

Introduction to Blockchains

Assignment 2

Devdatt N(200010012)

Version 1 :

Here, we have successfully deployed the contract after adding the requisite details:

The screenshot displays the Remix IDE interface during the deployment of a smart contract. The left sidebar contains the 'DEPLOY & RUN TRANSACTIONS' panel, which shows the environment set to 'Remix VM (London)', the account '0x5B3...edC4 (99.999999%)', a gas limit of '3000000', and a value of '0 Wei'. The contract selected is 'Auction - contracts/auction.sol'. The 'Deploy' button is highlighted in red. Below it, there are options to 'Publish to IPFS' or 'At Address'. The bottom of the sidebar shows 'Transactions recorded' and 'Deployed Contracts'.

The main editor displays the Solidity code for the 'Auction' contract. The code includes a pragma statement for Solidity version ^0.4.17, a contract definition for 'Auction', and several internal structures and mappings. The 'Data' section defines an 'Item' struct with 'itemId' and 'itemTokens' fields. The 'Person' struct defines 'remainingTokens', 'personId', and 'address' fields. A mapping 'tokenDetails' maps addresses to 'Person' objects. The 'bidders' array contains 'Person' objects. The 'items' array contains 'Item' objects. The 'winners' array contains addresses. The 'beneficiary' is an address. The 'bidderCount' is a counter. The 'constructor' is public and payable, initializing the beneficiary with the sender's address.

```
1 pragma solidity ^0.4.17;
2 contract Auction {
3
4     // Data
5     //Structure to hold details of the item
6     struct Item {
7         uint itemId; // id of the item
8         uint[] itemTokens; //tokens bid in favor of the item
9     }
10
11     //Structure to hold the details of a persons
12     struct Person {
13         uint remainingTokens; // tokens remaining with bidder
14         uint personId; // it serves as tokenId as well
15         address addr; //address of the bidder
16     }
17
18     mapping(address => Person) tokenDetails; //address to person
19     Person [4] bidders; //Array containing 4 person objects
20
21     Item [3] public items; //Array containing 3 item objects
22     address [3] public winners; //Array for address of winners
23     address public beneficiary; //owner of the smart contract
24
25     uint bidderCount=0; //counter
26
27     //functions
28
29
30     constructor() public payable{ //constructor
31
32         //Part 1 Task 1. Initialize beneficiary with address of smart contract's owner
33         //Hint. In the constructor, "msg.sender" is the address of the owner.
34         // ** start code here. 1 line approximately. **
35     }
```

Now, we will register with all four accounts. Account balances after registering the four bidding accounts:

```
0x617...5E7f2 (99.99999999)
0x5B3...eddC4 (99.999999999999386295 ether)
0xA8b...35cb2 (99.999999999999862553 ether)
0x4B2...C02db (99.999999999999839853 ether)
0x787...cabaB (99.999999999999839853 ether)
0x617...5E7f2 (99.999999999999839853 ether)
0x17F...8c372 (100 ether)
0x5c6...21678 (100 ether)
0x02C...D1F57 (100 ether)
```

Executing bid function for each of the four accounts: \

```
✓ [vm] from: 0x787...cabaB to: Auction.bid(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x598...00001 logs: 0 hash: 0xc98...31f8c
transact to Auction.bid pending ... Debug ▼

✓ [vm] from: 0x787...cabaB to: Auction.bid(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x598...00001 logs: 0 hash: 0x8f6...b0a7d
transact to Auction.bid pending ... Debug ▼

✓ [vm] from: 0x4B2...C02db to: Auction.bid(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x598...00001 logs: 0 hash: 0x35c...3a337
transact to Auction.bid pending ... Debug ▼

✓ [vm] from: 0x4B2...C02db to: Auction.bid(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x598...00001 logs: 0 hash: 0xf0e...05c47
transact to Auction.bid pending ... Debug ▼

✓ [vm] from: 0x4B2...C02db to: Auction.bid(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x598...00001 logs: 0 hash: 0x2a7...7b34f
transact to Auction.bid pending ... Debug ▼

✓ [vm] from: 0x4B2...C02db to: Auction.bid(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x598...00001 logs: 0 hash: 0x6df...f0f6c
transact to Auction.bid pending ... Debug ▼

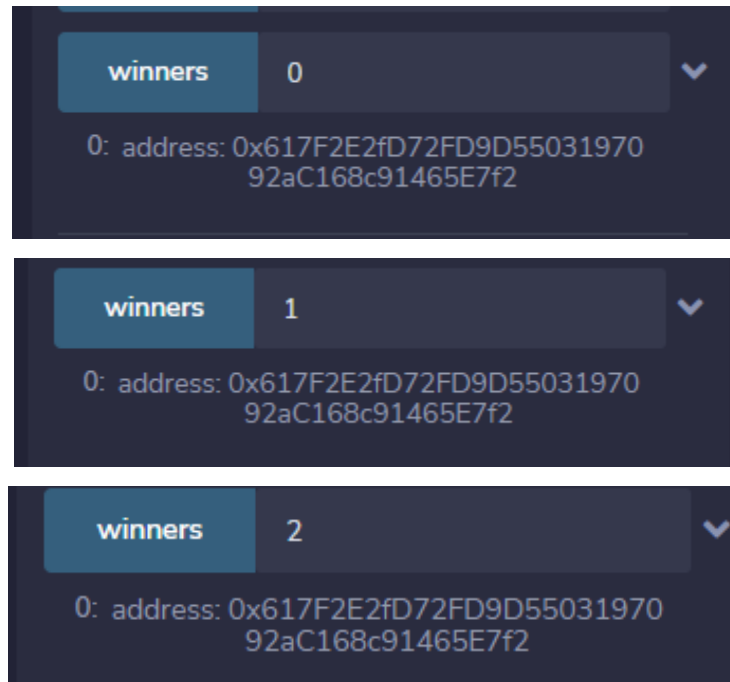
✓ [vm] from: 0x4B2...C02db to: Auction.bid(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x598...00001 logs: 0 hash: 0x465...2947f
transact to Auction.bid errored: VM error: revert.
revert
```

