This is an open-book, open-notes, open-laptop, closed-network exam. Please do not post anything on Piazza until the solutions are put up on the class website.

You have 1 hour 20 minutes to work on this exam. The examination has 6 problems worth 70 points. Please budget your time accordingly. Write your answers in the space provided on the exam. If you use additional scratch paper, please turn that in as well.

Your Name: ____________________ SUNet ID: ____________________

The following is a statement of the Stanford University Honor Code:

1. The Honor Code is an undertaking of the students, individually and collectively:

   (a) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;

   (b) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

2. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

3. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

Signature: ____________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>15</td>
<td>25</td>
<td>70</td>
</tr>
</tbody>
</table>
**Problem 1.** True or False? Briefly justify your answer in 1 to 5 lines [9 Points]

1. For a forward monotone dataflow framework, if OUT\([b_1]\) ≤ OUT\([b_2]\) for distinct basic blocks \(b_1\) and \(b_2\), then it must be that OUT\([s_1]\) ≤ OUT\([s_2]\), where \(s_1\) is a successor of \(b_1\), and \(s_2\) is a successor of \(b_2\).

   *False.* \(s_1\) and \(s_2\) may have different transfer function, so no monotonicity relationship can be applied. Even with the same transfer function, \(s_2\) could have other predecessors which, when meet’ed, decrease its value.

2. Given a monotone dataflow framework, if an input flow graph has no cycles, an iterative data flow algorithm will produce the maximum fixed point solution, even if all the interior program points are initialized to bottom. Note that the iterative algorithm may not be visiting the basic blocks in reverse post-order.

   *True,* interior initialization will never be used to compute anything due to acyclic nature of the graph. The transfer function will be applied to the initial condition and answers propagated to the rest of the graph.

3. Natural loops in an irreducible control flow graph are either disjoint or nested.

   *If you assumed that back edges going to the same header are merged into one loop, then this is true. Otherwise, this is false due to the case of two natural loops having the same header.*

**Problem 2.** PRE [6 points]

Can applying PRE to a program create more opportunities for constant propagation? If yes, provide a control flow graph that would benefit from applying PRE before constant propagation. Otherwise, explain why this is not possible.

*Yes,* consider the following CFG:
Problem 3. Register Allocation [6 Points]
Consider the following flow graph:

 ENTRY

 A = 0
 B = 0

 ... = B
 C = ...
 ... = A

 B = ...
 ... = C
 A = ...

 EXIT

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.

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?
   You can use the MOVE to split up the live range of C, with one split interfering with
   A and the other split interfering with B. Thus, you only need 2 registers.
Problem 4. Dominators [9 points]
Consider the following control flow graph:

![Control Flow Graph]

- Draw the immediate dominator tree for the flow graph above.

```
  A
  /\  
 B  G  C
  \  |
   H  D  F
    \  |
     E
```

Solution tree:

- What are the natural loops in the above flow graph? Indicate both the set of blocks and the corresponding back edge. $F \rightarrow C : \{C, D, F, H\}$ and $E \rightarrow C : \{C, D, E\}$

- Is the flow graph reducible? Explain your answer.

No, not all retreating edges in the different DFSTs are back edges.
Problem 5. Partial Redundancy Elimination [15 Points].

Show the result of running partial redundancy elimination. What’s the final optimized flow graph? You don’t need to show the intermediate steps.

Solution:
**Problem 6.** Atomic File Writes [25 points]

Suppose we wish to write a series of data to a text file. In case the program’s execution is interrupted, it is desirable that either the file does not exist, or it contains all the data to be written. One solution is to create a temporary file, write all the data, then rename the temporary file to the permanent file. Assuming that file renames are atomic, the permanent file will either not be created or it is in the fully modified state.

Your task is to create warnings to help a programmer write the code correctly. For simplicity, we assume that files are referred to by their literal names, not by file descriptors or variables. We use the convention that filenames with prefix "tmp_" are temporary; all others are permanent. There are only 3 relevant operations for this task:

- **create_temp_file(t):** creates a temporary file with name t
- **write_to_file(t, c):** writes contents c to a temporary file named t
- **rename(t, p):** renames a temporary file named t with a permanent file name p

![Figure 1: A sample program](image)

You can assume that the program is well-typed, meaning that all the file arguments to the functions have the right prefixes. Also, you can assume that each basic block contains only one instruction. Design an analysis to generate the following warnings in this problem:

(a) Issue Warning I on every write or rename operation that refers to a temporary file that may not have been created.

(b) Issue Warning II on every write or rename operation on a temporary file that may already have been renamed.
You can define one or more dataflow analyses to solve this problem. If you define multiple, answer the following questions for each dataflow analysis.

The way the problem is written, it seems as if create -> rename -> create -> write would throw Warning II for write as the file “may already have been renamed” at this point. However, this warning is extraneous as the file has now been recreated and is thus safe to write to. The solution described below does not throw this warning, but we accepted solutions that did since the problem could be interpreted that way.

1. Is it a forward or backward pass? What is the domain and meet operator? What is the transfer function?

   Forward pass. For every temporary file name \( t \) and basic block \( b \), let the domain, for \( \text{out}[b, t] \) and \( \text{in}[b, t] \) be the powerset of \{uninitialized, created, renamed\}. The meet is \( \cup \) for each file name \( t \). The boundary condition is to set all files to \{uninitialized\}. The transfer function is as below.

   - On \text{create}\_\text{tmp}\_\text{file}(t)\,\text{out}[b, f] = \{\text{created}\} if \( f = t \), otherwise \text{out}[b, f] = \text{in}[b, f].
   - On \text{write}\_\text{to}\_\text{file}(t, c)\,\text{for all } f, \text{out}[b, f] = \text{in}[b, f] \text{ (identity).}
   - On \text{rename}(t, p)\,\text{out}[b, f] = \{\text{renamed}\} if \( f = t \), otherwise \text{out}[b, f] = \text{in}[b, f].

2. Is your framework monotone?

   Yes - the identity function is monotone, and any constant function is monotone.

3. Is it distributive?

   Yes - the identity and constant functions are both distributive.

4. Describe your algorithm for generating each warning.

   Warning I: warn on statements \text{write}\_\text{to}\_\text{file}(t, c) or \text{rename}(t, p) in a basic block \( b \) if \text{uninitialized} \in \text{in}[b, t].

   Warning II: warn on statements \text{write}\_\text{to}\_\text{file}(t, c) or \text{rename}(t, p) in a basic block \( b \) if \text{renamed} \in \text{in}[b, t].

   In a few cases you will get two warnings, which is acceptable.