Experiment No.8

Aim: To develop a fungible token using the ERC-20 standard, familiarizing with token creation and manipulation on the Ethereum blockchain. (Token Contract).

I. Objectives:

A. Understand the ERC-20 token standard and its key functions. Such as totalSupply, balanceOf, transfer, approve, allowance, and transferFrom, as well as the Transfer and Approval events.

B. Gain experience in interacting with deployed smart contracts through the Remix IDE interface. This includes executing contract functions that change the state of the blockchain (such as transferring tokens) and querying the blockchain to retrieve current state information (such as token balances).

C. Prepare for the integration of the smart contract into decentralized applications (dApps) by adhering to the ERC-20 standard, facilitating the exchange and management of tokens within the Ethereum ecosystem.

II. Steps:

A. Open Remix IDE: Navigate to Remix IDE in your web browser.

B. Create the Contract File

- In the "File Explorers" pane, click the "Create New File" icon.
- Name your file MyToken.sol and paste the provided ERC-20 token contract code into it.

C. Compile the Contract

- Click on the "Solidity Compiler" tab on the left sidebar.
- Ensure the compiler version matches the version in your contract (^0.8.22). If necessary, select the correct version manually.
- Click "Compile MyToken.sol".

```
// Specifies the license under which this contract is released; MIT in this
case
// SPDX-License-Identifier: MIT

// Set the compiler version to be used for this contract
pragma solidity ^0.8.22;

// ERC-20 interface containing the standard functions and events of an
ERC-20 token
interface IERC20 {
   // Returns the total token supply
   function totalSupply() external view returns (uint256);

   // Returns the account balance of another account with address
`account`
   function balanceOf(address account) external view returns (uint256);
```

```
    // Transfers `amount` tokens to address `recipient`, and MUST fire
the `Transfer` event
    function transfer(address recipient, uint256 amount) external returns
(bool);

    // Returns the amount of tokens approved by the owner that can be
transferred to the spender's account
    function allowance(address owner, address spender) external view
returns (uint256);

    // Sets `amount` as the allowance of `spender` over the caller's tokens,
and MUST fire the `Approval` event
    function approve(address spender, uint256 amount) external returns
(bool);

    // Transfers `amount` tokens from address `sender` to address
`recipient`, and MUST fire the `Transfer` event
    function transferFrom(address sender, address recipient, uint256
amount) external returns (bool);

    // Events declaration
    event Transfer(address indexed from, address indexed to, uint256
value);
    event Approval(address indexed owner, address indexed spender,
uint256 value);
}

// MyToken contract implementing the IERC20 interface
contract MyToken is IERC20 {
    // Token metadata
    string public constant name = "MyToken"; // Token name
    string public constant symbol = "MTK"; // Token symbol
    uint8 public constant decimals = 18; // Token decimals

    // State variables
    mapping(address => uint256) balances; // Mapping of token balances
    mapping(address => mapping (address => uint256)) allowed; //
Mapping of token allowances
    uint256 totalSupply_ = 1000 ether; // Total supply of tokens, `ether`
is used for simplicity to denote 18 decimals

    // Constructor to initialize the contract with token supply assigned to
the deployer
```

```
    constructor() {
        balances[msg.sender] = totalSupply_;
    }

    // Returns the total token supply
    function totalSupply() public override view returns (uint256) {
        return totalSupply_;
    }

    // Returns the token balance of `tokenOwner`
    function balanceOf(address tokenOwner) public override view
returns (uint256) {
        return balances[tokenOwner];
    }

    // Transfers `numTokens` from the caller to `receiver`
    function transfer(address receiver, uint256 numTokens) public
override returns (bool) {
        require(numTokens <= balances[msg.sender], "Insufficient
balance");
        balances[msg.sender] -= numTokens;
        balances[receiver] += numTokens;
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    // Approves `delegate` to spend `numTokens` on behalf of the caller
    function approve(address delegate, uint256 numTokens) public
override returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }

    // Returns the number of tokens `delegate` is allowed to spend on
behalf of `owner`
    function allowance(address owner, address delegate) public override
view returns (uint) {
        return allowed[owner][delegate];
    }

    // Transfers `numTokens` from `owner` to `buyer`
```