Now, we execute revealWinners() :

```
✓ [vm] from: 0xA8b...35cb2 to: Auction.bid(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x598...00001 logs: 0 hash: 0xb19...744ab
transact to Auction.revealWinners pending ... Debug ▼

✓ [vm] from: 0x5B3...eddC4 to: Auction.revealWinners() 0xd91...39138 value: 0 wei data: 0x952...587d6 logs: 0 hash: 0x9cb...2216c
Debug ▼
```

After this, we can get the winners:



We can see that the winner is the same in all three cases. This might be because the random value we generate is in fact not random.

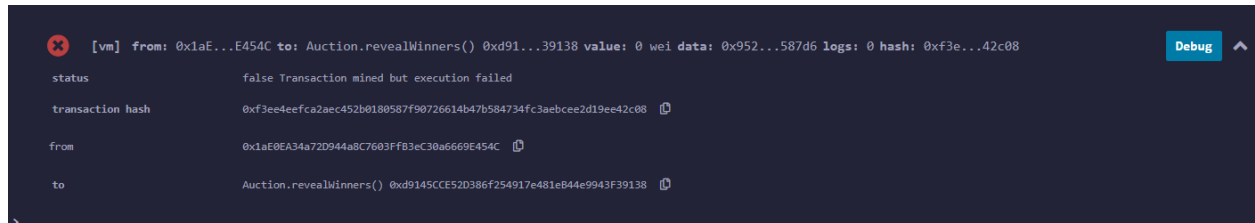
Version 2 :

We have added the `onlyOwner` modifier to `revealWinners()` so that only the auctioneer can call the function:

```
//Hint : Use require to validate if "msg.sender" is equ
modifier onlyOwner {
    // ** Start code here. 2 lines approximately. **
    |     require(beneficiary == msg.sender);
    |     _;
    //** End code here. **
}

function revealWinners() public onlyOwner{
    /*
```

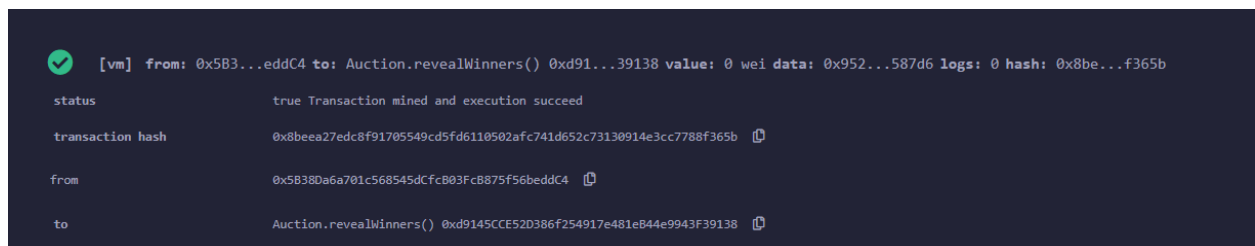
In this case, after trying to execute `revealWinners()` from an address that is not the auctioneer, we come across an error:



A screenshot of a debugger window showing a transaction error. The transaction is from `0x1aE...E454C` to `Auction.revealWinners()` with a value of `0 wei` and data `0x952...587d6`. The status is `false Transaction mined but execution failed`. The transaction hash is `0xf3ee4eefca2aec452b0180587f90726614b47b584734fc3aebcee2d19ee42c08`. The from address is `0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C` and the to address is `Auction.revealWinners() 0xd9145CCE52D386f254917e481eB44e9943F39138`. A `Debug` button is visible in the top right corner.

```
[vm] from: 0x1aE...E454C to: Auction.revealWinners() 0xd91...39138 value: 0 wei data: 0x952...587d6 logs: 0 hash: 0xf3e...42c08
status false Transaction mined but execution failed
transaction hash 0xf3ee4eefca2aec452b0180587f90726614b47b584734fc3aebcee2d19ee42c08
from 0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C
to Auction.revealWinners() 0xd9145CCE52D386f254917e481eB44e9943F39138
```

However, on calling the function from the auctioneer's account:



A screenshot of a debugger window showing a successful transaction. The transaction is from `0x5B3...eddC4` to `Auction.revealWinners()` with a value of `0 wei` and data `0x952...587d6`. The status is `true Transaction mined and execution succeed`. The transaction hash is `0x8beea27edc8f91705549cd5fd6110502afc741d652c73130914e3cc7788f365b`. The from address is `0x5B38Da6a701c568545dCfcB03FcB875f56beddC4` and the to address is `Auction.revealWinners() 0xd9145CCE52D386f254917e481eB44e9943F39138`.

```
[vm] from: 0x5B3...eddC4 to: Auction.revealWinners() 0xd91...39138 value: 0 wei data: 0x952...587d6 logs: 0 hash: 0x8be...f365b
status true Transaction mined and execution succeed
transaction hash 0x8beea27edc8f91705549cd5fd6110502afc741d652c73130914e3cc7788f365b
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to Auction.revealWinners() 0xd9145CCE52D386f254917e481eB44e9943F39138
```

We can see that the transaction is successful, therefore our smart contract is functioning.

Note: Version 1 smart contract code is `Auction__1.sol`, while Version 2 code is `Auction__2.sol`