EAT THE BLOCKS

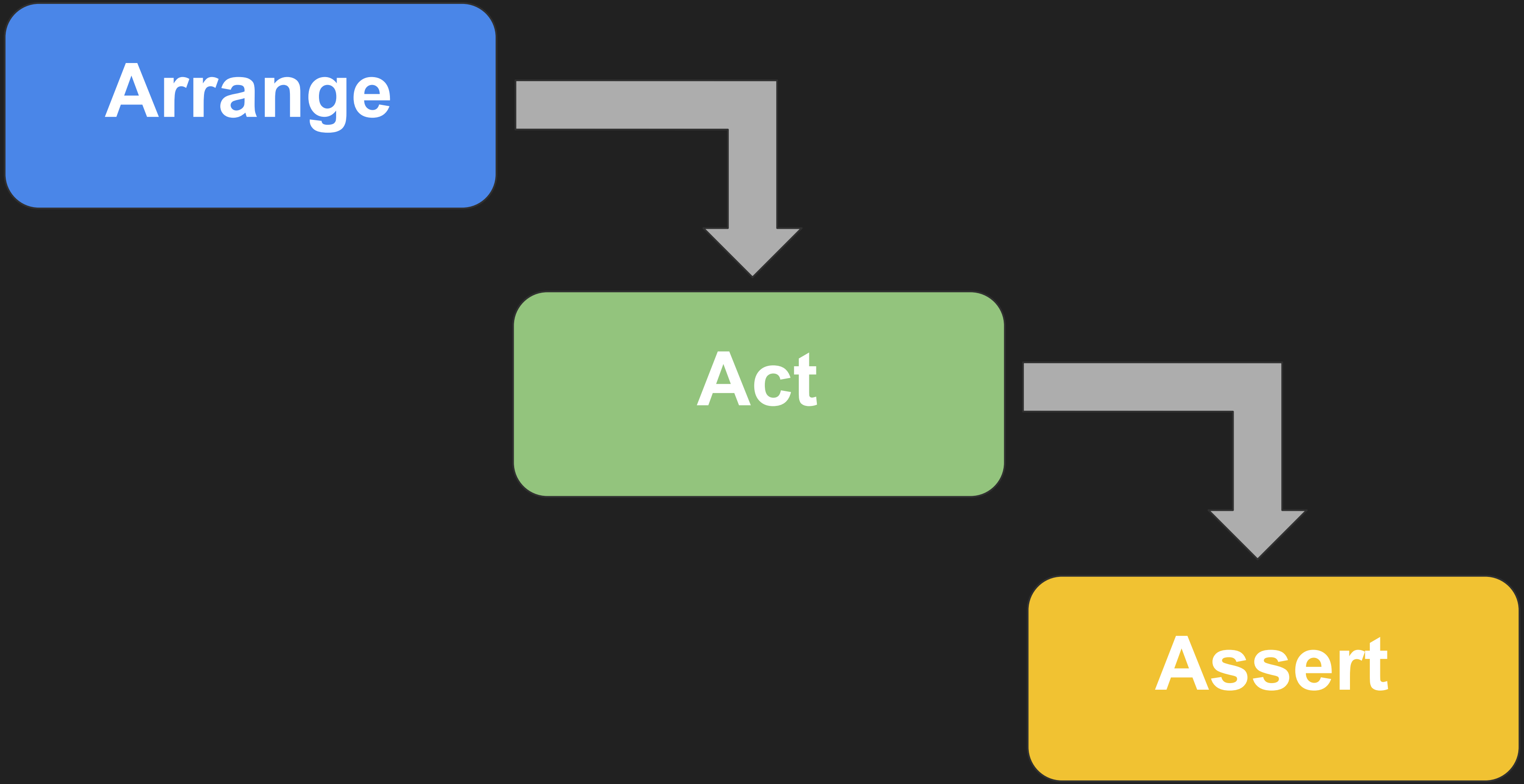
Solidity Mastery Bootcamp

DAY 8

Understand the Importance of Testing

Immutable Nature of Smart Contracts

Financial Risks

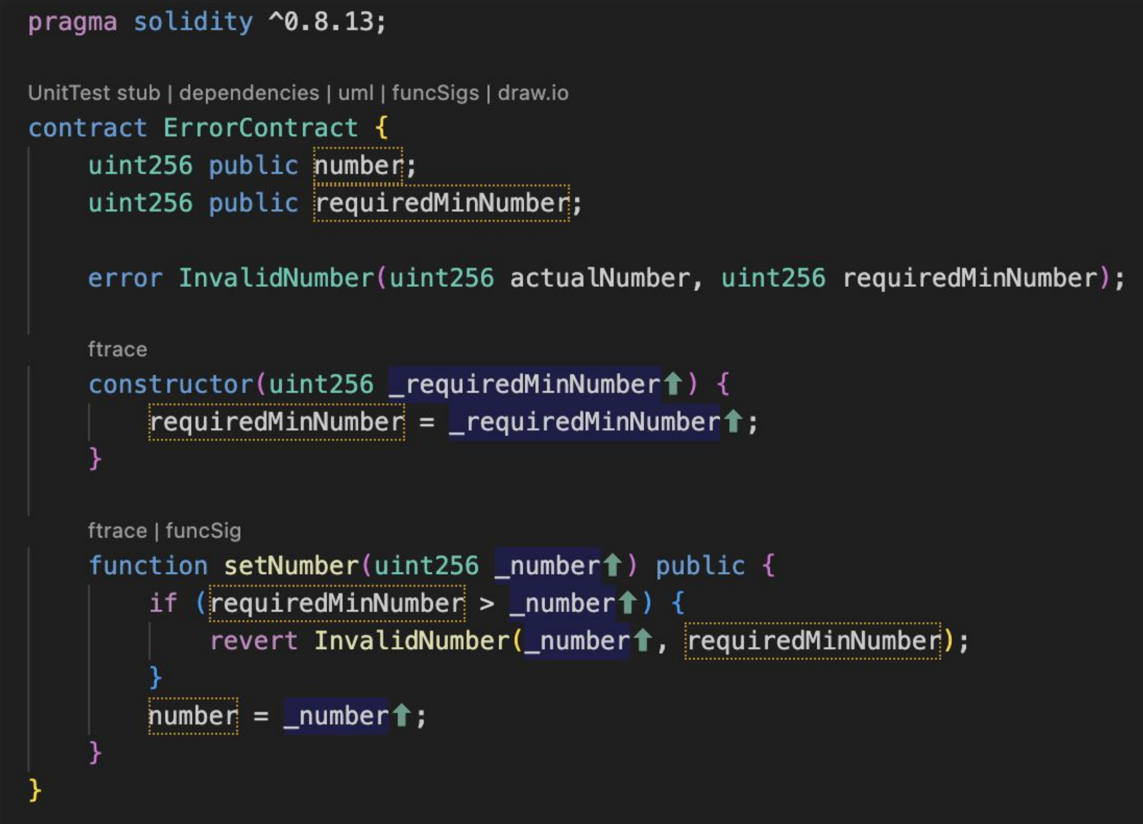

Arrange
Act
Assert

Testing errors

1. Contract example with revert statement (error).
2. Test contract.
3. A function with starting name "test_Revert[If|When]_Condition".
4. An vm.expectRevert(bytes calldata revertData) statement
   a. String error - we need "error text"
   b. Custom error without parameters - we need only the error selector*
   c. Custom error with parameters - we need the error selector* and parameters encoded using abi.encodeWithSelector(selector, param1, param2, ...)
