Compiler Construction
Professor Jeremy Siek

Quiz 3
October 1, 2012

Name: _____

This exam has 3 questions, for a total of 10 points.

1. [3 points] Fill in the code for the `IntMoveInstr` and `Stmt` cases in the following `liveness` function that computes the live-after and live-before sets for each intruction. The live-before set should be stored in an attribute of the AST node named `live_before` and it should also be returned, as is done in the `IntAddInstr` below. The helper function `var_refs` computes the set of variable names that are referenced within a given AST node. You do not need to define this helper function.

```
class IntAddInstr(Node):        # represents the operation lhs += rhs
    def __init__(self, lhs, rhs):
        self.lhs = lhs
        self.rhs = rhs

class IntMoveInstr(Node):       # represents the operation lhs = rhs
    def __init__(self, lhs, rhs):
        self.lhs = lhs
        self.rhs = rhs

def liveness(node, live_after):
    if isinstance(node, IntAddInstr):
        node.live_before = live_after | var_refs(node.lhs) | var_refs(node.rhs)
        return node.live_before

    elif isinstance(node, IntMoveInstr):




    elif isinstance(node, Stmt):




    elif isinstance(node, Module):
        liveness(node.node, set([]))
    ...
```

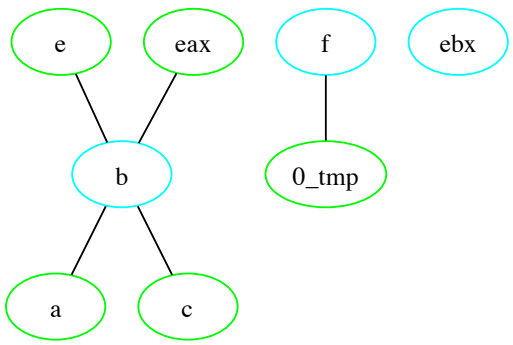
Solution:

```
    elif isinstance(node, IntMoveNode):
        node.live_before = (live_after - var_refs(node.lhs)) | var_refs(node.rhs)
        return node.live_before

    elif isinstance(node, Stmt):
        for s in reversed(node.nodes):
            live_after = liveness(s, live_after)
```

2. [3 points] Given the following result from liveness analysis, draw the interference graph. The live-before set is listed next to each instruction.

```
call input          {}
movl %eax, b        {eax}
call input          {b}
movl %eax, c        {eax,b}
movl c, a           {c,b}
addl 42, a          {a,b}
movl a, e           {a,b}
addl b, e           {b,e}
movl e, f           {e}
movl 1, 0_tmp       {f}
negl 0_tmp          {0_tmp,f}
subl 12, %esp       {f}
pushl f             {f}
call print_int_nl   {}
addl $16, %esp      {}
                    {}
```



Solution:

3. [4 points] Perform one iteration of register allocation for the program in question 2 using the greedy saturation-based algorithm. You are compiling for a computer architecture that only has two general purpose registers, %eax and %ebx, but is otherwise just like the x86 (it does have %esp and %ebp). Write down the following information for each step in the algorithm: 1) how many available registers there are for each variable at the beginning of the step, 2) what variable is chosen to be colored in this step, 3) which register or stack location is assigned to the variable.

Solution: The following is one of many possible solutions.

Name: _____

a	b	c	e	f	0_tmp	chosen variable	chosen location
2	1	2	2	2	2	b	%ebx
1	-	1	1	2	2	a	%eax
-	-	1	1	2	2	c	%eax
-	-	-	1	2	2	e	%eax
-	-	-	-	2	2	f	%ebx
-	-	-	-	-	1	0_tmp	%eax