BUỔI 3: SOLIDITY (2)

1. constant và immutable

constant: không thể sửa đổi, được gán cứng.

immutable: có thể được thiết lập trong hàm khởi tạo nhưng sau đó không thể sửa đổi.

2. Biến trạng thái

- local: khai báo trong hàm, không lưu trong blockchain
- state: khai báo ngoài hàm, lưu trong blockchain
- global: cung cấp thông tin về blockchain

Ghi vào một biến trạng thái thì cần gửi đi một giao dịch.

Đọc biến trạng thái thì không cần phí giao dịch.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.2 <0.9.0;

contract Variables {
    // Biến lưu trong blockchain
    string public text = "Hello";
    address public constant MY_ADDRESS =
0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
    uint public immutable MY_VALUE;

    constructor(uint _my_value) {
        MY_VALUE = _my_value;
    }
    function sayHello() public view returns(uint a, uint t, address ad) {
        uint x = 5;
        uint timestamp = block.timestamp;
        address sender = msg.sender;
        a = x;
        t = timestamp;
        ad = sender;
    }
}
```

3. Mapping

Cú pháp: mapping(keytype => valuetype)

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.2 <0.9.0;

contract MappingExample {
```

```
    mapping (address => uint) public balance;

    function get(address _addr) public view returns(uint) {
        return balance[_addr];
    }

    function set(address _addr, uint _val) public {
        balance[_addr] = _val;
    }
    //Reset về giá trị mặc định
    function remove(address _addr) public {
        delete balance[_addr];
    }
}
```

4. Array

Mảng có thể có kích thước cố định hoặc động

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.2 <0.9.0;

contract Array {
    //Khởi tạo một mảng
    uint16[] public arr1;
    uint16[] public arr2 = [1, 2, 3];
    uint16[] public arr3;

    function getArray(uint16 i) public view returns (uint16) {
        return arr1[i];
    }

    // Thêm một phần tử vào array
    function push(uint16 x) public {
        arr1.push(x);
    }
    //Lấy phần tử cuối ra khỏi array
    function pop() public {
        arr1.pop();
    }

    function getLength() public view returns(uint) {
        return arr1.length;
    }

    function remove(uint i) public {
        delete  arr1[i];
    }
}
```

Bài tập 1

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.2 <0.9.0;

contract Array {
    uint[] public arr;

    function remove(uint idx) public {
        //Viết hàm xóa một phần tử tại vị trí idx theo phương pháp ghi đè
        //Ví dụ mảng [1,2,3,4,5], idx = 1
        //Xóa thực hiện: [1, 3, 4, 5, 5] -> [1, 3, 4, 5]
    }

    function removeByReplace(uint idx) public{
        //Viết hàm xóa một phần tử tại vị trí idx bằng cách thay phần tử cuối
mảng vào vị trí muốn xóa
        //Và xóa phần tử cuối mảng
    }
}
```

5. msg

msg.sender: trả về address người đang call hàm của smart contract

address(this): trả về address người deploy smart contract

6. modifier

Có thể được chạy trước hoặc sau khi gọi 1 hàm.

Được sử dụng trong trường hợp cần hạn chế truy cập, xác thực đầu vào, chống xâm nhập lại

```
// SPDX-License-Identifier: MIT
pragma solidity > 0.8.2 < 0.9.1;
contract FunctionModifier {
    address public owner_add;
    uint public x = 10;
    bool public locked;

    constructor() {
        owner_add = msg.sender;
    }
    modifier checkOwner() {
        require(msg.sender == owner_add, "Khong phai owner");
        _;
    }

    modifier checkAddress(address _addr) {
        require(_addr != address(0), "Dia chi khong hop le");
```

```
        _;
    }
    function changeOwner(address _addr) public checkOwner checkAddress(_addr)
{
        owner_add = _addr;
    }

    modifier unlock() {
        require(!locked, "Khong truy cap lai duoc");
        locked = true;
        _;
        locked = false;
    }
    function increase(uint i) public unlock {
        x -= i;
    }
}
```

7. Vị trí dữ liệu

Các biến được khai báo là storage, memory hoặc calldata để chỉ định vị trí lưu trữ dữ liệu.

