CS 316 Fall 2009
All projects in this course must be completed **individually and independently**. All programs must be written in Java or C++.

PROJECT 2: Implementation of Lexical Analyzer
Due: 10/27/09, Tuesday, Midnight

The objective of this project is to implement a lexical analyzer that accepts the 22 token categories given in Project 1 **plus the keywords "if" and "else", both in lowercase letters only**. These keywords cannot be used as identifiers, but can be parts of identifiers, like "iff" and "delse". The implementation should be based on the (correct!) DFA constructed in Project 1. If your DFA wasn't correct, you may use this example DFA. Your lexical analyzer program should clearly separate the driver and the state-transition function so that the driver will remain invariant and only state-transition functions will change from DFA to DFA. The enumerated type or the integer type is suggested for representation of states. As discussed in one of the lectures, the class object representation of states would also be fine – create a generic root class *State* and derive subclasses corresponding to the DFA states. An interesting aspect of this method is that codes of the state-transition function are provided in the subclasses by taking advantage of inheritance polymorphism.

The following keyword recognition method is adequate for this project.

1. Create two additional DFA states *if* and *else*.
2. The DFA initially accepts the two keywords as identifiers.
3. Each time the DFA accepts an identifier, check to see if it is *if* or *else*, and if so, move the DFA to the corresponding state.

Alternatively, you may explicitly incorporate the step-by-step keyword recognition process into the state-transition function in the manner described in Problem 4 of Exercise Set #2. Extra credit will be given **provided** this method is implemented **correctly**. I suggest not to use this method unless you feel able to handle the increased number of states and state transitions correctly. (Many real-world lexical analyzers use this method for superior efficiency.)

The lexical analyzer program is to read an input text file, extract the tokens in it, and write them out one by one on separate lines. Each token should be flagged with its category. The output should be sent to an output text file. Whenever invalid tokens are found, error messages should be printed, and the reading process should continue. The program should read the input/output file names interactively from the terminal or as external arguments to the main function.

You may modify one of these sample Java programs into your solution; if you do so, modify the comments suitably as well:

Conditional Version. The state-transition function is implemented by "2-dimensional conditionals".
Array Version. The state-transition function is implemented by an array.

Here's a sample set of test input/output files:

in1 | out1
in2 | out2
in3 | out3
in4 | out4

You should make your own additional input files to test the program.

In Project 3 you will construct a top-down parser using this lexical analyzer.

Since the purpose of this project is to reinforce the understanding of lexical analyzer construction based on finite automata, you are **not** allowed to use any library functions/tools for lexical analysis (like the Java StringTokenizer).

Submission

Your source program must be emailed to yukawa@cs.qc.cuny.edu with the subject header:

CS 316, Project 2, your full name

The due date is 10/27/09, Tuesday, Midnight. Please include the program as attachment **without compression into .zip, .rar, etc**.