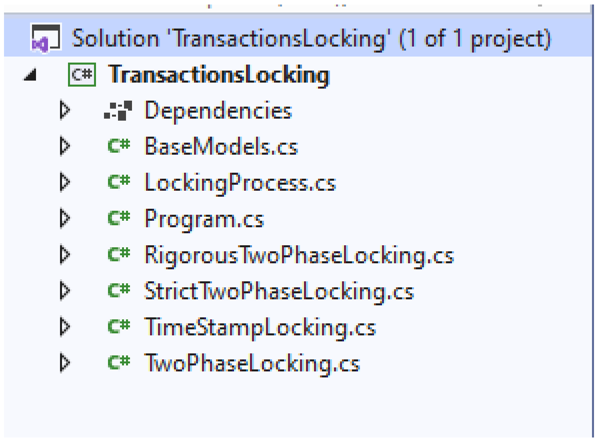
Project Report for Advanced Database Course

The University of Tehran

Student name: Sirvan Nasiri Kia

Professor: Dr. Keyhanipour

The current project aims to execute parallel transaction actions under locking algorithms including 2PL, Strict 2PL, Rigorous 2PL, and TimeStamp. The project is implemented in C# language using Visual Studio 2019 and under the .Net Core 3.1 platform. Note that input files are placed in the program's execution path, i.e. TransactionsLocking\bin\Debug\netcoreapp3.1. In the project structure, a separate file is considered for implementing each of the desired algorithms.



```
Solution 'TransactionsLocking' (1 of 1 project)
  TransactionsLocking
    Dependencies
    BaseModels.cs
    LockingProcess.cs
    Program.cs
    RigorousTwoPhaseLocking.cs
    StrictTwoPhaseLocking.cs
    TimeStampLocking.cs
    TwoPhaseLocking.cs
```

BaseModels contain basic classes and view models. LockingProcess is an abstract class that includes basic and common methods such as PrintOutput, InsertToWaintingList, InsertToStack, SetLock, and ReleaseLock. Other classes related to each of the protocols inherit from this abstract class.

_operationsList: A list of instructions for executing each operation is a line of input file. The type of operation (Read, Write, UnLock, Commit) is specified in each operation. Read and Write operations are read from the input file, but UnLock and Commit are added to the end of instructions based on the type of algorithm used.

_locksList: A list to hold created locks on data items.

_waitingsList: A list to hold a queue of transaction actions waiting for each data item in a list structure as follows:

_stacksList: A list of stacks that holds the execution flow of each transaction action from the first operation to the last executed operation.

Initiate: This function reads the input text file and initializes the list of operations (_operationsList) based on it.

PrintOutput: Used to display and print the output.

InsertToWaitingList: Adds a transaction action's waiting request to a data item in a queue structure.

InsertToStack: Adds an operation from a transaction action to its stack list (meaning that this operation has been executed).

SetLock: Locks a data item by a transaction action.

ReleaseWaitingItem: Used to release a waiting request on a data item by a transaction action (this method is called when a lock request is executed and its waiting request is removed from the queue).

ReleaseLock: Releases a lock on a data item by a transaction action.

CheckLock: Checks if a data item is locked.

CheckAvailability: Checks if the previous commands related to this operation have been executed in that transaction action.

CheckIsAvailable: Checks if the type of operation on this data item can be executed by this transaction action (not locked and previous commands have been executed).

Two Phase Locking

To execute this algorithm, in the first step, we determine the LockPoints on _operationsList using the SetLockPoints function. Then, the Calculate function performs the main operation. This algorithm continues its work until there are no executable commands left. It starts executing the available commands in a for loop from the first command. Depending on the type of operation and the data item it is working on, it decides whether to lock the desired data item and add the command to its transaction action's stack or add a waiting request for that data item or postpone its execution to the future. Finally, if no command has been executed, the schedule has been successfully completed. Otherwise, if no command is executed in one round (checked by availableItem), it means there is a DeadLock.

