**Department of Electrical Engineering**
**Sharif University of Technology**

**Foundations of Blockchain**
**by: Dr. Mohammad A. Maddah-Ali**
**Fall 1399(2020)**

---

# Practical Homework 3

---

If you have any questions about the problems please contact at this email address : kasraabbaszadeh@gmail.com

- Read each problem and code comments completely before coding. Provide exactly what the questions ask for.
- In the template exercise files that you are asked to complete:
  - You are allowed to add functions, variables and modifiers.
  - You are also allowed to add/delete "payable" keyword to/from functions and variables
  - You are **not** allowed to remove or rename functions
  - You are **not** allowed to change arguments passed to the functions
- Deliverables of this homework consist of a report and a set of smart contracts, and are specified in each section.
- Recommended editor for this exercise is Remix and you can compile and test your codes using Javascript VM as environment unless you're explicitly asked to use another environment (Problem 3 and Problem 5)
- If you have difficulties in submitting your answers, ask your questions in Quera.
- The policy for delayed submission is as follows: up til 24 hours after the deadline, 75 percent of the grade is given. Between 24 hours and 48 hours after the deadline, 50 percent of the grade is given. 48 hours after the deadline, submission is closed.

**Problem 1:** Hello World

Let's say Hello to the world of ethereum smart contracts. In this section, you will write a smart contract which its sole purpose is to greet people. Create a file in remix, paste Greeter.sol in it and Deploy the contract using "Hello From #YOUR_STUDENT_NUMBER" as argument (replace #YOUR_STUDENT_NUMBER with your student number).

**Deliverable:** Call greeting method and place a screenshot of the result in your report.

**Problem 2:** ERC20

In this section you are going to implement a custom version of ERC20 token. First of all, What is an ERC20 token?

ERC20, which stands for Ethereum Requests for Comment, is a set of programming rules that all Ethereum-based tokens are expected to follow. The Ethereum Developers agreed on these three (optional) variables, six functions, and two logging events as protocol standard for the minimal viable token, in order to normalize expected behaviors while communicating across the Ethereum network. By establishing this protocol, Ethereum developers are able to more easily integrate and work with token contracts already published to the network.

```
interface CustomERC20 {
contract ERC20Interface {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  function allowance(address owner, address spender) public view
returns (uint256);
  function transferFrom(address from, address to, uint256 value)
public returns (bool);
  function approve(address spender, uint256 value) public returns
(bool);

  event Approval(address indexed owner, address indexed spender,
uint256 value);
  event Transfer(address indexed from, address indexed to, uint256
value);
}
}
```

**Deliverable:** Complete CutomERC20.sol in order to satisfy requirements.

**Problem 3:** ERC721

ERC721 tokens are non fungible tokens (NFT), which essentially means they are all unique compared to each other, unlike the standard ERC20 tokens. Imagine trading one American dollar for another American dollar. This is allowed because they are the same currency that holds the value, which means you still end up with one dollar. Now, imagine trading one house with another house. This isn't allowed because they hold different values and have different properties, such as are in different locations, and have other values that make it unique, but it is still a house. This is the difference between standard tokens and NFTs.

```
Interface ERC721 {
event Transfer(address indexed _from, address indexed _to, uint256
_tokenId);
event Approval(address indexed _owner, address indexed _approved,
uint256 _tokenId);
event ApprovalForAll(address indexed _owner, address indexed _operator,
bool _approved);
function balanceOf(address _owner) public view returns (uint256);
function ownerOf(uint256 _tokenId) public view returns (address);
function safeTransferFrom(address _from, address _to, uint256 _tokenId,
bytes data) public;
function safeTransferFrom(address _from, address _to, uint256 _tokenId)
public;
function transferFrom(address _from, address _to, uint256 _tokenId)
public;
function approve(address _approved, uint256 _tokenId) public;
function setApprovalForAll(address _operator, bool _approved) public;
function getApproved(uint256 _tokenId) public view returns (address);
function isApprovedForAll(address _owner, address _operator) public
view returns (bool);
}
```

**Deliverable:** Complete CutomERC721.sol in order to satisfy requirements.

**Problem 4:** Funding Program

A blockchain company decides to grant developers to build a decentralized exchange on top of the ethereum and you are asked to code some smart contracts and build an auction Dapp to manage this funding program.the logic is as follows :

First the auction's admin creates a new contract and opens the auction by locking the max amount of funds he will pay we call this "ReserveFund". By creation the contract Commitment phase starts.

Now, developers must upload their proposal on IPFS submit it's address in addition to a commitment to their bid, not bid itself. Specifically, this commitment should be a SHA-3 hash of a 32 byte nonce and their bid value. then admin should choose 3 numbers of participants with the best proposals and submit their address. At this time the Commitment phase closes and the Opening phase starts.

In this phase all bidders should reveal their bid by sending the nonce used in their bid commitment. After sometime, admin can close the auction through calling finalize() function. Winner receives his fund from locked ether's and the rest will be sent back to Funder.

You should implement Ethereum smart contracts that supports this kind of auction. In addition to the reserve fund , your auction contract will take 2 additional parameters: length of Commitment phase and Opening phase measured in blocks.

**Deliverable:** Complete Auction.sol, CutomAuction.sol and Bid.sol in order to satisfy requirements.

**Bonus:** Proposal files are now visible for anyone but in real usecases only admin should have access them. One way to preserving privacy of data is encrypting the address of file with admins public Key to submit in bid commitment instead of address itself. Improve app to achive this goal.

**Problem 5:** Deploy to a testnet

In this section, we are going to deploy the Auction app to Reposten public testnet.

**Prerequisites:**

- Download MetaMask extension from Chrome Web Store.
- Go through MetaMask setup steps
- After setup select Ropsten Test Network from the drop down on top of the extension.
- Click on deposit and then GET ETHER to get your first test net ethers.
- in remix change the environment to Injected Web3.
- if everything goes right you should be able to see your account and its balance.

**Deploy the contract to Ropsten network:**

- Deploy the CustomAuction to Ropsten Network using remix.
- MetaMask catches your request and asks you to confirm it. You can tweak some variable here too.

If everything goes right, after the confirmation your contract will be deployed to Ropsten and you can trace your transaction on Etherscan using the links provided in remix as well as MetaMask.

Activate your CustomAuction and use 5 accounts as participants for bidding. Choose 3 of them as admin and finalize the Auction.

place screenshots of all steps in your report.

**Problem 6:** Example Dapp

In DApp folder you can find an application which connects a user interface to blockchain using ethereum's web3.js library. In order to test our DApp we are going to use a blockchain emulator called Ganache.

**Prerequisites:**

- Download Ganache from here and start it.
- In remix change the environment to Web3 Provider and accept the default endpoint (if you can't connect to the default endpoint try http://localhost:7545).
- If everything goes right you should be able to see ten accounts with balance 100 in the balance drop-down in remix.

**Deploy the contract to Ganache:**

- Deploy the voting contract (given in problem 4) to Ganache using remix.

**Connect the DApp to your contract:**

- Open DAPP/js/index.js
- Copy the contract address and paste it in the contractAddress field (line 12).
- Copy the contract ABI and past it in the abi field (line 16). You can copy it from "SOLIDITY COMPILER" tab in remix.
- Make sure web3 provider port is correct (line 9),
- Open voting.html

You should be able to see a working voting DApp.

**Deliverable:** vote to the candidates several times using the DApp UI and place a screenshot of the DApp (after your votes) in your report.

# References


[1] Install and work with IPFS

[2] Avanced Cryptocurrency Topics: ERC20 Interface

[3] ERC-20 vs ERC-721 Tokens