ECE 573 – Final Exam
December 14, 2009

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:	/30
Part 2:	/30
Part 3:	/15
Part 4:	/25
Total:	/100

Part 1: Dataflow analysis (30 pts)

In this problem, you will design a dataflow analysis that performs a *sign analysis*. A sign analysis determines at each program point whether a variable is definitely *positive* (+), definitely *negative* (−), definitely *zero* (0), or could be *anything* (?).

Problem 1 (10 pts): Provide a lattice for this dataflow analysis. A few things to keep in mind: (i) don't forget that having no information is different from a variable being "anything"; (ii) if a variable is 0, it also counts as both + and −; (iii) ? means that a variable might be positive or might be negative.

Problem 2 (1 pts): Should this analysis be forward or backward?

Problem 3 (1 pts): Does this analysis use meet or join at merge points?

Problem 4: For each statement, define the transfer function. Each subproblem shows one or more tables (except part (D)). The column headers give the value of y_{in} and the row headers (if the exist) give the value of z_{in}. For each subproblem, fill in the table with the value of x_{out} (*i.e.*, the value that x takes after the statement is processed) given the input values.

A) `x := y * z` (4.5 pts)

	+	−	0
+			
−			
0			

B) `x := y - z` (4.5 pts)

	+	−	0
+			
−			
0			

C) `x := y * y` (2 pts)

+	−	0	?

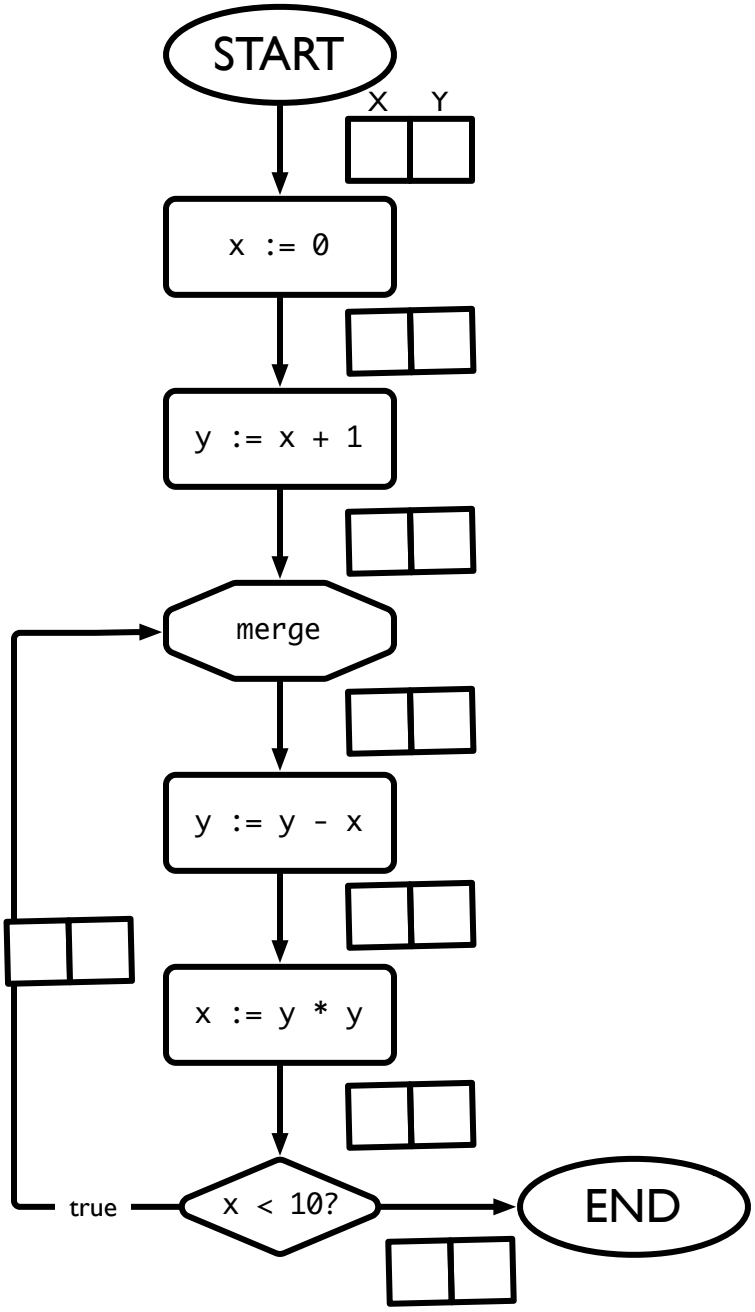
D) `if (x < 0) then { ... } else { ... }` (1 pt)

