Blockchain
Lab
Experiment 5

Mohit Patil
D20A 04

Aim: Deploying a Voting/Ballot Smart Contract

Theory

1. Relevance of `required` Statements in Solidity Programs

In Solidity, the `require` statement is used as a validation mechanism to enforce conditions before executing a function. It acts as a guard clause that ensures only valid inputs, authorized users, or correct states are allowed to proceed further in the program.

If the condition specified inside the `require` statement evaluates to **false**, the following actions occur:

- The function execution immediately stops.
- All state changes made during that transaction are reverted.
- The remaining gas is refunded (except the gas already consumed).
- An optional error message is returned.

This rollback mechanism is crucial in blockchain applications because it prevents invalid or malicious transactions from altering the contract's state permanently. Since blockchain data is immutable, such protective mechanisms are essential for maintaining integrity and security.

Example: Voting Smart Contract

In a Voting (Ballot) Smart Contract, `require` can be used in the following

scenarios: To check whether a voter has the right to vote:

```
require(voters[msg.sender].weight > 0, "Has no right to vote");
```

- To ensure that a voter has not already voted.
- To verify that only the chairperson can grant voting rights.

Therefore, `require` statements enhance:

- **Security** – Prevent unauthorized access.
- **Correctness** – Ensure logical conditions are satisfied.
- **Reliability** – Maintain consistency of blockchain data.
- **User Experience** – Provide meaningful error messages for debugging.

2. Important Keywords: `mapping`, `storage`, and `memory`

(a) `mapping`

A `mapping` in Solidity is a key-value data structure similar to a hash table or dictionary. Its syntax is:

```
mapping(keyType => valueType)
```

Example:

```
mapping(address => Voter) public voters;
```

In this case, each Ethereum address is associated with a `Voter` structure.

Characteristics of Mapping:

- Provides fast lookup of values using keys.
- Does not store keys explicitly.
- Does not have a length property.
- Cannot be iterated over directly.
- More gas-efficient for data retrieval compared to arrays.

Mappings are widely used in smart contracts such as voting systems, token balances, and ownership tracking.

(b) `storage`

`storage` refers to the permanent data area of a smart contract stored on the Ethereum blockchain. Variables declared at the contract level are stored in storage by default.

Key Features:

- Data is persistent across transactions.
- Changes remain recorded on the blockchain.
- Writing to storage consumes higher gas fees.
- Suitable for maintaining contract state.

For example, voter details stored inside a mapping remain available throughout the contract's lifecycle unless explicitly modified.

(c) `memory`

`memory` is a temporary data location used during function execution. Variables declared in memory exist only for the duration of the function call.

Key Features:

- Data is discarded after function execution.
- Less expensive than storage.
- Used for temporary variables, parameters, and intermediate computations.

Example use cases include:

- Temporary string manipulation.
- Handling function arguments.
- Processing proposal names before storing them permanently.

A smart contract developer must carefully decide whether to use `storage` or `memory` to optimize gas costs and ensure efficiency.

3. Why Use `bytes32` Instead of `String`?

In earlier implementations of Ballot smart contracts, `bytes32` was commonly used for storing proposal names instead of `string`. The primary reason for this choice was gas efficiency and performance optimization.

`bytes32`

- Fixed-size data type (exactly 32 bytes).
- Requires less storage management.
- Faster comparison operations.
- Lower gas consumption.
- Limited to 32 characters.

Due to its fixed size, `bytes32` is efficient but not flexible for longer or user-friendly names.

`string`

- Dynamically sized data type.
- Can store variable-length text.
- More user-friendly and readable.
- Requires complex memory handling in the Ethereum Virtual Machine (EVM).
- Higher gas consumption.

While `string` improves usability and flexibility, it increases computational and storage costs.