5. Tested contract executement.

*getting selector - MyContract.ErrorName.selector

EXERCISE 1

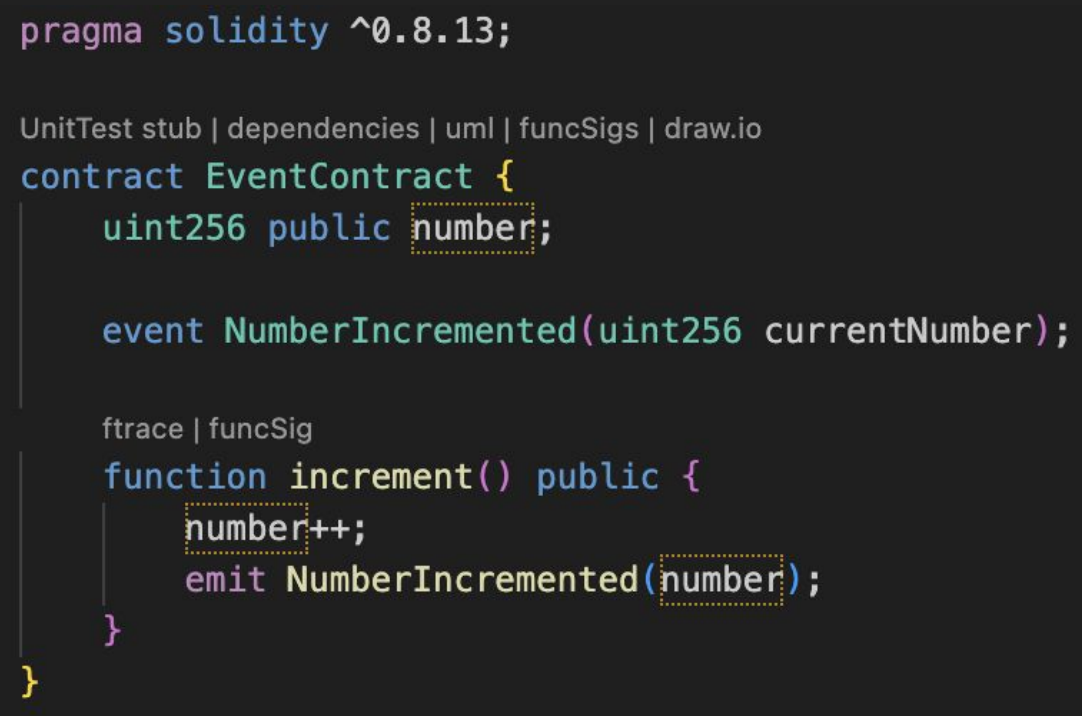


```
pragma solidity ^0.8.13;

UnitTest stub | dependencies | uml | funcSigs | draw.io
contract ErrorContract {
    uint256 public number;
    uint256 public requiredMinNumber;

    error InvalidNumber(uint256 actualNumber, uint256 requiredMinNumber);

    ftrace
    constructor(uint256 _requiredMinNumber) {
        requiredMinNumber = _requiredMinNumber;
    }

    ftrace | funcSig
    function setNumber(uint256 _number) public {
        if (requiredMinNumber > _number) {
            revert InvalidNumber(_number, requiredMinNumber);
        }
        number = _number;
    }
}
```

