

Blockchain and Cryptocurrency

Assignment 8: Writing a Smart Contract

8th assignment is about smart contract. It helps you understand the basic mechanism of Smart Contract. Through this assignment, you can practice to program Smart Contract and learn how it runs on the Ethereum network. To create smart contract, you have to use Solidity. You can refer this web site (☞ <http://solidity.readthedocs.io/en/v0.4.25/solidity-in-depth.html>) to understand Solidity.

Description of this assignment:

- This assignment includes creating a simple smart contract.
- This assignment includes creating three special-purpose smart contract.
- You must submit a report containing the results obtained as you go through each step and your contract codes.
 - ⇒ **Submit a report containing all results obtained from all steps and your contract codes.**

<Step. 1> Install Ethereum wallet, Mist

Refer to this URL ☞ <https://github.com/ethereum/mist/releases>, and download the correct version for your PC.

※ You can create and compile smart contracts using Mist.

<Step. 2> Connect to your own private network

Connect Mist to geth.

※ After executing geth console, enter the command line like "**Mist --rpc <http://ip:port>**" on new terminal (In the directory where Mist is located). Also, you can use ipc.

※ When you run geth, consider using these options(--rpc, --ipcpath).

<Step. 3> Create a simple smart contract

Test basic example of smart contract below and try to understand how the smart contract works on Ethereum network.

3-1) Create new smart contract that contains the code below and compile it.

3-2) If you compile successfully, publish the smart contract on the private network.

3-3) After the block contains the transaction generated by the corresponding smart contract is mined,

- You can check the status of the smart contract in detail. Capture the status and attach it to the report

3-4) Call functions included in the smart contract (Sending a transaction).

```
pragma solidity ^0.4.25;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

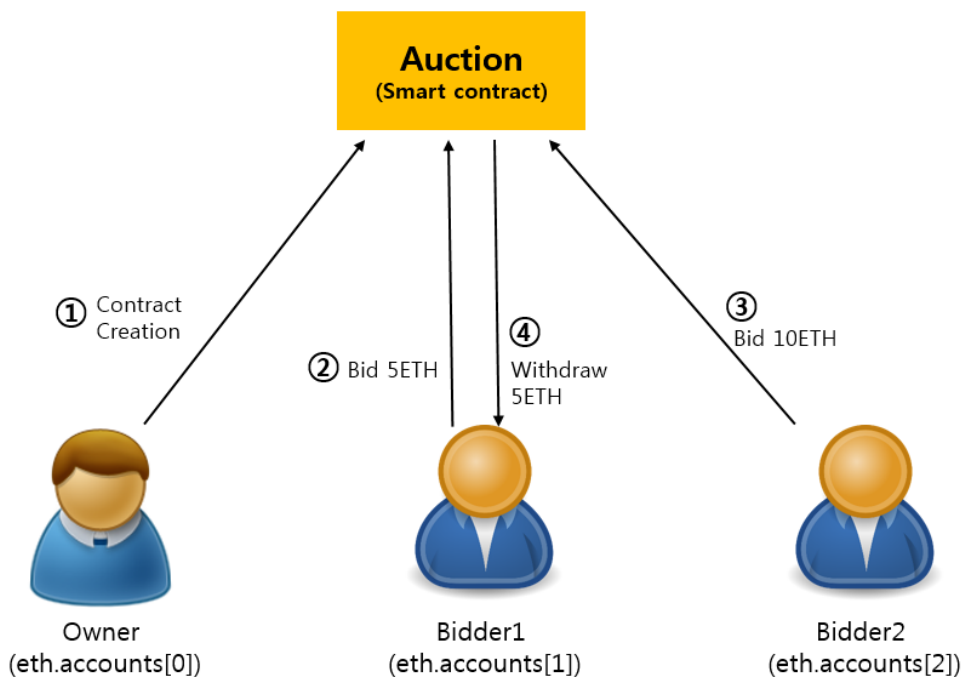
    function get() public constant returns (uint) {
        return storedData;
    }
}
```

3-5) Add logging function to keep track of who recently changed data
(Then, test it)

<Step. 4> Create a smart contract for auction

Scenario is the following:

1. Owner makes Auction smart contract
2. Bidder 1 bids 5ETH to Auction smart contract.
3. After that, Bidder 2 bids 10ETH to Auction smart contract.
4. Because the bid amount of Bidder 2 is higher than the bid amount of Bidder 1, Bidder 1 can withdraw the amount of Bidder 1's bid.



```

pragma solidity ^0.4.25;

contract Auction {
    address public highestBidder; // The highest bidder's address
    uint public highestBid; // The amount of the highest bid
    mapping(address => uint) public userBalances; // mapping for the amount to return

    constructor() public {
        // contractor
        // 1. Initialize highest bid and the bidder's address
    }

    function bid() public {
        // Funtion to process bid
        // 1. Check if the bid is greater than the current highest bid
        // 2. Update status variable and the amount to return
    }

    function withdraw() public {
        // Funtion to withdraw the amount of bid to return
        // 1. Check if the amount to return is greater than zero
        // 2. Update status variablle and return bid
    }
}

```

This is an incomplete code of smart contract for auction. You can implement Auction smart contract, filling contents of each function in this code. Also, in order to complete this contract, you can add and modify something such as state variables, functions, modifiers, etc.