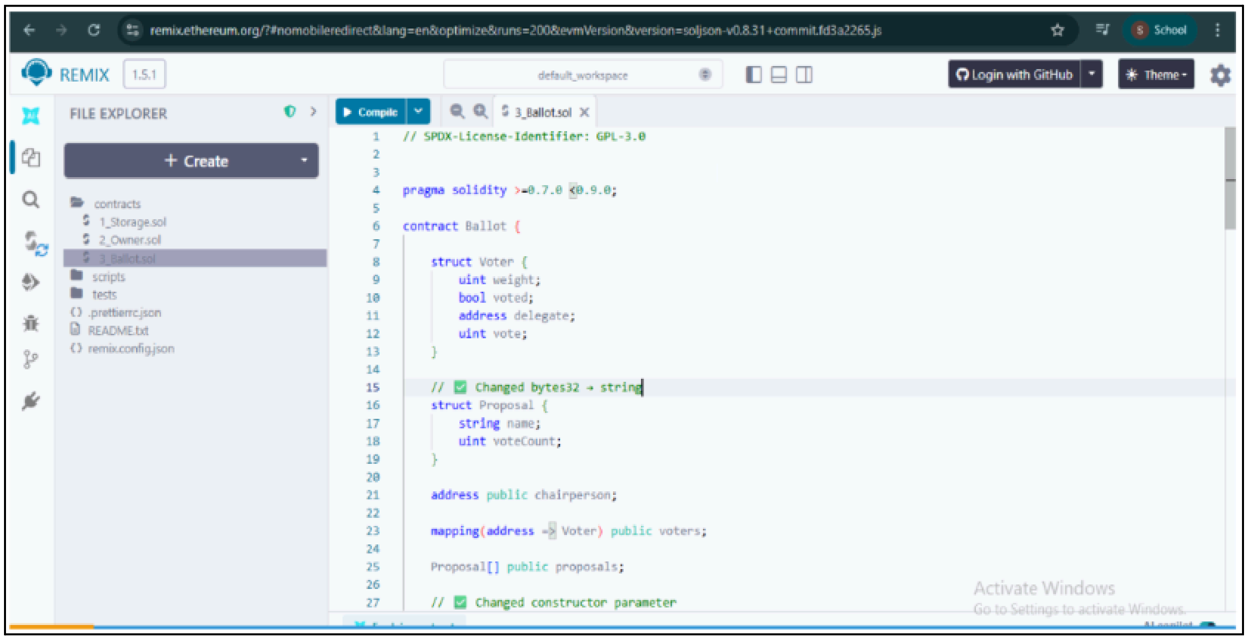
CODE:-

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }

    // ✅  Changed bytes32 →
    string struct Proposal {
        string name;
        uint voteCount;
    }

    address public chairperson;

    mapping(address => Voter) public voters;

    Proposal[] public proposals;

    // ✅  Changed constructor parameter
    constructor(string[] memory proposalNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

        for (uint i = 0; i < proposalNames.length; i++) {
            proposals.push(Proposal({
                name: proposalNames[i],
                voteCount: 0
            }));
        }
    }
```

```
function giveRightToVote(address voter) external {
    require(msg.sender == chairperson, "Only chairperson can give right to vote.");
    require(!voters[voter].voted, "The voter already voted.");
    require(voters[voter].weight == 0, "Voter already has the right to vote.");

    voters[voter].weight = 1;
}

function delegate(address to) external {
    Voter storage sender = voters[msg.sender];

    require(sender.weight != 0, "You have no right to vote");
    require(!sender.voted, "You already voted.");
    require(to != msg.sender, "Self-delegation is disallowed.");

    while (voters[to].delegate != address(0)) {
        to = voters[to].delegate;
        require(to != msg.sender, "Found loop in delegation.");
    }