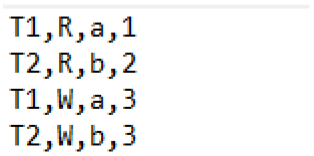
Project Report for Advanced Database Course

The University of Tehran

Student name: Sirvan Nasiri Kia

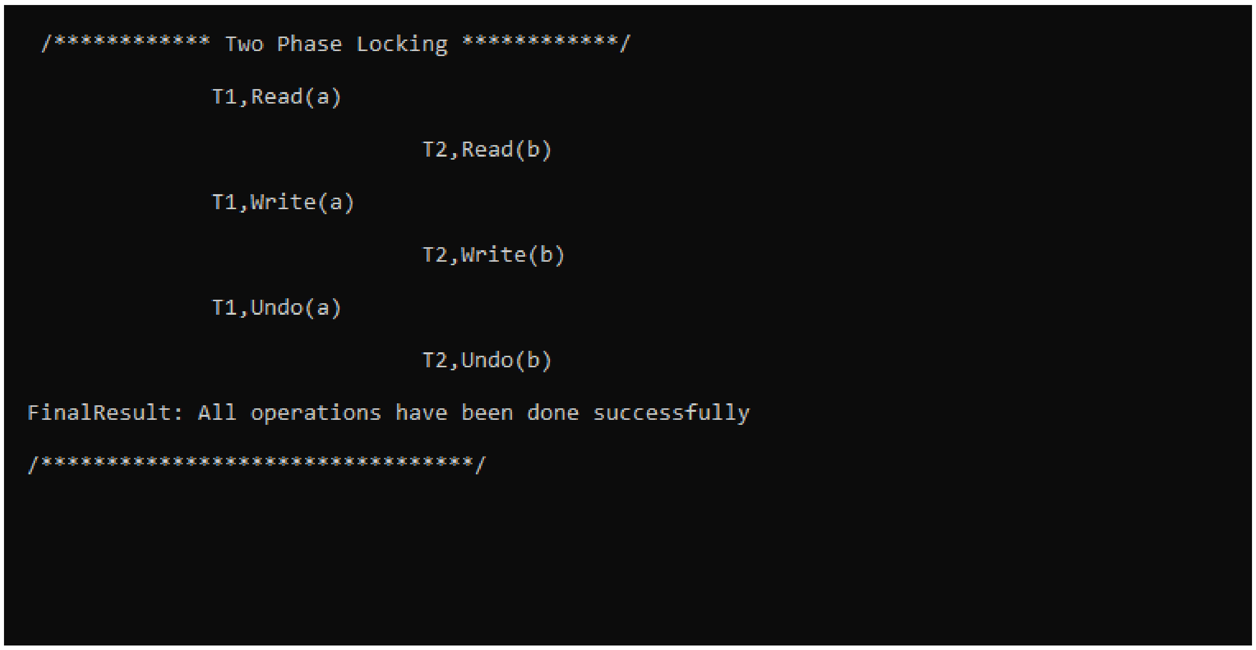
Professor: Dr. Keyhanipour

Input file 1:

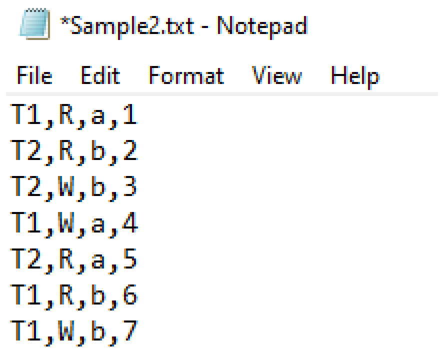


```
T1,R,a,1
T2,R,b,2
T1,W,a,3
T2,W,b,3
```

