Mr. Abdelhamid DEKHINET
University BATNA 2
Department of Computer Science
Master : Distributed Information Systems Engineering and Security

Exercises and Labs

Exo-Lab-1 : Compile and Deploy Smart Contract

I- Write a simple Solidity smart contract composed of three functions : get, increment and decrement a variable state.
Using Remix –Ethereum IDE and connection to the Ganach simulator:

a. Edit, Compile, Deploy and Run the contract
b. What is the address (Account) of the contact creator (Owner) ?
c. What is the contract address ?
d. Display the assembly code of the contract

```
pragma solidity ^0.8.17;
contract TestModifier {
    uint number;
    address public owner;
    constructor() {
        owner = msg.sender;
    }
    modifier isOwner() {
        require(msg.sender == owner, "Not the owner");
        _;
    }
    function setNumber(uint _number) public {
        number = _number;
    }
    function getNumber() public view returns(uint) {
        return number;
    }
    function getSender() public view returns(address) {
        return msg.sender;
    }
}
```

II- Modifiers in Solidity are special functions that modify the behavior of other functions. They allow developers to add extra conditions or functionality without having to rewrite the entire function. They can be used to restrict access, validate inputs, …
Consider the following solidity smart contract :

1- Describe the principle of the Modifiers based on the TestModifier contract.
2- Using Remix –Ethereum IDE and connection to the Ganach simulator:

a. Edit, Compile, Deploy and Run the contract
b. What is the account of the owner ?
c. Add the modifier isOwner to the function getNumber
   - Get the number with the account owner
   - Get the number with another account
     Has the transaction been reverted ?
     What is the effect of the modifier ?
d. Define a function to display the address of the user who interacts with the contract

Exo-Lab-2 : Simple Banking System

The purpose of the exercise and the lab is to create a basic decentralized banking system on the Ethereum blockchain. Initially and to simplify the deployment, we only consider the basic functionalities of the system, like:

- Deposit Amount
- Withdraw Amount
- Get Balance

1- Write a contract that provides these functions.
2- Modify the code so that only the account owner can withdraw an amount
3- The functionalities deployed are insufficient for the implementation of such a system. So, try to complete the system by defining other functionalities, such as :

- Account Creation
- Transfer Amount
- Grant Access to Account
- Revoke Access to Account
- …

```solidity
pragma solidity ^0.8.17;

contract SimpleBank {
    mapping (address => uint) balances;
    address public owner;

    event LogOperation(address from, uint amount);

    modifier AmountGrZero() {
        require(msg.value > 0, "Amount to deposit must be greater than Zero");
        _;
    }

    function deposit() public payable AmountGrZero {
        balances[msg.sender] += msg.value;
        emit LogOperation(msg.sender, msg.value);
    }

    function withdraw(uint Amount) public payable {
        // Check enough balance available, otherwise just return balance
        require(balances[msg.sender] > Amount, "insufficient balance");
        balances[msg.sender]-=Amount;
        emit LogOperation(msg.sender, Amount);
        payable(msg.sender).transfer(Amount);
    }

    function getBalance() public view returns(uint){
      return balances[msg.sender];
    }

}
```

Exo-Lab-3 : Simple Smart Contract

1- Increment and decrement operations

```
pragma solidity ^0.8.17;
contract Counter {
    uint public count;
    // Function to get the current count
    function get() public view returns (uint) {
        return count;
    }
    // Function to increment count by 1
    function inc() public {
        count += 1;
    }
    // Function to decrement count by 1
    function dec() public {
        // This function will fail if count = 0
        count -= 1;
    }
}
```

2- Sending Ether

```
pragma solidity ^0.8.17;
contract SendEther {

    function deposit() external payable {
        // Payable keyboard allows receiving Ether
        // Hide Code : address(this).balance += msg.value
    }
    function getBalance() external view returns(uint) {
        return address(this).balance;
    }
}
```

3- Spending Gas

```
pragma solidity ^0.8.17;
contract Gas {
   function GasConsumed() public returns(uint) {
        return tx.gasprice;
    }
   uint public i = 0;
   function forever() public {
        // Here we run a loop until all of the gas are spent
        // and the transaction fails
        while (true) {
            i += 1;
        }
    }
```

4- Dynamic array : Push and Pop operations

```
pragma solidity ^0.8.17;

contract Array {

    uint[] DynamicArray = [10,20,30,40];
    // uint[4] StaticArray = [10,20,30,40];

    function InsertPush(uint value) public {
        DynamicArray.push(value);
    }

    function DeletePop() public {
        DynamicArray.pop();
    }

    function DeleteValue(uint index) public {
        delete DynamicArray[index];
    }

    function UpdateValue(uint value, uint index) public {
        DynamicArray[index] = value;
    }

    function getArray() public view returns(uint[] memory){
      return DynamicArray;
    }

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

    function TransfertX2() public view returns (uint[] memory) {
        uint len = DynamicArray.length;
        uint[] memory ArrayX2 = new uint[](len);
        for (uint i = 0; i < len; i++)
        {
            ArrayX2[i] = DynamicArray[i]*2;
        }
        return ArrayX2;
    }

}
```

5- Iterable mapping : Mapping is not iterable

```
pragma solidity ^0.8.17;

contract IterableMapping {

    mapping (address => uint) public map;
    mapping (address => bool) public inserted;
    address[] public keys;

    function set(address key, uint value) public {
        map[key] = value;
        if (!inserted[key])
        {
            inserted[key]= true;
            keys.push(key);
        }
    }

    function getMapSize() public view returns (uint) {
        return keys.length;
    }

    function first() public view returns (uint) {
        return map[keys[0]];
    }

    function last() public view returns (uint) {
        return map[keys[keys.length-1]];
    }


    function any(uint index) public view returns (uint) {
        return map[keys[index]];
    }

    function getMapAdress() public view returns (address[] memory){
      return keys;
    }

}
```