CS243 Final Exam

8:30AM – 11:30AM
March 20, 2006

The exam is closed book, but you may use two single-sided 8.5 by 11 inch pieces of paper with whatever you wish written upon it. Answer all 8 questions on the exam paper itself.

Write your name here: _____

I acknowledge and accept the honor code.

(signed) _____

Question	Max	Score
1	20	
2	20	
3	20	
4	10	
5	20	
6	20	
7	20	
8	20	
TOTAL	150	

Problem 1 (20 pts.) Here is a small loop written in register-level intermediate code:

```
(A)   LD    r1, a        // load a into register r1
(B)   ADD   r2, r1, #1   // r2 gets 1 plus the value in r1
(C)   ST    a, r2        // store r2 into a
(D)   BNZ   r2, (A)      // go to the beginning if r2 is not zero
```

(a) Indicate each of the true dependences in this code. Give the first and second statements involved in the dependency and the register or variable that causes the dependency. Do not forget dependences that require going around the loop.

A->B (r1), B->C (r2), B->D (r2), and C->A (a)

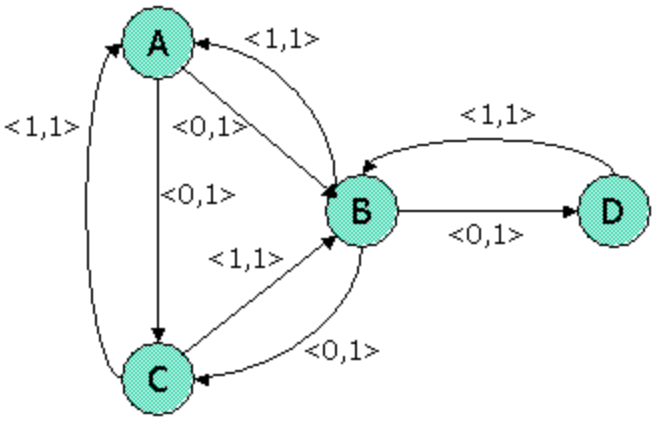
(b) Indicate each of the antidependences in the same manner.

A->C (a), C->B (r2), D->B (r2), and B->A (r1)

(c) Indicate each of the output dependences in the same manner.

Since no two statements write the same register or memory location, there are no nontrivial output dependencies. It is OK if you said that statements A, B, and C depended on themselves, but it is not possible for instances of a single statement to get out of order in a software pipeline.

(d) Assume that the machine allows any instruction to be executed in one clock cycle. Draw the data-dependence graph for the four statements, including labels on edges of the form <cycles, delay> as discussed in class and the text.



(e) Give the matrix of maximum acyclic delays, in terms of T, the delay between successive iterations of the loop --- again following the model used in class and in the text.

	A	B	C	D
A		1	2	2
B	2-T		1	1
C	2-2T	2-T		3-T
D	3-2T	1-T	3-2T	

Problem 2 (20 pts.) In this problem, we shall describe partially a simple data-flow framework and ask you to fill in additional details. Flow graphs have only one type of statement, of the form x=i, where x is a variable and *i* is a nonnegative integer. The effect of this statement is to give *x* the value *i*. We desire to compute a mapping *m* at each point in the program, so that $m(x)$ is the **largest** value that variable *x* can have at that point. The value of $m(x)$ can be any nonnegative integer or UNK, if there is not yet any value of *x* known.

(a) Define the appropriate meet (or confluence) operator on two mappings m_1 and m_2. That is, if $m = m_1 \wedge m_2$, describe $m(x)$ in terms of $m_1(x)$ and $m_2(x)$. **Hint**: things go much more succinctly in this and subsequent parts if you think of UNK as "negative infinity."

m (x) = max(m_1(x), m_1(x)).

(b) Define the $\leq$ relation on mappings that results from your meet in (a).

$m_1 \leq m_2$ iff for all x, $m_1(x) \geq m_2(x)$.

(c) Define the transfer function on mappings that describes the assignment x=i. Let f be the transfer function for x=i. Let m' = f(m), where m is a mapping. Then m'(x) = i, and m'(y) = m(y) for all variables y other than x.

(d) Is this framework distributive? Yes No (circle one). Justify your answer.

