

AIM: Deploying a Voting/Ballot Smart Contract

Theory:

Theory:

1. Relevance of require Statements in Solidity Programs

In Solidity, the require statement acts as a **guard condition** within functions. It ensures that only valid inputs or authorized users can execute certain parts of the code. If the condition inside require is not satisfied, the function execution stops immediately, and all state changes made during that transaction are reverted to their original state. This rollback mechanism ensures that invalid transactions do not corrupt the blockchain data.

For example, in a **Voting Smart Contract**, require can be used to check:

- Whether the person calling the function has the right to vote (`require(voters[msg.sender].weight > 0, "Has no right to vote");`).
- Whether a voter has already voted before allowing them to vote again.
- Whether the function caller is the **chairperson** before granting voting rights.

Thus, require statements enforce **security, correctness, and reliability** in smart contracts. They also allow developers to attach error messages, making debugging and contract interaction easier for users.

2. Keywords: mapping, storage, and memory

mapping:

A mapping is a special data structure in Solidity that links keys to values, similar to a hash table. Its syntax is `mapping(keyType => valueType)`. For example:

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

Here, each address (Ethereum account) is mapped to a Voter structure. Mappings are very useful for contracts like **Ballot**, where you need to associate voters with their data (whether they voted, which proposal they chose, etc.). Unlike arrays, mappings do not have a length property and cannot be iterated over directly, making them **gas efficient** for lookups but limited for enumeration.

storage:

In Solidity, storage refers to the **permanent memory** of the contract, stored on the Ethereum blockchain. Variables declared at the contract level are stored in storage by default. Data stored in storage is persistent across

transactions, which means once written, it remains available unless explicitly modified. However, because writing to blockchain storage consumes gas, it is more expensive. For example, a voter's information saved in the voters mapping remains available throughout the contract's lifecycle.

- **memory:**

In contrast, memory is **temporary storage**, used only for the lifetime of a function call. When the function execution ends, the data stored in memory is discarded. Memory is mainly used for temporary variables, function arguments, or computations that don't need to be permanently stored on the blockchain. It is cheaper than storage in terms of gas cost. For instance, when handling proposal names or temporary string manipulations, memory is often used.

Thus, a smart contract developer must **balance between storage and memory** to ensure efficiency and cost-effectiveness.

3. Why bytes32 Instead of string?

In earlier implementations of the Ballot contract, bytes32 was used for proposal names instead of string. The reason lies in **efficiency and gas optimization**.

- **bytes32** is a **fixed-size type**, meaning it always stores exactly 32 bytes of data. This makes storage simple, comparison operations faster, and gas costs lower. However, it limits proposal names to 32 characters, which is not very flexible for user-friendly names.
- **string** is a **dynamically sized type**, meaning it can store text of variable length. While it is easier for developers and users (since names can be written normally), it requires more complex handling inside the Ethereum Virtual Machine (EVM). This increases gas usage and may slow down comparisons or manipulations.

To make the system more user-friendly, modern implementations of the Ballot contract often convert from bytes32 to string. Tools like the **Web3 Type Converter** help developers easily switch between these two types for deployment and testing.

In summary, bytes32 is used when performance and gas efficiency are priorities, while string is preferred for readability and ease of use.

CODE:

```
contract Ballot {  
  
    struct Voter {  
        uint weight;  
        bool voted;  
        address delegate;  
        uint vote;  
    }  
  
    struct Proposal {  
        string name; // CHANGED from bytes32 → string  
        uint voteCount;  
    }  
  
    address public chairperson;  
    mapping(address => Voter) public voters;  
    Proposal[] public proposals;  
  
    /**  
     * @dev Create a new ballot  
     * @param proposalNames names of proposals  
     */  
    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 propose(string memory proposalName) external {  
        require(voters[msg.sender].voted == false,  
            "Voter has already voted");  
        require(proposals.length <= 100,  
            "Maximum number of proposals reached");  
        require(proposalName != "",  
            "Proposal name cannot be empty");  
        require(proposalName != proposalNames[  
            proposals.length - 1],  
            "Proposal name must be unique");  
  
        voters[msg.sender].voted = true;  
        voters[msg.sender].vote = proposals.length;  
        proposals.push(Proposal({  
            name: proposalName,  
            voteCount: 1  
        }));  
    }  
  
    function vote(uint proposalIndex) external {  
        require(voters[msg.sender].voted == false,  
            "Voter has already voted");  
        require(proposalIndex < proposals.length,  
            "Proposal index out of range");  
        require(proposals[proposalIndex].voteCount < 100,  
            "Proposal has already reached maximum weight");  
  
        voters[msg.sender].voted = true;  
        voters[msg.sender].vote = proposalIndex;  
        proposals[proposalIndex].voteCount++;  
    }  
  
    function getVotes() external view returns (uint totalWeight,  
        uint totalProposals) {  
        for (uint i = 0; i < proposals.length; i++) {  
            totalWeight += voters[proposals[i].vote].weight;  
            totalProposals++;  
        }  
    }  
}
```

```

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 right");

    voters[voter].weight = 1;
}

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

    require(sender.weight != 0, "No right to vote");
    require(!sender.voted, "Already voted");
    require(to != msg.sender, "Self-delegation not allowed");

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

    Voter storage delegate_ = voters[to];

