Instructions: The exam is taken without any material beside writing material and in silence. Answer the following problems, trying to be as clear and as accurate as possible. Take the time to read each statement carefully before answering. If you need more space, write on the back of your test and indicate it clearly.

Problem 1 (35 p.) Observe the flow graph on p. 7 and answer the following, assuming A is the entry node:

1. Part 1 - Dominators

(a) What are the nodes dominated by A?

All of them (including A itself), since the entry node is always in the path that starts at the entry node.

(b) What are the nodes dominated by C?

E, F. I or M are not dominated by C, since one can do A → B → D → H → I (→ M).

And C itself!

(c) What are the nodes dominating K?

K, G, D, B, A.

(d) Give the definition of an immediate dominator.

A node X is an immediate dominator of a node Y if

- X dominates Y
- X does not dominate any other dominator of Y.

(e) Draw the immediate dominator tree.
2. Part II - Loops

(a) The set \( \{ D, H \} \) is a loop. Give
- its header(s): \( D \) (because it dominates \( H \))
- its latch(es): \( H \) (because it has a back edge to the header)
- its exiting edge(s): \( D \rightarrow G, H \rightarrow I \)
- its exit block(s): \( G \) and \( I \).

(b) List the other loop(s) present in this graph.

\[ \{ B, D, G, H \} \text{ is another loop (containing the loop } \{ D, H \} \). \]

\[ \{ C, F, J \} \text{ is not a loop (2 entries), same for } \{ B, D, G \}. \]

\( C \) does not dominate \( J \)!

(c) (but everybody is welcome to try) One can define back edges and loops using dominators:

An edge from node \( W \) to node \( X \) is a back edge if \( X \) dominates \( W \). The body of the loop defined by a back edge from \( W \) to \( X \) includes \( W \) and \( X \), as well as all predecessors of \( W \) (direct and indirect) up to \( X \) (that is, \( X \)'s predecessors are not included).

\textbf{Def. 1} \textit{[Not a loop for Def. 2 (2 entries)]} \textit{(H is a prod. of G should have been included.)}

\textbf{Def. 2} \textit{Prove that the "other" definition of loop we studied in class (i.e. a subset of nodes such that a. it is strongly connected, b. all edges from outside of it points to the same node inside of it, c. is the maximal such subset) is equivalent to this one, or illustrate how they diverge.}

\textbf{Def 1} \Rightarrow \textbf{Def 2} \textit{Support: \{ H, I_1, ..., I_n, L_1, ..., L_k \} is a loop according to Def. 1, with \( H \) its header, I_1, ..., I_n its "inner node," and L_1, ..., L_k its latches. Then: \( L \) is strongly connected: I_1, ..., I_n are predecessors of L_1, ..., L_k, hence there is a path from them to L_1, ..., L_k, and L_1, ..., L_k are connected to H by their back edges.}

\( \text{To be clear: we will prove that } L \text{ respects a), b), c) of Def. 2.} \)
Let \( S = \{ S_1, \ldots, S_j \} \) be a set of nodes such that:

a) \( S \cap L = \emptyset \),
b) \( S \cup L \) is strongly connected,

c) no edge from a node not in \( S \cup L \) points to \( S \cup L \setminus H \) (that is, all "outside edge" points to \( H \)).

Our goal is to reach a contradiction, proving that \( L \) is the maximal subset with the properties from Def. 2.

As \( S \cup L \) is strongly connected, for all \( i \in \{ 1, \ldots, j \} \), there exists \( \{ 1, \ldots, k \} \) and a path \( S_i \to \cdots \to S_m \to \cdots \to H \)

We reason by case:

1. \( S_i \) is a predecessor of \( L_m \) but not of \( H \), and hence should belong to \( L \): a contradiction.