Yes. It suffices to show the point for a function f that is associated with some x=i. Let z be any variable. Now f(m $\wedge$ n)(z) = i if x=z and (m $\wedge$ n)(z) otherwise. On the other hand, (f(m) $\wedge$ f(n))(z) = max(i,i) if x=z and max(m(z), n(z)) otherwise. Since i = max(i,i), and (m $\wedge$ n)(z) = max(m(z), n(z)), we have shown that for all z, f(m $\wedge$ n)(z) = (f(m) $\wedge$ f(n))(z), i.e., , f(m $\wedge$ n) = (f(m) $\wedge$ f(n)). That statement is "distributivity."

(e) Is this framework monotone? Yes No (circle one). Justify your answer.

Yes. Every distributive framework is monotone.

Problem 3 (20 pts.)

Context-insensitive Andersen's pointer analysis described in class is expressed in Datalog using the following rules:

1. Pts(V,H) :- "H: V = new T".
2. Pts(V,H) :- "V=W" & Pts(W,H).
3. Pts(V,H) :- "V=W.F" & Pts(W,G) & Hpts(G,F,H).
4. Hpts(H,F,G) :- "V.F=W" & Pts(V,H) & Pts(W,G).

Given the program snippet below

```
void f(){
        String s1 = new String("hello");        // H1
        s1        = new String("world");        // H2
        String s2 = new String("!");            // H3
        String s3 = g(s1);
        String s4 = g(s2);
}

String g(String s){
        System.out.println("String s: " + s);
        return s;
}
```

a. (1 pt.) What is the points-to set for s1?
   s1: [H1, H2]

a. (1 pt.) What is the points-to set for s2?
   s2: [H3]

b. (2 pts.) What is the points-to set for s3?
   s3: [H1, H2, H3]

c. (2 pts.) What is the points-to set of for s4?
   s4: [H1, H2, H3]

d. (3 pts.) If the pointer analysis formulation were flow-sensitive but context-insensitive, which answer(s) among a—d would be affected (Assume that all points-to sets are computed at the end of method f)?

[a, b, c]

e. (3 pts.) If the pointer analysis formulation were context-sensitive but flow-insensitive, which answer(s) among a—d would be affected?

[b, c]

f. (4 pts.) Consider a context-sensitive version of the Andersen's analysis described in class. Consider the code snippet below:

```
class List {
        Node head;

        void add(Object data){
                Node n = new Node(data, head)  // N1
                head = n;
        }
}

class Node {
        Object data;
        Node next;

        Node(Object data, Node node) {
                this.data = data;
                this.next = node;
        }
}

void foo(){
        List t = new List();          // T1
        t.add(new Integer(1));        // I1
        t.add(new Integer(2));        // I2
}
```

What are the points-to sets for

t.head	[N1]	t.head.next	[N1]
t.head.data	[I1, I2]	t.head.next.data	[I1, I2]

g. (4 pts.) Why may the context-sensitive, flow-sensitive pointer analysis precision not suffice? Clearly state at least one specific reason.

Such an analysis doesn't distinguish between different dynamic objects created at the same allocation size. This can lead to imprecision.

Problem 4 (10 pts.) Array section analysis is used to determine which sections (or areas) of an array are *modified* or *references* in a given piece of code. For example, in the following code snippet:

```
for i = 2, 100
    A[i] = A[i-1] - 10
end_for
```

MOD(A) is the interval [2, 100] and REF(A) is the interval [1, 99]. MOD and REF values may be symbolic; i.e. for A[i+3] = 0, MOD(A) is i+3.

```
1  for i = 2, 10
2          A[i] = 0
3  end_for
4
5  for i = 2, 100
6          if i > 50
7                  A[i] = A[i-1] – 10
8          end_if
9  end_for
```

a. (4 pts.) For each region in the code above identified below compute MOD(A) and REF(A) for that region

Lines	MOD(A)	REF(A)
2	i	
1-3	[2,10]	
7	i	i-1
6-8	[50, ∞]	[49, ∞]
5-9	[50, 100]	[49, 99]
1-9	[2, 10] ∪ [50, 100]	[49, 99]

b. (4 pts.) Outline an *interprocedural* and *region-based* algorithm for computing MOD and REF functions. Assume your program is well-formed, i.e. consisting of properly nested for's, if's, etc. Make sure to address the case of recursion.

