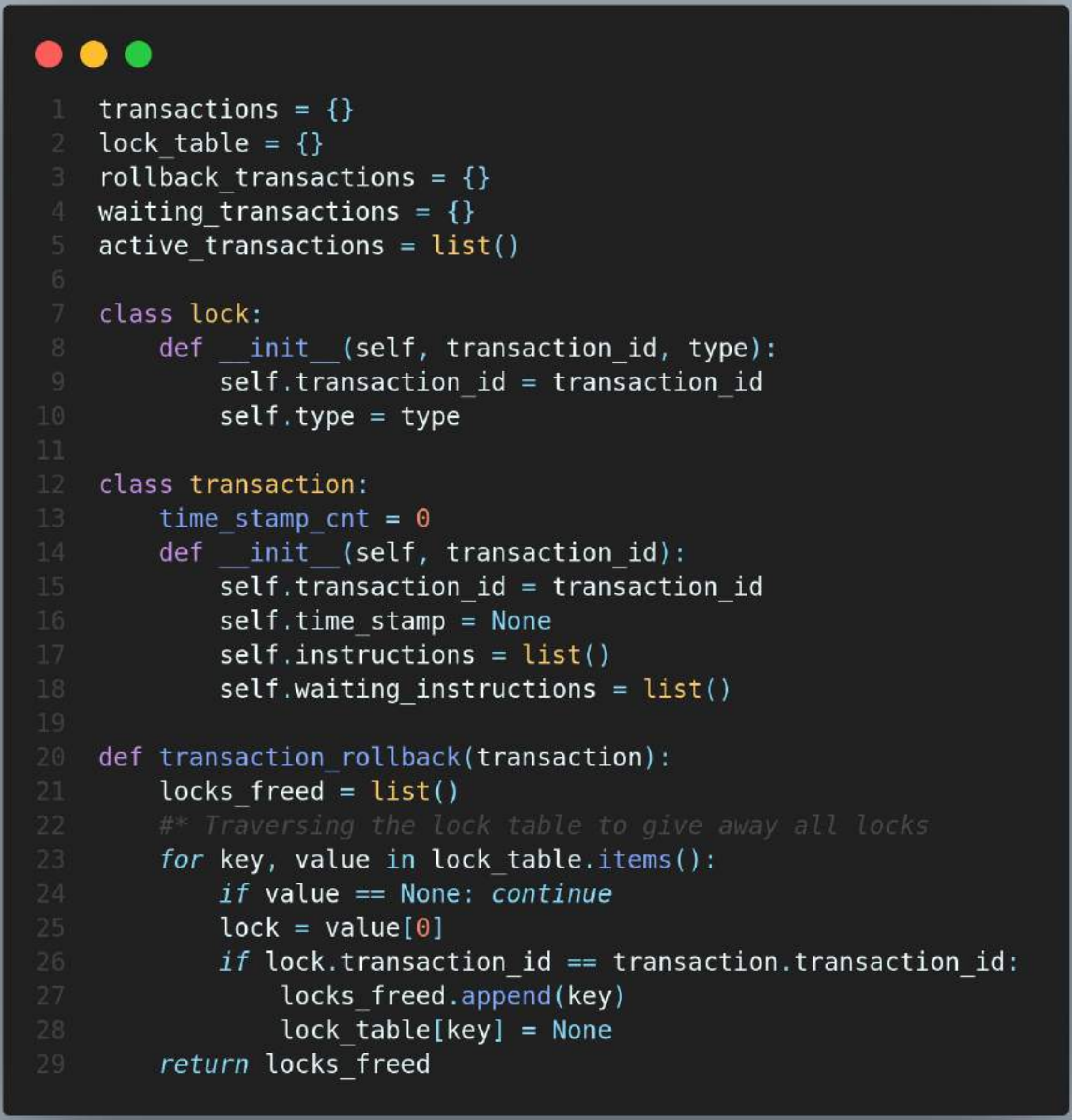
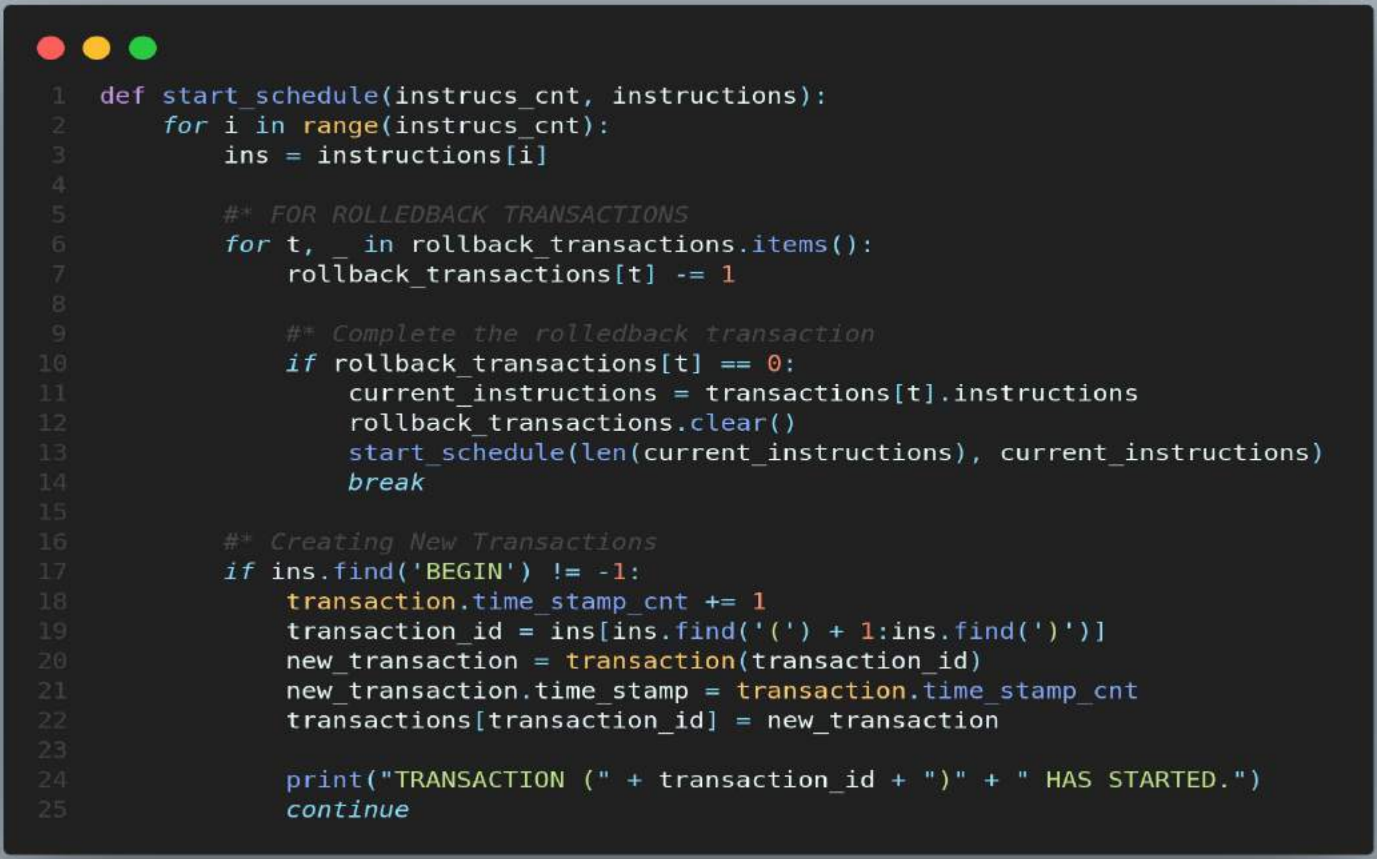
122022010 Shrayank Jai Mistry

Description: Write a program to simulate 2 phase locking protocal (rigorous 2 phase locking) Wound – Die Method For Deadlock Prevention.

