Assignment *N* Proposal

Description

The purpose of this assignment is to introduce the functionality of Solidity, writing / deploying Smart Contracts, and interacting with other contracts on the Ethereum Test Network.

Requirements

- Chrome / Firefox / Opera / Brave
- Metamask Browser Extension

Prerequisites

First, download and install the Metamask extension, and create a new account under the Ropsten Test Network to obtain a personal wallet address. Save your credentials in a secure place as this account will be reused throughout the course. Second, request *free* Ether via the Ropsten Ethereum Faucet using your new setup Ropsten wallet address.

For this assignment, the Remix browser based IDE will be used. This is an integrated compiler and Solidity runtime environment without server-side components.

You will work on the contract individually. Once completed, you will need to work with a partner to complete the remaining portion of the assignment. For another student to interact with your contract, you will need to exchange 1) Contract address and 2) Metamask wallet address.

Development

In Remix, create a new .sol file that will contain the assignment contract. This file can be saved in browser or loaded from a local file system.

Compiler version 0.4.24 will be used and can be selected from the compile tab on the upper right hand corner. When developing and running the contract use the Javascript VM Environment as this allows use of multiple accounts in the browser with pre-loaded ether for ease of testing.

Deploying to the Ropsten Test Network using the Injected Web3 Environment will only be done once code is finalized and ready to live on the Ropsten Test Network. Make note of the contract address when deploying to blockchain as this address will be used for the interaction and submission.

NOTE: Function and variable names should remain unchanged. Unique names will result in changes to the contract's ABI, thus not allowing use of the ABI that is provided below in the Interaction section.

```
pragma solidity >0.4.24;
contract Payout {

    // Create a private address variable 'admin'.
    // Create a private mapping 'accounts', which maps a KeyType address to a ValueType uint.

    // Create an event 'Sent' with 3 parameters: 'from' of type address, 'to' of type
    // address, and 'amount' of type uint.
    // Create an event 'Withdrawn' with 2 parameters: 'from' of type address, and 'amount'
    // of type uint.

    // This is the constructor whose code is run only when the contract is created.
    // Contract can be deployed assigning some value to the creator (admin).
    constructor() public payable {
        // Set 'admin' to be the current sender.
        // Set the admin's account (the mapping) to the value sent with the message.
    }

    // Fallback function if the contract is invoked with some value.
    function () public payable {
        // Add the value sent with the message to the admin's account.
    }

    // As the administrator, assign the value sent with the message to the receiver account.
    function mint(address receiver) external {
        // Require the current sender to be admin, throw an exception if the condition is not met.
        // Add the value sent with the message to the receiver's account.
    }

    // As a user, some amount (in wei) can be sent to some other receiver account
    function send(address receiver, uint amount) external {
        // Require the amount to be less than or equal to the value in the senders account,
        // throw an exception if the condition is not met.
        // Deduct the amount from the senders account.
        // Add the amount to the receivers account.
        // Emit the 'Sent' event with the correct arguments.
    }

    // As a user, some amount (in wei) can be withdrawn from the senders account to wallet
    // address.
    function withdraw(uint amount) external payable {
        // Require the amount to be less than or equal to the value in the senders account,
        // throw an exception if the condition is not met.
        // Deduct the amount from the senders account.
        // Transfer the amount to the senders wallet.
        // Emit the 'Withdrawn' event with the correct arguments.
    }

    // A viewable function is used to check an accounts balance.
    function checkBalance() external view returns (uint) {
        // Return the current senders account value.
    }
}
```

Interaction

Let yourself be **A** and another student be labeled **B**.

As **A**, complete the following tasks:

1. *Mint* a value from your wallet to **B**.
2. Share your unique contract address with **B**.

Two components are needed to interact with another contract in Remix; the deployed contract **address** and the **Application Binary Interface** (ABI). The below ABI defines a schema for the Payout contract, and will be *consistent* for all students.

```
[{"constant":false,"inputs":[{"name":"receiver","type":"address"}],"name":"mint","outputs":
[],"payable":true,"stateMutability":"payable","type":"function"},{"constant":false,"inputs":
[{"name":"receiver","type":"address"},
{"name":"amount","type":"uint256"}],"name":"send","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"anonymous":false,"inputs":
[{"indexed":false,"name":"from","type":"address"},{"indexed":false,"name":"to","type":"address"},
{"indexed":false,"name":"amount","type":"uint256"}],"name":"Sent","type":"event"},
{"payable":true,"stateMutability":"payable","type":"fallback"},{"inputs":
[],"payable":true,"stateMutability":"payable","type":"constructor"},{"constant":false,"inputs":
[{"name":"amount","type":"uint256"}],"name":"withdraw","outputs    “:
[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"anonymous":false,"inputs":
[{"indexed":false,"name":"from","type":"address"},
{"indexed":false,"name":"amount","type":"uint256"}],"name":"Withdrawn","type":"event"},
{"constant":true,"inputs":[],"name":"checkBalance","outputs":
[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"}]
```

As **B,** complete the following tasks:

1. Copy the above JSON into a new `test.abi` file. In the 'Run' tab, load contract from address (from **A**), and press ‘At Address’ to view and interact with the deployed contract.
2. *Check* your balance in the account (this value is the amount in Wei that **A** minted).
3. *Send* 1/2 of the available amount to **A**.
4. *Withdraw* the other 1/2 to your wallet.