Testing events

1. Contract example with event.
2. Test contract.
3. Copy-pasted event from tested contract.
4. A function with starting name “test_Function”.
5. An vm.expectEmit(bool checkTopic1, bool checkTopic2, bool checkTopic3, bool checkData) statement, where
   a. CheckTopic1 means checking first indexed parameter
   b. CheckTopic2 means checking second indexed parameter
   c. CheckTopic3 means checking third indexed parameter
   d. CheckData means checking exact values
6. Emitting expecting event with expecting data in test contract.
7. Tested contract executement.

EXERCISE 2

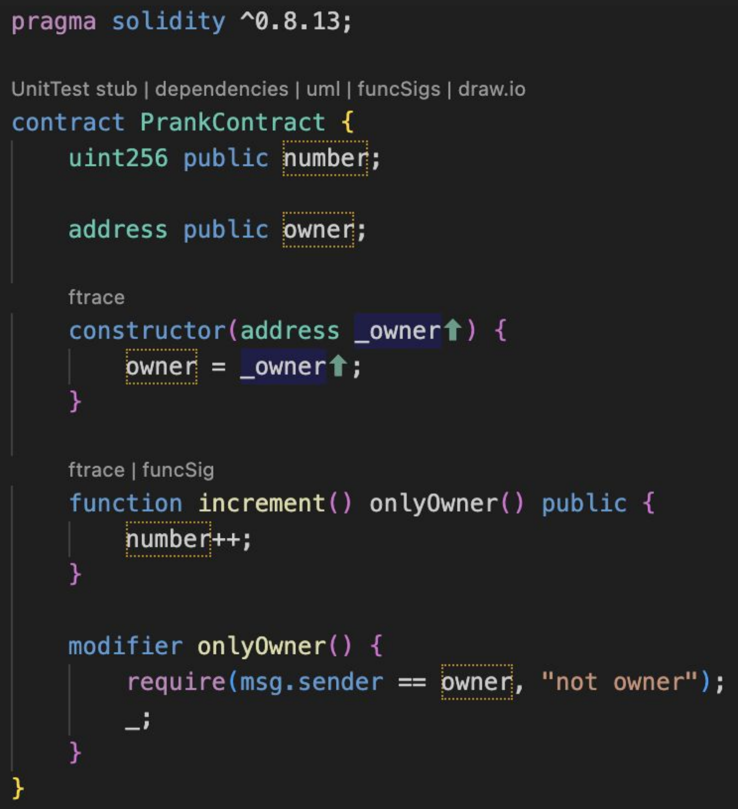


```
pragma solidity ^0.8.13;

UnitTest stub | dependencies | uml | funcSigs | draw.io
contract EventContract {
    uint256 public number;

    event NumberIncremented(uint256 currentNumber);

    ftrace | funcSig
    function increment() public {
        number++;
        emit NumberIncremented(number);
    }
}
```

Testing as another address

1. Contract example with access control.
2. Test contract.
3. A function with starting name "test_Function".
4. An vm.prank(address) statement, where address parameter is address which we want to do a call from.
    a. Alternative is vm.startPrank(address) which will set calling address until vm.stopPrank()
5. Tested contract executement.
6. Assertions.

