Experiment No. 11

Aim To demonstrate contract-to-contract interaction within the Ethereum ecosystem, simulating complex dApp structures (Interacting with Other Contracts).

I. Objectives:

A. Show how contracts can invoke functions of other contracts.
B. Outline best practices for managing contract dependencies.
C. Explore the potential for modular contract development.

II. Steps:

A. Access Remix IDE: Open Remix IDE in your web browser.

B. Create the Contract File

- Click on the "Create New File" icon in the "File Explorers" tab on the left side.
- Name your file CalledContract.sol and paste the provided Access Control contract code into it.

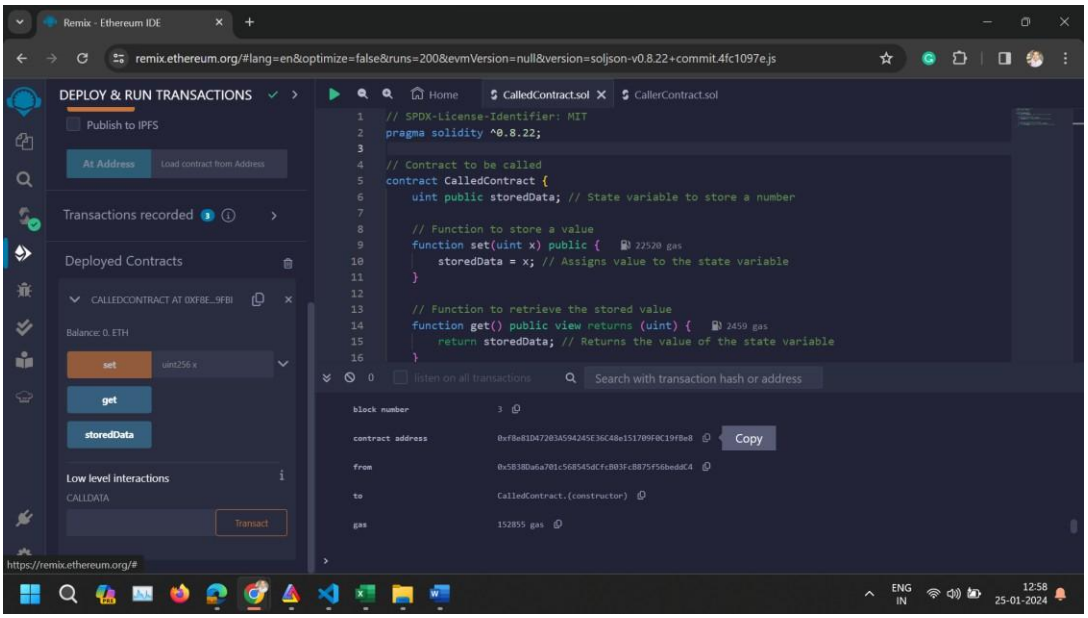
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.22;

// Contract to be called
contract CalledContract {
    uint public storedData; // State variable to store a number

    // Function to store a value
    function set(uint x) public {
        storedData = x; // Assigns value to the state variable
    }

