Project 1
Implementation of a Lexical Analyzer
Due October 28, Wednesday, 2020

Note: You need to COMPLETELY follow the instructions in this sheet.

1. **Problem:**

In this assignment you are requested to use the tool **flex** to write a lexical analyzer for a small language **Lotus**. **Lotus** contains the following six types of tokens:

1. This token type includes all identifiers. An identifier is a sequence of letters and digits; the first character must be a letter. All identifiers are returned as the same token. Namely, there is only one token in this token type. However, there are an infinite number of possible lexemes for this token. To distinguish these lexemes, the string of each identifier is returned as the attribute value of the token.
2. This token type includes the following seven keywords:
   else exit int if read while write
   These keywords are identifiers reserved for special use, and may not be used as general identifiers. Each keyword is returned as a different token. Namely, there are eight different tokens in this token type. Since there is only one lexeme for each token, no attribute value is returned for each keyword.
3. This token type includes all integer constants. An integer constant consists of a sequence of digits. All integer constants are returned as the same token. Namely, there is only one token in this token type. However, there are an infinite number of possible lexemes for this token. To distinguish these lexemes, the integer value of each integer constant is returned as the attribute value of the token.
4. This token type includes the following twenty-one operators:
   + - * / % == != > >= < <= && || ! = ; , ( ) { }
   Each operator is returned as a different token. Namely, there are twenty-one different tokens in this token type. Since there is only one lexeme for each token, no attribute value is returned for each operator.
5. This token type includes all white spaces. A white space may be a blank (‘ ’), a tab (‘\t’), or a newline (‘\n’). These white spaces are mainly served to separate tokens. These tokens are ignored and are not returned.
6. This token type includes all comments. A comment starts with the characters

// and ends with a newline. These tokens are ignored and are not returned.

The lexical analyzer needs to print the following error message

“Lexical error: line %d: unknown character %c”

when an unknown character is encountered. The error message is written to stderr and should include the line number where the error is detected.

The lexical analyzer is generated automatically by **flex** as the function yylex(). You should add a main function to read a **Lotus** program from stdin and repeatedly call the yylex() function to recognize tokens. When the lexical analyzer is executed with the option “-s”, it should print the recognized tokens to stdout. The format for printing tokens is as follows:

Identifiers: print "Identifier:", a space, then the string of the identifier.
Keywords: print "Keyword:", a space, then the string of the keyword.
Integer constants: print "Integer Constant:", a space, then the value of the integer constant.
Operators: print "Operator:", a space, then the string of the operator.

Each token is printed on a separate line. The executable lexical analyzer should be named scanner.

A sample Lotus program `test1` is given as follows:

```
// A program to sum 1 to n
sum( )
{
        int    n;
        int    s;

        read n;
        if (n < 0) {
                write -1;
                exit 0;
        } else {
                s = 0;
        }
        while (n > 0) {
                s = s + n;
                n = n – 1;
            }
            write s;
        }
```

The stdout output of test1 using –s option is as follows:

```
Identifier: sum
Operator: (
Operator: )
Operator: {
Keyword: int
Identifier: n
Operator: ;
Keyword: int
Identifier: s
Operator: ;
Keyword: read
Identifier: n
Operator: ;
Keyword: if
Operator: (
Identifier: n
Operator: <
Integer Constant: 0
Operator: )
Operator: {
Keyword: write
Operator: -
Integer Constant: 1
Operator: ;
Keyword: exit
Integer Constant: 0
Operator: ;
Operator: }
Keyword: else
Operator: {
```

```
Identifier: s
Operator: =
Integer Constant: 0
Operator: ;
Operator: }
Keyword: while
Operator: (
Identifier: n
Operator: >
Integer Constant: 0
Operator: )
Operator: {
Identifier: s
Operator: =
Identifier: s
Operator: +
Identifier: n
Operator: ;
Identifier: n
Operator: =
Identifier: n
Operator: -
Integer Constant: 1
Operator: ;
Operator: }
Keyword: write
Identifier: s
Operator: ;
Operator: }
```

Another sample Lotus program test2 is given as follows:

```
// A program to sum 1 to n
sum( )
{
    int   n;
    int   s;
```

```
	read n;
	if (n < 0) {
		write -1;
		exit 0;
	} else {
		s := 0;
	}
	while (n > 0) {
		s = s + n;
		n = n - 1;
	}
	write ##s;
}
```

The stdout output of test2 using –s option is as follows:

```
Identifier: sum
Operator: (
Operator: )
Operator: {
Keyword: int
Identifier: n
Operator: ;
Keyword: int
Identifier: s
Operator: ;
Keyword: read
Identifier: n
Operator: ;
Keyword: if
Operator: (
Identifier: n
Operator: <
Integer Constant: 0
Operator: )
Operator: {
Keyword: write
Operator: -
Integer Constant: 1
Operator: ;
Keyword: exit
Integer Constant: 0
Operator: ;
Operator: }
Keyword: else
Operator: {
Identifier: s
Operator: =
Integer Constant: 0
Operator: ;
Operator: }
Keyword: while
Operator: (
Identifier: n
Operator: >
Integer Constant: 0
Operator: )
Operator: {
Identifier: s
Operator: =
Identifier: s
Operator: +
Identifier: n
Operator: ;
Identifier: n
Operator: =
Identifier: n
Operator: -
Integer Constant: 1
Operator: ;
Operator: }
Keyword: write
Identifier: s
Operator: ;
Operator: }
```

The stderr output of test2 is as follows:

```
Lexical error: line 12: unknown cheracter :
Lexical error: line 18: unknown cheracter #
Lexical error: line 18: unknown cheracter #
```

2. Handing in your program:

You should provide a make file named “makefile” to build this assignment. See online manual make(1) for more information. The executable file should be named “scanner,” namely,

```
gcc -o scanner $(OBJ).
```

To turn in the assignment, upload a compressed file proj1.zip containing makefile, *.l, *.h, and *.c to Ecourse site.

3. Grading

The grading is based on the correctness of your program and the programming style of your program. The correctness will be tested by a number of test cases.

To facilitate the grading of teaching assistants, you should test your program on the machine csie1.

It is best to incrementally build your program so that you always have a partially-correct working program. It is also best to construct a shell script to systematically test your program.