Problem 1. Apply PRE to the following program. Assume that variables $w$, $v$, $x$, and $r$ are used in other portions of the code (not shown), possibly in the same basic block. You do not need to show the intermediate steps, just show the optimized code. You may add basic blocks to the flow graph, but only show those that are not empty in your solution (existing basic blocks are not empty, even if they appear to be).

Solution graph:
Problem 2.  

1. Draw the dominator tree for the graph.

Solution tree:

![Dominator Tree Diagram]

If entry and exit were included, there should be an edge from entry to \( b_1 \) and from \( b_{10} \) to exit.

2. What are the back edges and natural loops of the graph?

- \( b_8 \rightarrow b_2 : \{b_2, b_4, b_5, b_7, b_8\} \)
- \( b_{10} \rightarrow b_1 : \{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}\} \)
- \( b_4 \rightarrow b_2 : \{b_2, b_4\} \)
- \( b_7 \rightarrow b_5 : \{b_5, b_7, b_8\} \)

3. A node \( d \) strictly dominates a node \( n \) if \( d \) dominates \( n \) and \( d \neq n \). The dominance frontier of a basic block \( b \), \( DF(b) \), is the set of all blocks \( n \) such that (1) \( b \) dominates an immediate predecessor of \( n \) and (2) \( b \) does not strictly dominate \( n \). This is the boundary of the flow graph wherein the dominance of \( b \) terminates.

(a) Let \( \text{DOM\_BY}(b) \) be the set of all basic blocks dominated by a basic block \( b \) and \( \text{Succ}(b) \) be the set of blocks \( s \) such that there exists an edge \( b \rightarrow s \). Express \( DF(b) \) in terms of \( \text{DOM\_BY} \) and \( \text{Succ} \).

\[
DF(n) = \bigcup_{d \in \text{DOM\_BY}(n)} \text{Succ}(d) - \left( \text{DOM\_BY}(n) - \{n\} \right)
\]

(b) Can a block be in its own dominance frontier? If not, provide a brief explanation. Otherwise, provide an example of such a block in the graph. Yes
**Problem 3.** For the following program,

1. Use dominance frontiers and the algorithm described to generate its minimal SSA form. 
   **GRADING NOTE:** students who elected to further improve the SSA form by removing \( \phi \) functions with single arguments, for example, will also receive full credit. The critical \( \phi \) functions must be present to receive full points.

2. Show the results of constant propagation over the resulting SSA form. 
   **The only thing that changes is** \( x = y \ast 4 \) becomes \( x = 20 \).
**Problem 4.** For the following control flow graph, perform register allocation. Show the results of the following steps.

1. Assign each definition and use of a variable to a live range. For example, all instances of $A$ must be replaced with either $A_1$ or $A_2$ to signify one of two live ranges.

2. Draw the register interference graph with lines between nodes that represent live ranges.

3. Apply the heuristic-based register allocation algorithm with for a machine with 3 registers. Show the resulting “stack” of registers and show which ones, if any, are marked as spilled.

4. Assign the live ranges to registers.
1. Assign each definition and use of a variable to a live range. For example, all instances of $A$ must be replaced with either $A_1$ or $A_2$ to signify one of two live ranges.

```
A1 = 5;
EXIT
B1 = 3;
C1 = 2;
ENTRY
D1 = A1+2;
A2 = 9;
D1 = A1+2;
A2 = 9;
E1 = 2*B1;
E1 = 2*C1;
D2 = D1+A2;
E2 = E1+A2;
return D2 + E2;
```

2. Draw the register interference graph with lines between nodes that represent live ranges.
3. Apply the heuristic-based register allocation algorithm with a machine with 3 registers. Show the resulting “stack” of registers and show which ones, if any, are marked as spilled.

There are multiple answers. One possible answer is as follows:
E2 D2 E1 A1 B1 A2 C1 D1

4. Assign the live ranges to registers. There are many possible answers. One possible answer is as follows:

<table>
<thead>
<tr>
<th>Live Range</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>R0</td>
</tr>
<tr>
<td>A2</td>
<td>R2</td>
</tr>
<tr>
<td>B1</td>
<td>R1</td>
</tr>
<tr>
<td>C1</td>
<td>R1</td>
</tr>
<tr>
<td>D1</td>
<td>R0</td>
</tr>
<tr>
<td>D2</td>
<td>R0</td>
</tr>
<tr>
<td>E1</td>
<td>R1</td>
</tr>
<tr>
<td>E2</td>
<td>R1</td>
</tr>
</tbody>
</table>
Problem 5. Observe the control-flow graph below and answer the following questions.

1. What is the largest number of overlapping live ranges seen at any program point? 2 live ranges.

2. What is the minimum number of registers you need in order to successfully assign all variables without spilling? 3 registers are required. An answer of 2 registers is also accepted.

3. Now, imagine that, as part of register allocation, you can insert `MOVE x y` operations that copy a value from register x to another register y. Can you allocate all of the variables with fewer registers than before? If you insert a move on any of these edges, you can “split” a live range and thus allocate with only two registers. If the answer to the previous part was 2, then an acceptable answer for this part is that you can’t improve.
Problem 6. You are given the task of optimizing the code given below. You are only allowed to run the following four optimization techniques in any order and multiple times if necessary:

- PRE (as discussed in class)
- Constant Propagation (as discussed in class)
- Copy Propagation (as discussed in Section 9.1.5 of the textbook)
- Dead Code Elimination (liveness analysis, as discussed in class and in Homework 2)

You cannot modify the control flow graph or eliminate empty basic blocks, except to preprocess it for PRE. As in joeq, assume that an expression can take both registers and constants.

1. What is the order in which you should execute them to produce the best optimized code by running a minimum number of analyses? Answer in terms of a general advantageous ordering: something along at the lines of constant prop first, because constants will not change later; then DCE to remove spurious uses that would confuse PRE into
putting expressions too early; then copy propagation to maximize the number of identical expressions; then PRE, followed by copy propagation again to get rid of copies introduced by PRE; dead code elimination should be executed last, because all other passes introduce dead code.

Answer in terms of the specific control flow graph: constant propagation and DCE will get rid of almost all code. You could end up doing a lot of passes with a different ordering (for example, constant prop, PRE, copy prop, PRE, DCE, etc.). Other answers will receive incremental partial credit.

2. What is the final optimized program?
Solution program: