15-745: Optimizing Compilers

March 22, 2005 **Midterm** Version A

Name: _____

UserID: _____

Section: _____

Instructions:

- This is an open book test.
- Make sure that your exam is not missing any sheets, then write your full name and Andrew User ID on this sheet.
- Write your answers in the space provided beside or below the problem. If you make a mess, make sure that your final answer is neatly drawn, and then circle it.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.

1	2	3	4	5	6	7	Total
(15pt)	(20pt)	(20pt)	(20pt)	(26pt)	(5pt)	(9pt)	(115pt)

Problem 1: (15 pts)

Convert the following code to minimal SSA.

```
1   L0:    ; k is an argument
2        ; x is an argument
3          j ← 0
4          if k > 5 goto L3
5
6   L1:    if j >= 10 goto L6
7
8   L2:    if j <= 5 goto L4
9
10  L3:    k ← x * k + j
11         goto L5
12
13  L4:    k ← x + k * j
14
15  L5:    j ← j + 1
16         if j < 10 goto L2
17
18  L6:    return k
```

A) [5 pts] Before converting the above code to SSA, show the dominance frontier for each basic block. Refer to basic blocks by the linenumber of the first statement in each block.

1: _____ 6: _____

8: _____ 10: _____

13: _____ 15: _____

18: _____

B) [10 pts] Using the information from above, rewrite the above code in minimal SSA. You can represent the resulting code in a CFG if you prefer.

Problem 2: (20 pts)

In this question you are asked to create an interference graph for a fragment of code for a machine with 2 general purpose registers (`g0` and `g1`), 2 address registers (`a0` and `a1`), and 2 predicate registers (`p0` and `p1`). The general purpose registers can only be used for arithmetic operations. The address registers can only be used for memory operations. The predicate registers are the only registers that can be used for predication. All instructions can be predicated.

```
1   ;     argument 0 is A, the base of an array
2   ;     argument 1 is n, the length of the array
3   Entry:
4         i ← 0
5         sum ← 0
6   X:    t1 ← ld[A]
7         p1 ← t1 > 0
8   [p1]  goto Y
9         t1 ← t1 * -1
10  Y:    sum ← sum + t1
11        i ← i + 1
12        A ← A + 4
13        p2 <- i < n
14  [p2]  goto X
15        return sum
```

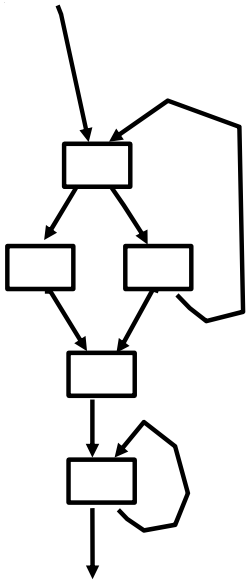
A) [8 pts] Rewrite the above code to prepare it for register allocation. In the process eliminate all possible branches. The first two arguments to any function are passed in `g0` and `g1`. There are no callee save registers. Finally, the return result is passed back in `g0`.

(Question 2 cont'd)

B) [12 pts] Show the interference graph for your rewritten code. Include edges which will ensure no registers will be used illegally.

Problem 3: (20 pts)

A) [5 pts] Perform a T1-T2 reduction on the CFG below. Enclose each region on the left CFG, and draw the control tree to the right.



B) [12 pts] Recall block X dominates block Y if X occurs on every path from the CFG entry to block Y. Dominators can be found using forward dataflow analysis. What is the meet operator? What are the Gen and Kill sets for a basic block?

C) [8 pts] Still considering finding dominators, pick two non-leaf nodes in the control tree; for each, calculate the transfer function of the corresponding region.

Problem 4: (20 pts)

A) [8 pts] Consider the following inner loop. Perform loop-invariant code motion, editing the code below.

```
1   do {
2           p1 = cost[i];
3           if (p1>0) {
4               sum += i ^ p;
5           } else {
6               sum += min * p;
7           }
8           i++;
9       } while (i<n);
```

B) [4 pts] Is this transformation always beneficial? Why or why not?

C) [4 pts] Suggest how profiling information could be used to improve this optimization.

D) [4 pts] When can loop-invariant code motion hoist an expression that might cause an exception? Consider both ”for” and ”do” loops.

Problem 5: (26 pts)
An expression, e, is very busy at point p if it computed along every path from p to the exit before its value is changed.

A) [2 pts] Is this analysis forward or backward?

B) [4 pts] define the elements of the lattice on which the analysis will run.

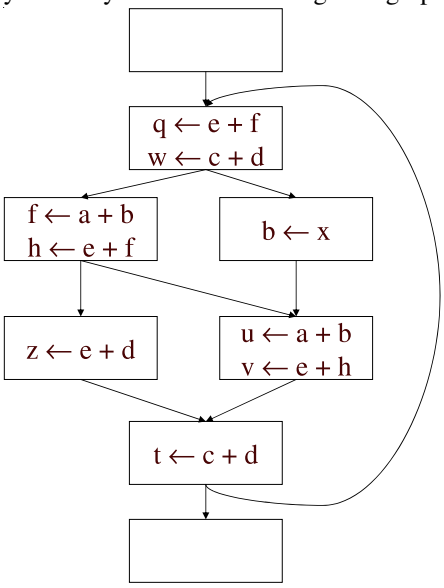
C) [6 pts] define the gen and kill operators.

D) [2 pts] Define the entry/exit initialization

(Question 5 cont'd)

E) [6 pts] define the dataflow equations

F) [6 pts] Show the result of running your analysis on the following flow graph.



Problem 6: (5 pts)

```
1   struct exam {
2                   struct exam *f;
3                   struct exam *g;
4                   int val;
5           };
6           struct exam *p;
```

Shape analysis has been performed on the heap structure pointed to by p. For each given shape determination, answer whether the two pointers shown can alias each other. Fill in the boxes with either the words **can** or **not**.

	tree	DAG	Cyclic
p→f, p			
p→f, p→g→g			
p→f→f, p→g→g			

Problem 7: (9 pts)

```
1   struct lah {
2                   struct lah *f;
3                   struct lah *g;
4                   int val;
5                   int sum;
6           };
```

A) [3 pts] Assume that it is determined that only field `f` is frequently accessed. Write the structure definitions for the above after hot/cold structure splitting has been applied.

B) [3 pts] Assuming intelligent rearrangement and a cold cache (nothing currently resident), how will the number of cache misses be affected?

C) [3 pts] Assuming no rearrangement but instead a malloc that performs simple sequential allocation, how will the number of cache misses be affected (again, assume a cold cache)?