DA51 Lab Session 9: Bespoke Ethereum Blockchain Tokens

Description:

This guide outlines the process of creating a custom ERC-20 token on the Ethereum blockchain using the Remix IDE. It covers setting up the development environment, defining the ERC-20 token contract, and implementing essential functions like transfer, approve, and balanceOf.

My contract file :

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

import "Utils.sol";

contract Lab9Token {
    using Utils for *;
    string public constant name = "DA51-Lab9";
    string public constant symbol = "Lab9";
    uint8 public constant decimals = 18;

    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
    event Transfer(address indexed from, address indexed to, uint token);
    mapping(address => uint256) balances;
    mapping(address => mapping(address=> uint256)) allowed;
    uint256 totalSupply_;


    constructor(uint256 initialSupply) {    infinite gas 820600 gas
        totalSupply_ = initialSupply * 10 ** uint256(decimals);
        balances[msg.sender] = totalSupply_;
    }

    function transfer(address to, uint tokens) public returns (bool) {    infinite gas
        require(to != address(0), "Invalid address");
        require(balances[msg.sender] >= tokens, "Insufficient balance");

        balances[msg.sender] -= tokens;
        balances[to] += tokens;
        emit Transfer(msg.sender, to, tokens);
        return true;
    }

    function approve(address spender, uint tokens) public returns (bool) {    infinite gas
        require(spender != address(0), "Invalid spender address");

        allowed[msg.sender][spender] = tokens;
        emit Approval(msg.sender, spender, tokens);
        return true;
    }
```

This first screen shot contains all the variables needed in this contract, the definition of the constructor and some functions like transfer and approve.

The second screen shot contains the last function needed as balanceOf, transferFrom and allowedT (to get the allowance)

```
    function balanceOf(address tokenOwner) public view returns (uint) {    2829 gas
        return balances[tokenOwner];
    }

    function transferFrom(address from, address to, uint tokens) public returns (bool) {
        require(to != address(0), "Invalid to address");
        require(from != address(0), "Invalid from address");
        require(balances[from] >= tokens, "Insufficient balance");
        require(allowed[from][msg.sender] >= tokens, "Allowance exceeded");

        balances[from] -= tokens;
        balances[to] += tokens;
        allowed[from][msg.sender] -= tokens;
        emit Transfer(from, to, tokens);
        return true;
    }

    function allowedT(address tokenOwner, address spender) public view returns (uint256)
    {
        return allowed[tokenOwner][spender];
    }
}
```

My test file:

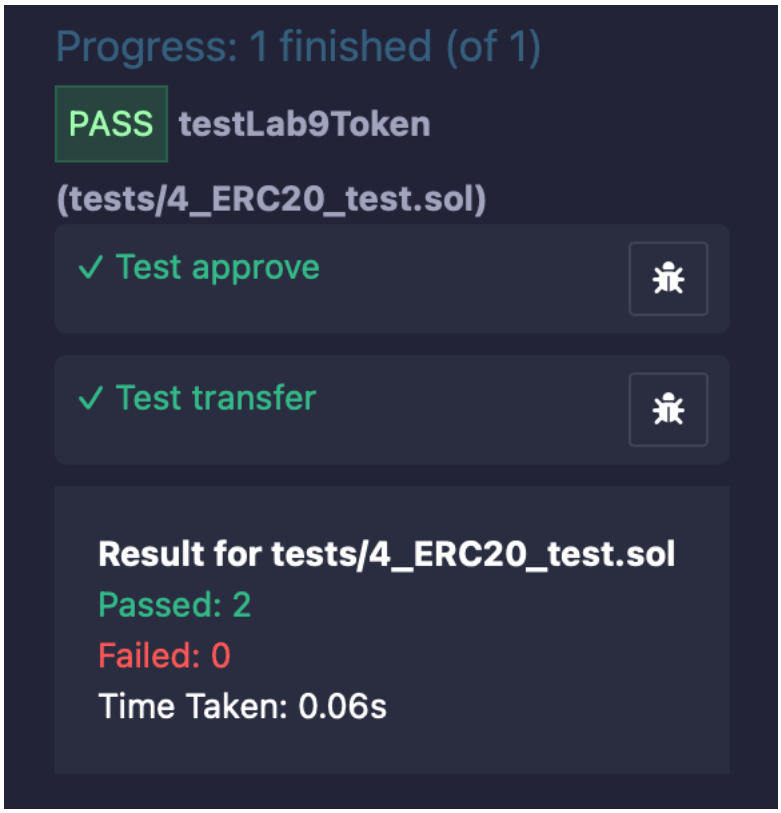
```solidity
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.4.22 <0.9.0;

import "remix_tests.sol";
import "../contracts/4_ERC20.sol";

contract testLab9Token {
    Lab9Token lab9Token;

    address owner;
    address account1 = address(0x123);
    address account2 = address(0x456);


    function beforeAll() public {    infinite gas
        lab9Token = new Lab9Token(1000);
        owner = address(this);
    }

    function testApprove() public {    infinite gas
        uint approveAmount = 1000;
        bool success = lab9Token.approve(account1, approveAmount);
        Assert.equal(success, true, "Approval should succeed");

        uint allowance = lab9Token.allowedT(owner, account1);
        Assert.equal(allowance, approveAmount, "Allowance is not set correctly");
    }
```

This first screen shot contains all the variables needed in this contract and the first test: tertApprove.

The second screen shot show the tow last tests : testTransfer to test if a transfer can be made based on several aspect and testTransferFailsOnInsufficientBalance to test if the we get an error when the balance is to low for a specific transfer.

```solidity
    function testTransfer() public {    infinite gas
        uint initialOwnerBalance = lab9Token.balanceOf(owner);
        uint transferAmount = 500;

        bool success = lab9Token.transfer(account1, transferAmount);
        Assert.equal(success, true, "Transfer should succeed");

        uint newOwnerBalance = lab9Token.balanceOf(owner);
        uint recipientBalance = lab9Token.balanceOf(account1);

        Assert.equal(
            newOwnerBalance,
            initialOwnerBalance - transferAmount,
            "Owner balance incorrect after transfer"
        );
        Assert.equal(
            recipientBalance,
            transferAmount,
            "Recipient balance incorrect after transfer"
        );
    }

    function testTransferFailsOnInsufficientBalance() public {    infinite gas
        uint transferAmount = 10000;

        try lab9Token.transfer(account1, transferAmount) {

        } catch Error(string memory reason) {
            Assert.equal(reason, "Insufficient balance", "Incorrect revert reason");
        }
    }
}
```

The results of the tests are correct as we can see in this screen shot:



Conclusion:

At the end of this Lab session, we are able to create a custom ERC-20 token for our Ethereum-based applications, understanding the requirements and functionalities of the ERC-20 standard.