

**Name:**

Please put your name on every sheet. The total number of possible points is 100.

Total points: 100, 20% of course grade

Good luck!

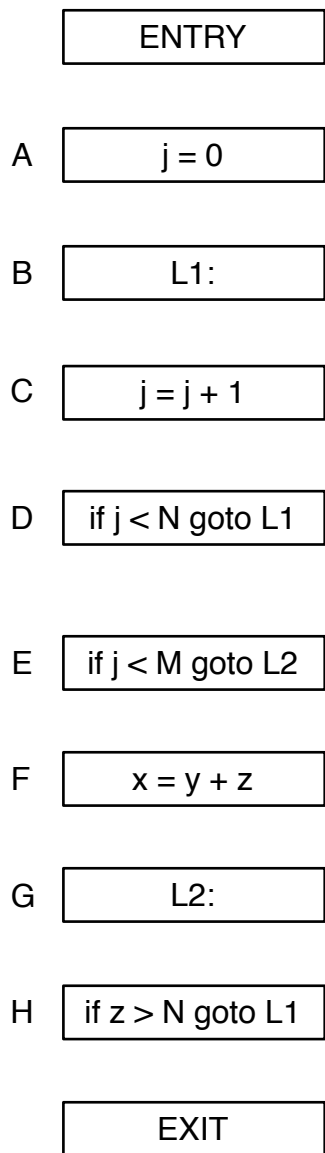
1. [20 points] Garbage Collection. For the following code,

```
class Main {
    public static void main(String[] a){ System.out.println(new Foo().testing());}
}
class Foo {
    Foo next;
    public int nextFoo(Foo p) { next = p; return 7; }
    public int testing() {
        Foo local; int i; i = 0;
        while (i<N) {
            local = new Foo(); local.nextFoo(local);
            i = i + 1;
        }
        return 42;
    }
}
```

- a) indicate the number of objects created by the whole program as a function of N,
- b) indicate the number of objects as a function of N that will leak (not be destroyed even though it is unreachable) if a reference counting scheme is used,
- c) and indicate the number of times a mark and sweep collector will be called as a function of N assuming 12 bytes of GC overhead (as in the MiniJava GC assignment), 4 bytes to store a pointer, and a 36 byte heap.

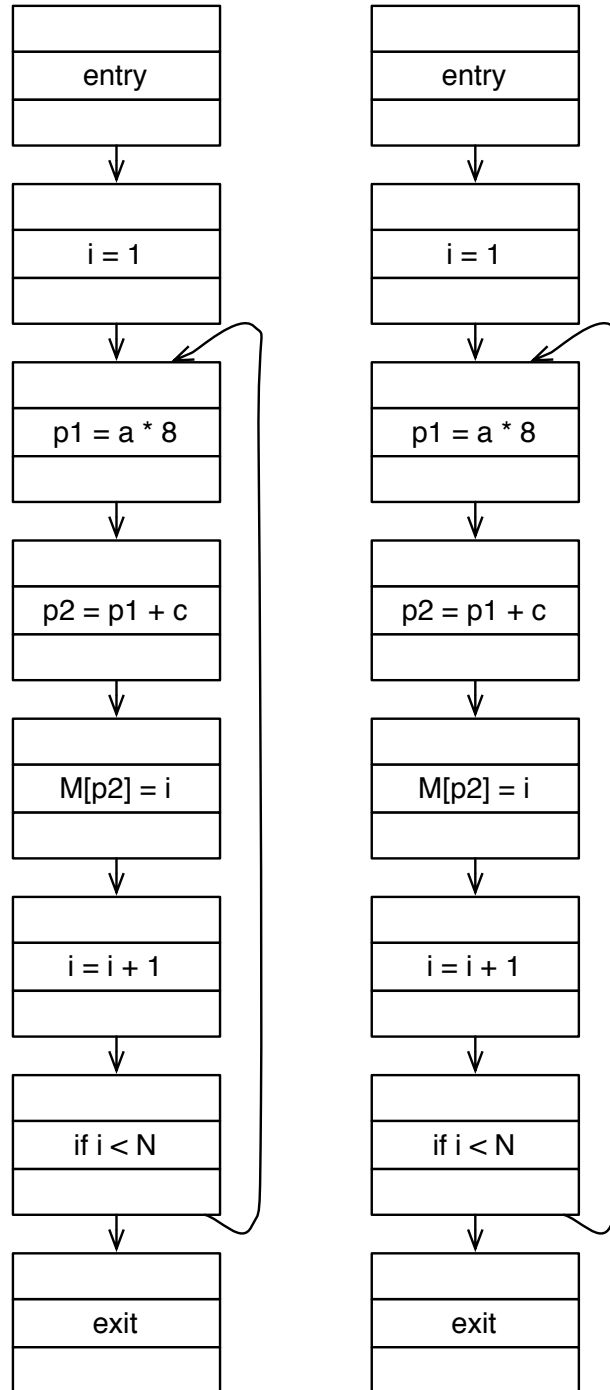
2. [20 points] Control flow and loops. For the given 3-address code,