4-0) Prepare three accounts for testing your contract.

(Account 1: Owner, **Account 2:** Bidder 1, **Account 3:** Bidder 2)

4-1) Complete Auction smart contract and save it as "Auction.sol".

4-2) Compile the code using solc. (After installing sole, use sole --abi -bin. Then, you can check **contract ABI** and **bytecode** of yours)

4-3) Send transaction to create the contract, on account 1(owner).

4-4) Check the balance of contract account and account 2, the amount of the highest bid and address of the highest bidder.

4-5) Send transaction to execute bid() function in other to bid 5ETH, on account 2(Bidder 1).

4-6) Check the balance of contract account, account 2 and account 3, the amount of the highest bid and address of the highest bidder.

4-7) Send transaction to execute bid() function in other to bid 10ETH, on account 3(Bidder 2).

4-8) Check the balance of contract account and account 3, the amount of the highest bid and address of the highest bidder.

4-9) Send transaction to execute withdraw() function in other to withdraw the amount of bid, on account 2.

4-10) Check the balance of contract account and account 2.

<Step. 5> Create a smart contract for crowd funding

Crowdfunding smart contract creates contracts for fundraising purposes only and collects ETH. An investor can invest in a way that creates a transaction that sends an investor to this contract.

The contract sets up the deadline and target amount for the fundraising activities and remits the collected ETH to the owner of the contract when the target value is achieved at the closing date. If the target value is not achieved, return ETH to the investor.

```
pragma solidity ^0.4.25;

contract CrowdFunding {
    // Investor struct
    struct Investor {
        address addr; // investor's address
        uint amount; // investment amount
    }
    address public owner; // contract owner
    uint public numInvestors; // the number of investors
    uint public deadline; // deadline for this contract to be closed
    string public status; // "Funding", "Campagin Success", "Campagin Failed"
    bool public end; // the end of funding
    uint public goalAmount; // target amount
    uint public totoalAmount; //total ammount
    mapping(uint => Investor) public investors;

    // 1. Create modifier to limit to owner

    // Constructor
    constructor(uint _duration, uint _goalAmount) public {
        // Initialize owner, deadline, goalAmount, status, end, numInvestors, totoalAmount
    }

    // Function to be called when investing
    function fund() public {
        // 1. Check if this crowd funding ended or not
        // 2. Set invest-related info and process funding
    }

    function checkFoalReached () public {
        // 1. Check if this crowd funding ended or not

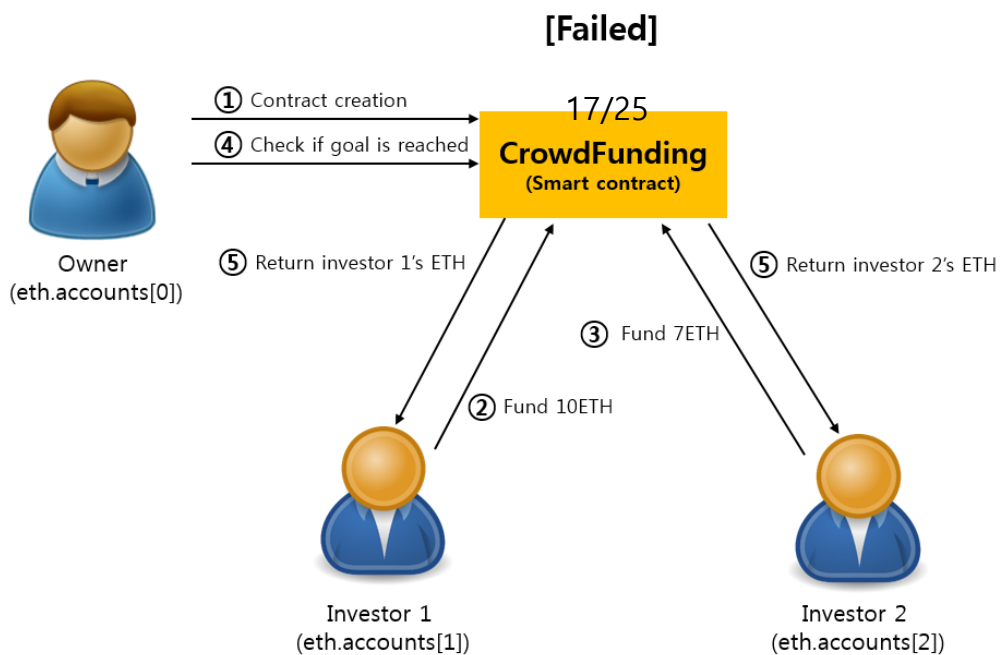
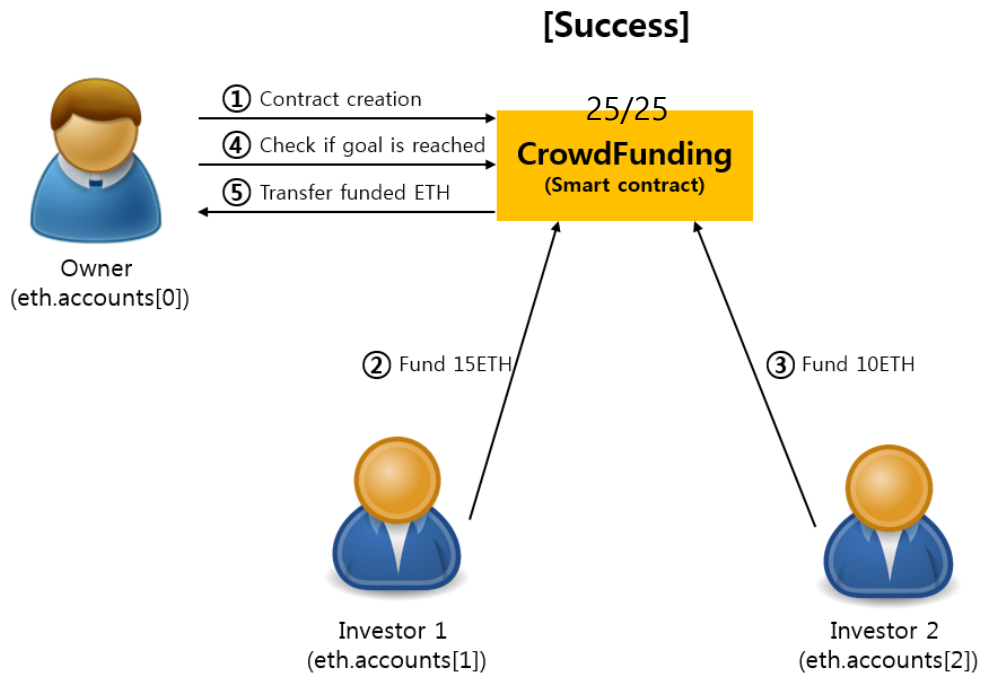
        // 2. Check if the deadline is past or not

        // 3-1. If this crowd funding is successful, send funded ETH to owner
        // 3-2. If not, return fund-raising to each investor
        // Consider updating status and end
    }

    // 1. Create function to destroy this contract
}
}
```

