Experiment No.7

Aim: To create a simple auction contract where users can place bids and the highest bidder wins at the auction's conclusion (Auction Contract).

I. **Objectives:**
   A. Show how to manage Ethereum transactions and handle Ether within contracts.
   B. Implement time-based logic to manage auction start and end times.
   C. Utilize events to log important contract activities.

II. **Steps:**
   A. Access Remix IDE: Open your web browser and navigate to Remix IDE.
   B. Create and Prepare the Contract File
   - In the "File Explorers" tab, click on the "Create New File" icon and name your file SimpleAuction.sol.
   - Copy and paste the SimpleAuction contract code into SimpleAuction.sol.

```
// Specifies the license under which this contract is released; MIT in
this case
// SPDX-License-Identifier: MIT

// Set the compiler version to be used for this contract
pragma solidity ^0.8.22;

// Declares a new contract named SimpleAuction
contract SimpleAuction {
    // Address that will receive the funds of the auction
    address payable public beneficiary;
    // Timestamp for when the auction ends
    uint public auctionEndTime;

    // Address of the bidder with the highest bid
    address public highestBidder;
    // Amount of the highest bid
    uint public highestBid;

    // Mapping to keep track of funds each address has contributed,
    // to be returned if they are outbid
    mapping(address => uint) pendingReturns;

    // Events to emit when a bid is higher than the previous ones,
    // and when the auction ends
    event HighestBidIncreased(address bidder, uint amount);
    event AuctionEnded(address winner, uint amount);

    // Constructor to initialize the auction with a bidding time
    // and the beneficiary's address
    constructor(uint _biddingTime, address payable _beneficiary) {
        beneficiary = _beneficiary;
        auctionEndTime = block.timestamp + _biddingTime;
    }

    // Function to place a bid on the auction
    function bid() public payable {
        // Requires that the auction has not yet ended
        require(block.timestamp <= auctionEndTime, "Auction already
ended.");
        // Requires that the bid is higher than the current highest bid
        require(msg.value > highestBid, "There already is a higher
bid.");

        // If there was a previous bid, it adds the previous highest bid to
        // the pending returns of the previous highest bidder
        if (highestBid != 0) {
            pendingReturns[highestBidder] += highestBid;
        }

        // Updates the state with the new highest bid and bidder
        highestBidder = msg.sender;
        highestBid = msg.value;
        emit HighestBidIncreased(msg.sender, msg.value);
    }

    // Function to withdraw a bid that was overbid by another bidder
    function withdraw() public returns (bool) {
        uint amount = pendingReturns[msg.sender];
        if (amount > 0) {
            // Resets the pending return before sending to prevent re-
entrancy attacks
            pendingReturns[msg.sender] = 0;

            // Attempts to send the amount back to the sender
            if (!payable(msg.sender).send(amount)) {
                // If sending fails, reset the pending return to allow retrying
                pendingReturns[msg.sender] = amount;
                return false;
            }
        }
        return true;
    }

    // Function to end the auction, can only be called after the
auctionEndTime
    function auctionEnd() public {
        // Requires that the current time is past the auction end time
        require(block.timestamp >= auctionEndTime, "Auction not yet
ended.");
        // Requires that auctionEnd has not already been called
        require(!ended, "auctionEnd has already been called.");

        // Sets the auction as ended and emits the AuctionEnded event
        ended = true;
        emit AuctionEnded(highestBidder, highestBid);

        // Transfers the highest bid to the beneficiary of the auction
        beneficiary.transfer(highestBid);
    }

    // State variable to keep track if the auction has ended
    bool ended = false;
}
```

