```
tnc = 0; // current transaction id
void tbegin {
    read_set = new Set();
    write_set = new Set();
    start_tn = tnc;  //the transaction that finished just before this one started
}

// Serial Validation
boolean tend(Transaction T[], int start_tn, Set my_read_set, Set my_write_set) {
    lock();
    int finish_tn = tnc;  //the transaction that finished just before this one
    bool valid = true;

    for(int t = start_tn + 1; t <= finish_tn; t++)
        if(T[t].write_set intersects with my_read_set)
            valid = false;
    if(valid) {
        write_phase();
        tnc = tnc+1;
        tn = tnc;
    }
    unlock();
    if(valid)
        cleanup();
    else
        backup();

    return valid;
}

// Two-pass Serial Validation
boolean tend(Transaction T[], int start_tn, Set my_read_set, Set my_write_set) {
    int mid_tn = tnc;
    bool valid = true;

    for(int t = start_tn + 1; t <= mid_tn; t++)
        if(T[t].write_set intersects with my_read_set)
            valid = false;
    lock();
    int finish_tn = tnc;

    for(int t = mid_tn + 1; t <= finish_tn; t++)
        if(T[t].write_set intersects with my_read_set)
            valid = false;
    if(valid) {
        write_phase();
        tnc = tnc+1;
        tn = tnc;
    }
    unlock();

    if(valid)
        cleanup();
    else
        backup();

    return valid;
}
```

```
// Parallel Validation
List<Transaction> active = new List();
boolean tend(Transaction T[], int start_tn, Set my_read_set, Set my_write_set) {
    lock();
    int finish_tn = tnc;
    List<Transaction> finish_active = active.copy();  //transactions in tend concurrently
    active.append(me.id);
    unlock();

    bool valid = true;

    for(int t = start_tn + 1; t <= finish_tn; t++)
        if(T[t].write_set intersects with my_read_set)
            valid = false;

    for (id t : finish_active)
        if(T[t].write_set intersects with (my_read_set Union my_write_set))
            valid = false;

    if(valid) {
        write_phase();
        lock();
        tnc = tnc+1;
        tn = tnc;
        active.remove(me.id);
        unlock();
        cleanup();
    } else {
        active.remove(me.id);
        backup();
    }

    return valid;
}
```