    Voter storage delegate_ = voters[to];
    require(delegate_.weight >= 1);

    sender.voted = true;
    sender.delegate = to;

    if (delegate_.voted) {
        proposals[delegate_.vote].voteCount += sender.weight;
    } else {
        delegate_.weight += sender.weight;
    }
}

function vote(uint proposal) external {
    Voter storage sender = voters[msg.sender];

    require(sender.weight != 0, "Has no right to vote");
    require(!sender.voted, "Already voted.");
```

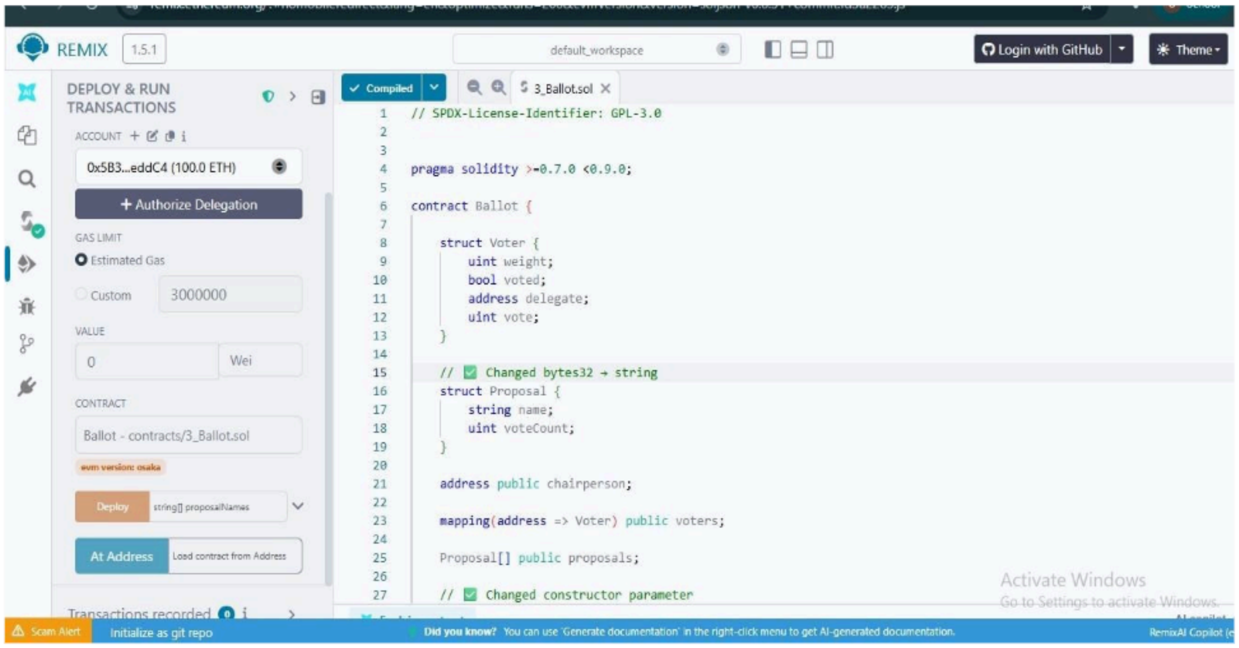
```
        sender.voted = true;
        sender.vote = proposal;

        proposals[proposal].voteCount += sender.weight;
    }

    function winningProposal() public view returns (uint winningProposal_) {
        uint winningVoteCount = 0;

        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
    }

    // ✅  Changed return type

    function winnerName() external view returns (string memory winnerName_) {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```

OUTPUT:-



```
// SPDX-License-Identifier: GPL-3.0


pragma solidity >=0.7.0 <0.9.0;

contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }

    // ✅ Changed bytes32 → string
    struct Proposal {
        string name;
        uint voteCount;
    }

    address public chairperson;

    mapping(address => Voter) public voters;

    Proposal[] public proposals;

    // ✅ Changed constructor parameter
```

Deploying and running of the contract



ACCOUNT 0x5B3...eddC4 (100.0 ETH)