2. This case is a bit more involved: as all edges going into \( S \cup L \) must point to \( H \), it implies that \( \partial H \) dominates \( S_i \), and since \( \partial H \) \( H \) is an edge (or at least, a path) from \( S_i \) to \( H \) (since \( S \cup L \) is strongly connected), then \( S \cup L \) is actually an outer loop surrounding the inner loop \( L \). Since maximality is to be read as "\( L \) is not a larger loop containing the loop under study," we have reached a contradiction (\( S \cup L \) is a larger subset satisfying def. 2, but it is itself a loop, & hence not considered to break maximality).

\[ \Rightarrow \text{Hence } L \text{ satisfy def. 2!} \]

\[ \text{Def. 2} \Rightarrow \text{Def. 1} \]

We must now prove that if \( L = \{ H, I_1, \ldots, I_n, L_1, \ldots, L_k \} \) is a loop according to Def. 2, then it satisfies the conditions of Def. 1. All we have to prove is that \( H, I_1, \ldots, I_n, L_1, \ldots, L_k \) are predecessors of \( I_1, \ldots, L_k \) but not of \( H \). Since all edges outside of \( L \) point to \( H \), \( I_1, \ldots, I_n \) cannot be predecessors of \( H \). Since \( I_1, \ldots, I_n \) do not have edges to \( H \) (as they are not latches) & since they are connected to one of \( L_1, \ldots, L_k \) (as the graph \( L \) is strongly connected), they are predecessors of one of \( L_1, \ldots, L_k \), which concludes this part.
Problem 2 (35 p.) Consider the following code, assuming that \texttt{int} variables \(x, y\) and \(z\) have been declared and initialized.

```plaintext
if (x < 10) \{ x = 15; \}
while (x > 10) {
    x--;
    if (y > 0) \{ y = 3; \}
    else \{ y = 5; \}
} // Do nothing
```

1. Give the final values of \(x, y\) and \(z\) assuming the following initial values of \(x, y\) and \(z\):

<table>
<thead>
<tr>
<th>Before:</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>After:</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>10</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0</td>
<td>-1</td>
<td></td>
<td>10</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0</td>
<td>1</td>
<td></td>
<td>10</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

2. Draw the control flow graph of this code, writing the actual code in each basic block.

3. Are there any instructions that can be hoisted (moved from inside the body of the loop to outside of it)?
   If yes, indicate them and where you would move them, if no, explain why.

   \textbf{Nothing can be hoisted unless there is a guard making sure that } x > 10. \textbf{Using a conditional ("if } x > 10\text{"), then all of B2 could be hoisted (either before or after the loop, it makes no difference). (} y = 1\text{ can be purely unmoved, but that's not hoisting, that's dead code elimination).}
4. Using the semi-colon (;) as a "do nothing" instruction, transform your flow graph so that your loop has a pre-header and only one latch (sometimes called a "postbody"). You can abbreviate the code if you want.

5. Convert your flow graph to SSA form, writing the actual code in each basic block.
6. Rotate this loop: transform it(s non-SSA form) into a do...while(...) loop, making sure the semantics is exactly preserved.

```
if (x > 10) {
    do {
        x--; 
        y = 1;
        if (y > 0) y = 3;
        else y = 5;
    } while (x > 10);
}
```

This way, we know the body will be executed only if the condition is met when entering it.

And since we don't know if this condition will be true when we enter the original loop, we need...

7. Optimize this code “as much as you can”, avoiding useless repetition, branchings or assignments.

```
if (x != 10) { x = 10;
    if (y > 0) y = 3;
    else y = 5;
}
```

You can study question 1 again to convince yourself that if x = 10, nothing happens and that if x ≠ 10, • it is possibly incremented to 15 if less than 10, and decremented to 10 in any case,
• the body is executed, and y = 1; is useless as the value of y is always impacted by the if statement.
Problem 3 (30 p.) Consider the code on p. 7 (courtesy of https://anooopsarkar.github.io/compilers-class/llvm-practice.html), and answer the following:

1. What is the return type of `gcd`?
   
   \[ \text{i32, no, integer} \]

2. Explain what the following do:
   
   \[
   \%a1 = alloca i32
   \text{store i32 \%a, i32* \%a1}
   \]
   
   "Allocates memory on the stack frame" for an int, and stores its address in \%a1. This is used for "automatic variables".

3. Explain what the following do:
   
   \[
   \text{br label \%ifstart}
   \]
   
   IV unconditionally transfer the control flow to \%ifstart. It is a branching with only 1 branch.

4. How many basic blocks there is in this code?
   
   \[ 5, all in the same function \]

5. Draw the control flow graph for this code. No need to copy the code, simply use the labels of the blocks.

6. "Retro-translate" this code into C code, knowing that

   The `\text{\%a \%r \text{rem}}` instruction returns the remainder from the signed division of its two operands.

   ```c
   int gcd (int a, int b) {
       if (b == 0) { return a; }
       else { gcd (b, a \% b); }
       return 0;
   }
   ```