```

```
require(delegate_.weight >= 1, "Delegate has no right");

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, "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;
}

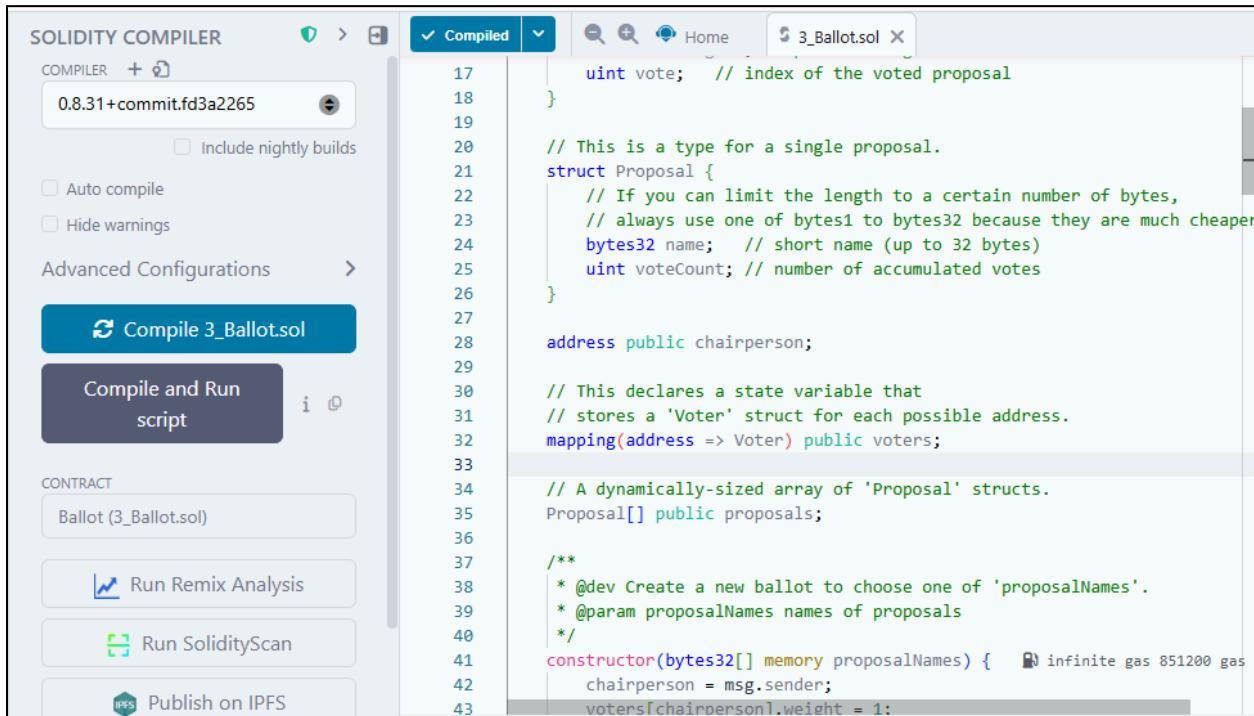
}

}

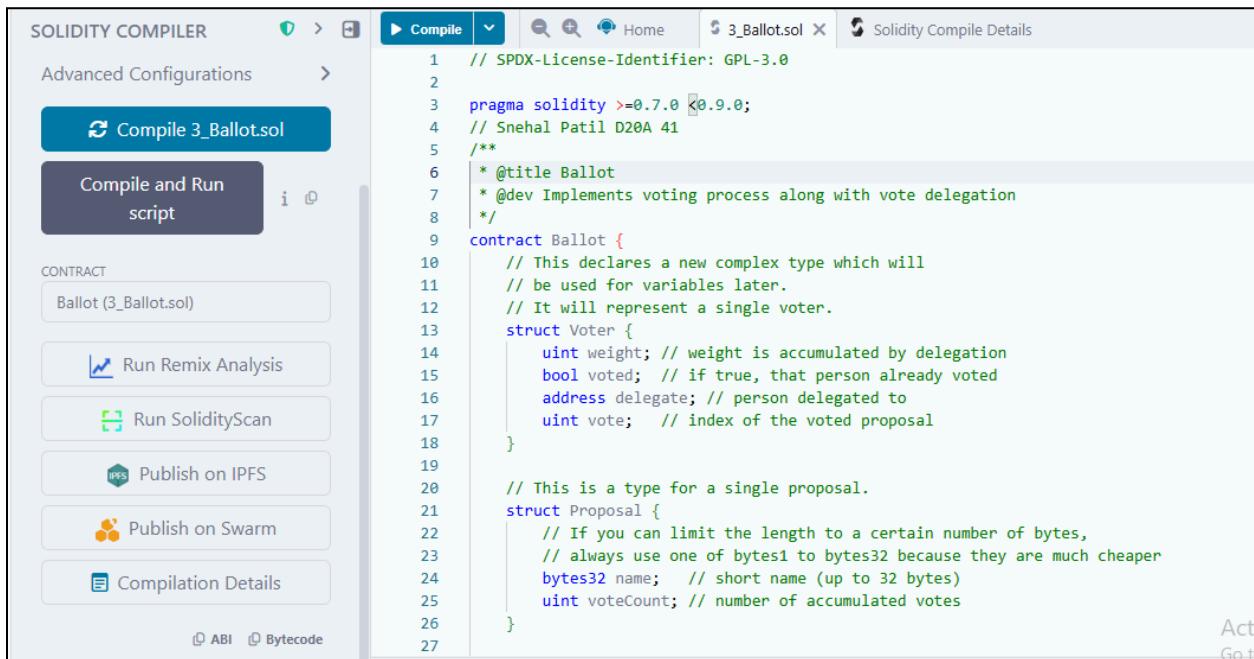
function winnerName() external view returns (string memory) {
    return proposals[winningProposal()].name;
}
}
```

