Written Assignment 4
Due Tuesday, November 30, 2010 at 5pm

This assignment asks you to prepare written answers to questions on code generation, operational semantics, optimization, register allocation, and garbage collection. Each of the questions has a short answer. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work. Written assignments can be turned in at the start of lecture. Alternatively, assignments can be turned in at Professor Aiken's office in Gates 411, or submitted electronically in PDF format by following the electronic submission instructions at `http://www.stanford.edu/class/cs143/policies/submit.html`, by 5:00 PM on the due date.

1. (10 points) Suppose that we want to add the following conditional expression to Cool.

   `cond <p1> => <e1>; <p2> => <e2>; ... ; <pn> => <en>; dnoc`

   There must be at least one predicate and expression pair (that is, $n \geq 1$). The evaluation of a `cond` expression begins with the evaluation of the predicate `<p1>`, which must have static type `Bool`. If `<p1>` evaluates to `true`, then `<e1>` is evaluated, and the evaluation of the `cond` expression is complete. If `<p1>` evaluates to `false`, then `<p2>` is evaluated, and this process is repeated until one of the predicates evaluates to `true`. The value of the `cond` expression is the value of the expression `<ei>` corresponding to the first predicate `<pi>` that evaluates to `true`. If all the predicates evaluate to `false`, then the value of the `cond` expression is `void`.

   Write operational semantics rules for this conditional expression in Cool.

2. (10 points) Write a code generation function `cgen(cond <p1> => <e1>; ... ; <pn> => <en>; dnoc)` for the conditional expression described in Question 1. For concreteness, assume that $n = 2$. Use the stack machine architecture and conventions from the lectures.

3. (10 points) Consider the following basic block, in which all variables are integers, and `**` denotes exponentiation.

```
a := b + c
z := a ** 2
x := 0 * b
y := b + c
w := y * y
u := x + 3
v := u + w
```

   Assume that the only variables that are live at the exit of this block are `v` and `z`. In order, apply the following optimizations to this basic block. Show the result of each transformation.

   (a) algebraic simplification
   (b) common sub-expression elimination
   (c) copy propagation
   (d) constant folding

(e) dead code elimination

When you've completed part (e), the resulting program will still not be optimal. What optimizations, in what order, can you apply to optimize the result of (e) further?

4. (10 points) Consider the following program.

```
L0: e := 0
    b := 1
    d := 2
L1: a := b + 2
    c := d + 5
    e := e + c
    f := a * a
    if f < c goto L3
L2: e := e + f
    goto L4
L3: e := e + 2
L4: d := d + 4
    b := b - 4
    if b != d goto L1
L5:
```

This program uses six temporaries, `a-f`. Assume that the only variable that is live on exit from this program is `e`.

Draw the register interference graph. (Drawing a control-flow graph and computing the sets of live variables at every program point may be helpful.)

5. (10 points) Suppose that the following Cool program is executed.

```
class C {
  x : C; y : C;
  setx(newx : C) : C { x <- newx };
  sety(newy : C) : C { y <- newy };
  setxy(newx : C, newy : C) : SELF_TYPE { { x <- newx; y <- newy; self; } }; };
class Main {
  x : C;
  main() : Object {
    let a : C <- new C, b : C <- new C, c : C <- new C, d : C <- new C,
        e : C <- new C, f : C <- new C, g : C <- new C, h : C <- new C in {
      f.sety(g); a.setxy(e, c); b.setx(f); g.setxy(f, d); c.sety(h); h.setxy(e, a);
      x <- c;
  } }; };
```

(a) Draw the heap at the end of the execution of the program, identifying objects by the names to which they are bound in the `let` expression. Assume that the root is the `Main` object created at the start of the program, and that this object is not in the heap. If the value of an attribute is `void`, don't show that attribute as a pointer on the diagram.

(b) For each of the garbage collection algorithms discussed in class (Mark and Sweep, Stop and Copy, and Reference Counting), show the heap after garbage collection. When the pointers of an object are processed, assume that the processing order is the order of the attributes in the source program. Assume that the heap has space for at least 16 objects.