Output



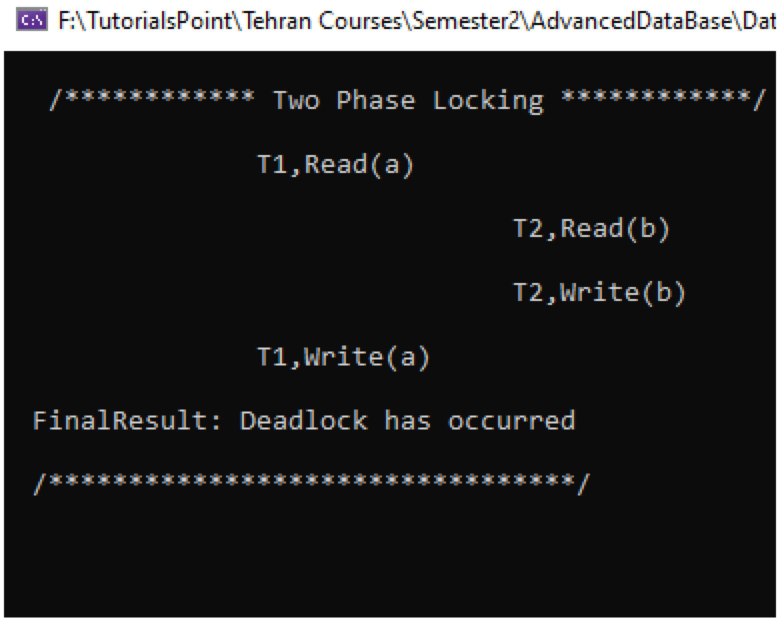
```
 /************ Two Phase Locking ************/

             T1,Read(a)

                             T2,Read(b)

             T1,Write(a)

                             T2,Write(b)

             T1,Undo(a)

                             T2,Undo(b)

FinalResult: All operations have been done successfully

/*********************************/
```



*Sample2.txt - Notepad

```
T1,R,a,1
T2,R,b,2
T2,W,b,3
T1,W,a,4
T2,R,a,5
T1,R,b,6
T1,W,b,7
```

Input File 2:

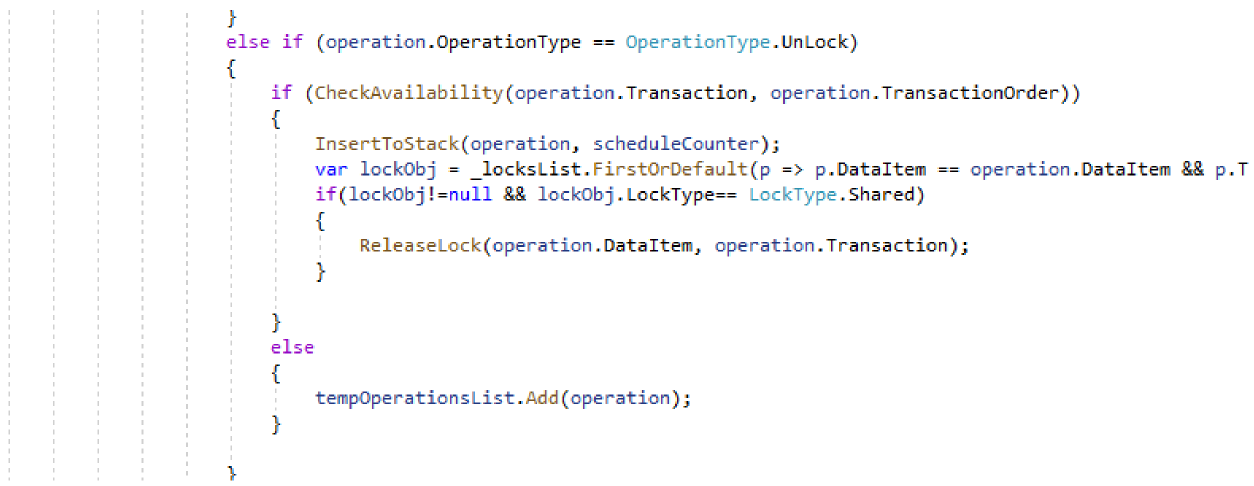


F:\TutorialsPoint\Tehran Courses\Semester2\AdvancedDataBase\Dat

```
  /************ Two Phase Locking ************/

              T1,Read(a)

                               T2,Read(b)

                               T2,Write(b)

              T1,Write(a)

 FinalResult: Deadlock has occurred

 /**********************************/
```

