Experiment No.5
To design a smart contract using Solidity and Remix IDE
Date of Performance:
Date of Submission:

AIM: To design a smart contract using Solidity and Remix IDE

Objective: To develop a program in solidity to demonstrate smart contract in ethereum blockchain

Theory:

Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met.

Smart contracts work by following simple "if/when…then…" statements that are written into code on a blockchain. A network of computers executes the actions when predetermined conditions have been met and verified. These actions could include releasing funds to the appropriate parties, registering a vehicle, sending notifications, or issuing a ticket. The blockchain is then updated when the transaction is completed. That means the transaction cannot be changed, and only parties who have been granted permission can see the results.

Within a smart contract, there can be as many stipulations as needed to satisfy the participants that the task will be completed satisfactorily. To establish the terms, participants must determine how transactions and their data are represented on the blockchain, agree on the "if/when...then…" rules that govern those transactions, explore all possible exceptions, and define a framework for resolving disputes.

Then the smart contract can be programmed by a developer – although increasingly, organizations that use blockchain for business provide templates, web interfaces, and other online tools to simplify structuring smart contracts.

Solidity for smart contracts

Solidity is an object-oriented programming language created specifically by the Ethereum Network team for constructing and designing smart contracts on Blockchain platforms.

It's used to create smart contracts that implement business logic and generate a chain of transaction records in the blockchain system.

It acts as a tool for creating machine-level code and compiling it on the Ethereum Virtual Machine (EVM).

It has a lot of similarities with C and C++ and is pretty simple to learn and understand. For example, a "main" in C is equivalent to a "contract" in Solidity.

Like other programming languages, Solidity programming also has variables, functions, classes, arithmetic operations, string manipulation, and many other concepts.

- The Ethereum Virtual Machine (EVM) provides a runtime environment for Ethereum smart contracts.

- It is primarily concerned with ensuring the security and execution of untrusted programs through the use of an international network of public nodes.

- EVM is specialized in preventing Denial-of-Service attacks and certifies that the programs do not have access to each other's state, as well as establishing communication, with no possible interference.

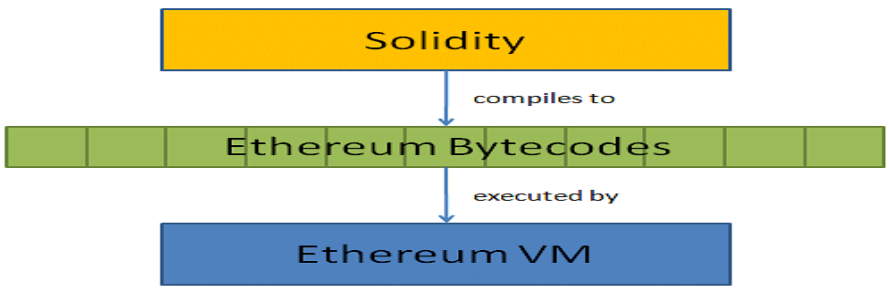


Fig.5.1 EVM in Ethereum Blockchain

- Smart contracts refer to high-level program codes compiled into EVM before being posted to the Ethereum blockchain for execution.

- It enables you to conduct trustworthy transactions without the involvement of a third party; these transactions are traceable and irreversible.

Process:

Step 1. Open Remix **IDE** by typing URL https://remix.ethereum.org/.

Step 2. In Remix IDE select 'Solidity' plugins

Step 3. Click on File Explorer

Step 4. In the default workspace click on 'create new file'

Step 5. Give suitable name to the *file* with extension .sol

Step 6. Type the (smart contract) code in the editor section of the Remix IDE for newly created file (.sol)