EXERCISE 3



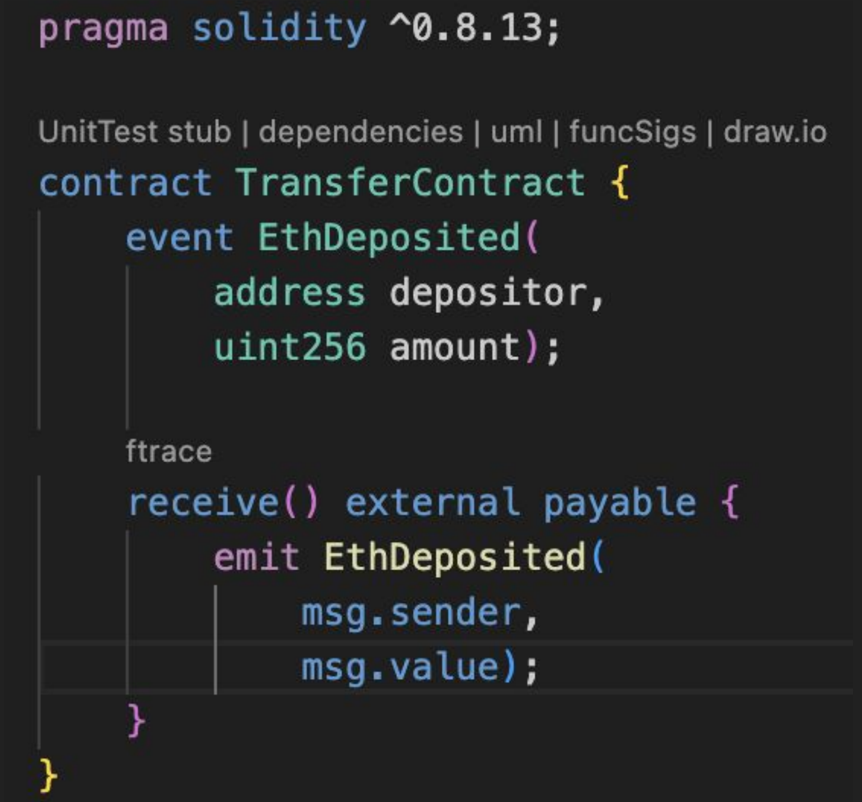
```
pragma solidity ^0.8.13;

UnitTest stub | dependencies | uml | funcSigs | draw.io
contract PrankContract {
    uint256 public number;

    address public owner;

    ftrace
    constructor(address _owner) {
        owner = _owner;
    }

    ftrace | funcSig
    function increment() onlyOwner() public {
        number++;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "not owner");
        _;
    }
}
```

Testing ETH transfers

1. Contract example with payable function.
2. Test contract.
3. A function with starting name "test_Function".
4. Top up account with Ethers using either
   a. deal(address, amount) + vm.prank(address)
   b. hoax(address, amount)

   where address is where we want to give Ethers and amount is amount in Wei.
5. Use .call{ value: amount }("") to transfer Ethers to tested contract.
6. Assertions.

EXERCISE 4

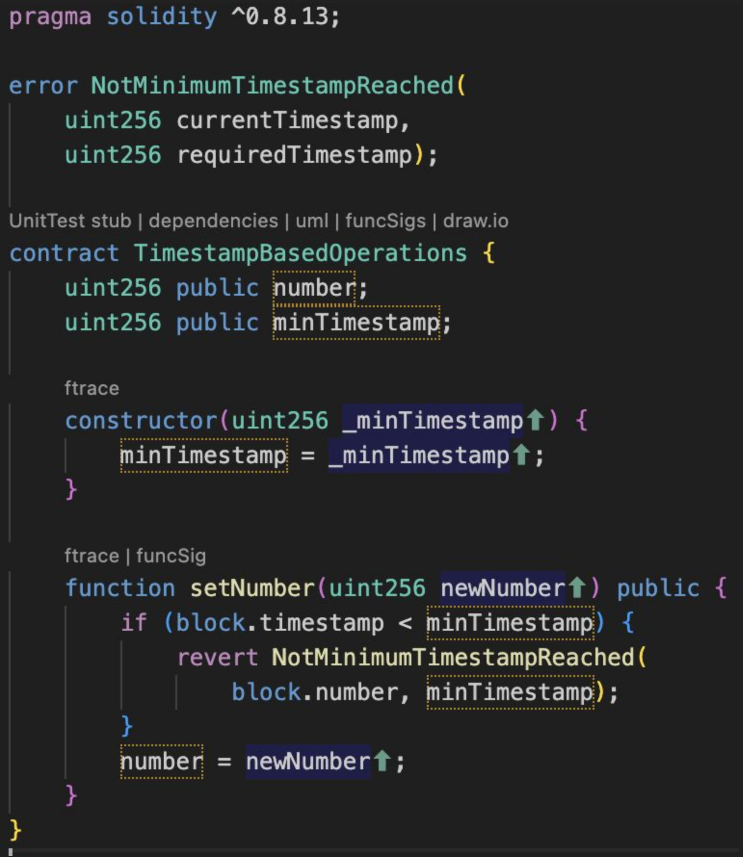


```
pragma solidity ^0.8.13;

UnitTest stub | dependencies | uml | funcSigs | draw.io
contract TransferContract {
    event EthDeposited(
        address depositor,
        uint256 amount);

    ftrace
    receive() external payable {
        emit EthDeposited(
            msg.sender,
            msg.value);
    }
}
```

Testing timestamp

1. Contract example with timestamp based requirement function.
2. Test contract.
3. A function with starting name “test_Function”.
4. An vm.warp(newTimestamp).
5. Tested contract executement.
6. Assertions.

