**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?

   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?

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

   And C itself!

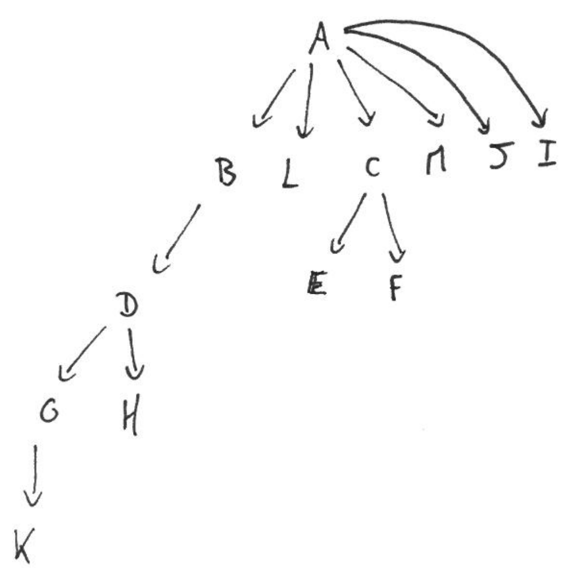
   (c) What are the nodes dominating K? (J is not either, as we can do $A \to B \to D \to H \to I$)

   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.
   - $X \neq Y$

   (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 ede(s):~~ D→G, H→I
- its exit block(s): G and I.

2

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

{B, D, G, H} is another loop (containing the loop {D, H}).

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

↳ C does not dominate J!

Not a loop for Def. 2 (2 "entries") nor a loop for Def. 1. (H is a pred. of G & should have been included).

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

Def. 1 — An edge from node W to node X is a back edge if X dominates ~~N~~ 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).

Def. 2. — 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.

- **Def 1 ⇒ Def. 2.**

Suppose $L = \{H, I_1, \dots, I_n, L_1, \dots, L_k\}$ is a loop according to Def. 1, with H its header, $I_1, \dots, I_n$ its "inner nodes" and $L_1, \dots, L_k$ the latches. Then:

we let loops having multiple back edges being loops. Take $k = 1$ if you would rather not. The proof stays the same.

To be clear: we will prove that L respects a), b) & c) of Def. 2.

- L is strongly connected: $I_1, \dots, I_n$ are predecessors of $L_1, \dots, L_k$, hence there is a path from them to $L_1, \dots, L_k$, and $L_1, \dots, L_k$ are connected to H by their back edges.

- If a node not in L was pointing to $I_1, \dots, I_n, L_1, \dots,$ or $L_k$, then as $I_1, \dots, I_n$ are (indirect) predecessors of $L_1, \dots, L_k$, this node would be a predecessor of one of the $L_i$'s, and should have been in L: a contradiction.

• Let $S = \{S_1, \dots, S_j\}$ 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 $i \in \{1, \dots, j\}$, $\exists\, m \in \{1, \dots, k\}$ and either a path

ⓐ $S_i \cdots \to H \cdots \to L_m$ or a path ⓑ $S_i \cdots \to L_m \cdots \to H$. We reason by case:

ⓑ $S_i$ is a predecessor of $L_m$ but not of $H$, and hence should belong to $L$: a contradiction.

ⓐ This case is a bit more involved: as all edges going into $S \cup L$ must point to $H$, it implies that (i) $H$ dominates $S_i$, and since (ii) there 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 "... that 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).

$\Rightarrow$ Hence $L$ satisfy Def. 2!

---

• <u>Def. 2 $\Rightarrow$ Def. 1</u>

We must now prove that if $L = \{H, I_1, \dots, I_n, L_1, \dots, L_k\}$ is a loop according to Def. 2, then it satisfies the conditions of Def. 1. All we have to prove is that the $I_1, \dots, I_n$ are predecessors of $L_1, \dots, L_k$ but not of $H$. Since all edges outside of $L$ points to $H$, $I_1, \dots, I_n$ cannot be predecessors of $H$. Since $I_1, \dots, I_n$ do not have edges to $H$ (as they are not latches) & since they are connected to one of $L_1, \dots, L_k$ (as the graph $L$ is strongly connected), they are predecessors of one of $L_1, \dots, L_k$, which concludes this part.

