CS243 Midterm Examination

Winter 2003-2004

You have 1 hour 15 minutes to work on this exam. The examination has 75 points, one point for every minute. Please budget your time accordingly.

Write your answers in the space provided on the exam. If you use additional scratch paper, please turn that in as well.

Your Name: _____

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

Signature: _____

Problem	Points	Score
1	20	_____
2	15	_____
3	15	_____
4	25	_____
Total	**75**	_____

1. (20 points) True or false. No explanation is necessary.

   a. Applying constant propagation before lazy code motion can improve the applicability of lazy code motion.

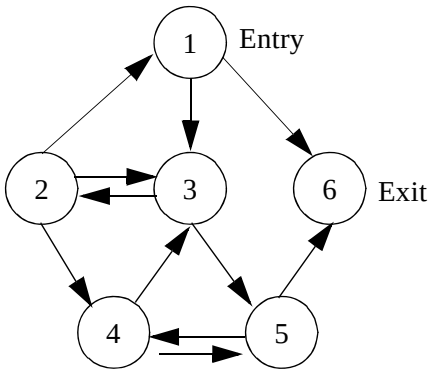
   b. Applying lazy code motion before constant propagation can improve the applicability of constant propagation.

   c. If reverse post-order is used, the data-flow algorithm for constant propagation converges in at most d+2 iterations, where d is the loop depth of the input program.

   d. A reverse control-flow graph is like a control-flow graph, except that the directions on the edges are reversed. That is, given a control-flow graph $G = <N, E>$, $G'=<N, E'>$ is a reverse control-flow graph of G if $(n_1, n_2) \in E$ iff $(n_2, n_1) \in E'$.
   True or false: G' is reducible if G is reducible.

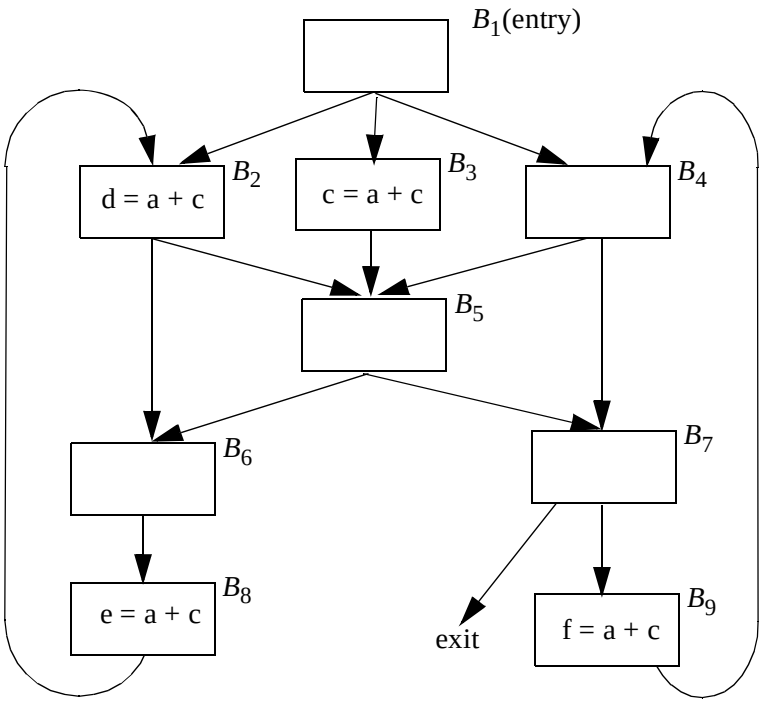
2. (15 points)

a. What are the natural loop(s) in the following flow graph?



b. Is the above flow graph reducible? Explain your answer.

3. (15 points) Apply the lazy code motion algorithm as described in class to the program below. Show the result of the optimization---it is not necessary to show any intermediate steps. (Introduce new basic blocks as necessary).



4. (25 points) Method inlining is the optimization of replacing a method invocation by the actual code to be executed. This optimization is especially useful for object-oriented programs; because such programs tend to have many small methods, inlining can greatly reduce the method invocation overhead and increase the effectiveness of data-flow optimizations, register allocation and instruction scheduling.

   However, because of virtual methods, it is generally not possible to determine statically which methods will be invoked. **Describe a "virtual method resolution" algorithm that specifies, for each invocation site, whether if it can invoke only one method, and if so, the class in which the method is defined.** This information can be used to determine which methods can be inlined. You only need to come up with an **intraprocedural** algorithm that analyzes methods one at a time. For the example below:

```
class B {
     public int m () { ... };
     public int n () { ... };
}
class C extends B {
     public int m () { ... };
}
...
public void p () {
    B x;
    B y;
    int z;
    x = new C();
    y = x;
    z = y.m ();
}
```

   your analysis should deduce that the method invoked by `y.m()` is method `m` defined in class `C`.

   a. (20 points) There are two parts to this question. In the first part, assume that the target language is a subset of the Java programming language with statements:

      class v; (declaration)

      if *cond* goto *L*;

      v = new *class* ();

      v_1 = v_2.*method* ();

      v_1 = v_2;

      return *v*;

   You may ignore the *cond* in the goto statement and assume that all paths may be taken. The language allows only single inheritance. You may assume that an earlier pass has analyzed the class declarations to collect the class hierarchy information and a list of methods defined in each class. (Please turn over).

If your solution uses data-flow analysis, specify the algorithm fully by answering the following questions:

i. What is the direction of your data flow-analysis?

ii. What is the set of values in the semi-lattice?

iii. Draw a diagram of the lattice, identifying the top and bottom elements clearly.

iv. How would you initialize the interior points?

v. Define the transfer function of a basic block. You may assume that each basic block contains only one statement.

vi. How would you initialize the information at the entry/exit nodes?

vii. Is your data-flow framework monotone? (No explanation is necessary).

viii. Is your data-flow framework distributive? (No explanation is necessary).

ix. Will your algorithm necessarily converge? If so, why?

Please state clearly how you identify the sites that can only invoke one method, and the class in which the invoked method is defined.

b. (5 points) Notice that the language above is missing the important operations of field accesses. Explain **briefly** how you would extend your analysis to handle the addition of these two statements:

$v_1.field = v_2$

$v_1 = v_2.field$

***** Extra Space : Don't forget that Question 4 has parts (a) and (b) *****