EXERCISE 5

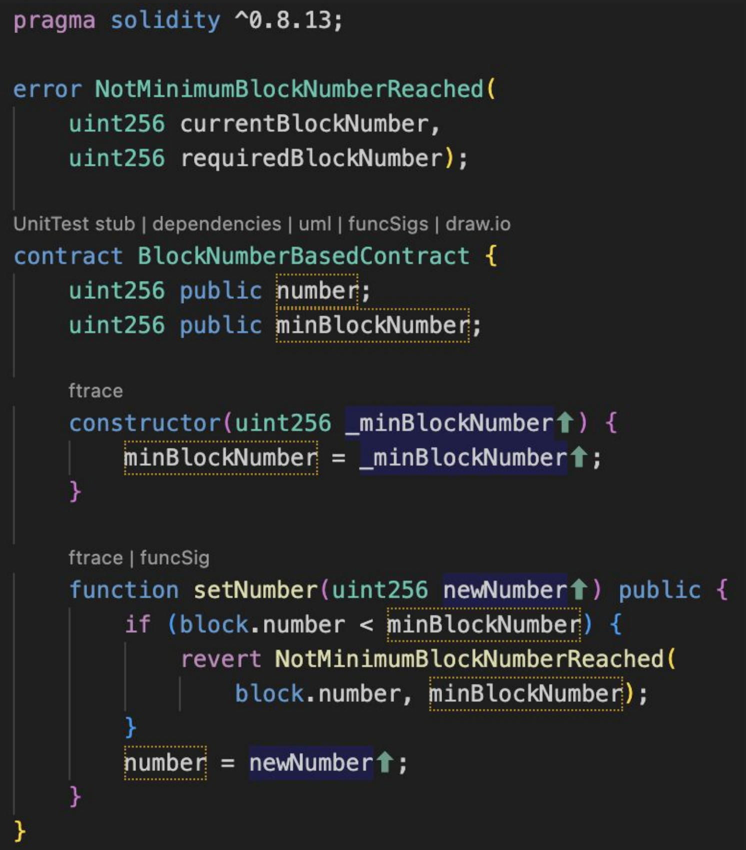


```
pragma solidity ^0.8.13;

error NotMinimumTimestampReached(
    uint256 currentTimestamp,
    uint256 requiredTimestamp);

UnitTest stub | dependencies | uml | funcSigs | draw.io
contract TimestampBasedOperations {
    uint256 public number;
    uint256 public minTimestamp;

    ftrace
    constructor(uint256 _minTimestamp) {
        minTimestamp = _minTimestamp;
    }

    ftrace | funcSig
    function setNumber(uint256 newNumber) public {
        if (block.timestamp < minTimestamp) {
            revert NotMinimumTimestampReached(
                block.number, minTimestamp);
        }
        number = newNumber;
    }
}
```

Testing block number

1. Contract example with block number based requirement function.
2. Test contract.
3. A function with starting name “test_Function”.
4. An vm.roll(newBlockNumber).
5. Tested contract executement.
6. Assertions.

EXERCISE 6



```solidity
pragma solidity ^0.8.13;

error NotMinimumBlockNumberReached(
    uint256 currentBlockNumber,
    uint256 requiredBlockNumber);

UnitTest stub | dependencies | uml | funcSigs | draw.io
contract BlockNumberBasedContract {
    uint256 public number;
    uint256 public minBlockNumber;

    ftrace
    constructor(uint256 _minBlockNumber) {
        minBlockNumber = _minBlockNumber;
    }

    ftrace | funcSig
    function setNumber(uint256 newNumber) public {
        if (block.number < minBlockNumber) {
            revert NotMinimumBlockNumberReached(
                block.number, minBlockNumber);
        }
        number = newNumber;
    }
}
```

