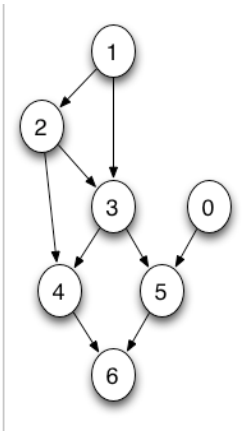
Compilers Preliminary Examination 2012

B1. (25 points)

1. (10 points)

a. (7 points) Construct a straight-line (i.e., no branches) block of instructions that exhibits the data dependence DAG below. The instructions in the block should also conform to the following rules:

- Must use three-address code like the following:
  r5 = r3 + r7
  r6 = r5 * 8
- Can use up to eight registers (r1 through r8)
- Up to a maximum of two registers can be live coming into the block
- Instructions 1 and 5 should use the divide operator ,%, and all other instructions can use operators +, -, or multiply, *.



b. (3 points) Give the best instruction ordering assuming instructions 1 and 5 take three cycles each, and the others take one cycle.

2. (8 points) Explain why the following grammar is LL(1) but not SLR(1).

X -> YaYb | ZbZa
Y -> ε
Z -> ε

3. (7 points) The C++ language uses the ">>" character sequence to denote the right-shift operator. This character sequence can also appear when a template (generic) type takes another template as its argument, as in "stack<list<int>>" that is supposed to denote a stack of integer lists.

What problem can this cause?
Describe how you might solve this problem.

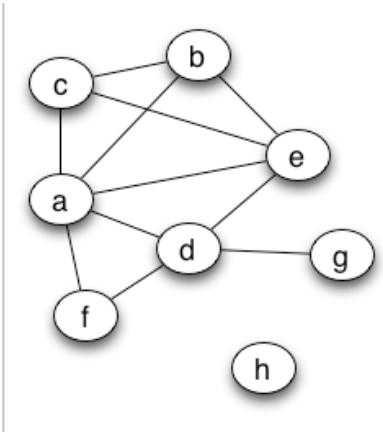
B2. (25 points)

1. (15 points)

a. (7 points) Fill in the code for procedure P below so that the variables produce the following interference graph.

```
Procedure P:
c = 1
b = 2
...

return h
```



b. (3 points) Is the interference graph 3-*colorable*? Justify your answer.

c. (5 points) Does graph coloring produce an optimal register allocation? Why or why not?

2. (5 points) Give an example program that gives different results in static and dynamic scoping. In each case, explain how an interpreter or compiled code might perform function execution.

3. (5 points) Define when a context-free grammar is ambiguous, and give an example to demonstrate your knowledge.

B3. (25 points)

1. (6 points) Each of the following statements may be true, false, or nonsensical. Indicate which and (respectively) provide a (one-sentence) justification of why it holds, a counterexample or other explanation of why it fails, or corrected statement.

a. In a language with stack-allocated free variables, the static chain pointer always points to the caller's stack frame.

b. Any regular grammar is also a context-free grammar. Hence any lexer generated by lex (or JLex) could instead be generated by yacc (or Cup).

c. All SLR(1) grammars that are not LR(1) exhibit a reduce-reduce conflict in the LR(1) DFA.

d. Structural equivalence implies name equivalence. That is, if two types are structurally equivalent, they are also name equivalent.

2. (14 points)

a. (5 points) What is an activation record? When is one created and where? List five things that could be stored in an activation record.

b. (4 points) Could you design a language without needing activation records? Why or why not? If so, give an example.

c. (5 points) Describe language properties that would prevent a compiler designer from being able to use a stack for activation records.

3. (5 points) For each of the following types of variables, state all the possible places where in memory the compiler might allocate the space for such a variable. Possible answers include registers, activation records, static data areas (with different visibilities, either local or global), and the runtime heap.

a. A variable local to a procedure

b. A global variable

c. A dynamically allocated global variable

d. A formal parameter

e. A compiler-generated temporary variable

B4. (25 points)

1. (10 points) Describe advantages and disadvantages of the following storage allocation strategies: (a) static allocation, (b) stack allocation, and (c) heap allocation.

2. (5 points) Describe advantages and disadvantages of garbage collection versus explicit memory management.

3. (5 points) Reference counting is an efficient strategy of dynamic storage reclamation, but it has one major problem. Describe the problem and a solution that could be used to overcome this problem.

4. (5 points) Consider the following grammar:

$S \rightarrow aS \mid Ab$
$A \rightarrow XYZ \mid \varepsilon$
$X \rightarrow cS \mid \varepsilon$
$Y \rightarrow dS \mid \varepsilon$
$Z \rightarrow eS$

Explain why it is that if we add the production $X \rightarrow bS$, the grammar is no longer LL(1).