

CS153 Final Exam

January 20, 2005

Instructions: Please use the blue books to write your answers. Make sure that you write your name on each book that you use.

- [8 questions at 5 points = 40 points]** For each of the following optimizations, briefly (in a sentence or two) explain what the optimization does and why the optimization improves code.
 - constant folding
 - constant propagation
 - common sub-expression elimination
 - dead code elimination
 - loop unrolling
 - strength reduction
 - loop invariant removal
 - inlining
- [3 questions at 5 points = 15 points: running total 55]** Briefly explain the differences between (a) reference counting, (b) mark-sweep, and (c) copying garbage collection. In particular, what are the advantages and disadvantages of each approach?
- [5 points: running total 60]** Consider the following Java class definitions:

```
class A {
    int x = 0;
    int y = 0;
    void setx(int v) { x := v; }
    int getx() { return x; }
    void sety(int v) { y := v; }
    int gety() { return y; }
}

class B extends A {
    int c = 0;
    void setx(int v) { c := c + 1; x := v; }
    void reset() { c := 0; }
}
```

Draw a box-and-pointer diagram with an instance of each class.

4. **[15 points: running total 75]** Translate the function below into low-level, MIPS-like assembly code. Try to make the resulting code execute fast. In particular, assume that most instructions take 1 cycle, except multiplications and stores to memory, which take 5 cycles, and loads from memory, which take 10 cycles.

```
int foo(int n, int step, int[] A, int[] B) {
    int i;
    int s = 0;
    for (i = 0; i < n; i = i + step*2) {
        s = A[i] + B[2*i-1] + A[i-step*2];
    }
    return s;
}
```

5. **[15 points: 100]** Liveness analysis is useful for computing the set of temps that are needed in the future of a computation. It is possible to formulate liveness as a dataflow analysis that computes *gen* and *kill* sets for each node in a control-flow graph. Describe how you would compute *gen* and *kill* sets for each of the following kinds of statements, and then describe how you would compute for each node in a CFG what the live-in and live-out sets would be in terms of the *gen* and *kill* sets.

Statements include:

<code>t := t1 ⊕ t2</code>	<i>add, subtract, ...</i>
<code>t := Mem[t1 + c]</code>	<i>loads</i>
<code>Mem[t1 + c] := t</code>	<i>stores</i>
<code>if t1 > t2 goto L1 else goto L2</code>	<i>conditionals</i>
<code>goto L</code>	<i>jumps</i>
<code>L:</code>	<i>labels</i>
<code>t := call f(t1, ..., tn)</code>	<i>function calls</i>

6. **[5 points: 80]** Modern compilers use Static Single-Assignment (SSA) as their intermediate representation, instead of control-flow graphs built out of statements such as those in question 5. The essence of SSA is that there is at most one definition of a temporary (i.e., the code is *functional*.) How would this simplify your liveness analysis (and in fact any dataflow analysis)?
7. **[5 points: 85]** In Appel's formulation of graph-coloring register allocation, he does *coalescing* (a.k.a. copy propagation) as part of the coloring process. Why doesn't he just run a coalescing pass before doing the register allocation?

8. If you could go back to the beginning of the semester, and if you could make changes to the topics covered or the projects, what changes would you make?