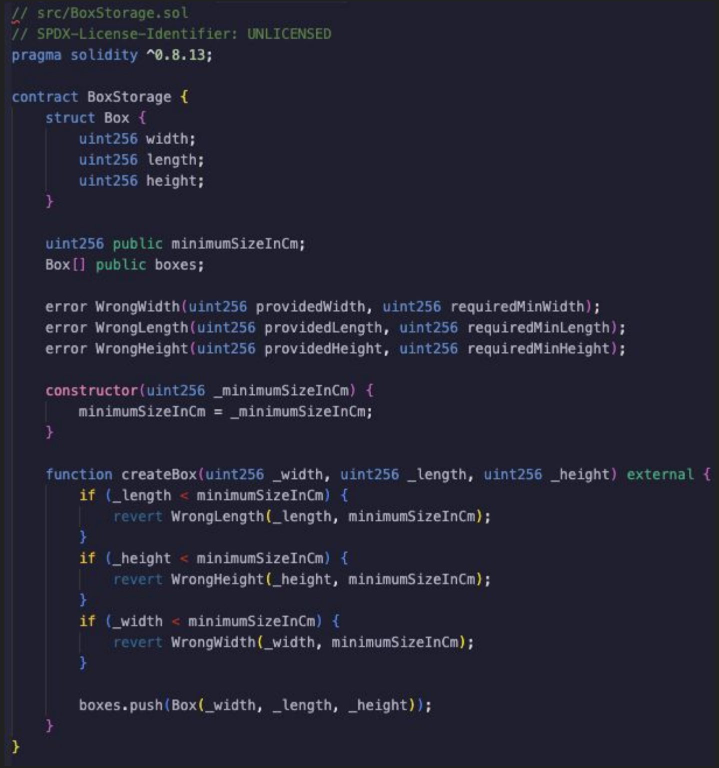
Homework

Quiz

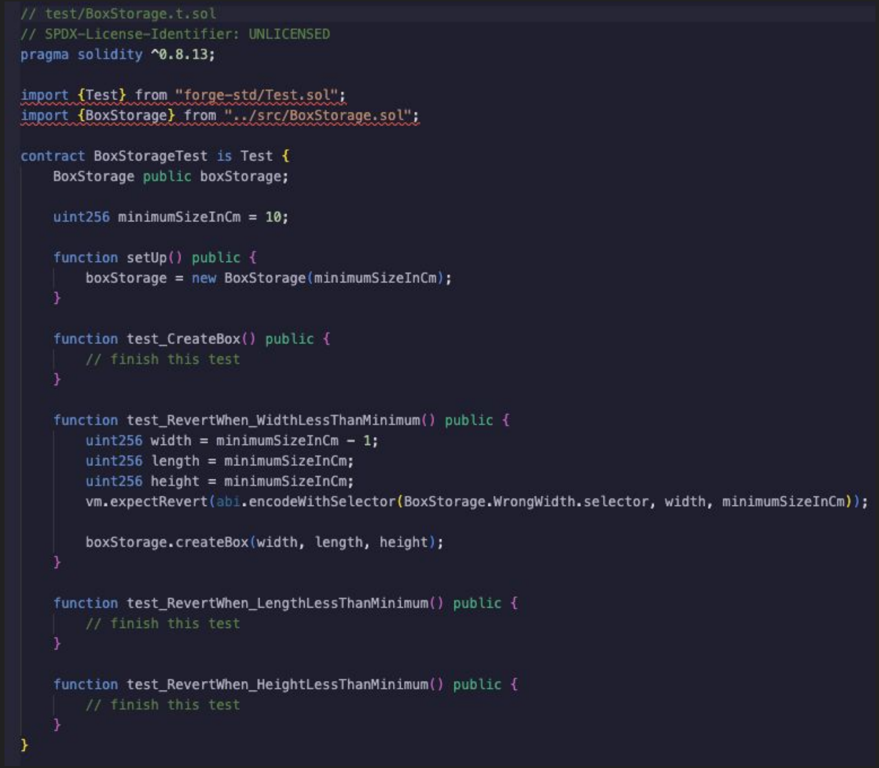
- Answer the quiz below

Homework 1:

Finish missing tests.



```solidity
// src/BoxStorage.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract BoxStorage {
    struct Box {
        uint256 width;
        uint256 length;
        uint256 height;
    }

    uint256 public minimumSizeInCm;
    Box[] public boxes;

    error WrongWidth(uint256 providedWidth, uint256 requiredMinWidth);
    error WrongLength(uint256 providedLength, uint256 requiredMinLength);
    error WrongHeight(uint256 providedHeight, uint256 requiredMinHeight);

    constructor(uint256 _minimumSizeInCm) {
        minimumSizeInCm = _minimumSizeInCm;
    }

    function createBox(uint256 _width, uint256 _length, uint256 _height) external {
        if (_length < minimumSizeInCm) {
            revert WrongLength(_length, minimumSizeInCm);
        }
        if (_height < minimumSizeInCm) {
            revert WrongHeight(_height, minimumSizeInCm);
        }
        if (_width < minimumSizeInCm) {
            revert WrongWidth(_width, minimumSizeInCm);
        }

        boxes.push(Box(_width, _length, _height));
    }
}
```