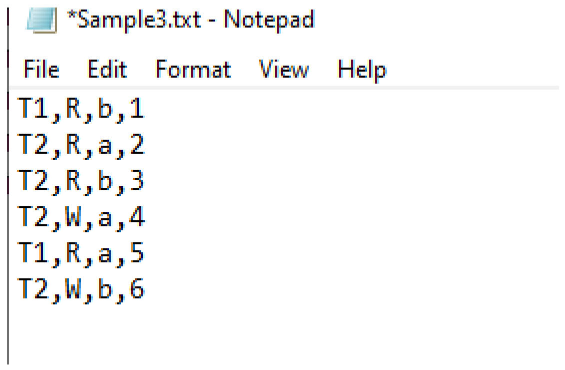
Strict Two Phase Locking

The structure of the strict two phase locking algorithm is exactly like the basic 2PL. The difference is that when the setrelatedoperations function is executed at the beginning of program execution, in addition to initializing the unlock points, it also executes the corresponding commit points at the end of the commands. The release of exclusive locks only occurs at the commit point. The changes made only release share locks at the unlock point, and other lock releases occur at the commit point.



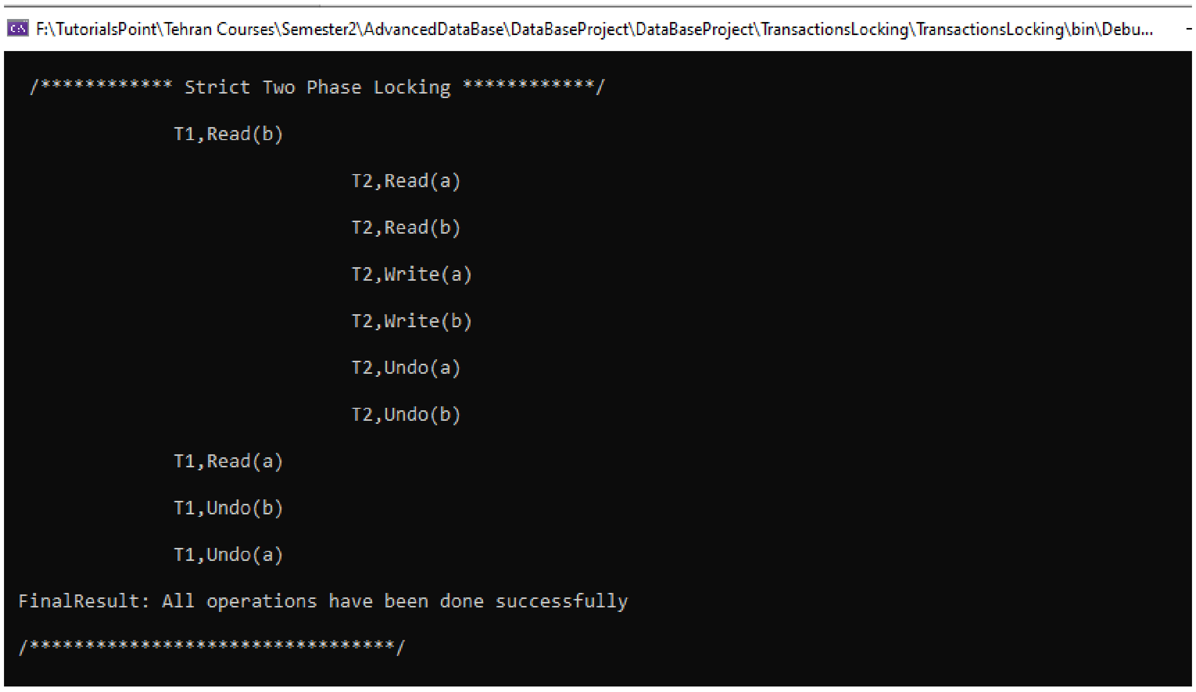
```
}
else if (operation.OperationType == OperationType.UnLock)
{
    if (CheckAvailability(operation.Transaction, operation.TransactionOrder))
    {
        InsertToStack(operation, scheduleCounter);
        var lockObj = _locksList.FirstOrDefault(p => p.DataItem == operation.DataItem && p.T
        if(lockObj!=null && lockObj.LockType== LockType.Shared)
        {
            ReleaseLock(operation.DataItem, operation.Transaction);
        }

    }
    else
    {
        tempOperationsList.Add(operation);
    }

}
```