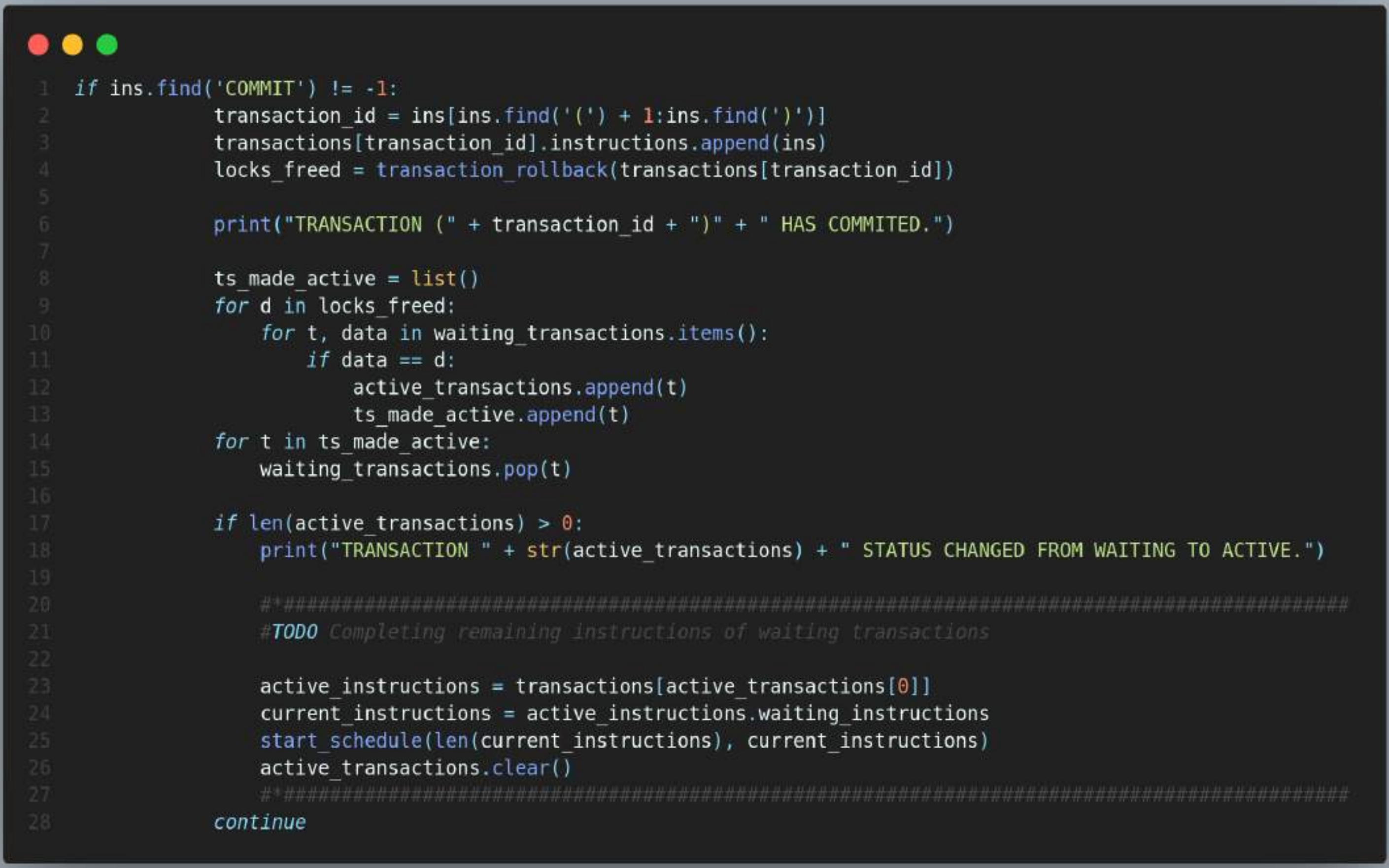
Programming Language: Python



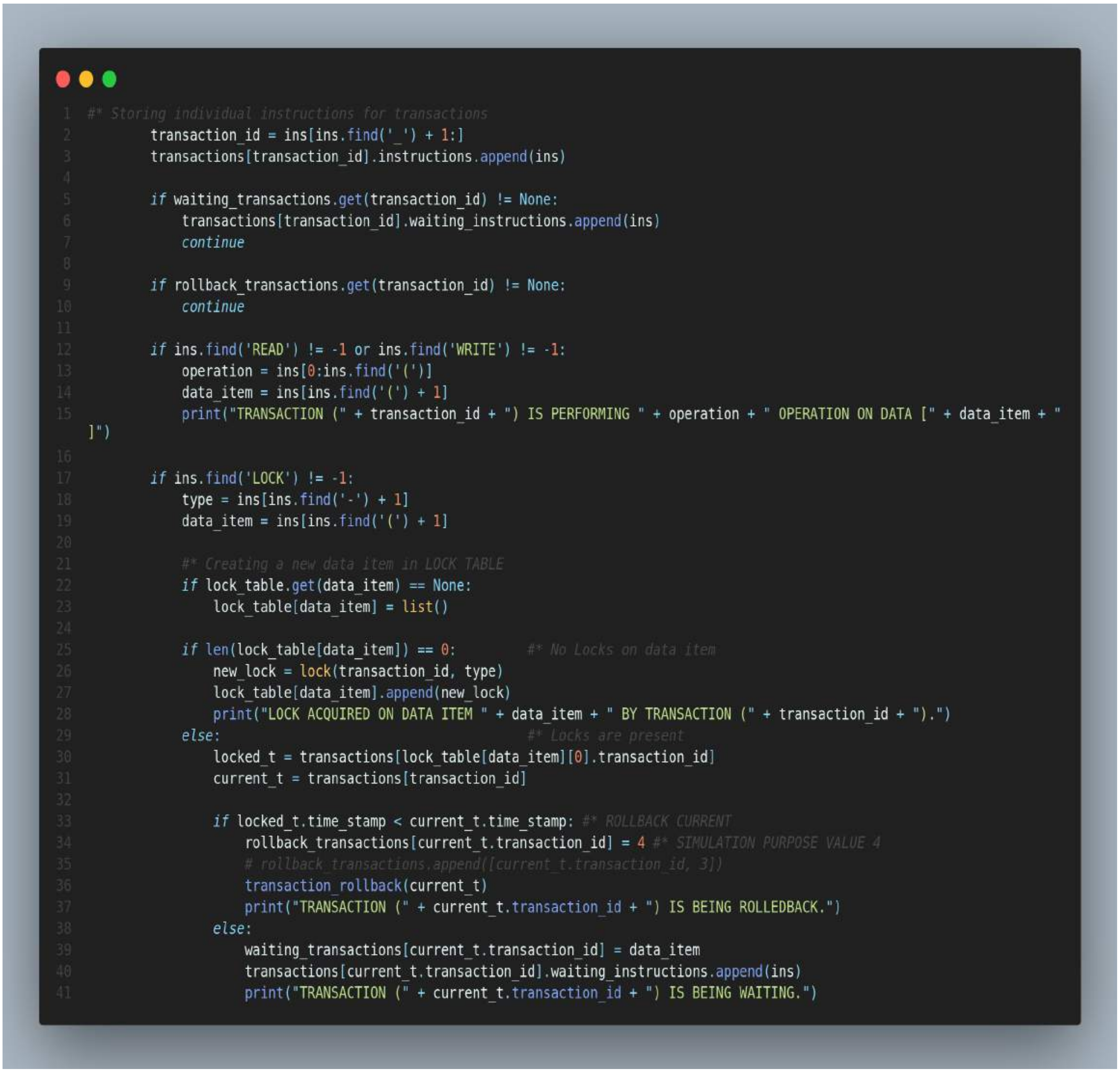
```python
transactions = {}
lock_table = {}
rollback_transactions = {}
waiting_transactions = {}
active_transactions = list()

class lock:
    def __init__(self, transaction_id, type):
        self.transaction_id = transaction_id
        self.type = type

class transaction:
    time_stamp_cnt = 0
    def __init__(self, transaction_id):
        self.transaction_id = transaction_id
        self.time_stamp = None
        self.instructions = list()
        self.waiting_instructions = list()

def transaction_rollback(transaction):
    locks_freed = list()
    #* Traversing the lock table to give away all locks
    for key, value in lock_table.items():
        if value == None: continue
        lock = value[0]
        if lock.transaction_id == transaction.transaction_id:
            locks_freed.append(key)
            lock_table[key] = None