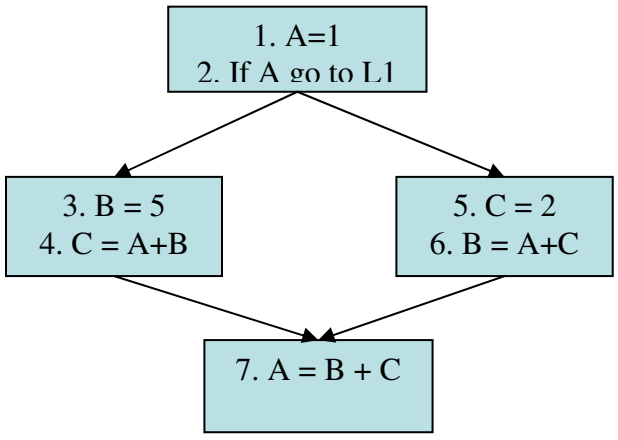
   Perform a bottom up analysis collecting accessed to all arrays involved. Whenever we ascend up a region (as in the case of the if on lines 6-8), combine the if or for condition with the current constraints. When propagating up a loop or a method call, project out the local variables (as in the case of the for on lines 1-3). Recursion requires a fixed point-iteration until all the MOD and REF values stabilize.

c. (2 pts.) Why may intraprocedural (within one procedure) parallelization be insufficient for real programs?

   Interprocedural parallelism is limited to how much is happening within a procedure. Parallelizing across procedures may uncover much greater parallelization opportunities in practice.

Problem 5 (20 pts.) Coloring for CS students

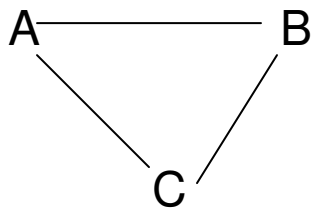
Below is a flow graph for a program. Each instruction is labeled by its instruction number on its left.



a) What are the (merged) live ranges in this program above? (Use the instruction numbers to describe the live ranges; specify clearly whether you are referring to the beginning or the end of an instruction).

	1	2	3	4	5	6	7
IN		**A**	**A**	**A,B**	**A**	**A,C**	**B,C**
OUT	**A**	**A**	**A,B**	**B,C**	**A,C**	**B,C**	**A**

b) Draw the interference graph for this program.

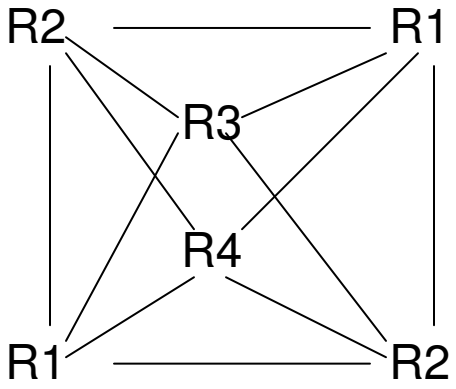


c) Are three registers sufficient to avoid spilling? If the answer is yes, show a mapping; if the answer is no, explain why not?

This question makes sense for two registers but for three the answer is trivially:

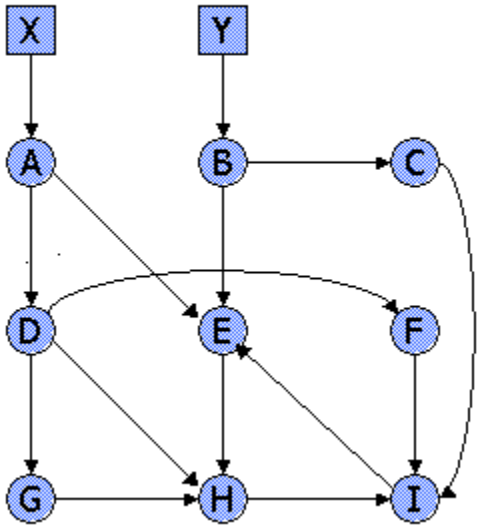
R1: A
R2: B
R3: C

d) Is color mapping algorithm optimal? If yes explain why? If not, give a counter example for a 4 colorable case.
No, it is not.