    // Function to retrieve the stored value
    function get() public view returns (uint) {
        return storedData; // Returns the value of the state variable
    }
}
```

C. Compile the Contract

- Click on the "Solidity compiler" tab on the left sidebar.
- Ensure the compiler version is set to match the version specified in your contract (^0.8.22). If necessary, select the correct version manually.
- Click "Compile CalledContract.sol".

D. Deploy the Contract

- Switch to the "Deploy & Run Transactions" tab.
- In the "Environment" dropdown, select "JavaScript VM" for testing purposes.
- Click "Deploy". The contract does not require any constructor parameters, so deployment should be straightforward.
- Copy CallerContract Address.



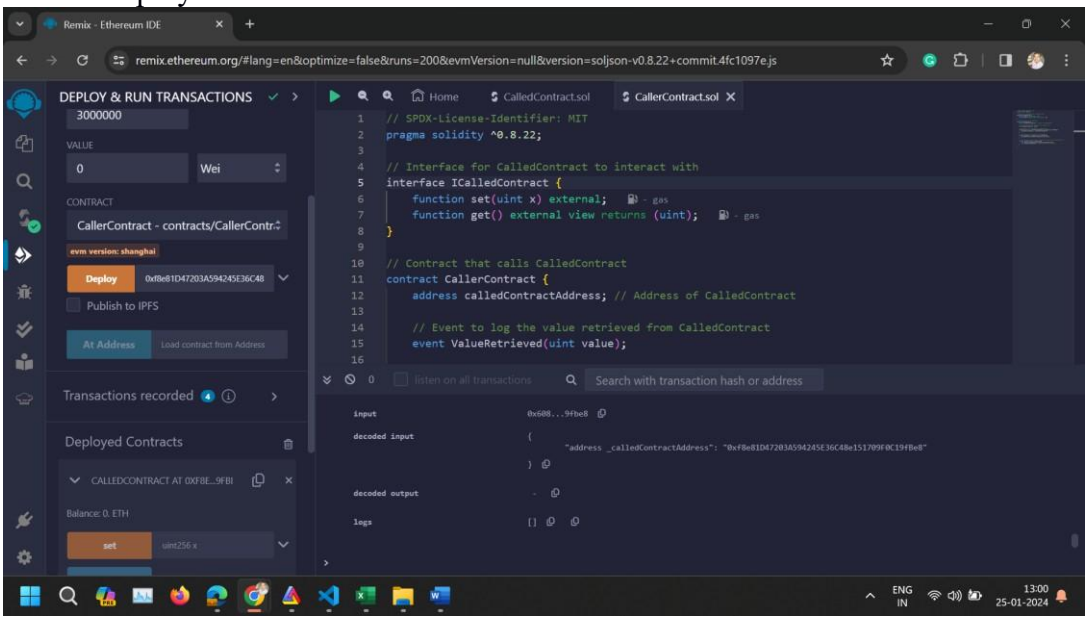
E. Deploy CallerContract with CalledContract's Address

- Create a new file named CallerContract.sol and paste the CallerContract code.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.22;

// Interface for CalledContract to interact with
interface ICalledContract {
    function set(uint x) external;
    function get() external view returns (uint);
}

// Contract that calls CalledContract
contract CallerContract {
    address calledContractAddress; // Address of CalledContract

    // Event to log the value retrieved from CalledContract
    event ValueRetrieved(uint value);

    // Constructor to set the address of CalledContract
    constructor(address _calledContractAddress) {
        calledContractAddress = _calledContractAddress;
    }

    // Function to call set function of CalledContract
    function callSet(uint _value) public {
        ICalledContract(calledContractAddress).set(_value);
    }
```

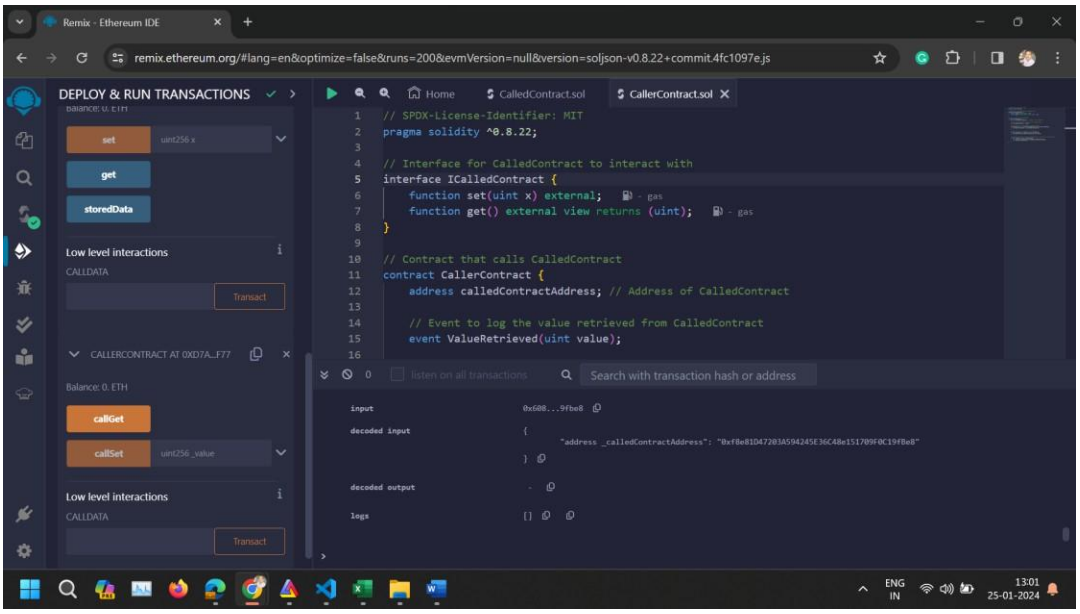
```
    // Function to call get function of CalledContract and emit the retrieved
value
    function callGet() public {
        uint value = ICalledContract(calledContractAddress).get();
        emit ValueRetrieved(value); // Emitting event with the value
    }
}
```

- Compile CallerContract.sol.
- In the "Deploy & Run Transactions" panel, paste the CalledContract's address as the _calledContractAddress parameter for the CallerContract constructor.
- Deploy CallerContract.



F. Interact with the Contracts

- Call callSet: Use CallerContract's callSet function to store a value in CalledContract.
- Call callGet: Then, use CallerContract's callGet function to retrieve and log the stored value.



III. Conclusion: This experiment demonstrates contract-to-contract interaction within the Ethereum blockchain using Remix IDE. This setup illustrates how contracts can leverage functionality of other contracts, a foundational concept for building complex decentralized applications.