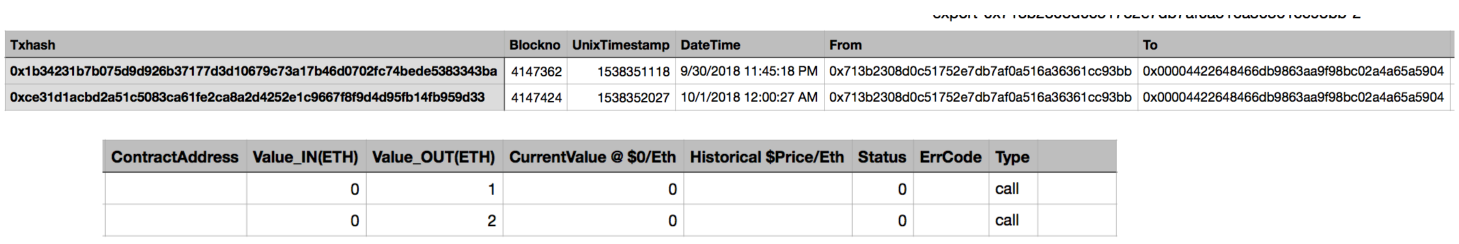
Each student must complete both **A** and **B** tasks to receive credit.

Submission

Contract details and transactions can be verified using https://ropsten.etherscan.io. Follow this link and copy your unique contract address into the search bar. Under the "Internal Txns" tab, download the CSV export and submit it to Canvas.

Grading

Note for Dr. Gersch: With the submitted CSV you can confirm the assignment is completed by using to and from address. The contract address is listed in the From cell, and the wallet address in the To cell. The students wallet address they submitted at the beginning of class should match in the To cell



Txhash	Blockno	UnixTimestamp	DateTime	From	To
0x1b34231b7b075d9d926b37177d3d10679c73a17b46d0702fc74bede5383343ba	4147362	1538351118	9/30/2018 11:45:18 PM	0x713b2308d0c51752e7db7af0a516a36361cc93bb	0x00004422648466db9863aa9f98bc02a4a65a5904
0xce31d1acbd2a51c5083ca61fe2ca8a2d4252e1c9667f8f9d4d95fb14fb959d33	4147424	1538352027	10/1/2018 12:00:27 AM	0x713b2308d0c51752e7db7af0a516a36361cc93bb	0x00004422648466db9863aa9f98bc02a4a65a5904

ContractAddress	Value_IN(ETH)	Value_OUT(ETH)	CurrentValue @ $0/Eth	Historical $Price/Eth	Status	ErrCode	Type
	0	1	0		0		call
	0	2	0		0		call

SOLUTION

```
pragma solidity >0.4.24;
contract Payout {

    // An admin is initialized and used to hold the creator of the contract.
    // A mapping is used to hold internal addresses and some value amount.
    address private admin;
    mapping (address => uint) private accounts;

    // Events allow light clients to react to changes efficiently.
    event Sent(address from, address to, uint amount);
    event Withdrawn(address from, uint amount);

    // This is the constructor whose code is run only when the contract is created.
    // Contract can be deployed assigning some value to the creator (admin).
    constructor() public payable {
        admin = msg.sender;
        accounts[admin] = msg.value;
    }

    // Fallback function if the contract is invoked with some value.
    function () public payable {
        accounts[admin] += msg.value;
    }

    // As the administrator, assign the value sent with the message receiver account.
    function mint(address receiver) external payable {
        require(msg.sender == admin, "Permissions Denied.");
        accounts[receiver] += msg.value;
    }

    // As a user, some amount (in wei) can be sent to some other receiver account
   function send(address receiver, uint amount) external {
        require(amount <= accounts[msg.sender], "Can not send.  Insufficient balance.");
        accounts[msg.sender] -= amount;
        accounts[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }

    // As a user, some amount (in wei) can be withdrawn from an internal user address.
    function withdraw(uint amount) external {
        require(amount <= accounts[msg.sender], "Can not withdraw. Insufficient balance.");
        accounts[msg.sender] -= amount;
        msg.sender.transfer(amount);
        emit Withdrawn(msg.sender, amount);

    }

    // A viewable function is used to check an accounts internal balance.
    function checkBalance() external view returns (uint) {
        return accounts[msg.sender];
    }
}
```