    return locks_freed
```



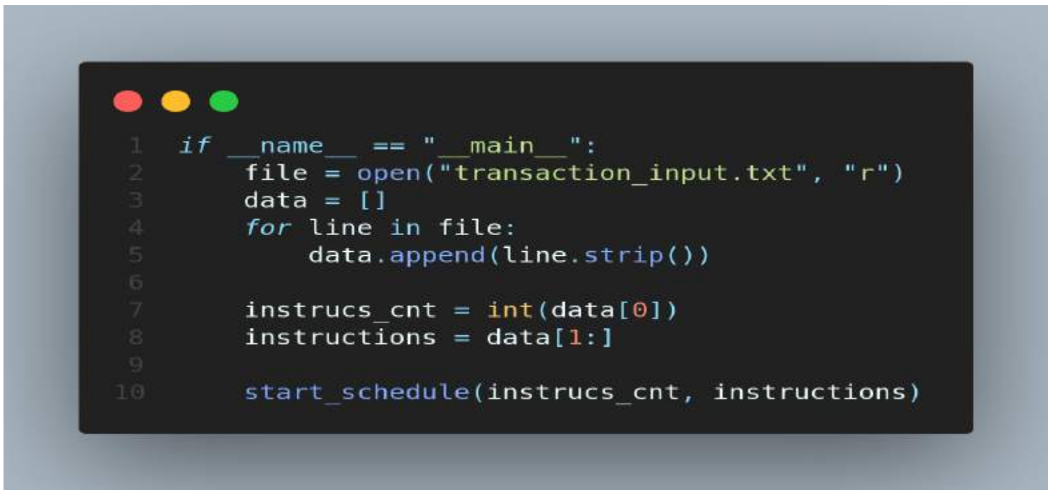
```python
def start_schedule(instrucs_cnt, instructions):
    for i in range(instrucs_cnt):
        ins = instructions[i]

        #* FOR ROLLEDBACK TRANSACTIONS
        for t, _ in rollback_transactions.items():
            rollback_transactions[t] -= 1

            #* Complete the rolledback transaction
            if rollback_transactions[t] == 0:
                current_instructions = transactions[t].instructions
                rollback_transactions.clear()
                start_schedule(len(current_instructions), current_instructions)
                break

        #* Creating New Transactions
        if ins.find('BEGIN') != -1:
            transaction.time_stamp_cnt += 1
            transaction_id = ins[ins.find('(') + 1:ins.find(')')]
            new_transaction = transaction(transaction_id)
            new_transaction.time_stamp = transaction.time_stamp_cnt
            transactions[transaction_id] = new_transaction

            print("TRANSACTION (" + transaction_id + ")" + " HAS STARTED.")
            continue
```



```python
if ins.find('COMMIT') != -1:
            transaction_id = ins[ins.find('(') + 1:ins.find(')')]
            transactions[transaction_id].instructions.append(ins)
            locks_freed = transaction_rollback(transactions[transaction_id])

            print("TRANSACTION (" + transaction_id + ")" + " HAS COMMITED.")

            ts_made_active = list()
            for d in locks_freed:
                for t, data in waiting_transactions.items():
                    if data == d:
                        active_transactions.append(t)
                        ts_made_active.append(t)
            for t in ts_made_active:
                waiting_transactions.pop(t)

            if len(active_transactions) > 0:
                print("TRANSACTION " + str(active_transactions) + " STATUS CHANGED FROM WAITING TO ACTIVE.")

