Project 1

In this project, you will implement a program that simulates the behavior of the two-phase locking (2PL) protocol for concurrency control. The particular protocol to be implemented will be **rigorous 2PL**, with the *wound-wait* method for dealing with deadlock.

The input to your program will be a file of transaction operations in a particular sequence. Each line has a single transaction operation. The possible operations are: b (begin transaction), r (read item), w (write item), and e (end transaction). Each operation will be followed by a transaction id that is an integer between 1 and 99. For r and w operations, an item name follows between parentheses (item names are single letters from A to Z). An example is given below.

Examples of two input files:

```
b1;
r1 (Y);
w1 (Y);
r1 (Z);
b3;
r3 (X);
w3 (X);
w1 (Z);
e1;
r3 (Y);
b2;
r2 (Z);
w2 (Z);
w3 (Y);
e3;
r2 (X);
w2 (X);
e2;
```

```
b1;
r1(Y);
w1(Y);
r1(Z);
b2;
r2(Y);
b3;
r3(Z);
w1(Z);
e1;
w3(Z);
e3;
```

The transaction timestamps will be based on the transactions start order, and are integer numbers: 1, 2, 3, …. For example, in the first example input file (on left above), TS(T1) = 1, TS(T3) = 2, TS(T2) = 3.
You should do the following steps for Project 1:

1. Design and implement appropriate data structures to keep track of transactions (**transaction table**) and locks (**lock table**).
2. In the **transaction table**, you should keep relevant information about each transaction. This includes transaction id, transaction timestamp, transaction state (active, blocked (waiting), aborted (cancelled), committed, etc.), list of items locked by the transaction, plus *any other relevant information*. (It is part of your work to determine and specify other relevant information needed.) For blocked transaction, you should also keep an ordered list of the operations of that transaction that are waiting to be executed once the transaction can be resumed.
3. In the **lock table**, you should keep relevant information about each locked data item. This includes item name, lock state (read (share) locked, or write (exclusive) locked), transaction id for the transaction holding the lock (for write locked) or list of transaction ids for the transactions holding the lock (for read locked), list of transaction ids for transactions waiting for the item to be unlocked (if any), plus *any other relevant information*. (It is part of your work to determine and specify other relevant information.)
4. Write a program that reads the operations from the input schedule and simulates the appropriate actions for each operation by referring to and updating the entries in the transaction and lock tables. Your program should print a short summary of the action it takes to simulate each command, including information on any updates to the system tables (transaction table and lock table), and if the simulation will commit or abort or block a transaction, or just allow the operation to execute.

Some additional information about the actions that your program should take are as follows (this list is *not* an exhaustive list):

5. A transaction record should be created in the transaction table when a begin transaction operation (b) is processed (the state of this transaction should be active).
6. Before processing a read operation (r(X)), the appropriate **read lock(X)** request should be processed by your program, and the lock table should be updated appropriately. If the item X is already locked by a conflicting write lock, the transaction is either: (i) blocked and its transaction state is changed to blocked (in the transaction table), or (ii) the transaction is aborted (and its state is changed to aborted) if the deadlock prevention protocol (*wound-wait*) determines that the conflicting transaction should be aborted. If the item is already locked by a non-conflicting read lock, the transaction is added to the list of transactions that hold the read lock on the requested item (in the lock table).
7. Before processing a write operation (w(X)), the appropriate **write lock(X)** request should be processed by your program (lock upgrading is permitted if the upgrade conditions are met – that is, if the item is read locked by only the transaction that is requesting the write lock). The lock table should be updated appropriately. If the item is already locked by a conflicting read or write lock, the transaction is either: (i) blocked and its state is changed to blocked (in the transaction table), or (ii) the transaction is aborted (and its state is changed to aborted) if the deadlock prevention protocol (*wound-wait*) determines that the conflicting transaction should be aborted.
8. Before processing an operation in the input list, you should check if the transaction is in a blocked state or aborted state. (i) If it is blocked, add the operation to the ordered list of the operations of that transaction that are waiting to be executed (in the transaction table); these operations will be executed once the transaction is resumed, subject to the rules of the locking protocol. (ii) If it is aborted, its subsequent operations in the input list are ignored.
9. Before changing a transaction state to blocked, your program should check the *deadlock prevention* protocol (*wound-wait*) rules to determine if the transaction should wait (be blocked) or die (abort). The transaction timestamps are used to decide on which action to take.
10. The process of aborting a transaction should release (unlock) any items that are currently locked by the transaction, one at a time, and changing the transaction state to aborted in the transaction table. Any subsequent operations of the aborted transaction that are read from the input file should be ignored.
11. If a transaction reaches its end (e) operation successfully, it should be committed. The process of committing a transaction should release (unlock) any items that are currently locked by the transaction, one at a time, and changing the transaction state to committed in the transaction table.
12. The process of unlocking an item should check if any transaction(s) are blocked because of waiting for the item. If any transactions are waiting, it should remove the first waiting transaction and grant it access to the item (note that this will relock the item) and thus resume the transaction. All waiting operations of the transaction are processed (as discussed above, subject to the locking protocol) before any further operations from the input file are processed.