Lab 2: Smart Contract Programming

The learning objective of this lab is for students to acquire programming skills in smart contracts and to develop basic programs. The lab will consider an educational scenario of smart contract, that is, the rock-paper-scissors game. The lab will be developed on Ethereum/Solidity platform using existing programming platform such as Remix. There will be programming exercises for students to implement two-party computation (e.g., `max(x,y)`) and three-party rock-paper-scissors game.

Lab Description: This module consists of four lab exercises where the first two exercises (1a and/or 1b) are to set up programming environments. Exercise 1b is a bonus one (with 20% extra points). The last two exercises require you to design a contract program based on the given requirement. There are two programming environments you can choose from: the web-based Remix IDE and the command-line-based Solc. Use of Solc will give you 20% bonus.

Solidity

Solidity is an object-oriented programming language for writing smart contracts mainly on Ethereum. We provide a crash course on Solidity Syntax at [link]. You can find other tutorials [here].

To write a Solidity program, you have to have an account payable, which is used as the constructor. Otherwise, you will not be able to make deposits/transfers in the contract.

Exercise 1a: Hello-world Contract with Remix

In Exercises 1, we will learn to compile and execute a given Solidity program. It is a hello-world program given as below. Function `greeter` takes a string argument and stores it in Variable `greeting` . Function `greet` take no argument and returns the value of Variable `greeting` .

```
pragma solidity ^ 0.4.13;
   contract hello { /* define variable greeting of the type string */
    string greeting;
       function greeter(string _greeting) public {
              greeting = _greeting;
       }
       /* main function */
       function greet() public constant returns(string) {
              return greeting;
       }
}
```

In this exercise, we will use Remix which is a web-based IDE for contract development. It supports the full programming life cycle including contract compilation, debugging, development, and execution. More details about the Remix IDE can be found in [link].

1.1 Compile the code using `Start to Compile` button provided in the Remix IDE. Check for the errors (if any) and resolve them.

1.2 Deploy the contract using `Deploy` button provided under `Run` tab. You can see the deployed contracts and functions deployed on the right-bottom corner. Provide the input for `greeter` function and click on "transact" button. You can see transaction being successful in the "Remix Transactions" section.

1.3 Click on "greet" button/function, you can see the string value set for "greeting" using "greeter" function will be displayed. Submit the final screenshot of running this Solidity program.

Exercise 1b (with 20% bonus): Hello-world Contract with Solc and On-Campus Ethereum

In this exercise, we will use a command-line programming framework `solc` to compile and deploy the hello-world contract to Ethereum. We will use our on-campus Ethereum network (as in the previous labs).

Follow the instructions [here] to compile and execute the hello-world contract. Submit the screenshot of contract-execution results.

Exercise 2: Find the Maximum

In this exercise, you are asked to write a Solidity program to find the maximum of two values, x and y, and return that value.The Smart contact should have the following functionalities:

1. A function that takes x and y as input
2. Returns the maximum of x and y as output. Deploy the program and run the program in the Remix IDE [link]

In exercise 1, if you took 1a option, you should do exercise 2 with Remix. If you took option 1b, then you can do exercise 2 with Solc.

Exercise 3: Rock-paper-scissors game

Write a Smart contract to implement the Rock-Paper-Scissors game in solidity. You can use variables to keep track of the deposit and player values.The contract should have the following functionalities:

1. There should be two players. Consider one specific address as the owner address (where both players will deposit their money).
2. Each player deposits an initial amount of 5 Ethers into the owner account.
3. Once both the players deposit the money, allow them to play. While depositing the money, make sure you keep track of who is depositing and make him/her the player1 or player2 accordingly.
4. Write a function `play` which takes the string parameter (Choice of the player - Rock, paper, scissors) and consider their choice only if they have deposited successfully.
5. Once both the players have input their choices, find the winner and transfer the money as below:

- a. If player1 wins, send bid amount ie, 10 Ethers to player1.
- b. If player2 wins, send bid amount ie, 10 Ethers to player2.
- c. If both the players win, divide the bid amount and send to players equally. Once the game is finished, `Account` values (on the right-top corner of the IDE) of the designated player addresses should be updated. Make sure the player is depositing exactly 5 Ethers else the transaction should be rejected. While depositing the amount (5 Ethers in our case), `Value` on the right-top corner must be equivalent to 5 Ethers, in-order for the Remix to send the transaction successfully.

Deliverable

- For all exercises, you should show the screenshot of you executing the program successfully. For instance, in Exercise 1, your screenshort should show the execution of a sequence such as `greeter("X")` and `greet()` which prints the value of `X`.
- In Exercise 2,3, you should also submit the solidity program you wrote.