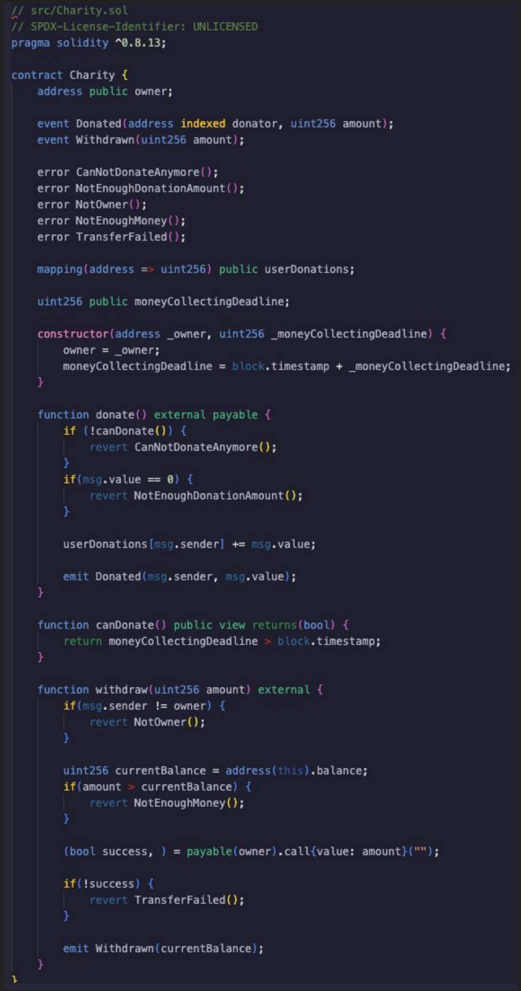
```solidity
// test/BoxStorage.t.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test} from "forge-std/Test.sol";
import {BoxStorage} from "../src/BoxStorage.sol";

contract BoxStorageTest is Test {
    BoxStorage public boxStorage;

    uint256 minimumSizeInCm = 10;

    function setUp() public {
        boxStorage = new BoxStorage(minimumSizeInCm);
    }

    function test_CreateBox() public {
        // finish this test
    }

    function test_RevertWhen_WidthLessThanMinimum() public {
        uint256 width = minimumSizeInCm - 1;
        uint256 length = minimumSizeInCm;
        uint256 height = minimumSizeInCm;
        vm.expectRevert(abi.encodeWithSelector(BoxStorage.WrongWidth.selector, width, minimumSizeInCm));

        boxStorage.createBox(width, length, height);
    }

    function test_RevertWhen_LengthLessThanMinimum() public {
        // finish this test
    }

    function test_RevertWhen_HeightLessThanMinimum() public {
        // finish this test
    }
}
```

Homework 2:

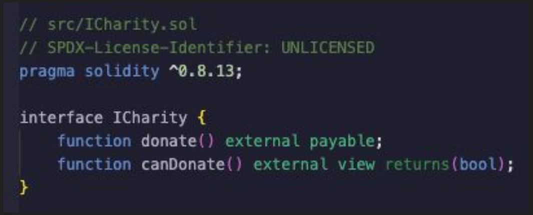
Write tests for `Charity` and `Wallet` contracts.

Use separate files for each contract.

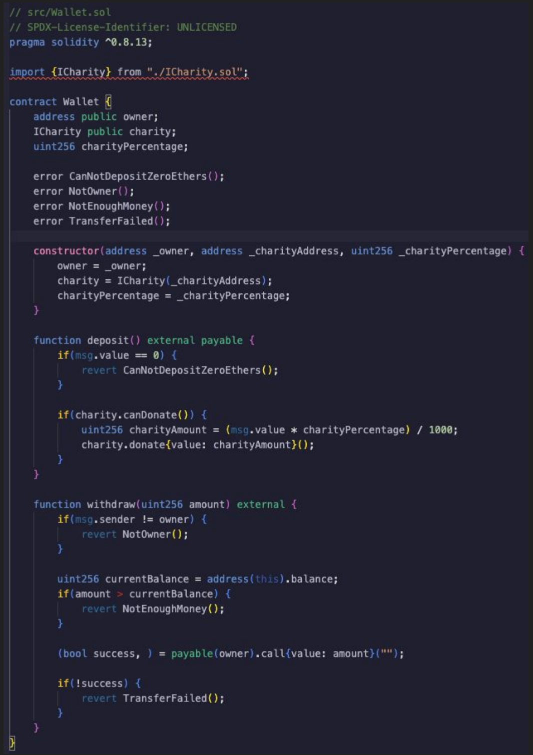
Try to reach 100% coverage (use `forge coverage` to check)