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

B1.

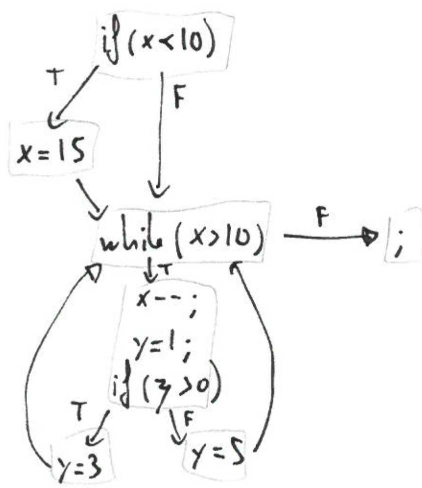
```
if (x < 10) {x = 15;}
while (x > 10) {
  x--;
  y = 1;
  if (z > 0) {y = 3;}
  else {y = 5;}
}
; // Do nothing
```

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 | 3 | 1 |
| | 10 | 10 | 10 | | 10 | 10 | 10 |
| | 11 | 0 | −1 | | 10 | 5 | −1 |
| | 11 | 0 | 1 | | 10 | 3 | 1 |

observe that in this case the value of y did not change: we need to be carefull if we want to hoist y=1 or the if block inside the loop!

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

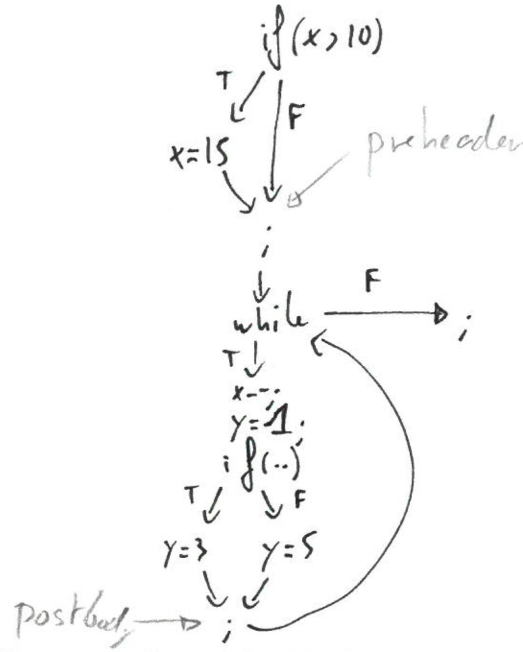



3. Are there any instructions that can be hoisted (moved from inside the body of the loop to outside of it)? If yes, indicate them and where you would move them, if no, explain why.