GAS LIMIT: Estimated Gas; Custom 3000000

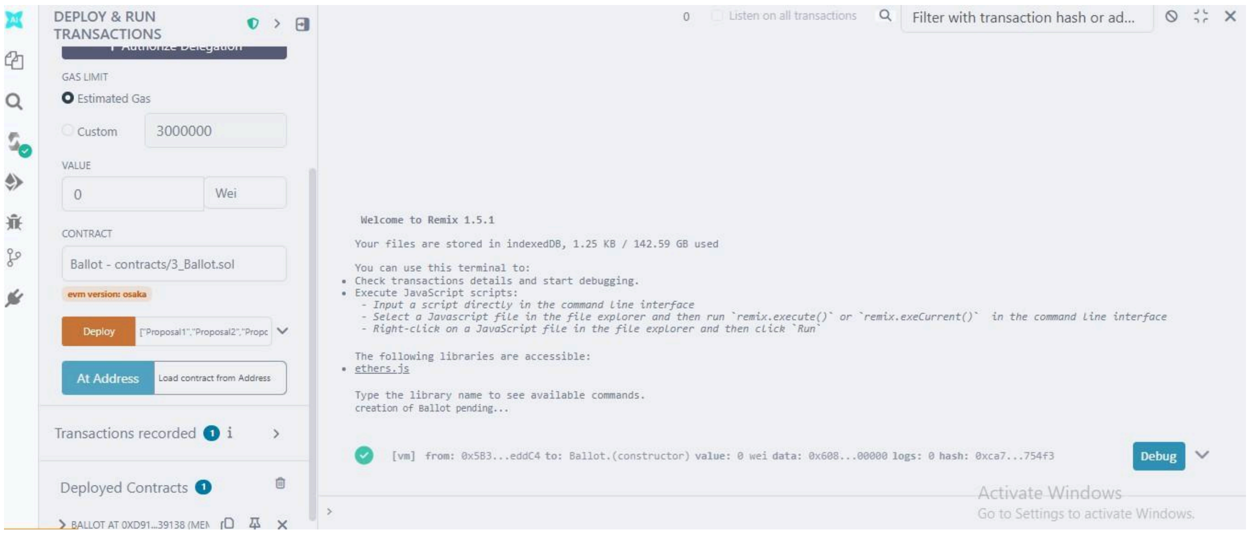
VALUE 0 Wei

CONTRACT Ballot - contracts/3_Ballot.sol

evm version: osaka

Deploy string[] proposalNames

```
// SPDX-License-Identifier: GPL-3.0


pragma solidity >=0.7.0 <0.9.0;

contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }

    // ✅ Changed bytes32 → string
    struct Proposal {
        string name;
        uint voteCount;
    }

    address public chairperson;

    mapping(address => Voter) public voters;

    Proposal[] public proposals;

    // ✅ Changed constructor parameter
```



