Assignment # 1
COP 3402 Fall 2011
Lexical Specification of KOOL Language

In this course, we will write a compiler for a simple programming language called **KOOL** (Knights Object-Oriented Language). The language is powerful but trimmed down from typical languages in a variety of ways to make the exercise easier without limiting the learning outcomes. Your current task is to write a lexical analyzer for **KOOL**.

KOOL has a relatively small set of keywords. At present (and life can change over the semester), these reserved words are:
void int bool string null true false class new this extends
for do while until if then else break return
print println read readln

Special characters in **KOOL** are:
`+ - * / % = += -= *= /= %= < <= > >= == # ||`
`&& ! ; , . [ ] ( ) { }`
Notice that neither " nor \ are special characters as they can only appear as part of a string.

An integer constant can either be specified in decimal (base 10) or binary (base 2).
A decimal integer is a sequence of decimal digits. A binary integer must begin with 0B or 0b and is followed by a sequence of binary digits.

An identifier is a sequence of letters, digits, and underscores, starting with a letter. At present the language is case insensitive, but I could change my mind so I would isolate the case sensitivity to the lexical phase (in other words have it alter all identifiers and keywords to lower case so the keyword lookup and, eventually, the symbol table lookup is even unaware of this decision). Identifiers that match in their first 8 letters are considered the same, so you will want to truncate them to this maximum length, if necessary. Anything that is a blank or that collates to being less than a blank is whitespace. These characters separate tokens. Keywords and identifiers must be separated by whitespace or a token that is neither a keyword nor an identifier. Thus, whiledo is an identifier rather than two keywords. MyData(123#x) might make no sense syntactically, but is viewed lexically as IDENT(mydata), LPAREN, INTEGER(123), NOTEQUAL, IDENT(x), RPAREN.

A string constant is a sequence of characters enclosed in double quotes. Strings can contain any character except a newline or double quote, unless the double quote is preceded by an escape character (\). The escape character also needs to be escaped. If a \ appears vbefore any character other than a " or \, then it is just ignored. A string cannot span multiple lines nor can it exceed 80 characters, including the quotes and escape characters – this means strings are limited to 78 characters each.

An end-of-line terminated comment is started by // and extends to the end of the line. Any symbol is allowed after the // with the end-of line signifying its end. C-style comments start with /* and end with the first subsequent */. Any symbol is allowed in a C-style comment except the sequence */ which ends the current comment. C-style comments can span multiple lines, but also can end in the middle of a line.

Your lexical analyzer must report on each token, giving the token type and, where appropriate, the token value. I will create a reference solution that you can use to check your answers out.

Turn in a C or C++ program (source) that runs on a standard C or C++ compiler (no Windows sugar; just a console application). Name your source with your First Initial followed by your Last Name followed by "Asn1" and the C or C++ extension. E.g., Steven would be SZittrowerAsn1.c or perhaps SZittrowerAsn1.cpp.

Due Dates:

Due on Monday, September 19 before the end of day (11:59PM)

Turn in must be done via Webcourses (it will be set up by next week). Only the source is to be turned in. We will provide some test cases by next week, but we will also run many other test cases as part of the grading.

Grading Criteria

EXECUTION POINTS (60 POINTS)

The student's program should compile, run and produce the correct output.

Test Case 1 (*hanoi.kool*) – 20 points

o Every line containing incorrect tokens is -1 point

Test Case 2 (*errors.kool*) – 20 points

o Main focus here is that the lexical analyzer correctly identifies and prints the errors

o There are 5 errors printed in this program, each one is worth 3 points

o Overall correctness of the other tokens is 5 points

Test Case 3 (*biggie.kool*) – 20 points

o We are not checking every token

o Overall soundness of the output is important

IMPLEMENTATION DETAILS (30 POINTS)

We are looking for correct functionality (at least conceptually). The actual structure or naming of the functions is irrelevant.

Distinguishes identifiers and keywords – 10 points

Distinguishes ints, strings and bools– 5 points

Correct handling of strings and comments – 10 points

Handling of special characters – 5 points

CODE POINTS (10 POINTS)

Code formatting (readable indentation, variable names, etc.) – 5 points

Overall input / output is setup correctly – 5 points

There is an optional up to 5 point bonus if your coding style demonstrates elegance.

NOTES:

If the assignment is submitted up to 24 hours late, there is a 10% deduction from the final grade; submitting between 24 – 48 hours late results in a 20% deduction.

If the program does not compile or crashes, at least 25 points will be deducted from the execution points; if only a small fix is required 5 points are taken off. (This can happen if you use some non-standard C or C++ feature that we can access quickly.

IDE:

I talked to Remo who GTA-ed last semester and found that most students preferred Code::Blocks and so that will be the preferred environment, although you may also use Eclipse (my fave) or Visual Studio. Using any other environment will take special arrangement with one of the GTAs.