ECE 468 – Final Exam
December 12, 2011

Name: _____

Purdue Email: _____

Please sign the following:

I affirm that the answers given on this test are mine and mine alone. I did not receive help from any person or material (other than those explicitly allowed).

X _____

Part 1:	/25
Part 2:	/25
Part 3:	/15
Part 4:	/35
Total:	/100

Part 1: Dataflow analysis (25 pts)

In this problem, you will design a dataflow analysis that detects unused declarations. An unused declaration is a declaration of a variable that is never *used* in a program (though it may be defined).

Problem 1 (2 pts): This is a bit vector analysis. Is it forward or backward? Explain your answer.

Problem 2 (3 pts): What data do you need to keep track of at each program point?

Problem 3 (1 pts): Should you use union or intersection at merge points?

Problem 4 (4 pts): Assume that each statement in the program has two sets defined, DEF(s), of all the variables *defined* in the statement, and USE(s), of all the variables *used* in the statement. Using these two sets, define what GEN(s) and KILL(s) would be for a statement.

Problem 5 (5 pts): Explain how your analysis would be used to detect unused declarations.

Problem 6 (10 pts): For each statement in the following code, show what information your analysis would compute (IN sets, OUT sets, GEN sets and USE sets) if you used your analysis.

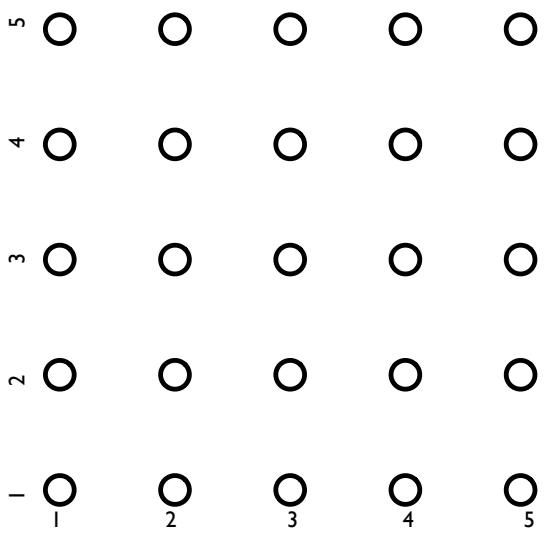
Code	GEN	KILL	IN	OUT
`1: int x = 3;`				
`2: int y = 6;`				
`3: x = READ();`				
`4: int z = 8;`				
`5: if (x < y) goto 8;`				
`6: x = z + 1;`				
`7: goto 10;`				
`8: int w = 4;`				
`9: z = w;`				
`10: halt;`				

Part 2: Depedence analysis and loop optimization (25 pts)

For the next four problems, consider the following loop:

```
for (int i = 1; i < 6; i++) {
  for (int j = 1; j < 6; j++) {
     A[i+2][j+1] = A[i-1][j] + A[i+2][j-1];
  }
}
```

Problem 1 (10 pts): Below is the iteration space graph for the loop nest (i along the horizontal axis, j along the vertical axis). Draw the dependence arrows that arise from analyzing the loop. Use solid arrows for flow dependences, arrows with a slash through them (as in class) for anti-dependences, and arrows with a circle over them for output dependences.



Problem 2 (2 pts): List the distance vectors that arise from this loop nest, and mark whether they are flow, anti or output.

Problem 3 (1 pt): List the direction vectors that arise from this loop nest, and mark whether they are flow, anti or output.

Problem 4 (1 pt): Can the loops in this nest be interchanged? Why or why not?

Problem 5 (3 pts): Assume that loop interchange is legal for the above loop nest. Suppose array `A` were laid out in *column-major* order in memory. Would you *want* to do interchange? Why or why not?

Problem 5 (8 pts): What are the distance vectors in the following loop nest? Give the type of dependence that each vector represents.

```
for (int i = 1; i < 6; i++) {
  for (int j = 1; j < 6; j++) {
     A[i+1][j] = A[i][j+1];
     A[i+1][j] = A[i+2][j-1];
  }
}
```

Part 3: Pointer analysis (15 pts)

Problem 1 (5 pts): In a flow-sensitive pointer analysis, why do we perform a *weak* update when determining how the expression `*x = y` changes the points-to graph?

Problem 2 (10 pts): Consider performing instruction scheduling. Which will result in a better schedule: (a) using a flow-*sensitive* pointer analysis before scheduling; (b) using a flow-*insensitive* pointer analysis before scheduling; or (c) would it not make a difference? Justify your answer.

Part 4: Review (35 pts)

ECE 468 student Cam Piler thinks the following grammar is LR(0) but not LL(1).

$$
\begin{aligned}
1.S &\rightarrow AB\$ \\
2.A &\rightarrow xz \\
3.A &\rightarrow x \\
4.B &\rightarrow w
\end{aligned}
$$

Problem 1 (10 pts): Argue that the grammar is not LL(1). Your argument should make reference to a parse table and predict sets.

Problem 2 (10 pts): Argue that Cam Piler is wrong, and the language is not even LR(0). Show that he's wrong in two steps. First, build the LR(0) machine for the grammar:

Problem 3 (2 pts): Use the LR(0) machine you built above to argue that the grammar is not LR(0).

Problem 4 (3 pts): Cam Piler wrongly thinks that if a grammar is not LL(1), it can't be regular (there is no regular expression that can capture the language). Show that he's wrong by providing a regular expression that captures all the strings from the above grammar.

Problem 5 (10 pts): Consider performing strength reduction (the loop optimization, not the peephole optimization) on a piece of code. If your goal is to parallelize that loop, is strength reduction a good idea or a bad idea? Justify your answer.