1. We want to optimize the following program snippet written in the Decaf language by eliminating common subexpressions:

```plaintext
i = callout("get_int_035");
j = i + 1;
k = i;
l = k + 1;
```

where the `get_int_035` function reads an integer form standard input and returns it.

(a) (5 points) What does the optimized code look like when we use value numbering to find common subexpressions?

(b) (5 points) What does the optimized code look like when we use available expression analysis to find common subexpressions?
2. We want to compute reaching definitions for the following program:

```
L1:   i = m - 1;
L2:   j = n;
L3:   a = x;
    do {
L4:       i = i + 1;
L5:       j = j - 1;
        if (i < j) {
L6:            a = y;
        } else {
L7:                i = z;
        }
    } while(j > m);
```

(a) (5 points) Draw the control flow graph of this program.
(b) (5 points) Compute GEN[n] and KILL[n] for each basic block n.

\[
\begin{align*}
\text{GEN[1]} &= 11111111 \\
2 &= 00011100 \\
3 &= 00000010 \\
4 &= 00000001 \\
\text{KILL[1]} &= 00011111 \\
0 &= 11000001 \\
2 &= 00100000 \\
3 &= 00100000 \\
4 &= 10110000 \\
\end{align*}
\]

(c) (5 points) Set up data-flow equations (IN[n] = \ldots and OUT[n] = \ldots) for each basic block n.

\[
\begin{align*}
\text{IN[1]} &= 11111111 \\
0 &= \text{OUT[2]} \cup \text{OUT[5]} \\
2 &= \text{OUT[2]} \\
3 &= \text{OUT[2]} \\
4 &= \text{OUT[2]} \\
5 &= \text{OUT[3]} \cup \text{OUT[4]} \\
6 &= \text{OUT[5]} \\
\text{OUT[n]} &= \text{GEN[n]} - \text{KILL[n]} \cup \text{GEN[n]} \\
\end{align*}
\]

\(\text{IN or OUT not both}\)

(d) (5 points) Find out the solution of the data-flow equations.

\[
\begin{align*}
\text{IN[1]} &= 00000000 \\
\text{IN[2]} &= 11111111 \\
0 &= 00000000 \\
2 &= 00101111 \\
3 &= 00101111 \\
4 &= 00101111 \\
5 &= 00101111 \\
\end{align*}
\]
3. Design a data-flow analysis that determines which expressions are very busy at each program point. An expression is very busy at a program point \( p \) if along every path from \( p \) the expression is always used before a redefinition of any of the variables occurring in it.

(a) (5 points) Is the data-flow analysis a forward analysis? Or a backward analysis?

Backward

(b) (5 points) Draw the Hasse diagram of the lattice used in the analysis, assuming that there are just three expression \((e_1, e_2 \text{ and } e_3)\) in the target program.

(c) (5 points) What is the confluence operator?

\[ a \cup b = a \]

\[ a \cap b = b \]

(d) (5 points) What is the initial value of IN[entry] or OUT[exit], depending on your answer to (a).
4. (a) (10 points) Prove that the "greater than or equal" relation ($\geq$) is a partial order on the set of integers with both positive and negative infinity ($\mathbb{Z} \cup \{-\infty, \infty\}$).

1. $a \leq a$
2. $a \leq b \land b \leq c \Rightarrow a \leq c$
3. $a \leq b \land b \leq a \Rightarrow a = a$

True for all $\mathbb{Z}$

Consider $a = \infty$, then 1, 2, 3 all hold, $b = c = \infty$

Consider $b = \infty$, then 1, 2, 3 all hold, $c = \infty$

Consider $c = \infty$, then 1, 2, 3 all hold, $a = \infty$

Similar case analysis for $-\infty$

(b) (5 points) Draw the Hasse diagram for it, and mark its greatest element ($\top$) and least element ($\bot$).
5. If your proof is not correct, you will not get a score even when your yes/no answer is correct.

(a) (8 points) If a lattice has a greatest or least element, is it always unique? Prove it or disprove it.

   Yes.  
   
   Proof: contradiction.
   
   Assume 2 greatest elements \( g_1 \), \( g_2 \)
   
   Consider \( g = g_1 \lor g_2 \). \( g_1 \leq g \), \( g_2 \leq g \) so
   
   \[ \begin{align*}
   \text{either } g &= g_1 \quad \text{or } g = g_2 \quad \text{or } g \neq g_1, g \neq g_2, \quad \text{so}
   
   g_1, g_2 \text{ not greatest.}
   \end{align*} \]

(b) (7 points) Does a complete lattice always have both greatest and least elements? Prove it or disprove it.

   Yes.  
   
   Consider \( \lor \) \( L \) = greatest element
   
   \( \land L \) = least element
6. Alice designed a mysterious data-flow analysis on programs written in the following language:

\[ \begin{align*}
S & \rightarrow \text{id := E | if (E < E) then S else S} \\
E & \rightarrow \text{id + id | c}
\end{align*} \]

where \( \text{id} \) and \( c \) denote a variable and a non-negative integral constant. It is known that she modeled a program state at each program point as a function that maps each variable to its value. For example, the \( \{x \mapsto 1, y \mapsto 2\} \) program state means that \( x \) and \( y \) have 1 and 2 at the program point, respectively. Also, her abstraction function is as follows:

\[ \text{AF}[\{\text{id}_1 \mapsto v_1, \text{id}_2 \mapsto v_2, \ldots, \text{id}_n \mapsto v_n\}] = \{\text{id}_1 \mapsto (v_1 \% 3), \text{id}_2 \mapsto (v_2 \% 3), \ldots, \text{id}_n \mapsto (v_n \% 3)\} \]

where \( \% \) is the remainder operator. Let's restore her data-flow analysis from the abstract function.

(a) (5 points) Define the base lattice for her data-flow analysis. Note that the actual lattice is defined using element of the base lattice as follows:

\[ \{\text{id}_1 \mapsto \bar{v}_1, \text{id}_2 \mapsto \bar{v}_2, \ldots, \text{id}_n \mapsto \bar{v}_n\} \]

where \( \bar{v}_1, \bar{v}_2 \) and \( \bar{v}_n \) are element of the base lattice.

(b) (10 points) Define the transfer functions for the following basic blocks.

\[ f(\text{id}) = \begin{cases} 
\text{id} & \text{if id = x} \\
\text{f}(\text{id}) & \text{otherwise}
\end{cases} \]

\[ f'(\text{id}) = \begin{cases} 
(f(y) + x \% 3) & \text{if id = x} \\
\text{f}(\text{id}) & \text{otherwise}
\end{cases} \]
(c) (5 points) Will the analysis always produce the meet-over-path solution? Justify your answer by proof sketch or example. If your justification is not correct, you will not get a score even when your yes/no answer is correct.

**No.** Have control flow impression.

\[ x = 1 \]
\[ y = 2 \]

\[ i < n \]
\[ x = 2 \]
\[ y = 1 \]

\[ z = x + y \] \[ z = T, \cdots \] in analysis

\[ z = 0, \cdots \] is m0p solution