C. Compile the Contract

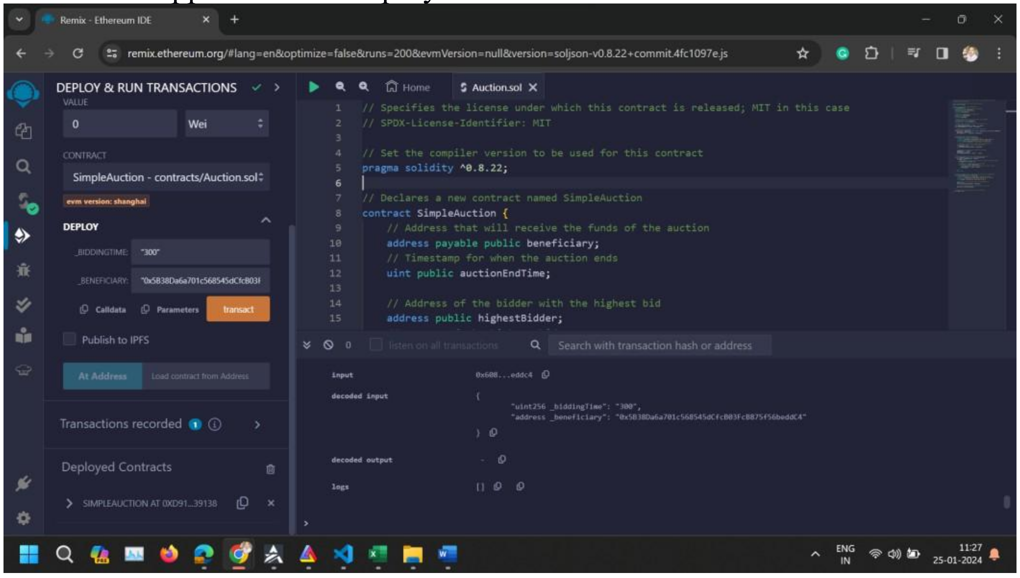
- Click on the "Solidity compiler" tab on the left sidebar.
- Ensure the compiler version is set correctly to match the version specified in your contract (^0.8.22). Remix might automatically select the appropriate version based on your code. If not, select it manually.
- Click "Compile SimpleAuction.sol" to compile your contract.

D. Deploy the Contract:

- Switch to the "Deploy & run transactions" tab by clicking its icon on the left sidebar.
- Select an environment from the "Environment" dropdown. For testing, "JavaScript VM" is recommended as it provides a simulated blockchain.
- Before deploying, you need to provide initial parameters to the constructor. The SimpleAuction constructor requires:
- _biddingTime: The duration of the auction in seconds. For testing, you can use a value like 300 (for 5 minutes).
- _beneficiary: The address that will receive the funds of the auction. In the "JavaScript VM" environment, you can use one of the test accounts provided by Remix. Copy one of the account addresses from the account dropdown.
- Enter these parameters in the "Deploy" field. It should look something like this (replace the address with one from your environment):

```
300, "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
```

- Click "Deploy" to deploy your contract. After deployment, the contract will appear under "Deployed Contracts".



DEPLOY & RUN TRANSACTIONS
VALUE: 0 Wei
CONTRACT: SimpleAuction - contracts/Auction.sol
evm version: shanghai
DEPLOY
_BIDDINGTIME: "300"
_BENEFICIARY: "0x5B38Da6a701c568545dCfcB03F
Publish to IPFS
Transactions recorded 1
Deployed Contracts
SIMPLEAUCTION AT 0XD91...39138