                #*##############################################################################################
                #TODO Completing remaining instructions of waiting transactions

                active_instructions = transactions[active_transactions[0]]
                current_instructions = active_instructions.waiting_instructions
                start_schedule(len(current_instructions), current_instructions)
                active_transactions.clear()
                #*##############################################################################################
            continue
```

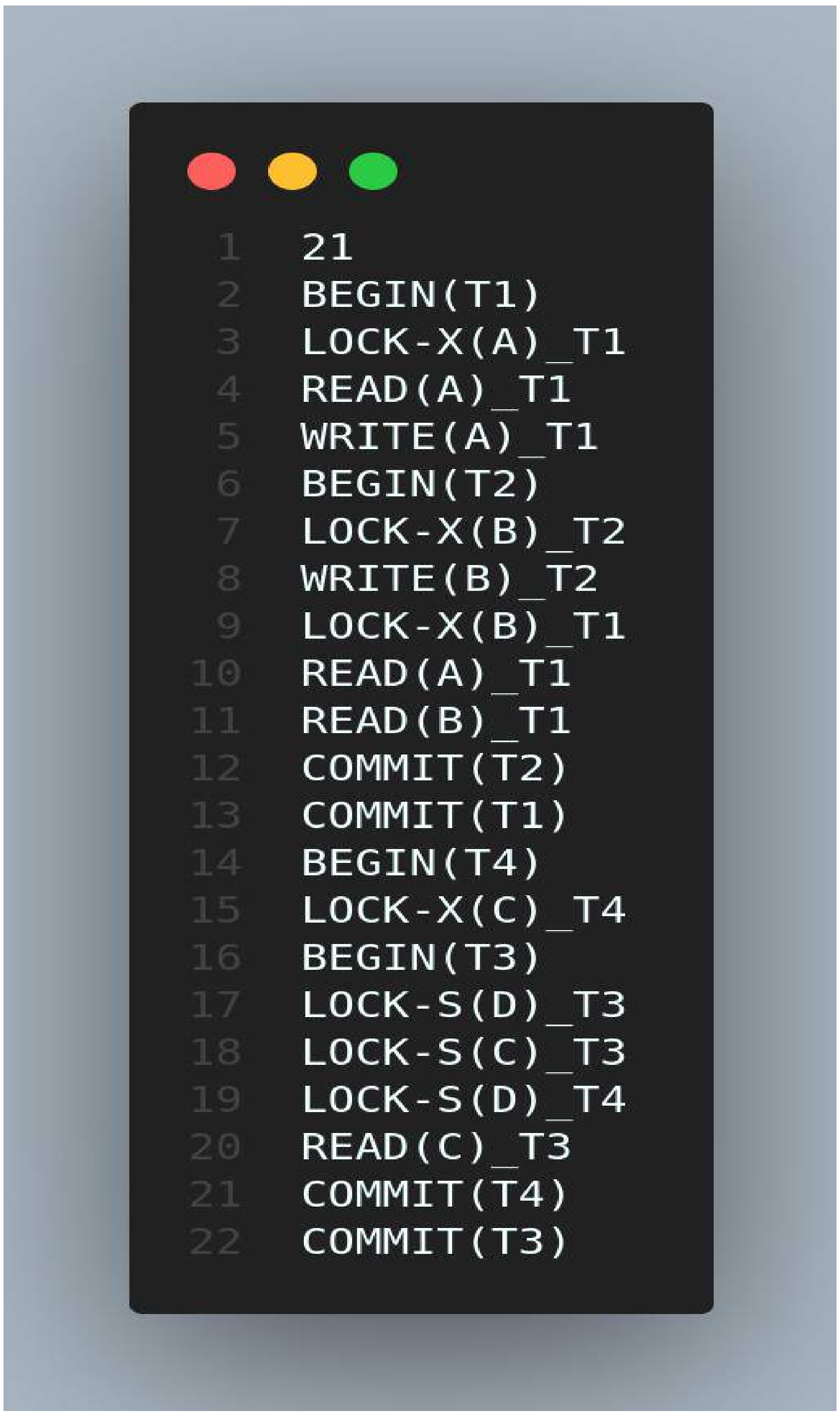


```python
#* Storing individual instructions for transactions
        transaction_id = ins[ins.find('_') + 1:]
        transactions[transaction_id].instructions.append(ins)

        if waiting_transactions.get(transaction_id) != None:
            transactions[transaction_id].waiting_instructions.append(ins)
            continue

        if rollback_transactions.get(transaction_id) != None:
            continue

        if ins.find('READ') != -1 or ins.find('WRITE') != -1:
            operation = ins[0:ins.find('(')]
            data_item = ins[ins.find('(') + 1]
            print("TRANSACTION (" + transaction_id + ") IS PERFORMING " + operation + " OPERATION ON DATA [" + data_item + " ]")

        if ins.find('LOCK') != -1:
            type = ins[ins.find('-') + 1]
            data_item = ins[ins.find('(') + 1]

            #* Creating a new data item in LOCK TABLE
            if lock_table.get(data_item) == None:
                lock_table[data_item] = list()

            if len(lock_table[data_item]) == 0:          #* No Locks on data item
                new_lock = lock(transaction_id, type)
                lock_table[data_item].append(new_lock)
                print("LOCK ACQUIRED ON DATA ITEM " + data_item + " BY TRANSACTION (" + transaction_id + ").")