GAS LIMIT: Estimated Gas; Custom 3000000

VALUE 0 Wei

CONTRACT Ballot - contracts/3_Ballot.sol

evm version: osaka

Deploy ["Proposal1","Proposal2","Propc

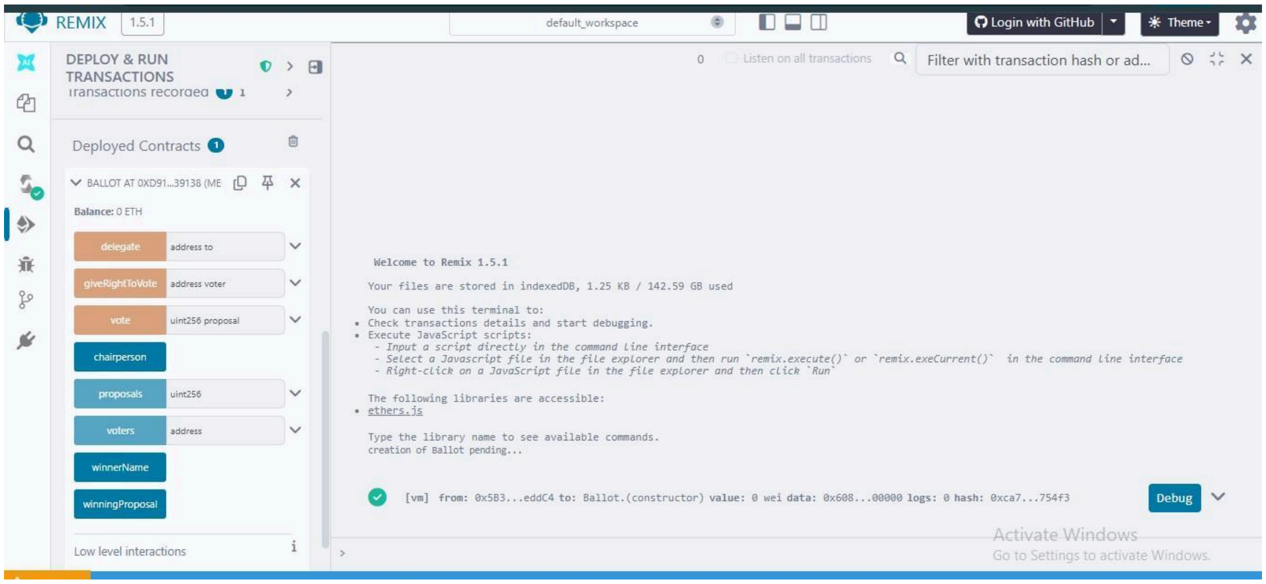
Transactions recorded 1

Deployed Contracts 1

BALLOT AT 0XD91...39138 (MEN

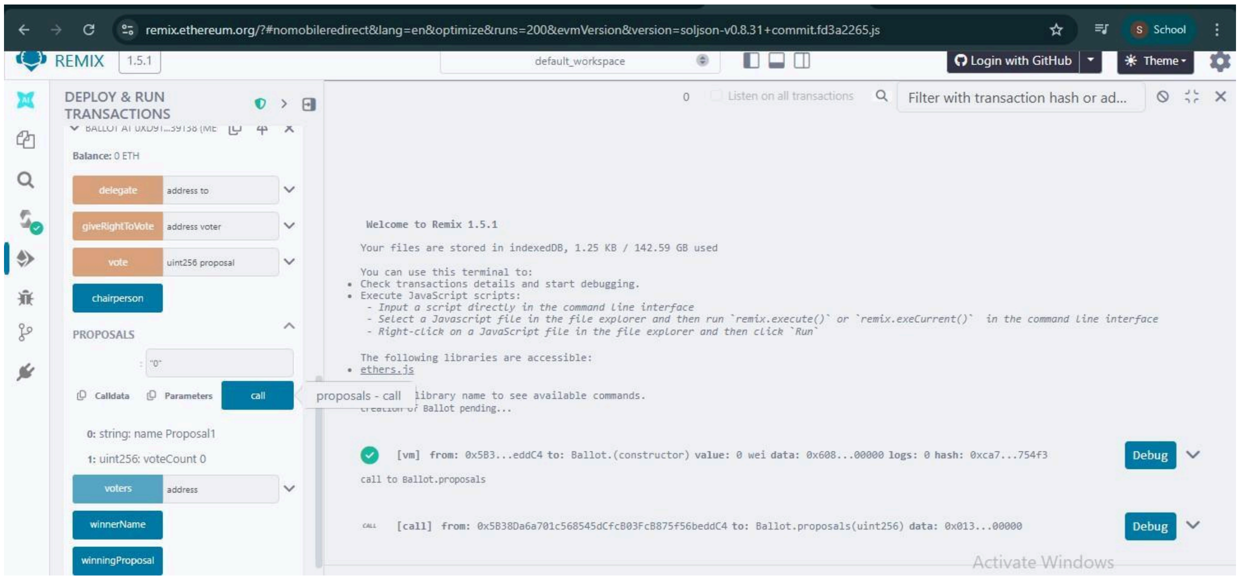
```
Welcome to Remix 1.5.1

Your files are stored in indexedDB, 1.25 KB / 142.59 GB used

You can use this terminal to:
• Check transactions details and start debugging.
• Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()`  in the command line interface
  - Right-click on a JavaScript file in the file explorer and then click `Run`

The following libraries are accessible:
• ethers.js

Type the library name to see available commands.
creation of Ballot pending...

✅ [vm] from: 0x5B3...eddC4 to: Ballot.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xca7...754f3
```