Problem 6. (20 pts) Time to take out the garbage

a) Here is a network of objects in a heap, with root objects X and Y.



The heap is managed by an incremental algorithm that uses the four lists *Unreached, Unscanned, Scanned*, and *Free*, as in Baker's algorithm. To be specific, the *Unscanned* list is managed as a queue, and when more than one object is to be placed on this list due to the scanning of one object, we do so in alphabetical order. Suppose also that we use write barriers to assure that no reachable object is made garbage. Starting with A and B on the *Unscanned* list, the following events occur:

1. The pointer B→C is rewritten to be B→I.
2. A is scanned.
3. B is scanned.
4. The next object on the *Unscanned* queue is scanned.
5. The pointer A→D is rewritten to be A→G.

Simulate the entire incremental garbage collection, assuming no more pointers are rewritten. In particular, which objects are placed on the *Scanned* list, and in which order?

Scan Order	Event	Unscanned Queue
-	**Initial**	**A,B**
-	**B→C becomes B→I**	**A,B**
1	**SCAN A**	**B,D,E**
2	**SCAN B**	**D,E,I**
3	**SCAN D**	**E,I,F,G,H**
-	**A→D becomes A→G**	**E,I,F,G,H**

4	**SCAN E**	**I,F,G,G**
5	**SCAN I**	**F,G,H**
6	**SCAN F**	**G,H**
7	**SCAN G**	**H**
8	**SCA N H**	**EMPTY**

Which objects wind up on the *Free* list and Unreachable list?

Free: C
Unreachable:

b) Why is mark-and-compact garbage collector better than mark-and-sweep garbage collector? Is it possible for a mark-and-sweep garbage collector to outperform a mark-and-compact garbage collector on some programs. If the answer is yes, describe a scenario for which that is true.

Mark-and-compact is better because it defragments memory allows allocation of bigger chunks of memory thus more spatial locality.

Yes. A heap that has every object live will take as long to mark. The sweep is unnecessary, but the compaction will take non-zero time. Another acceptable answer, when all memory allocations are of the same size, compaction does not add any additional benefit but still has the overhead.

d) Is reference counting always successful in collecting inaccessible space? If yes, explain why? If not, give an example where it would fail.

Reference counting will fail anytime there is a circular reference.

Problem 7 (20 pts.) Consider the following program:

```
for i = 2, 100
  for j = 1, 200
    for k = 10, 200
      A[i, j, k] = A[i-1, j-2, k+3]+4
    end_for
  end_for
end_for
```

(b)

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} -2 \\ 100 \\ -1 \\ 200 \\ -10 \\ 200 \end{bmatrix} \geq 0$$

(c)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} -1 \\ -2 \\ 3 \end{bmatrix}$$

(d) New iteration space:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} -2 \\ 100 \\ -1 \\ 200 \\ -10 \\ 200 \end{bmatrix} \geq 0$$

Array access for A[i-1,j-2,k+3]:

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} -1 \\ -2 \\ 3 \end{bmatrix}$$

(e) Generate the new loop nest using the transformation in (c).

```
for u = 1, 200
  for v = 10, 200
    for w = 2, 100
      A[w, u, v] = A[w-1, u-2, v+3]+4
    end_for
  end_for
end_for
```

Problem 8 (20 pts.) Consider the following program. Use the GCD test to analyze data dependencies.

```
for i = 20, 100
  A[3i] = A[2i-1]+4
  A[4i+2] = A[6i+7]
end_for
```

1. List all true dependencies, and explain why.

_____

_____A[3i] and A[2i-1]. The equation 3x -2y = -1 has solutions because GCD(3, -2) divides -1. _____

2. List all anti-dependencies, and explain why.

_____None.

_____

3. List all output dependencies, and explain why.

_____

_____ A[3i] and A[4i+2]. The equation 3x -4y = 2 has solutions because GCD(3, -4) divides 2. _____

4. List all input dependencies, and explain why.

_____ A[2i-1] and A[6i+7]. The equation 2x -6y = 8 has solutions because GCD(2, -6) divides 8.

_____

_____