For this subproblem, give the values of x along the true branch and the false branch:

x_{true}:

x_{false}:

Problem 5 (6 pts): For the following CFG, fill in the boxes with the value of x and y that you get *after* performing your dataflow analysis (assume that x and y are arguments to the function, and hence could have any value at the beginning).

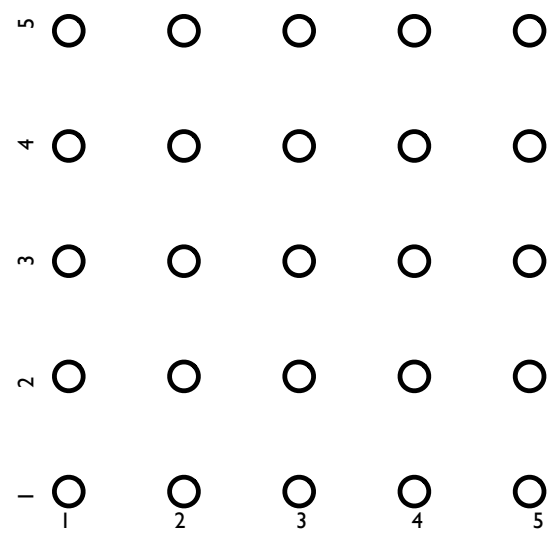


Part 2: Depedence analysis and loop optimization (30 pts)

For the next four problems, consider the following loop:

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

Problem 1 (10 pts): Below is the iteration space graph for the loop nest (i along the x-axis, j along the y-axis). Draw the dependence arrows that arise from analyzing the loop.



Problem 2 (2 pts): List the distance vectors that arise from this loop nest.

Problem 3 (2 pts): List the direction vectors that arise from this loop nest.

Problem 4 (2 pts): Can the loops in this nest be interchanged? Why or why not?

Problem 5 (2 pts): Give an example of a direction vector which prevents loop interchange.

For the next two problems, consider the following loop:

```
for (int i = 0; i < N; i++) {
  A[2i + 1] = A[4i - 2];
}
```

Problem 6 (2 pts): Give an informal (1-2 sentence) argument for why there would be no loop-carried dependences in this loop.

Problem 7 (8 pts): Use the GCD test to *prove* there are no dependences in this loop (show your work!)

Problem 8 (2 pts): Why might the GCD test say that there is a dependence in a loop even when there isn't?

Part 3: Pointer analysis (15 pts)

Problem 1 (2 pts): What is the difference between a weak update and a strong update when doing pointer analysis?

Problem 2 (5 pts): Why does `*x = y` require a *weak* update when doing flow-sensitive pointer analysis?