Loading the Proposal Candidate's Names (string)



```
Welcome to Remix 1.5.1

Your files are stored in indexedDB, 1.25 KB / 142.59 GB used

You can use this terminal to:
• Check transactions details and start debugging.
• Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()`  in the command line interface
  - Right-click on a JavaScript file in the file explorer and then click `Run`

The following libraries are accessible:
• ethers.js

Type the library name to see available commands.
creation of Ballot pending...

[vm]  from: 0x5B3...eddC4 to: Ballot.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xca7...754f3
```

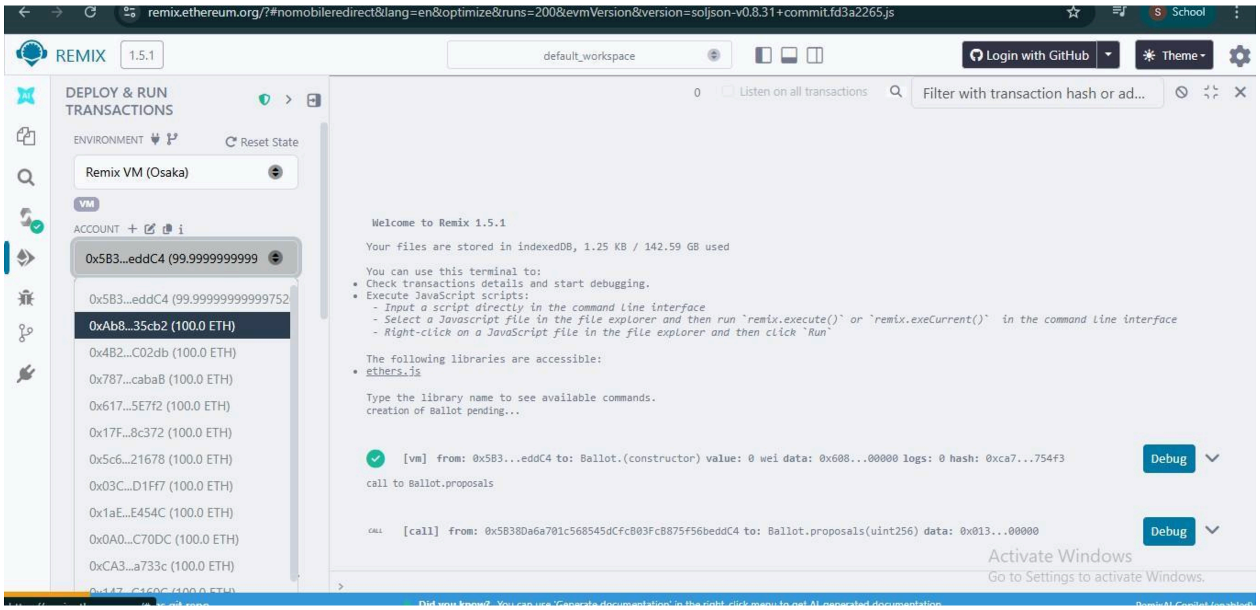
Viewing the details of the Proposal Candidate and Giving the right to an account other than and by the chairman



```
0: string: name Proposal1
1: uint256: voteCount 0
```