This is an incomplete code of smart contract for crowd funding. You can implement CrowdFunding smart contract, filling contents of each function in this code. Also, in order to complete this contract, you can add and modify something such as state variables, functions, modifiers, etc.

There are two scenarios, **Success** and **Failed** of funding.



5-0) Prepare three accounts for testing your contract.

(**Account 1:** Owner, **Account 2:** Investor 1, **Account 3:** Investor 2)

5-1) Complete CrowdFunding smart contract and save it as "CrowdFunding.sol".

5-2) Compile the code using solc. (After installing sole, use sole --abi -bin. Then, you can check **contract ABI** and **bytecode** of yours)

5-3) Send transaction to create the contract, on account 1(owner).

(When creating the contract, set _duration to 300 (5min) and _goalAmount to 25ETH)

5-4) Check deadline and end state.

<<<<<**Case of successful fund**>>>>>

5-5) Send transaction to execute fund() function in order to fund 15ETH and 10ETH from Investor 1 and Investor 2 respectively, on account 2 and 3.

5-6) Check investment of Investor 1, 2.

5-7) Check total investment of the contract and balance of contract account.

5-8) Check owner's balance (Account 1)

5-9) Send transaction to execute checkGoalReached() function to check the fundraising results
(Do it after the deadline!!!)

5-10) Check deadline and end state.

5-11) Check balance of contract account and owner.

<<<<<**Case of failed fund**>>>>>

5-12) Follow steps from 5-3 to 5-4.

5-13) Send transaction to execute fund() function in order to fund 10ETH and 7ETH from Investor 1 and Investor 2 respectively, on account 2 and 3.

5-14) Check investment of Investor 1, 2.

5-15) Check total investment of the contract and balance of contract account.

5-16) Check balance of Investor 1, 2 and owner.

5-17) Send transaction to execute checkGoalReached() function to check the fundraising results
(Do it after the deadline!!!)

5-18) Check deadline and end state.

5-19) Check balance of contract account.

5-20) Check balance of Investor 1, 2 and owner.

<TA>

Kyungchan Ko <kkc90@postech.ac.kr>

Changhoon Kang <chkang@postech.ac.kr>