- storage: biến state (lưu trong blockchain)

- memory: biến lưu trong bộ nhớ, tồn tại khi hàm được gọi

- calldata: vị trí dữ liệu đặc chứa các đối số hàm

8. View và pure

- view: không thay đổi giá trị của biến state

- pure: không thay đổi hoặc đọc biến state

9. payable

Hàm hay address được khai báo payable có thể nhận ether trong contract

10. enum và struct

Enum: kiểu liệt kê, có thể khai báo ngoài contract

Struct: cho phép người dùng tự định nghĩa kiểu

```
// SPDX-License-Identifier: MIT
pragma solidity > 0.8.2 < 0.9.1;
contract StructExample {
    struct Task {
        string text;
        bool completed;
    }

    Task[] public  taskList;
```

```
    function createTask(string calldata _text) public {
        //Các cách khởi tạo
        //Gọi như một hàm
        taskList.push(Task(_text, false));
        //Sử dụng mapping
        taskList.push(Task({text: _text, completed: false}));
        //Khởi tạo một struct rỗng và cập nhật giá trị
        Task memory tsk;
        tsk.text = _text;
        tsk.completed = false;
        taskList.push(tsk);
    }

    function getTask(uint idx) public view returns(string memory text, bool
completed) {
        Task storage tsk = taskList[idx];
        return (tsk.text, tsk.completed);
    }

    function updateTask(uint idx, string calldata _text) public {
        Task storage tsk = taskList[idx];
        tsk.text = _text;
    }

    function toogleCompleted(uint idx) public {
        Task storage tsk = taskList[idx];
        tsk.completed = true;
    }
}
```

```
// SPDX-License-Identifier: MIT
pragma solidity > 0.8.2 < 0.9.1;

contract EnumExample{
    enum  Status {
        Pending, //0
        Rejected, //1
        InProcess, //
        Ready,
        Shipped
    }

    Status public status;

    function get() public view returns (Status) {
        return status;
    }

    function set(Status _stat) public {
        status = _stat;
```

```
    }

    function cancel() public {
        status = Status.Rejected;
    }

    function reset() public {
        delete status;
    }

}
```

Bài tập 2

Viết 1 contract trong đó:

- 1 biến state array kiểu uint

- các hàm như thêm phần tử, xóa phần tử, trả về giá trị phần tử tại 1 vị trí

- hàm kiểm tra phần tử có được sắp xếp theo thứ tự tăng/giảm dần?

- hàm trả về giá trị trung bình của mảng

Bài tập 3

Xây dựng contract TicTacToe

- Kiểm tra có người nào thắng hay không với một bàn cờ cho trước

- Các người chơi là 1 và 0

Bài tập 4

Xây dựng contract trong đó có 1 enum là các ngày trong tuần (tuần bắt đầu là ngày thứ 2)

Viết một hàm kiểm tra xem với 1 chỉ số cho trước thì ngày có phải ngày cuối tuần không.

Ví dụ: isWeekend(0) trả về false, isWeekend(6) trả về true

Bài tập 5

Viết một contract trong đó có một kiểu struct Student gồm tên và tuổi, danh sách các sinh viên; các hàm bao gồm thêm sinh viên, tìm kiếm sinh viên theo tên, xóa sinh viên theo index, xóa sinh viên theo tên

Bài tập 6

Xây dựng một contract mô phỏng hoạt động của một ngân hàng đơn giản.

Các biến state của contract bao gồm: balances (kiểu mapping address của người gọi transaction và tiền đang có)

Các hàm bao gồm: gửi tiền, rút tiền, chuyển tiền.