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 I - Dominators
   (a) What are the nodes dominated by A?

   (b) What are the nodes dominated by C?

   (c) What are the nodes dominating K?

   (d) Give the definition of an immediate dominator.

   (e) Draw the immediate dominator tree.

2. Part II - Loops

   (a) The set $\{D, H\}$ is a loop. Give

   - its header(s):
   - its latch(es):
   - its exiting edge(s):
   - its exit block(s):

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

   (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).

   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.

Problem 2 (35 p.) Consider the following code, assuming that `int` variables `x`, `y` and `z` have been declared and initialized.

```
if (x < 10) {x = 15;}
while (x > 10) {
  x--;
  y = 1;
  if (z > 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`:

Before:	x	y	z	After:	x	y	z
	1	1	1				
	10	10	10				
	11	0	-1				
	11	0	1				

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.

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.

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

Problem 3 (30 p.) Consider the code on p. 7 (courtesy of https://anoopsarkar.github.io/compilers-class/llvm-practice.html), and answer the following:

1. What is the return type of `gcd`?

2. Explain what the following do:

```
%a1 = alloca i32
store i32 %a, i32* %a1
```

3. Explain what the following do:

```
br label %ifstart
```

4. How many basic blocks there is in this code?

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 `srem` instruction returns the remainder from the signed division of its two operands.

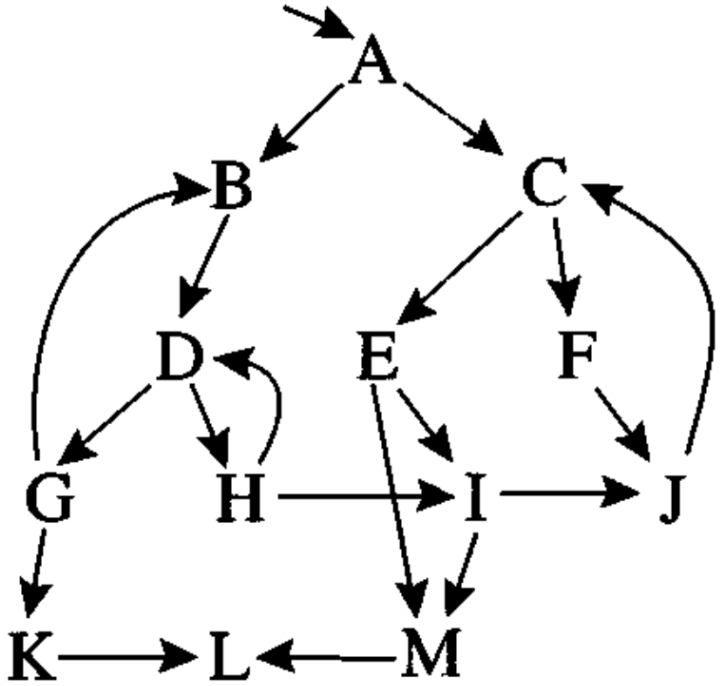

Figure 1: Flow Graph

```
define i32 @gcd(i32 %a, i32 %b) {
    entry:
        %a1 = alloca i32
        store i32 %a, i32* %a1
        %b2 = alloca i32
        store i32 %b, i32* %b2
        br label %ifstart

    ifstart:                                          ; preds = %entry
        %b3 = load i32, i32* %b2
        %eqtmp = icmp eq i32 %b3, 0
        br i1 %eqtmp, label %iftrue, label %iffalse

    iftrue:                                           ; preds = %ifstart
        %a4 = load i32, i32* %a1
        ret i32 %a4
        br label %end

    end:                                              ; preds = %iffalse, %iftrue
        ret i32 0

    iffalse:                                          ; preds = %ifstart
        %b5 = load i32, i32* %b2
        %a6 = load i32, i32* %a1
        %b7 = load i32, i32* %b2
        %modtmp = srem i32 %a6, %b7
        %calltmp = call i32 @gcd(i32 %b5, i32 %modtmp)
        ret i32 %calltmp
        br label %end
}
```