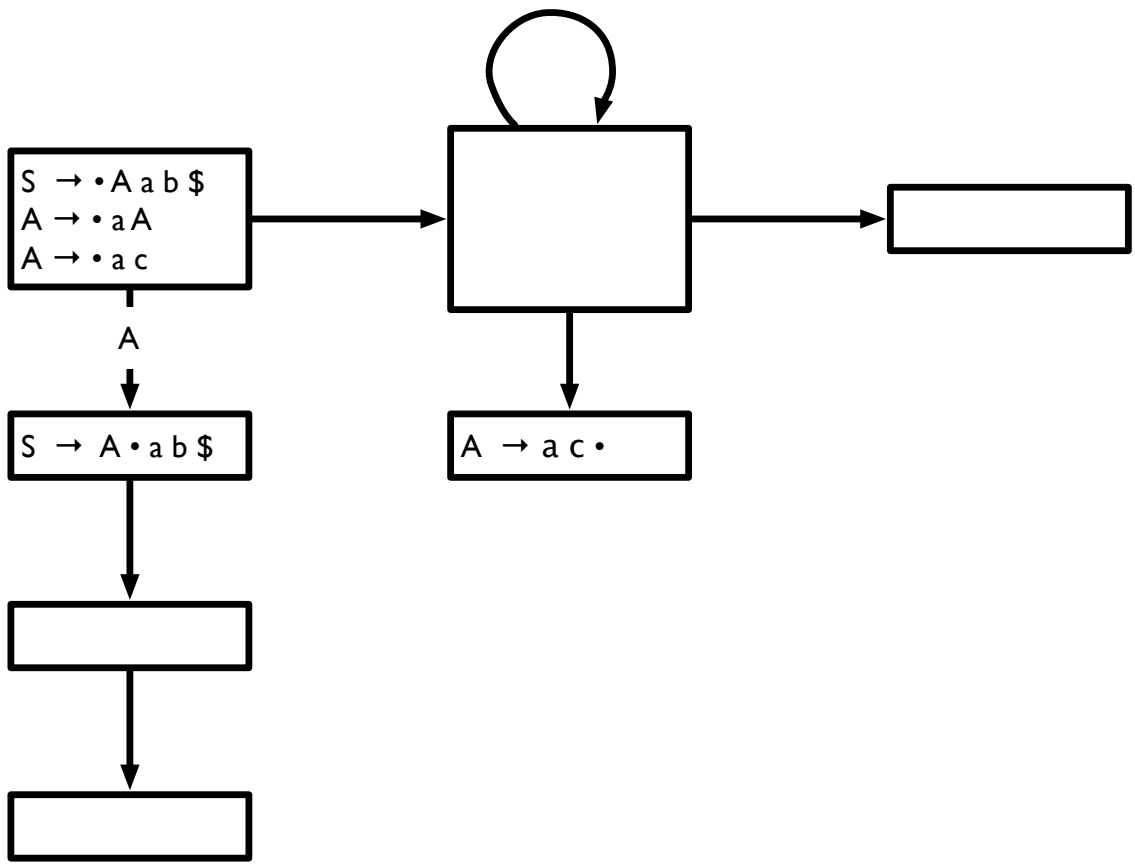
Problem 3 (8 pts): Consider a flow-sensitive analysis where *all* strong updates have been replaced with weak updates. Will this analysis produce different results from a flow-*insensitive* analysis like Andersen's? Why or why not? (Hint: consider applying both analyses to a function with no branching control flow—*i.e.*, a single basic block).

Part 4: Review (25 pts)

Problem 1 (5 pts): Give the grammar for a language that *cannot* be parsed by an LL(1) parser, but *can* be parsed by an LL(2) parser.

Problem 2 (10 pts): Fill in the CFSM for the following grammar (label lines with their transitions, fill in any missing boxes):

$$\begin{array}{lcl} S & \rightarrow & Aab\$ \\ A & \rightarrow & aA \\ A & \rightarrow & c \end{array}$$



Consider the problem of allocating temporaries to registers for the following piece of code:

```
T1 = A + B;

T2 = C + D;

T3 = T2 + E;

T4 = T1 + T2;

T5 = F + T2;

T6 = T1 + T4;

T7 = T6 + T5;
```

Problem 3 (5 pts): How many registers do you need to allocate temporaries to registers without spilling?

Problem 4 (5 pts): Draw the interference graph for the code.