1. (10 points) Using the method described in the textbook or in class, convert the regular expression
   \((a(b|c))^*\)
   into an equivalent NFA (which can have \(\varepsilon\)-moves).
   **Answer:** The NFA should look something like this (vertex labels are not required):

   Slight variations are permitted. Note that this is not the smallest equivalent NFA. The smallest has only two states and is not a correct answer, because it does not follow the construction method in the book or in class.

2. (15 points) In C, the binary infix + and − operators both associate from left to right and have equal precedence. The binary infix = operator associates from right to left and has lower precedence than the + operator. Give a context-free grammar in abstract form with start symbol \(E\) that parses C expressions built from terminals \(v\) (for variables) and \(c\) (for constants) and involving only the operators +, −, = and parentheses. Do not restrict the kinds of subexpressions allowed with an operator (for example, \(v + (c - v) = c + v\) is allowed). Your grammar should be suitable for conflict-free LALR parsing and should reflect the precedences and associativities of the operators.
   **Answer:**

   \[
   \begin{align*}
   E & \rightarrow T = E \mid T \\
   T & \rightarrow T + F \mid T - F \mid F \\
   F & \rightarrow (E) \mid c \mid v
   \end{align*}
   \]
3. (20 point total) Consider the following syntax-directed definition with start symbol $E$. (The grammar describes a very small fragment of LISP that can nonetheless serve as a general-purpose programming language.) Here, $f$, $g$, $h$, and $k$ are binary functions.

$$
E \rightarrow (E_1 E_2) \\
E \rightarrow (v . E_1) \\
E \rightarrow v
$$

Note that $E$ has two attributes: $E.x$ and $E.y$. The terminal $v$ also has an $lval$ attribute.

(a) (5 points) Draw a parse tree for the token stream

$$
((v . (v v))(v . (v v)))
$$

**Answer:**

![Parse Tree](image)

(b) (10 points) Redraw a fresh copy of your parse tree above and give the corresponding dependency graph for the attributes in this tree. Place the vertices near their associated nodes of the tree. (Recall that for any two attributes $u$ and $v$, there is a directed edge $u \rightarrow v$ just in the case that $v$ directly depends on $u$.)

**Answer:**

![Dependency Graph](image)

(c) (5 points) Give a possible order of evaluation for the attributes above by placing numbers 1, 2, 3, . . . next to each node. (A 1 means the node can be evaluated first, a 2 means the node can be evaluated second, etc.)

**Answer:** See the diagram above. There are other correct answers.
4. (20 points) Consider the following grammar with start symbol $E'$:

$$
E' \to E \\
E \to v \\
E \to v \ (L) \\
L \to E \\
L \to L, E
$$

Using the method described in class or the text, construct the set of states for a simple LR(0) (SLR) parser for this grammar, defining the transition function at the same time. Note that in class I denoted the transition function as $\text{trans}$. The textbook denotes the same function as $\text{goto}$ on page 224.

To ensure a unique correct answer, you must stick to the following rules of order, which mirror the order I used for my example in class:

(a) Give each state as a list of LR(0) items, omitting the brackets.

(b) Give the start state first, and denote it by $s_0$. Denote other states $s_1, s_2, \ldots$ in the order they are constructed.

(c) List the kernel items first in each state. List additional nonkernel items in the order that they enter the closure.

(d) For each $i \geq 0$, define all transitions out of $s_i$ before defining those out of $s_{i+1}$.

(e) When finding the transitions out of a state, or computing a closure, consider each item of the state in the order you listed it.

(f) Do not list the empty set as a state.
Answer:

\[ s_0 : \]
\[ E' \rightarrow .E \]
\[ E \rightarrow .v \]
\[ E \rightarrow .v(L) \]
\[ s_1 = \text{trans}(s_0, E) : \]
\[ E' \rightarrow E. \]
\[ s_2 = \text{trans}(s_0, v) : \]
\[ E \rightarrow v. \]
\[ E \rightarrow v.(L) \]
\[ s_3 = \text{trans}(s_2, '(') : \]
\[ E \rightarrow v.(L) \]
\[ L \rightarrow .E \]
\[ L \rightarrow .L,E \]
\[ E \rightarrow .v \]
\[ E \rightarrow .v(L) \]
\[ s_4 = \text{trans}(s_3, L) : \]
\[ E \rightarrow v(L.) \]
\[ L \rightarrow L,.E \]
\[ s_5 = \text{trans}(s_3, E) : \]
\[ L \rightarrow E. \]
\[ s_2 = \text{trans}(s_3, v) \]
\[ s_6 = \text{trans}(s_4, ')) : \]
\[ E \rightarrow v(L) \]
\[ s_7 = \text{trans}(s_4, ')) : \]
\[ L \rightarrow L,.E \]
\[ E \rightarrow .v \]
\[ E \rightarrow .v(L) \]
\[ s_8 = \text{trans}(s_7, E) : \]
\[ L \rightarrow L,E. \]
\[ s_2 = \text{trans}(s_7, v) \]

5. (10 points) Describe briefly and in general terms how to use liveness (next use) analysis to warn about uninitialized variables.

**Answer:** An un-initialized variable warning should be given if a variable is live at the entry point of the program (or function or procedure). This means that there is some computational path in which the variable will be used before it is set. Liveness analysis can determine this.
6. (25 points total) Consider the following three-address code (line numbers added):

1  L1:  i := a
2   goto L5
3  L2:  s := s + 1
4   goto L5
5  L3:  if j <= i then goto L1
6  L4:  i := i + 1
7   goto L1
8  L5:  n := n - 1
9   if n > 0 then goto L3

Assume that control enters at line 1 and that there are no other entry points.

(a) (5 points) Describe the basic blocks $B_1, B_2, \ldots$ by giving an inclusive range of line numbers for each block.

**Answer:**

<table>
<thead>
<tr>
<th>Block</th>
<th>from line</th>
<th>to line</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$B_2$</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$B_3$</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$B_4$</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>$B_5$</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

(b) (5 points) Draw the flow diagram as a directed graph, labeling the nodes $B_1, B_2, \ldots$. Give dangling arrows both for the entry point and for the exit point of the code as a whole.

**Answer:**

(c) (5 points) Using the strict definition of a loop as defined in class and in the text, list sets of vertices that constitute loops. Label any inner loop(s) as such.

**Answer:** The only loop is \{$B_1, B_2, B_3, B_4, B_5$\}, and thus this is an inner loop. The set \{$B_1, B_3, B_4, B_5$\} is not a loop because it does not have a unique entry point (the set can be entered at either $B_1$ or $B_3$).

(d) (10 points) Reduce the size of the control flow graph as much as possible to produce an equivalent control flow graph.

**Answer:** Note that removing the unreachable $B_2$ allows $B_1$ and $B_5$ to be coalesced into a single block.

\[ B_3 \rightarrow B_4 \]

\[ B_1; B_5 \rightarrow \]