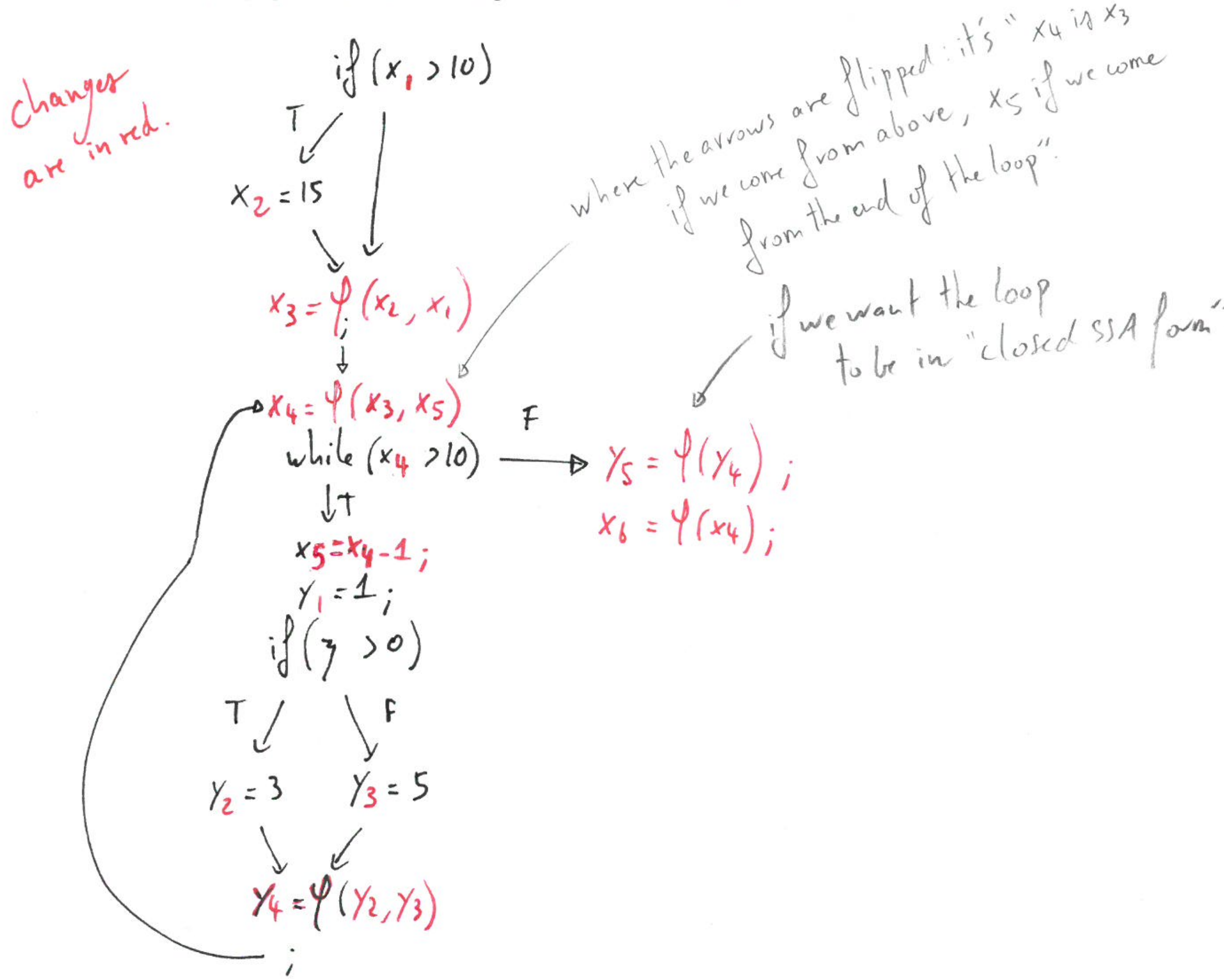
Nothing can be hoisted unless there is a guard making sure that x>10. Using a conditional ("if x>10"), then all of B1 could be hoisted (either before or after the loop, it makes no difference).

(y=1 can be purely removed, 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);
}
```

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

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 $x \neq 10$,

- 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://anoopsarkar.github.io/compilers-class/llvm-practice.html`), and answer the following:

1. What is the return type of gcd? i32, so, integer.

2. Explain what the following do:

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

"Allocates memory on the stack frame" for an int, and stores its adduss in %a1. This is used for "automatic variables".

the value of the argument %a is stored at this newly created adduss.

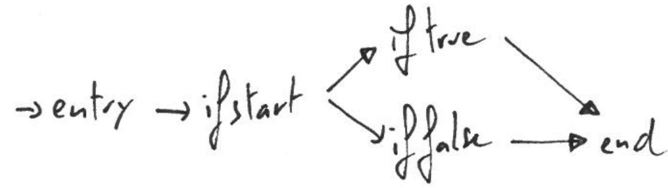
3. Explain what the following do:

```
br label %ifstart
```

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

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

this is the end block, that is actually never reached but added by llvm.