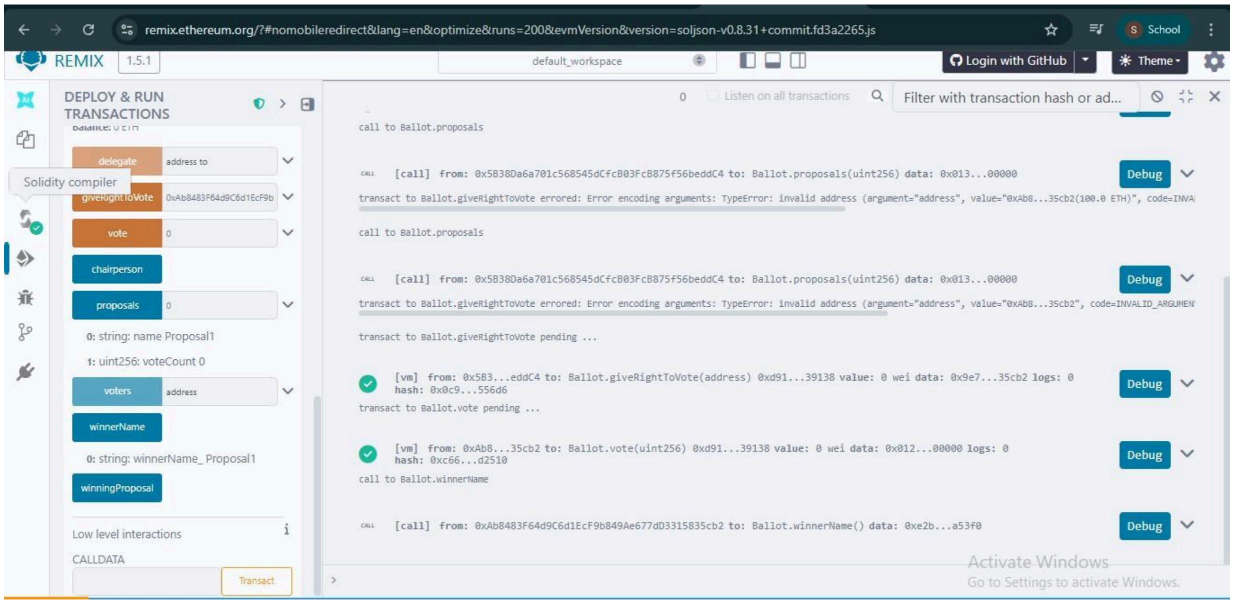
```
Welcome to Remix 1.5.1

Your files are stored in indexedDB, 1.25 KB / 142.59 GB used

You can use this terminal to:
• Check transactions details and start debugging.
• Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()`  in the command line interface
  - Right-click on a JavaScript file in the file explorer and then click `Run`

The following libraries are accessible:
• ethers.js

Type the library name to see available commands.
creation of Ballot pending...

[vm]  from: 0x5B3...eddC4 to: Ballot.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xca7...754f3
call to Ballot.proposals

CALL  [call]  from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ballot.proposals(uint256) data: 0x013...00000
```

Selecting the account which was given the right to vote and then writing the proposal candidate's index to vote.

