FILS
Course: Compiler Techniques

Homework #3
Lexical Analysis

1. Suppose we have the following tokens: (1) the keyword **if** , (2) the keyword **while**, (3) the keyword **when**, and (4) identifiers consisting of strings of letters and digits, beginning with a letter. Show:
a) The NFA for these tokens, and
b) The DFA for these tokens.

2. Write a Lex (or flex) program that copies a file, replacing each nonempty sequence of white space by a single blank.

3. Write a *Lex* (or *flex*) specification for a lexical analyzer for a simple calculator language. This language contains integer numbers, the operators plus, minus, multiply, and divide, and parentheses for grouping. Additionally, the symbol "=" is in the language to terminate an expression. These symbols and their corresponding token names are shown in the table below.

Symbol in Language	Token Name
integer number (e.g., "0", "12", "1719")	NUMBER XXXX [where XXXX is the number itself]
+	PLUS
-	MINUS
*	MULT
/	DIV
(	L_PAREN
)	R_PAREN
=	EQUAL

The calculator language itself is very simple. There is only one type of phrase in the language: "Expression=", where "Expression" is defined

```
21=
2+3*4=
(2+3)*4=
30/3/5=
-250/50=
(10+2)*-(3-5)=
40-20-5=
4*(1/1-3/3+10/5-21/7+45/9-121/11+26/13-45/15+34/17-38/19+63/21-1/1+2002/1001)=
```

Note, however, that lexical analysis only scans for valid tokens in the calculator language, not valid expressions. The parsing phase (phase 2, the next phase of the class project) is where sequences of tokens will be checked to ensure that they adhere to the specified language grammar. Thus, for this exercise which deals only with lexical analysis, even such phrases as */2=10++-(+ and ***101***())(- can still be tokenized successfully.

Task 1: Create a Lex (or *flex*) specification to recognize tokens in the calculator language. Print out an error message and exit if any unrecognized character is encountered in the input. Use flex to compile your

specification into an executable lexical analyzer that reads text from standard-in and prints the identified tokens to the screen, one token per line.
Task 2: Enhance your Lex (or *flex*) specification so that input text can be optionally read from an input file, if one is specified on the command line when invoking the lexical analyzer.
Task 3 (optional): Enhance your Lex (or *flex*) specification so that in addition to printing out each encountered token, the lexical analyzer will also count the following.
The number of integers encountered
The number of operators encountered: +, -, *, /
The number of parentheses encountered: (,)
The number of equal signs encountered
The total counts should be printed to the screen after all input text has been tokenized. The counts need not be printed if an unrecognized character is encountered in the input (since the lexical analyzer should just terminate after issuing the error message).

Note:
You can find Lex resources here:
http://dinosaur.compilertools.net/lex/

A free variant for Lex, which runs on Windows , is *flex*:
http://flex.sourceforge.net/
http://dinosaur.compilertools.net/flex/