Input File:



```
T1,R,b,1
T2,R,a,2
T2,R,b,3
T2,W,a,4
T1,R,a,5
T2,W,b,6
```

Output:

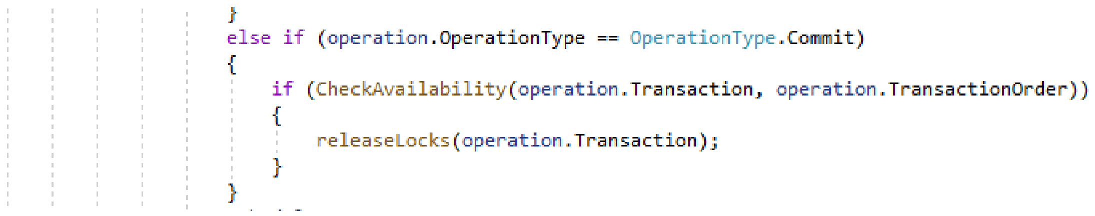


F:\TutorialsPoint\Tehran Courses\Semester2\AdvancedDataBase\DataBaseProject\DataBaseProject\TransactionsLocking\TransactionsLocking\bin\Debu...

```
 /************ Strict Two Phase Locking ************/
             T1,Read(b)
                             T2,Read(a)
                             T2,Read(b)
                             T2,Write(a)
                             T2,Write(b)
                             T2,Undo(a)
                             T2,Undo(b)
             T1,Read(a)
             T1,Undo(b)
             T1,Undo(a)
FinalResult: All operations have been done successfully
/**********************************/
```

Rigorous Two Phase Locking

The structure of this algorithm is exactly like the strict mode, with the difference that no lock is released until the commit.



```
}
else if (operation.OperationType == OperationType.Commit)
{
    if (CheckAvailability(operation.Transaction, operation.TransactionOrder))
    {
        releaseLocks(operation.Transaction);
    }
}
```

Based Time Stamp Protocol

The following variables and functions are used to execute this algorithm:

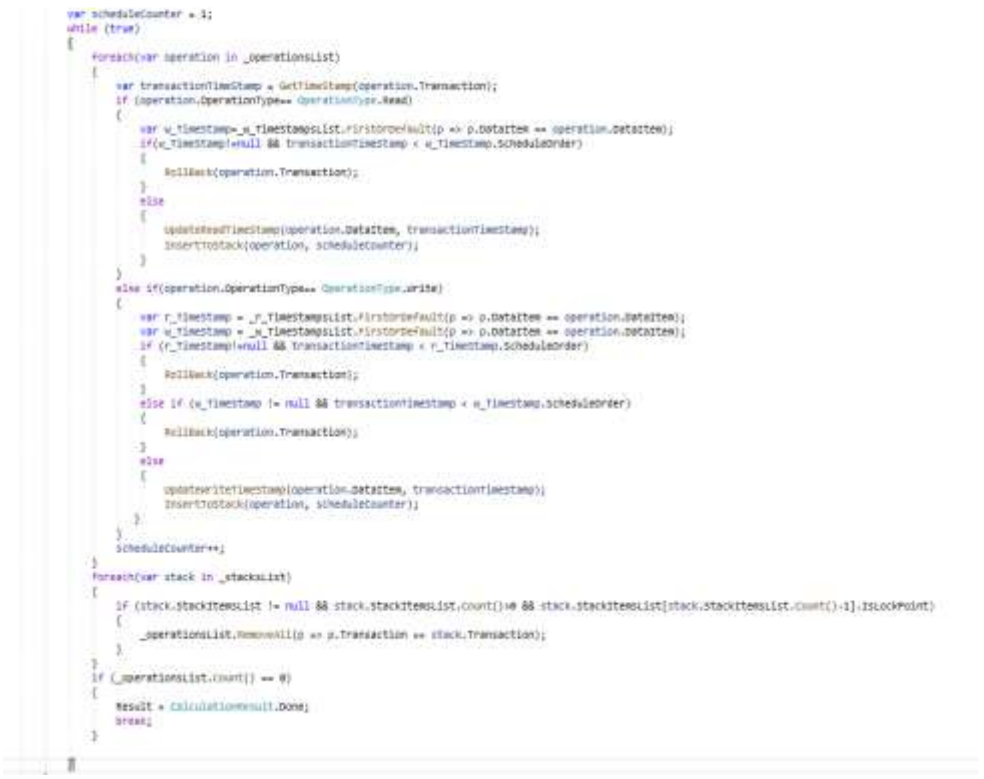
_w_TimeStampsList: A list of TimeStapModelDataItem class to store the last value of w_TimeStamp for each data item.

_r_TimeStampsList: A list of TimeStapModelDataItem class to store the last value of r_TimeStamp for each data item.

_timeStampsList: A list of TimeStampModelTransaction class to store the TimeStamp of each transaction.

RollBack(string transaction): To completely roll back a transaction, releasing the locks taken by it and emptying the execution stack.

This algorithm continues to work until an operation is available for execution. Otherwise, it stops. In each execution, a for loop is applied to the available operations, and based on the type of operation and the values of transactionTimeStamp, w_TimeStamp, and r_TimeStamp, it decides whether to execute the operation or perform a rollback.



```
var scheduleCounter = 1;
while (true)
{
    foreach(var operation in _operationsList)
    {
        var transactionTimeStamp = GetTimeStamp(operation.Transaction);
        if (operation.OperationType== OperationType.Read)
        {
            var w_timestamp=_w_TimestampsList.FirstOrDefault(p => p.DataItem == operation.DataItem);
            if(w_TimeStamp!=null && transactionTimeStamp < w_TimeStamp.ScheduleOrder)
            {
                RollBack(operation.Transaction);
            }
            else
            {
                UpdateReadTimeStamp(operation.DataItem, transactionTimeStamp);
                InsertToStack(operation, scheduleCounter);
            }
        }
        else if(operation.OperationType== OperationType.Write)
        {
            var r_TimeStamp = _r_TimestampsList.FirstOrDefault(p => p.DataItem == operation.DataItem);
            var w_TimeStamp = _w_TimestampsList.FirstOrDefault(p => p.DataItem == operation.DataItem);
            if (r_TimeStamp!=null && TransactionTimeStamp < r_TimeStamp.ScheduleOrder)
            {
                RollBack(operation.Transaction);
            }
            else if (w_Timestamp != null && transactionTimestamp < w_Timestamp.ScheduleOrder)
            {
                RollBack(operation.Transaction);
            }
            else
            {
                UpdateWriteTimestamp(operation.DataItem, transactionTimestamp);
                InsertToStack(operation, scheduleCounter);
            }
        }
        scheduleCounter++;
    }
    foreach(var stack in _stacksList)
    {
        if (stack.StackItemsList != null && stack.StackItemsList.Count()>0 && stack.StackItemsList[stack.StackItemsList.Count()-1].IsLockPoint)
        {
            _operationsList.RemoveAll(p => p.Transaction == stack.Transaction);
        }
    }
    if (_operationsList.Count() == 0)
    {
        Result = CalculationResult.Done;
        break;
    }
}
```