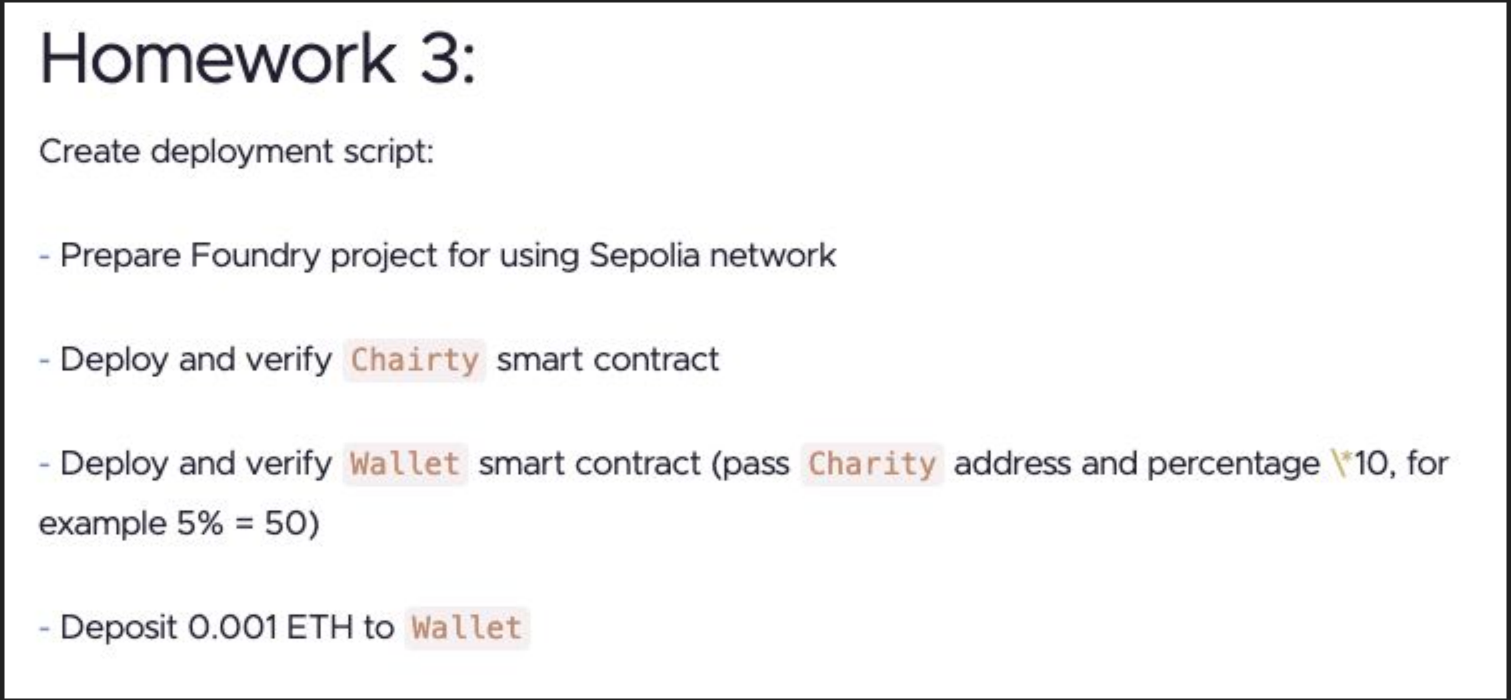
```solidity
// src/Charity.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Charity {
    address public owner;

    event Donated(address indexed donator, uint256 amount);
    event Withdrawn(uint256 amount);

    error CanNotDonateAnymore();
    error NotEnoughDonationAmount();
    error NotOwner();
    error NotEnoughMoney();
    error TransferFailed();

    mapping(address => uint256) public userDonations;

    uint256 public moneyCollectingDeadline;

    constructor(address _owner, uint256 _moneyCollectingDeadline) {
        owner = _owner;
        moneyCollectingDeadline = block.timestamp + _moneyCollectingDeadline;
    }

    function donate() external payable {
        if (!canDonate()) {
            revert CanNotDonateAnymore();
        }
        if(msg.value == 0) {
            revert NotEnoughDonationAmount();
        }

        userDonations[msg.sender] += msg.value;

        emit Donated(msg.sender, msg.value);
    }

    function canDonate() public view returns(bool) {
        return moneyCollectingDeadline > block.timestamp;
    }

    function withdraw(uint256 amount) external {
        if(msg.sender != owner) {
            revert NotOwner();
        }

        uint256 currentBalance = address(this).balance;
        if(amount > currentBalance) {
            revert NotEnoughMoney();
        }

        (bool success, ) = payable(owner).call{value: amount}("");

        if(!success) {
            revert TransferFailed();
        }

        emit Withdrawn(currentBalance);
    }
}
```



```solidity
// src/ICharity.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

interface ICharity {
    function donate() external payable;
    function canDonate() external view returns(bool);
}
```



```solidity
// src/Wallet.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {ICharity} from "./ICharity.sol";

contract Wallet {
    address public owner;
    ICharity public charity;
    uint256 charityPercentage;

    error CanNotDepositZeroEthers();
    error NotOwner();
    error NotEnoughMoney();
    error TransferFailed();

    constructor(address _owner, address _charityAddress, uint256 _charityPercentage) {
        owner = _owner;
        charity = ICharity(_charityAddress);
        charityPercentage = _charityPercentage;
    }

    function deposit() external payable {
        if(msg.value == 0) {
            revert CanNotDepositZeroEthers();
        }

        if(charity.canDonate()) {
            uint256 charityAmount = (msg.value * charityPercentage) / 1000;
            charity.donate{value: charityAmount}();
        }
    }

    function withdraw(uint256 amount) external {
        if(msg.sender != owner) {
            revert NotOwner();
        }

        uint256 currentBalance = address(this).balance;
        if(amount > currentBalance) {
            revert NotEnoughMoney();
        }

        (bool success, ) = payable(owner).call{value: amount}("");

        if(!success) {
            revert TransferFailed();
        }
    }
}
```



Homework 3:

Create deployment script:

- Prepare Foundry project for using Sepolia network
- Deploy and verify `Chairty` smart contract
- Deploy and verify `Wallet` smart contract (pass `Charity` address and percentage *10, for example 5% = 50)
- Deposit 0.001 ETH to `Wallet`