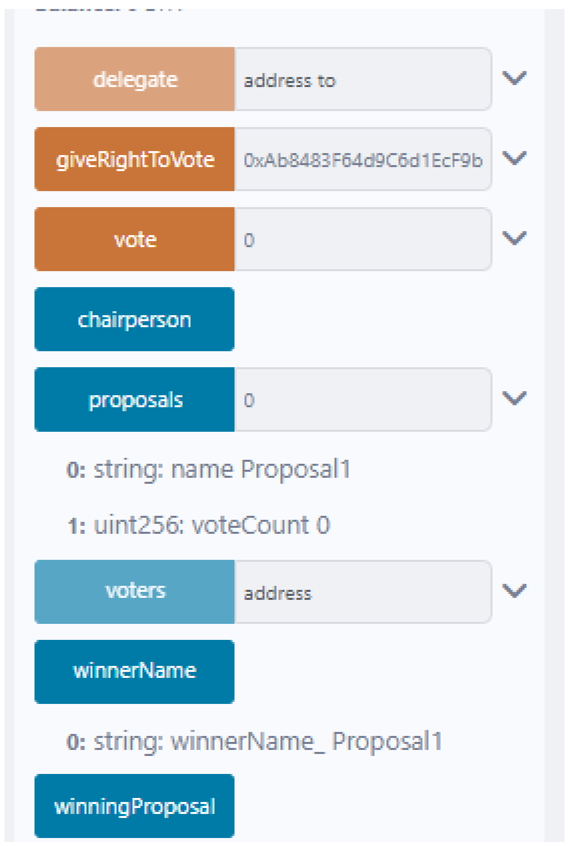

remix.ethereum.org/?#nomobileredirect&lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js
REMIX 1.5.1
default_workspace
Login with GitHub
Theme
DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
Reset State
Remix VM (Osaka)
ACCOUNT
0x5B3...eddC4 (99.9999999999
0x5B3...eddC4 (99.99999999999752
0xAb8...35cb2 (100.0 ETH)
0x4B2...C02db (100.0 ETH)
0x787...cabaB (100.0 ETH)
0x617...5E7f2 (100.0 ETH)
0x17F...8c372 (100.0 ETH)
0x5c6...21678 (100.0 ETH)
0x03C...D1Ff7 (100.0 ETH)
0x1aE...E454C (100.0 ETH)
0x0A0...C70DC (100.0 ETH)
0xCA3...a733c (100.0 ETH)
Listen on all transactions
Filter with transaction hash or ad...

```
Welcome to Remix 1.5.1

Your files are stored in indexedDB, 1.25 KB / 142.59 GB used

You can use this terminal to:
• Check transactions details and start debugging.
• Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()`  in the command line interface
  - Right-click on a JavaScript file in the file explorer and then click `Run`

The following libraries are accessible:
• ethers.js

Type the library name to see available commands.
creation of Ballot pending...

[vm] from: 0x5B3...eddC4 to: Ballot.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xca7...754f3
call to Ballot.proposals

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ballot.proposals(uint256) data: 0x013...00000
```

In this final screenshot we can see that after the delegation's vote the weight of one vote of the one voter who was delegated is the number of people who delegated the voter as their voter (Default weight of any voter is 1).


remix.ethereum.org/?#nomobileredirect&lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js
REMIX 1.5.1
default_workspace
Login with GitHub
Theme
DEPLOY & RUN TRANSACTIONS
Solidity compiler
delegate
address to
giveRightToVote
0xAb8483F64d9C6d1EcF9b
vote
0
chairperson
proposals
0
0: string: name Proposal1
1: uint256: voteCount 0
voters
address
winnerName
0: string: winnerName_ Proposal1
winningProposal
Low level interactions
CALLDATA
Transact
Listen on all transactions
Filter with transaction hash or ad...

```
call to Ballot.proposals

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ballot.proposals(uint256) data: 0x013...00000
transact to Ballot.giveRightToVote errored: Error encoding arguments: TypeError: invalid address (argument="address", value="0xAb8...35cb2(100.0 ETH)", code=INVA

call to Ballot.proposals

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ballot.proposals(uint256) data: 0x013...00000
transact to Ballot.giveRightToVote errored: Error encoding arguments: TypeError: invalid address (argument="address", value="0xAb8...35cb2", code=INVALID_ARGUMEN

transact to Ballot.giveRightToVote pending ...

[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...35cb2 logs: 0
hash: 0x0c9...556d6
transact to Ballot.vote pending ...

[vm] from: 0xAb8...35cb2 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
hash: 0xc66...d2510
call to Ballot.winnerName

CALL [call] from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: Ballot.winnerName() data: 0xe2b...a53f0
```



0: string: name Proposal1

1: uint256: voteCount 0

0: string: winnerName_ Proposal1

CONCLUSION:-

In this practical assignment, a Voting (Ballot) Smart Contract was implemented and deployed using Solidity through the Remix IDE environment. The experiment enabled a deeper understanding of how decentralized applications function on the Ethereum blockchain. Essential features such as `require` statements, `mapping`, and data location specifiers (`storage` and `memory`) were utilized to maintain validation, proper data management, and reliable execution of the contract.

Additionally, the comparison between `bytes32` and `string` data types demonstrated the balance that developers must maintain between minimizing gas consumption and improving readability.

Overall, this exercise enhanced conceptual clarity as well as practical knowledge of smart contract architecture, highlighting the importance of efficiency, security, and thoughtful design in blockchain-based voting systems.