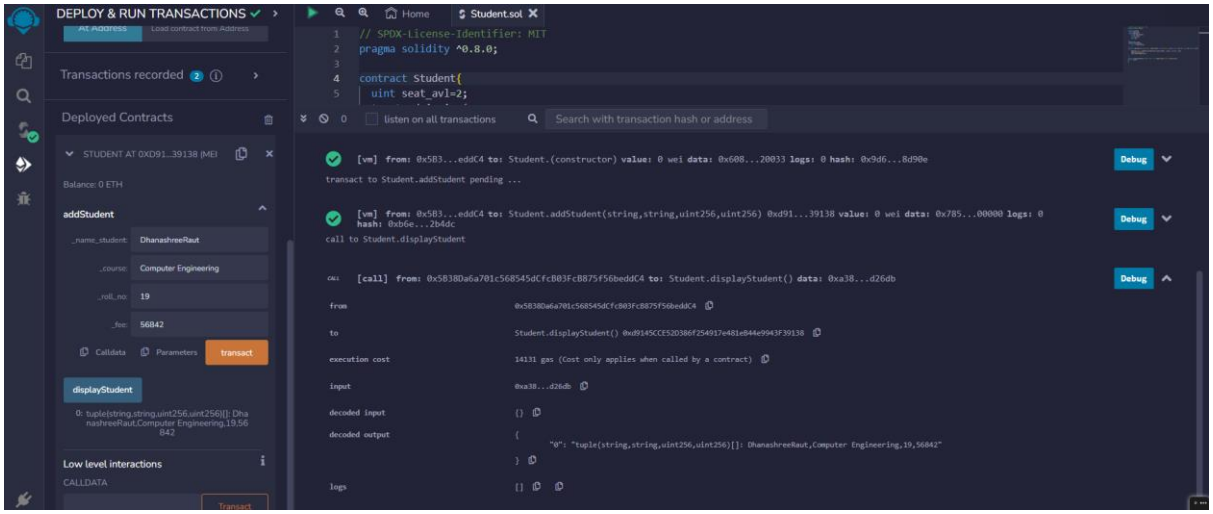
Step 7. After typing the code for the smart contract in the newly creates .sol file,click on the compiler option available and then compile the file

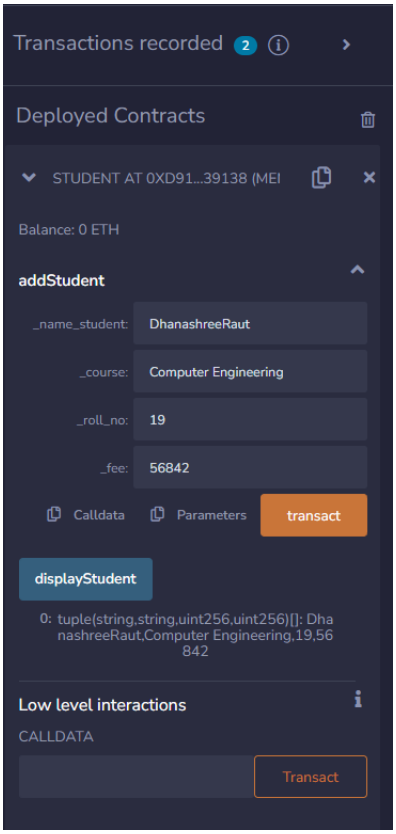
Step 8. If no error, then click on the 'Deploy and Run' to execute the smart contract