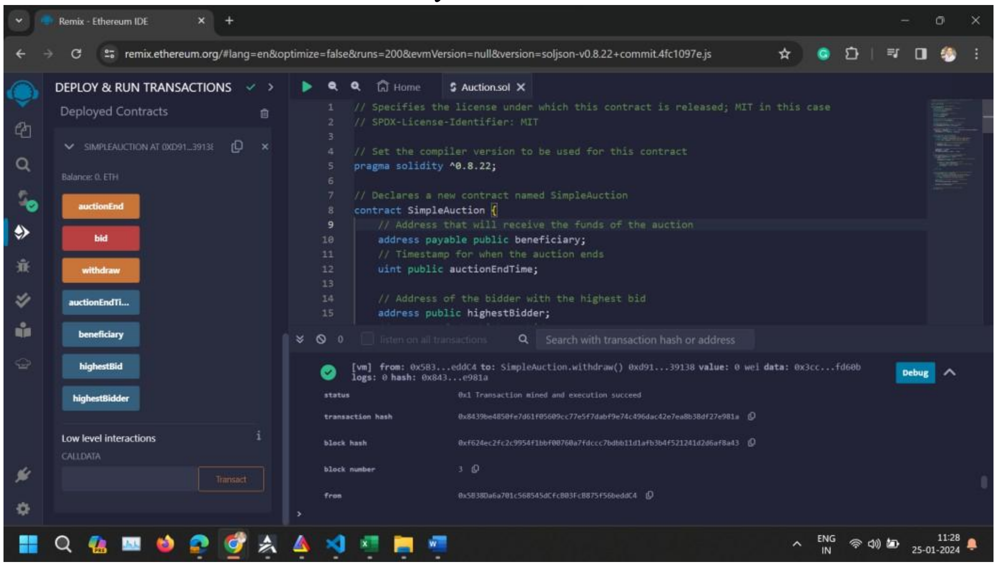
```
// Specifies the license under which this contract is released; MIT in this case
// SPDX-License-Identifier: MIT

// Set the compiler version to be used for this contract
pragma solidity ^0.8.22;

// Declares a new contract named SimpleAuction
contract SimpleAuction {
    // Address that will receive the funds of the auction
    address payable public beneficiary;
    // Timestamp for when the auction ends
    uint public auctionEndTime;

    // Address of the bidder with the highest bid
    address public highestBidder;
```

```
input               0x608...eddc4
decoded input       {
                        "uint256 _biddingTime": "300",
                        "address _beneficiary": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
                    }
decoded output      -
logs                []
```

Remix - Ethereum IDE
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.22+commit.4fc1097e.js
Home
Auction.sol
Calldata
Parameters
transact
At Address
Load contract from Address
listen on all transactions
Search with transaction hash or address
11:27
25-01-2024

E. Interact with the Contract

- **To Place a Bid:** Find your deployed SimpleAuction contract in the "Deployed Contracts" section. Use the bid function to place a bid. Since bid is a payable function, you need to specify the value to send along with the transaction. In the "Value" field (above the "Deploy" button), enter the amount you want to bid and select the correct unit (e.g., ether). Then, click the bid button within your contract's UI in Remix.
- **To Withdraw:** If a bidder has been outbid, they can withdraw their funds using the withdraw function. Simply click the withdraw button.
- **To End the Auction:** Use the auctionEnd function after the bidding time has passed. Click the auctionEnd button to finalize the auction and transfer the funds to the beneficiary.



DEPLOY & RUN TRANSACTIONS
Deployed Contracts
SIMPLEAUCTION AT 0XD91...3913E
Balance: 0. ETH

```
// Specifies the license under which this contract is released; MIT in this case
// SPDX-License-Identifier: MIT

// Set the compiler version to be used for this contract
pragma solidity ^0.8.22;

// Declares a new contract named SimpleAuction
contract SimpleAuction {
    // Address that will receive the funds of the auction
    address payable public beneficiary;
    // Timestamp for when the auction ends
    uint public auctionEndTime;

    // Address of the bidder with the highest bid
    address public highestBidder;
```

```
[vm] from: 0x5B3...eddC4 to: SimpleAuction.withdraw() 0xd91...39138 value: 0 wei data: 0x3cc...fd60b
logs: 0 hash: 0x843...e981a
status              0x1 Transaction mined and execution succeed
transaction hash    0x8439be4850fe7d61f05609cc77e5f7dabf9e74c496dac42e7ea8b38df27e981a
block hash          0xf624ec2fc2c9954f1bbf00760a7fdccc7bdbb11d1afb3b4f521241d2d6af8a43
block number        3
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
```

Conclusion: This experiment demonstrates the dynamic and interactive capabilities of smart contracts, especially in handling financial transactions and time-dependent operations. It provides insights into building more complex decentralized applications (dApps) involving financial logic. It showcases the basics of creating a timed auction system on the Ethereum blockchain. By following these steps, you can deploy and test the auction contract in a simulated environment using Remix IDE. This process provides hands-on experience with contract deployment, interaction, and understanding key concepts like payable functions and time-based logic in Solidity