

Experiment No.9

Aim: To implement access control mechanisms within a smart contract to restrict function execution to authorized addresses. (Access Control).

I. Objectives:

- A. Introduce modifiers for function access control.
- B. Develop functions that can only be executed by the contract owner.
- C. Learn to transfer contract ownership in a secure manner.

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 AccessControl.sol and paste the provided Access Control contract code into it.

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

// Define a contract named "AccessControl"
contract AccessControl {
    // Declare a state variable to store the address of the contract owner
    address public owner;

    // Event declaration for ownership transfer
    event OwnershipTransferred(address indexed previousOwner,
    address indexed newOwner);

    // New state variable added for demonstration
    uint public restrictedData;

    event RestrictedDataChanged(uint newValue);

    // The constructor sets the deployer as the initial owner
    constructor() {
        owner = msg.sender;
        emit OwnershipTransferred(address(0), msg.sender);
    }

    // Modifier to restrict function access to only the owner
    modifier onlyOwner() {
        require(msg.sender == owner, "Caller is not the owner");
        _; // Continue execution
    }
}
```

```

        // Function to transfer ownership to a new address
        function transferOwnership(address newOwner) public onlyOwner
        {
            require(newOwner != address(0), "New owner is the zero
address");
            emit OwnershipTransferred(owner, newOwner);
            owner = newOwner;
        }

        // Example function that is restricted to the owner
        // This function changes the value of 'restrictedData'
        function restrictedAction(uint _newValue) public onlyOwner {
            restrictedData = _newValue;
            // Emit an event whenever 'restrictedData' is changed
            emit RestrictedDataChanged(_newValue);
        }
    }
}

```

C. Compile the Contract

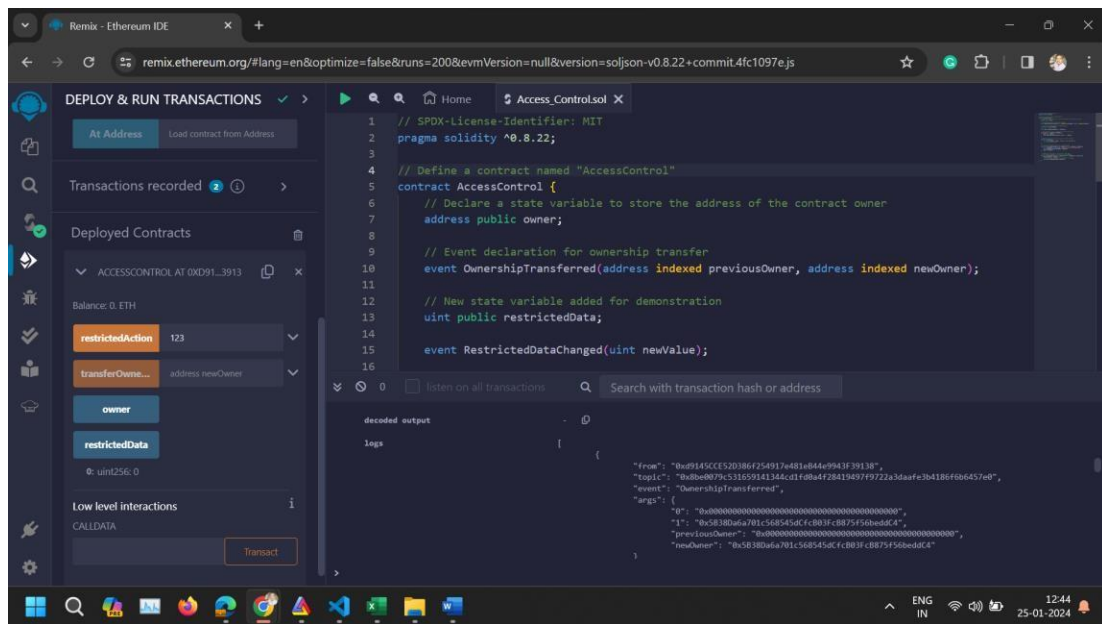
- Click on the "Solidity compiler" tab on the left sidebar.
- Ensure the compiler version is set to match the version specified in your contract (^0.8.22). If necessary, select the correct version manually.
- Click "Compile AccessControl.sol".

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.

E. Interact with the Contract

- Once deployed, you'll see the contract under "Deployed Contracts".
- You can interact with the owner variable to see the current owner.
- Use the transferOwnership function to change the owner. Input a new address from the Remix accounts to simulate transferring ownership.
- Try calling restrictedAction to see how the onlyOwner modifier works in practice. If you've transferred ownership, ensure you're interacting as the current owner.



- III. **Conclusion:** The Access Control contract demonstrates a fundamental security pattern in smart contract development, ensuring that sensitive functions can only be executed by authorized accounts. By deploying and interacting with this contract in Remix, you gain practical experience with modifiers and event emissions, essential tools in the Ethereum developer's toolkit.