Assignment 2

- Due: January 26, 1999. Email your solution to nspring@cs.washington.edu and philbe@microsoft.com.
- Read Sections 6.1 - 6.2 and the Appendix of Chapter 6 of the textbook.
- Background reading on recoverability and strictness is attached.

Problem 1

Consider the following history:

H = $w_0[x]$ $w_0[y]$ $w_0[z]$ c_0 $r_1[x]$ $r_2[x]$ $r_5[z]$ $r_4[z]$ $w_1[y]$ $w_2[x]$ c_2 $w_5[z]$ $r_4[y]$ $r_3[z]$ $w_3[x]$ $w_3[y]$ c_1 c_5 c_4 c_3

a. Draw the serialization graph for H. (If you're submitting in text format, just list all the edges.)
b. List all serial histories that are equivalent to H.
c. List the operation sequences in H, if any, that cause H to violate recoverability.
d. List the operation sequences in H, if any, that cause H to violate strictness.

Problem 2

RecMan is a record management system that stores records on pages. A record can either reside on one page, or span two pages (occupying the bottom of one page and the top of the next). RecMan implements five operations, ReadRecord, WriteRecord, Start, Commit, and Abort, as follows:

- ReadRecord(t, x, ptr) –
  - lock record x in read mode on behalf of transaction t
  - read the page or pages that contain record x into the caller's private memory and return a pointer, ptr, to the first byte of x
- WriteRecord(t, x, newval) –
  - lock record x in write mode on behalf of transaction t
  - if the page or pages that contain record x are not in the caller's private memory, then read them into the caller's private memory
  - replace the contents of x in caller's private memory by newval
  - write the page(s) that contain record x back to the file system
- Start(t) – create a new transaction and return its transaction identifier in parameter t
- Commit(t) – commit all of transaction t's writes; release all of t's locks
- Abort(t) – undo all of transaction t's writes; release all of t's locks

RecMan is two-phase locked but does not always produce serializable executions.

a. Give an example that shows RecMan is not serializable.
b. What broken assumption is causing two-phase locking to not guarantee serializability?

Problem 3

Give three good reasons why it's a bad idea to store locks in the database itself, rather than in a lock table. For concreteness, you may assume that

- the database consists of a set of files, each of which contains a set of records, each of which is divided up into fields;
- locks for each record are stored in the record itself; and
- the database system buffers (in main memory) those records that have been recently accessed, and uses a least-recently-used buffer replacement scheme.