Programming Assignment 2: Pseudocode
CS5280

Darpan Gaur
CO21BTECH11004

BTO

variables

```
class transaction
{
    mutex tLock // lock for transaction
    int id; // transaction id
    // transaction status:  0: active, 1: commit, 2: abort
    int tStatus;
    vector<int> localmem; // local memory
    set<int> read_set;
    set<int> write_set;
}
class item
{
    mutex itemLock // lock for item
    int val; // value of item
    set<int> read_list;
    int R_TS; // read timestamp
    int W_TS; // write timestamp
}

// scheduler variables
mutex sch_lock; // lock for scheduler
vector<item> items; // vector of items
vector<transaction> tList; // vector of transactions
```

begin_trans

```
begin_trans()
{
    // returns the id for the transaction
    lock(sch_lock);
    int id = idCounter++;
    // create a new transaction
    transaction t= new transaction(id);
    // initialize the variables
    unlock(sch_lock);
    return id;
}
```

read(i, x, l)

```
read(i, x, l)
{
    // i is the transaction id
    // x is the variable to be read
    // store value of x in l
    lock(items[x]->itemLock); // lock the item
    lock(i->tLock); // lock the transaction
    if (i->id < items[x]->W_TS) {
        i->status = 2
        unlock(i->tLock);
        unlock(items[x]->itemLock);
        return -1;
    }
    if (i->status == 2) {
        items[x].read_list.erase(i->id);
        unlock(i->tLock);
        unlock(items[x]->itemLock);
        return -1;
    }
    items[x]->R_TS = i->id; // update the write timestamp
    *l = items[x].val; // read the value

    i->read_set.insert(x); // update the read set

    unlock(i->tLock);
    unlock(items[x]->itemLock);
    return 0;
}
```

write(i, x, l)

```
write(i, x, l)
{
    // i is the transaction id
    // x is the variable to be written
    // l is the value to be written
    lock(items[x]->itemLock); // lock the item
    lock(i->tLock); // lock the transaction
    if (i->id < items[x]->R_TS || i->id < items[x]->W_TS) {
        i->status = 2
        unlock(i->tLock);
        unlock(items[x]->itemLock);
        return -1;
    }
    items[x]->W_TS = i->id; // update the read timestamp
    i->localmem[x] = l; // write the value
    i->write_set.insert(x); // update the write set

    items[x]->read_list.insert(i->id); // update the read list

    unlock(i->tLock);
    unlock(items[x]->itemLock);
    return 0;
}
```

try_commit(i)

```
try_commit(i)
{
    // i is the transaction id
    lock(i->tLock); // lock the transaction
    if (i->status == 2) {
        for (auto x : i->read_set) {
            lock(items[x]->itemLock); // lock the item
            items[x]->read_list.erase(i->id);
            unlock(items[x]->itemLock); // unlock the item
            return -1;
        }
    }
    for (auto x : i->write_set) {
        lock(items[x]->itemLock); // lock the item
```

```
        items[x]->val = i->localmem[x]; // write the value
        unlock(items[x]->itemLock); // unlock the item
        unlock(i->tLock); // unlock the transaction
    }
    i->status = 1; // commit the transaction
    reutrn 0;
}
```

free_trans(i)

```
free_trans(i) {
    delete localMem
    delete read_set
    delete write_set
    remove i from read_list
}
```