1. (a) Given the regular expression,
   \[(a|b)*(c|d)\] 
   write, by hand in pseudo-code (or a real language if you prefer), a procedure that reads in a character at a time and recognizes strings in the language described by this regular expression.

   (b) Since we can write recognizers for regular languages by hand, why did we even bother to discuss NFA’s and DFA’s in class?

   (c) Write a regular expression for which you think it would be difficult to hand-code a recognizer (although obviously not impossible). Explain why it would be difficult, and then construct an NFA to recognize it.

2. (a) Give the canonical collection of sets of LR0 items for the following grammar:
   \[S \rightarrow aA\beta \mid a\beta \]
   \[A \rightarrow d\beta \]
   \[B \rightarrow c \]

   (b) The closure operation used to convert a NFA to a DFA is similar to the closure operation used to construct an SLR parse table. Explain briefly the purpose of each of these closure operations and give an intuitive explanation of why they are similar.

   (c) Give an intuitive explanation, along with a simple example, of why - following a reduce operation in an SLR parser - the goto operation looks at the state symbol below the topmost non-terminal.

3. In C, there’s a do-while construct of the form:
   \[
   \text{do}
   \{
   \quad \text{statements}
   \}
   \text{while (expression)}
   \]

   Write pseudo-code for the \text{gen()} \ function, within an intermediate code generator, to generate 3-address intermediate code for a do-while loop. Assume that \text{gen()} \ is already implemented for expressions and blocks of statements. \text{gen()} \ applied to an expression emits the quads for the expression and returns the name of the variable containing the result of the expression. For a block of statements, assume that \text{gen()} \ just emits the quads for the all the statements in the block. (Note: If you prefer to generate intermediate code in the style described in the dragon book, rather than as I described in class, you can do so.)

4. (a) Why is it, for the source language that you implemented in your compiler, that global variables can be placed in the data section but local variables need to be placed on the stack?

   (b) Draw the layout of an activation record for the following procedure in the source language that you implemented.
procedure f(x, y: integer);
var i: integer;
a: array [3..10] of integer;
begin
...
end

(c) If the intermediate instruction

\[ x := a[i] \]

appears within the intermediate code for the procedure f (above), what assembly code should be generated for this intermediate instruction? (You don’t have to write perfect Intel X86 assembly code, it just has to be at the level of assembly code rather than intermediate code.)

5. (a) Using your own words (not the book’s), define the terms basic block and flow graph.

(b) What is it about a basic block that makes it amenable to being considered for optimization?

(c) A common peephole optimization is to convert, for example,

\[
\text{jmp L1} \\
\text{...} \\
\text{L1: jmp L2}
\]

to

\[
\text{jmp L2} \\
\text{...} \\
\text{L1: jmp L2}
\]

Give three reasons that this might improve program performance on a modern (hint: pipelined) processor, and briefly explain each reason.