COP5621 Exam 4 - Spring 2005

Name: ______________________________ (Please print)

Put the answers on these sheets. Use additional sheets when necessary. Show how you derived your answer when applicable (this is required for full credit and helpful for partial credit). You can collect 100 points in total for this exam. A bonus question for an additional 15 points is included. If the total number of points for this exam exceed 100, the excess points are carried over to the next exams.

1. Match the terms below with the given sentences so as to best complete each sentence. Use no term more than once. Some terms will go unused. (10 points)

(1)	static link	(7)	stack pointer
(2)	access link	(8)	control link
(3)	state	(9)	environment
(4)	local	(10)	global
(5)	live	(11)	dead
(6)	code motion	(12)	reduction in strength

   (a) A transformation on a program is called ___ if it can be performed by looking only at the statements in a basic block rather than an entire region of code.

   (b) An implementation of lexical scope for nested procedures is obtained by adding a pointer called ___ to each activation record.

   (c) In liveness analysis, a variable is said to be ___ if it has no next use.

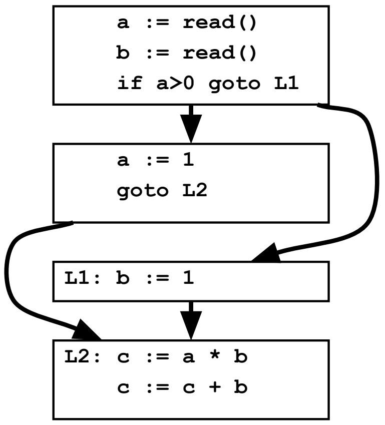
   (d) The loop optimization called ___ moves loop-invariant statements to the loop header.

   (e) In programming language semantics, the term ___ refers to a function that maps a name to a storage location (i.e. to an ℓ-value) and the term ___ refers to a function that maps a storage location to the value held (i.e. to an r-value).

2. Name three *peephole optimizations.* (10 points)

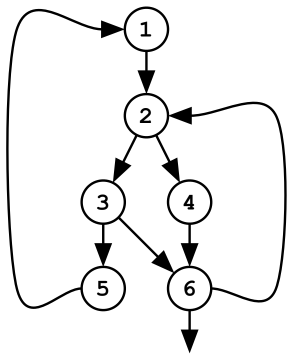
3. Describe a typical *calling sequence* to invoke a function. (10 points)

4. Apply *register allocation and assignment* using *graph coloring* to the following CFG:



To determine a coloring, show the live ranges of the variables `a`, `b`, and `c` in the CFG, assuming that the variables are dead at the exit from the CFG Then draw the *register-interference graph* (*conflict graph*) for the variables and determine the minimum number of colors necessary to color the graph. (15 points)

5. Consider the following CFG:



(a) Draw the *dominator tree* of the CFG. (10 points)

(b) Identify the *natural loops.* (5 points)

(c) Is the CFG reducible? Explain why or why not. (5 points)

6. (a) Organize the following fragment of three-address code into basic blocks and construct the CFG. (10 points)

```
L0:  i := 0
L1:  i := i+1
     if i>m goto L2
     n := n<<1
     if n<k goto L1
L2:  k := k-1
     if k<=0 goto L3
     goto L0
L3:  goto L4
L4:  halt
```

(b) Apply branch chaining optimization followed by dead-code elimination. After these optimizations, draw the modified CFG with its three-address code. (10 points)

7. Consider the following program:

```
program P(input, output)
   var n : integer;
   procedure Q(k : integer)
      procedure S(i : integer)
         begin
           ... (* body of S *)
         end;
      begin
         S(k)
      end
   procedure R(j : integer);
      var m : integer
      begin
         Q(n + m)
      end;
begin
   R(n)
end
```

(a) Program `P` calls `R`, `R` in turn calls `Q`, and `Q` in turn calls `S`. Draw the resulting stack layout with activation records. Show the arguments and local variables in each record and draw the access links. (10 points)

