTCONST    LEXEME: -42
TOKEN: STRCONST    LEXEME: 'another str constant'
```

You may find it useful to use the `diff` command to compare your output with the sample output posted on the course website. To do this, first flex and compile your program, and run

```
flex mfpl.l
g++ lex.yy.c -o mfpl_lexer
mfpl_lexer < hw1_mfpl.txt > myOutput.out

diff myOutput.out hw1_mfpl.txt.out --ignore-space-change --side-by-side --ignore-case\
 --ignore-blank-lines
```

To learn more about the `diff` command, see `http://ss64.com/bash/diff.html`

Submission

You will submit this assignment using `cssubmit`. Your ***single*** lexer file ***must*** have a `.l` extension. If it does not, you will receive a zero for this assignment. From the directory containing the `.l` file, you will run
`cssubmit 256 a 1`
on the cs213 Linux machines. This will collect your submission and submit it to me. You may submit as many times as you desire; only your last submission will be graded (previous submissions are overwritten). **READ** the output of cssubmit; it may have changed since you last used it.