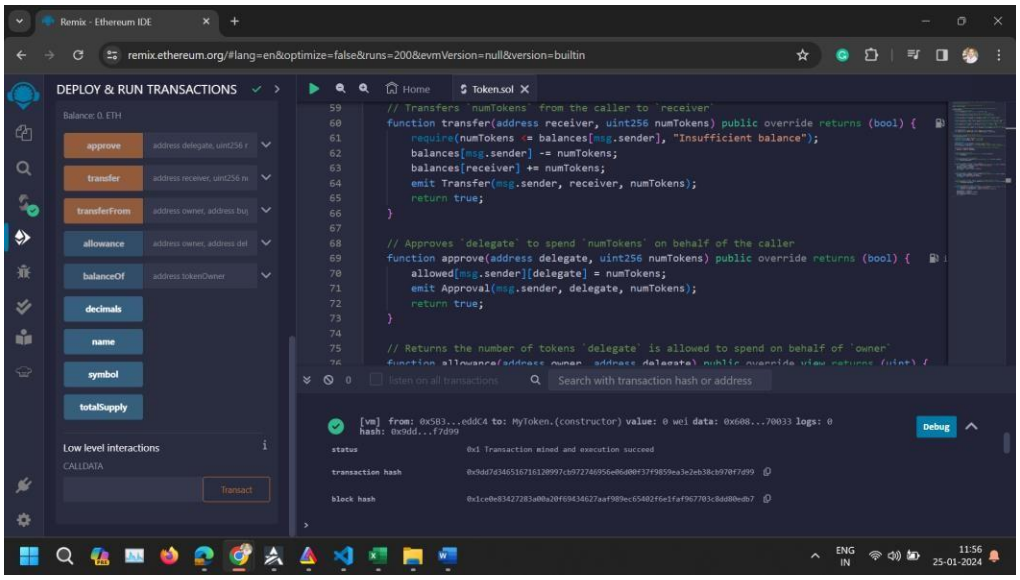
```
        function transferFrom(address owner,   address buyer,   uint256
    numTokens) public override returns (bool) {
            require(numTokens <= balances[owner], "Insufficient balance");
            require(numTokens        <=        allowed[owner][msg.sender],
    "Insufficient allowance");

            balances[owner] -= numTokens;
            allowed[owner][msg.sender] -= numTokens;
            balances[buyer] += numTokens;
            emit Transfer(owner, buyer, numTokens);
            return true;
        }
}
```

D. Deploy the Contract

- Switch to the "Deploy & Run Transactions" tab.
- In the "Environment" dropdown, select "JavaScript VM" for testing purposes.
- Click "Deploy". The initial total supply is set in the contract constructor and will be assigned to the account that deploys the contract.

E. Interact with the Contract

- Once deployed, you'll see the contract under "Deployed Contracts".
- Use the provided functions to interact with your token:
- balanceOf: Enter an account address to get its token balance.
- transfer: Enter a recipient address and amount to transfer tokens.
- approve and transferFrom: Use these to allow and then transfer tokens on behalf of another account.



```
    // Transfers `numTokens` from the caller to `receiver`
    function transfer(address receiver, uint256 numTokens) public override returns (bool) {
        require(numTokens <= balances[msg.sender], "Insufficient balance");
        balances[msg.sender] -= numTokens;
        balances[receiver] += numTokens;
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    // Approves `delegate` to spend `numTokens` on behalf of the caller
    function approve(address delegate, uint256 numTokens) public override returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }

    // Returns the number of tokens `delegate` is allowed to spend on behalf of `owner`
```

```
[vm] from: 0x5B3...eddC4 to: MyToken.(constructor) value: 0 wei data: 0x608...70033 logs: 0
hash: 0x9dd...f7d99
status              0x1 Transaction mined and execution succeed
transaction hash    0x9dd7d346516716120997cb972746956e06d00f37f9859ea3e2eb38cb970f7d99
block hash          0x1ce0e83427283a00a20f69434627aaf989ec65402f6e1faf967703c8dd80edb7
```

III. Conclusion: This ERC-20 token contract demonstrates how to create and interact with a fungible token on the Ethereum blockchain. By following the steps above, you can deploy and test the token contract in Remix, providing a hands-on experience with the ERC-20 standard. This foundational knowledge is crucial for developing more complex decentralized applications and understanding the tokenomics within the Ethereum ecosystem.