(b) Which variables are visible (in scope) in the body of `S` and how many access links must be traversed to reach the nonlocal data? (5 points)

Var	*visible (Y/N)*	*#links*
`i`		
`j`		
`k`		
`n`		
`m`		

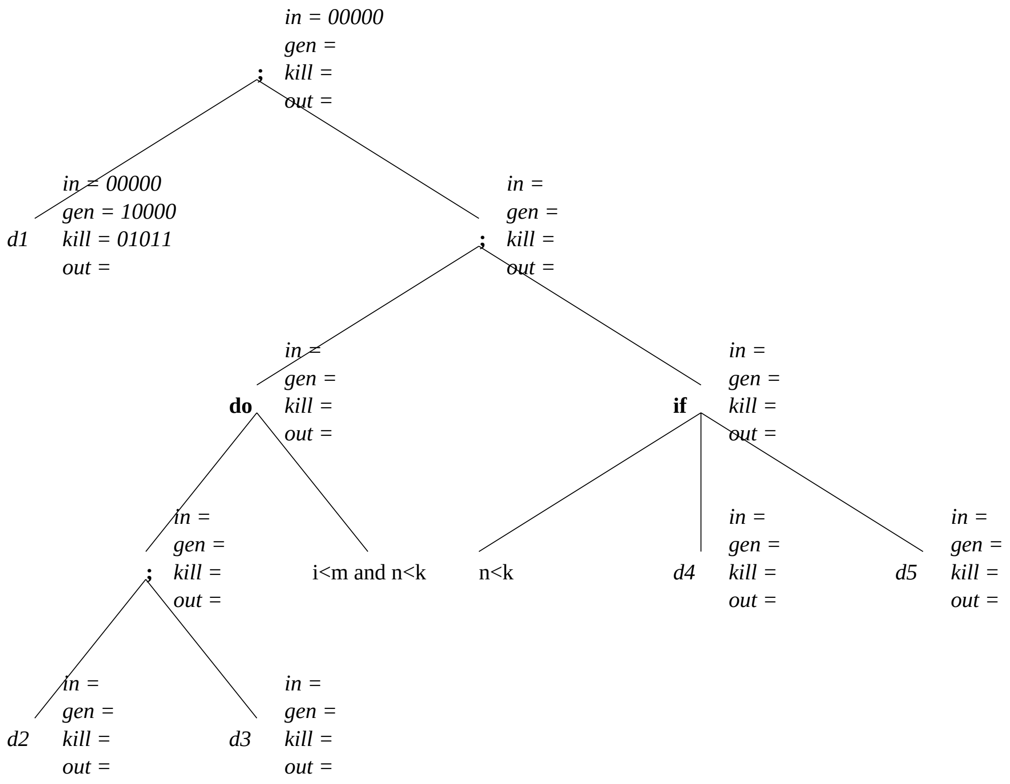
8. **(Bonus question).**

(a) Give the data-flow equations for reaching definitions as described in the book and illustrated in class for the four example programming constructs (assignment, statement composition, if-then-else, and do-while). (8 points)

(b) Consider the following program:

```
d1 :  i := 0;
      do
d2 :     i := i + 1;
d3 :     n := n << 1
      while i<m and n<k;
      if n<k then
d4 :     i := i - 1
      else
d5 :     i := -1
```

Annotate the syntax tree of the above program with *in*, *gen*, *kill*, and *out* bit-vectors:



Note: compute the *gen* and *kill* vectors bottom-up first, i.e. start at the leaves. For example, *gen=01000* and *kill=10011* of d_2, because d_2 kills all other definitions of variable `i` (i.e. d_1, d_4, and d_5). Then, go up by applying the equations for reaching definitions. When the *gen* and *kill* sets are determined, compute the *in* and *out* vectors in a top-down, left-to-right traversal. (*in* is inherited, while *out* is synthesized.) (7 points)