            else:                                        #* Locks are present
                locked_t = transactions[lock_table[data_item][0].transaction_id]
                current_t = transactions[transaction_id]

                if locked_t.time_stamp < current_t.time_stamp: #* ROLLBACK CURRENT
                    rollback_transactions[current_t.transaction_id] = 4 #* SIMULATION PURPOSE VALUE 4
                    # rollback_transactions.append([current_t.transaction_id, 3])
                    transaction_rollback(current_t)
                    print("TRANSACTION (" + current_t.transaction_id + ") IS BEING ROLLEDBACK.")
                else:
                    waiting_transactions[current_t.transaction_id] = data_item
                    transactions[current_t.transaction_id].waiting_instructions.append(ins)
                    print("TRANSACTION (" + current_t.transaction_id + ") IS BEING WAITING.")
```



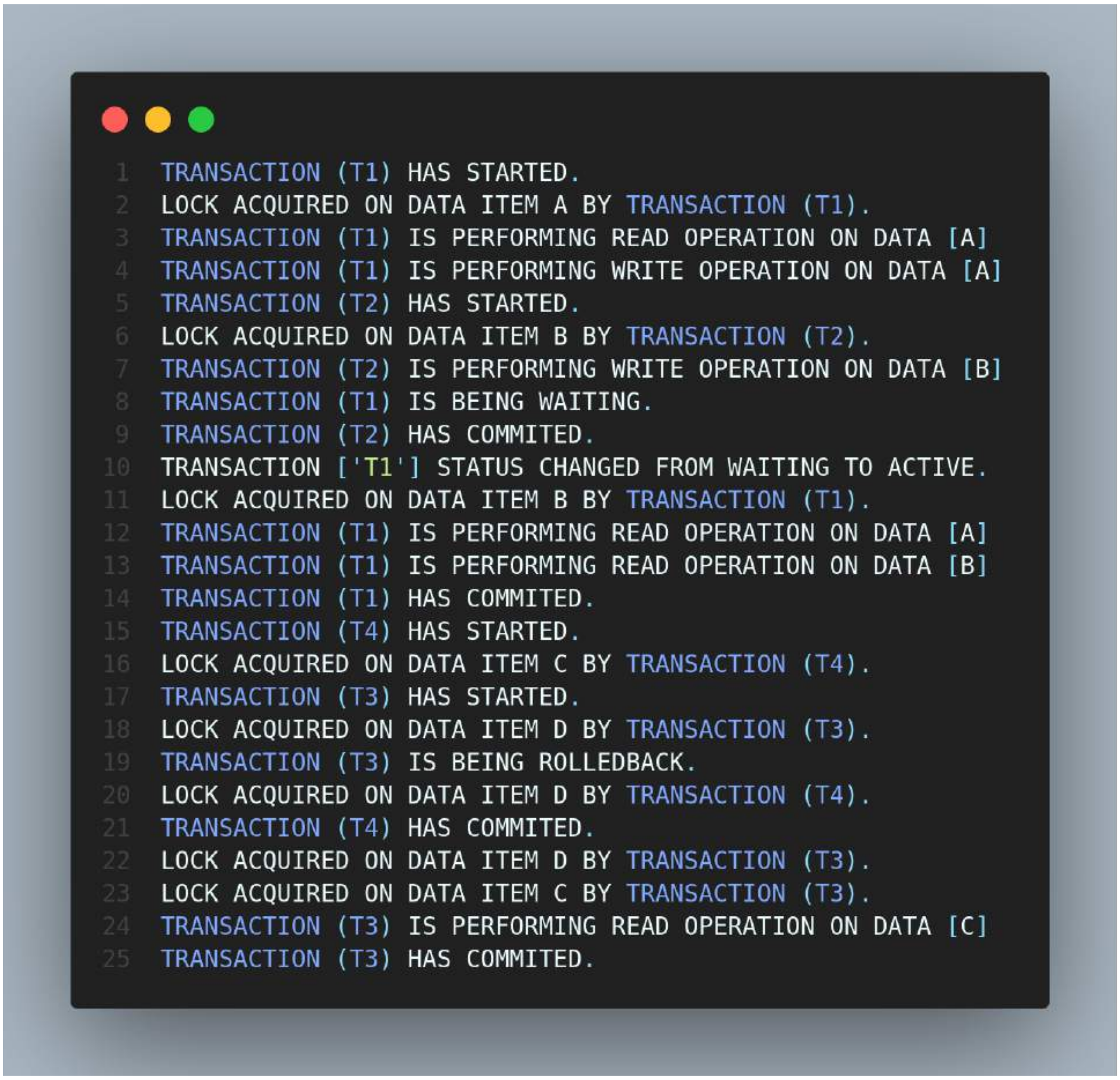
```python
if __name__ == "__main__":
    file = open("transaction_input.txt", "r")
    data = []
    for line in file:
        data.append(line.strip())

    instrucs_cnt = int(data[0])
    instructions = data[1:]

    start_schedule(instrucs_cnt, instructions)
```

INPUT FILE:



```
21
BEGIN(T1)
LOCK-X(A)_T1
READ(A)_T1
WRITE(A)_T1
BEGIN(T2)
LOCK-X(B)_T2