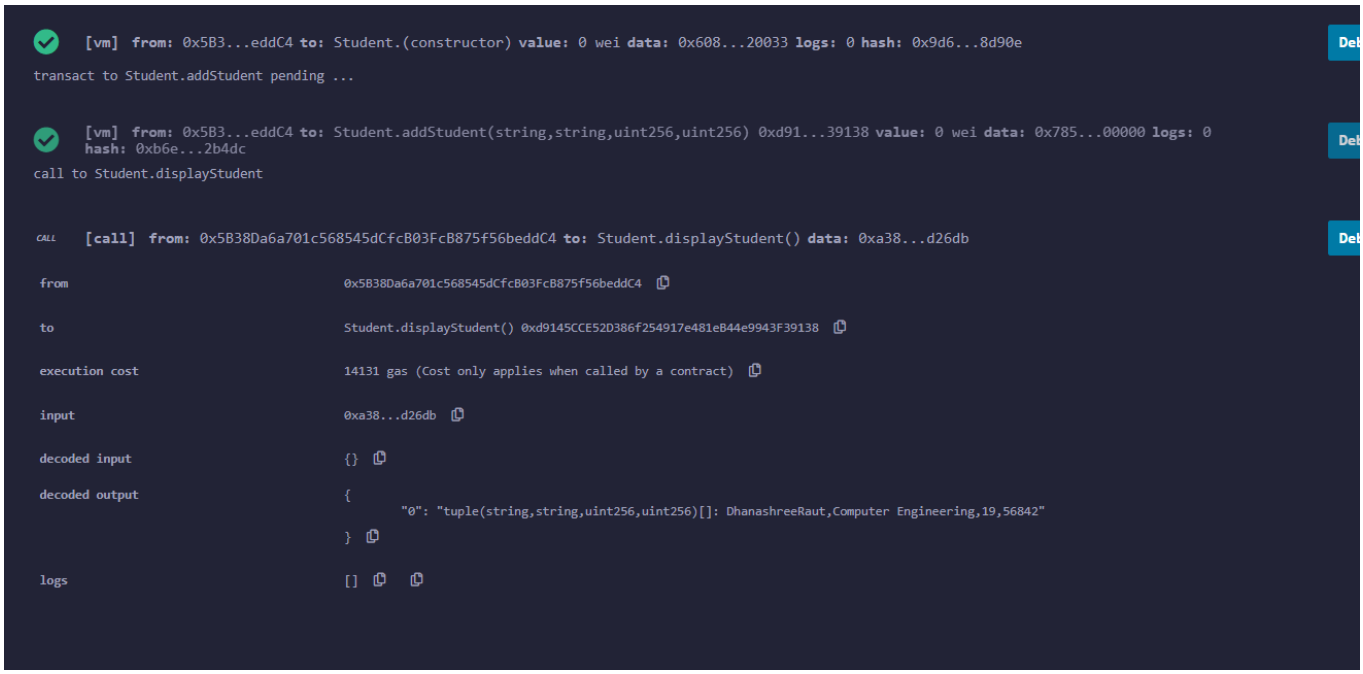
Code: Student.sol

```
pragma solidity ^0.8.0;
contract Student{
  uint seat_avl=2;
 struct admission{
     string name_student;
     string course;
     uint roll_no;
     uint fee;
 }
 admission[] stud1;
 modifier constraint1{
     require (seat_avl>0);
     _;
 }
 function addStudent(string memory _name_student, string memory _course, uint _roll_no, uint _fee) public constraint1
 {
     admission memory newAdmission=admission(_name_student, _course, _roll_no, _fee);
     seat_avl=seat_avl-1;
     stud1.push(newAdmission);
 }
function displayStudent() public view returns(admission[] memory)
{return stud1;
}
}
```

Output:


DEPLOY & RUN TRANSACTIONS
Transactions recorded 2
Deployed Contracts
STUDENT AT 0XD91...39138 (MEI
Balance: 0 ETH
addStudent
_name_student: DhanashreeRaut
_course: Computer Engineering
_roll_no: 19
_fee: 56842
Calldata
Parameters
transact
displayStudent
0: tuple(string,string,uint256,uint256)[]: DhanashreeRaut,Computer Engineering,19,56842
Low level interactions
CALLDATA
Transact
Home
Student.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Student{
uint seat_avl=2;
listen on all transactions
Search with transaction hash or address
[vm] from: 0x5B3...eddC4 to: Student.(constructor) value: 0 wei data: 0x608...20033 logs: 0 hash: 0x9d6...8d90e
transact to Student.addStudent pending ...
[vm] from: 0x5B3...eddC4 to: Student.addStudent(string,string,uint256,uint256) 0xd91...39138 value: 0 wei data: 0x785...00000 logs: 0 hash: 0xb6e...2b4dc
call to Student.displayStudent
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Student.displayStudent() data: 0xa38...d26db
from
to
execution cost
14131 gas (Cost only applies when called by a contract)
input
decoded input
decoded output
logs
Debug


Transactions recorded 2
Deployed Contracts
STUDENT AT 0XD91...39138 (MEI
Balance: 0 ETH
addStudent
_name_student: DhanashreeRaut
_course: Computer Engineering
_roll_no: 19
_fee: 56842
Calldata
Parameters
transact
displayStudent
0: tuple(string,string,uint256,uint256)[]: DhanashreeRaut,Computer Engineering,19,56842
Low level interactions
CALLDATA
Transact



Conclusion:

Solidity is chosen for Ethereum smart contracts due to its Ethereum-specific nature, security features, extensive ecosystem, and widespread adoption. The provided Solidity code defines a "Student" contract that manages student admissions. It tracks available seats, defines a student admission structure, and includes a constraint to ensure seats are available before adding a student. The addStudent function adds students, and displayStudent allows viewing the list of admitted students. Interacting with this contract, you can add students, and the contract will maintain seat availability and store student data. The output will show the list of admitted students when displayStudent is called, reflecting the state of the contract.