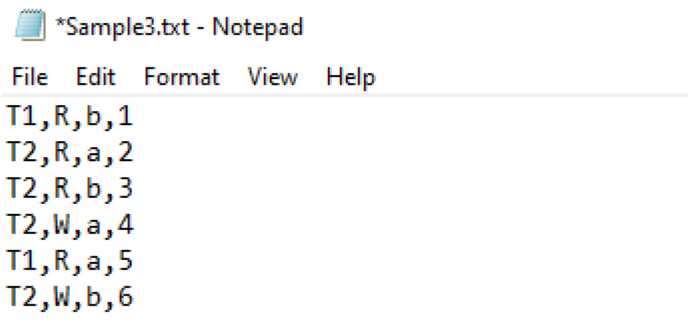
Input:

Project Report for Advanced Database Course

The University of Tehran

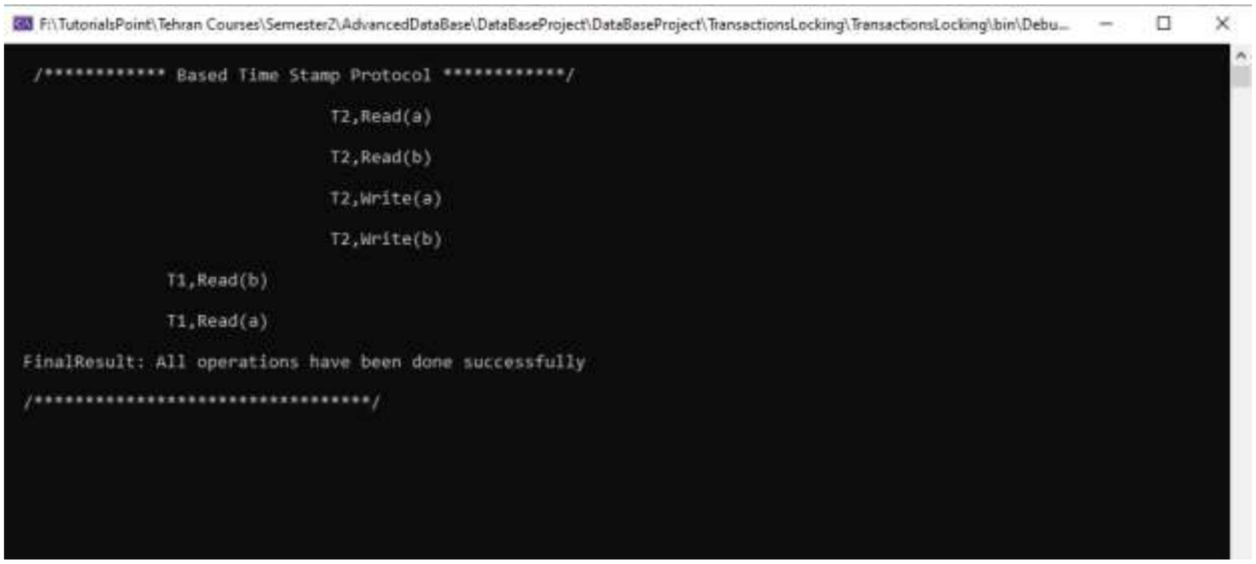
Student name: Sirvan Nasiri Kia

Professor: Dr. Keyhanipour



```
T1,R,b,1
T2,R,a,2
T2,R,b,3
T2,W,a,4
T1,R,a,5
T2,W,b,6
```

Output:


F:\TutorialsPoint\Tehran Courses\Semester2\AdvancedDataBase\DataBaseProject\DataBaseProject\TransactionsLocking\TransactionsLocking\bin\Debu...

```
 /************ Based Time Stamp Protocol ************/

                              T2,Read(a)

                              T2,Read(b)

                              T2,Write(a)

                              T2,Write(b)

               T1,Read(b)

               T1,Read(a)

FinalResult: All operations have been done successfully

/**********************************/
```