WRITE(B)_T2
LOCK-X(B)_T1
READ(A)_T1
READ(B)_T1
COMMIT(T2)
COMMIT(T1)
BEGIN(T4)
LOCK-X(C)_T4
BEGIN(T3)
LOCK-S(D)_T3
LOCK-S(C)_T3
LOCK-S(D)_T4
READ(C)_T3
COMMIT(T4)
COMMIT(T3)
```

OUTPUT :



```
TRANSACTION (T1) HAS STARTED.
LOCK ACQUIRED ON DATA ITEM A BY TRANSACTION (T1).
TRANSACTION (T1) IS PERFORMING READ OPERATION ON DATA [A]
TRANSACTION (T1) IS PERFORMING WRITE OPERATION ON DATA [A]
TRANSACTION (T2) HAS STARTED.
LOCK ACQUIRED ON DATA ITEM B BY TRANSACTION (T2).
TRANSACTION (T2) IS PERFORMING WRITE OPERATION ON DATA [B]
TRANSACTION (T1) IS BEING WAITING.
TRANSACTION (T2) HAS COMMITED.
TRANSACTION ['T1'] STATUS CHANGED FROM WAITING TO ACTIVE.
LOCK ACQUIRED ON DATA ITEM B BY TRANSACTION (T1).
TRANSACTION (T1) IS PERFORMING READ OPERATION ON DATA [A]
TRANSACTION (T1) IS PERFORMING READ OPERATION ON DATA [B]
TRANSACTION (T1) HAS COMMITED.
TRANSACTION (T4) HAS STARTED.
LOCK ACQUIRED ON DATA ITEM C BY TRANSACTION (T4).
TRANSACTION (T3) HAS STARTED.
LOCK ACQUIRED ON DATA ITEM D BY TRANSACTION (T3).
TRANSACTION (T3) IS BEING ROLLEDBACK.
LOCK ACQUIRED ON DATA ITEM D BY TRANSACTION (T4).
TRANSACTION (T4) HAS COMMITED.
LOCK ACQUIRED ON DATA ITEM D BY TRANSACTION (T3).
LOCK ACQUIRED ON DATA ITEM C BY TRANSACTION (T3).
TRANSACTION (T3) IS PERFORMING READ OPERATION ON DATA [C]
TRANSACTION (T3) HAS COMMITED.
```