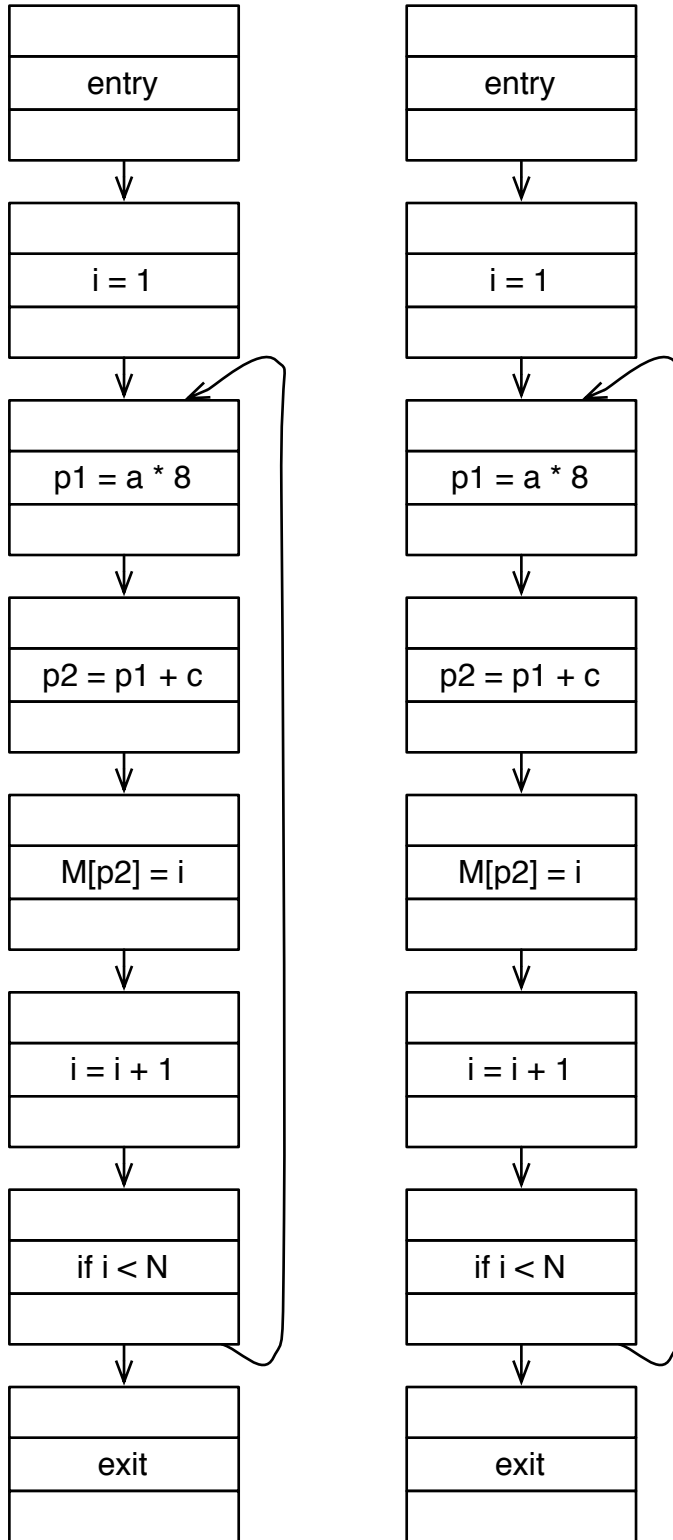
**Dominators**



- a) draw the control-flow graph (preferably in place)
- b) calculate the dominator set for each node
- c) label each of the following (there can be more than one of each): pre-header, header, back edge, exit node,
- d) indicate the loop(s) as set(s) of nodes.

3. [30 points] Partial Redundancy Elimination. Fill in the given control-flow graphs with anticipated, PRE availability, earliest, postponable, latest, and used expression data-flow analysis results. Label the top of each control-flow graph with the property being computed. latest and earliest should be shown next to one of the control-flow graphs. Based on the results perform PRE with the lazy code motion algorithm and write out the resulting 3-address code.





4. [20 points] After PRE. What other data-flow optimizations should follow PRE? Why? Perform the two most important on your 3-address code from problem 4.

5. [20 points] Instruction scheduling.

- a) Rewrite the code resulting from problem 4 with the loop unrolled twice. Use a different set of temporaries (p#) for the second loop body. Recall that there should only be one set of loop overhead statements.
- b) Draw the DAG for the new loop body. Do NOT include the branch. Assume a latency of 2 between any instructions with a flow dependence, a latency of one between output dependences, and a latency of zero between anti dependences.
- c) Use list scheduling with the following priorities to find a schedule. How many cycles does the schedule require assuming only one instruction can be issued each cycle?

**Priorities**

- 1) Avoid stalls with previously scheduled instructions.
- 2) Pick instructions that interlock with the most immediate successors.
- 3) All else being equal pick the initial instruction ordering.