Distribute the following questions across the members of your group. You will share your solutions (and most importantly the *method* of your solutions) during the next lecture period. Divide up the questions so that **each** question has at least two solutions from different group members.

1. Implement the DFA optimizing algorithm discussed today in lecture. You may use your favorite programming language,[1] and you will be expected to compare and verify the results of your coding with your other group members.

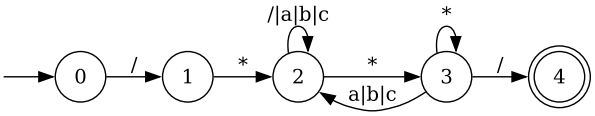
   The input for your program will be the non-empty lines of a regular file (or `stdin`). Each line has the following whitespace separated token format (leading and trailing whitespace should be ignored):

   1. A plus or minus symbol: plus means the state is an accepting state, minus means the state is a non-accepting state.
   2. A non-negative integer written without a positive sign. This first integer on the line is the state ID for the row being read.
   3. The remainder of the line are the columns of the transition table for this row. Again, all entries are whitespace delimitted; an entry for each column must be either a non-negative state ID or the letter E. An integer is the state ID transitioned to on input of the appropriate character; and E (error) means there is no edge from this state for the appropriate character.

   There must be the same number of transition table columns for each row; or the file is considered flawed. The starting state for the DFA must have the ID of zero.

   Here is an example of a C/C++ multiline comment DFA over the alphabet { `/, *, a, b, c` } (although the number of states cannot be reduced by the *MergeStates* algorithm).

-	0	1	E	E	E	E
-	1	E	2	E	E	E
-	2	2	3	2	2	2
-	3	4	3	2	2	2
+	4	E	E	E	E	E

   Your program should be able to read a formatted file representing a DFA, and output a file of the DFA after being optimized. You may read and write files in whatever user interface you prefer: command line argument, `stdin` and `stdout` pipes, or (???) GUI.

2. Take a critical hand in the solution of question 1 by providing (**promptly!**) a set of input files for 3–4 different DFAs with mergable states. These DFAs should **not have dead or unreachable states**, but you should be sure they have at least two "levels" of state merges. Provide the final "optimized" result as well (in a different file).

3. Read §3.7–§3.7.3 of text and summarize the practical issues of scanner implementation discussed, along with plausible solutions.

4. Read §3.7.4–§3.7.6 of text and summarize the practical issues of scanner implementation discussed, along with plausible solutions. (This question is not a copy and paste error of the previous, the portion of reading is different.)

[1] But it might be wise to make it one suitable for course projects.