Output:

Compiled ballot



```
17     uint vote; // index of the voted proposal
18 }
19
20 // This is a type for a single proposal.
21 struct Proposal {
22     // If you can limit the length to a certain number of bytes,
23     // always use one of bytes1 to bytes32 because they are much cheaper
24     bytes32 name; // short name (up to 32 bytes)
25     uint voteCount; // number of accumulated votes
26 }
27
28 address public chairperson;
29
30 // This declares a state variable that
31 // stores a 'Voter' struct for each possible address.
32 mapping(address => Voter) public voters;
33
34 // A dynamically-sized array of 'Proposal' structs.
35 Proposal[] public proposals;
36
37 /**
38 * @dev Create a new ballot to choose one of 'proposalNames'.
39 * @param proposalNames names of proposals
40 */
41 constructor(bytes32[] memory proposalNames) {
42     chairperson = msg.sender;
43     voters[chairperson].weight = 1;
```



```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4 // Snehal Patil D20A 41
5 /**
6  * @title Ballot
7  * @dev Implements voting process along with vote delegation
8  */
9 contract Ballot {
10     // This declares a new complex type which will
11     // be used for variables later.
12     // It will represent a single voter.
13     struct Voter {
14         uint weight; // weight is accumulated by delegation
15         bool voted; // if true, that person already voted
16         address delegate; // person delegated to
17         uint vote; // index of the voted proposal
18     }
19
20     // This is a type for a single proposal.
21     struct Proposal {
22         // If you can limit the length to a certain number of bytes,
23         // always use one of bytes1 to bytes32 because they are much cheaper
24         bytes32 name; // short name (up to 32 bytes)
25         uint voteCount; // number of accumulated votes
26     }
```

The screenshot shows the Solidity Compiler interface. In the top left, there's a button labeled "Compile 3_Ballot.sol". Below it, a section titled "Compile and Run script" is highlighted. On the left sidebar, under "CONTRACT", the file "Ballot (3_Ballot.sol)" is selected. The main panel displays the "Ballot" contract details, including its metadata and bytecode. The bytecode section shows the following JSON structure:

```
{
  "functionDebugData": {
    "@_71": {
      "entryPoint": null,
      "id": 71,
      "parametersSlots": 1,
      "returnSlots": 0
    }
  }
}
```

- Loading the Proposal Candidate's Names (string)

The screenshot shows the Remix IDE interface. The title bar says "DEPLOY & RUN TRANSACTIONS". Under "VALUE", the input field is set to "0 Wei". In the "CONTRACT" section, "Ballot - contracts/3_Ballot.sol" is selected, and the "evm version: osaka" is specified. In the "DEPLOY" section, the "proposalNames: ["candidate1","candidate2"]" field is populated. Below it, there are buttons for "Calldata", "Parameters", and a large orange "transact" button. At the bottom, there are buttons for "At Address" and "Load contract from Address". The "Transactions recorded" section shows one transaction. The "Deployed Contracts" section lists "BALLOT AT 0xD91...39138 (MEM)".

Giving right to vote

The screenshot shows the Truffle UI interface for a deployed contract named BALLOT. The sidebar lists several functions: `delegate`, `giveRightToVote`, `vote`, `chairperson`, `proposals`, `voters`, `winnerName`, and `winningProposal`. The `giveRightToVote` function is currently selected. The main pane displays the Solidity code for this function:

```
55 while (voters[to].delegate != address(0)) {
56     to = voters[to].delegate;
57     require(to != msg.sender, "Delegation loop detected");
58 }
```

An "Explain contract" panel is open, showing the execution flow:

- creation of Ballot pending...
- [vm] from: 0x5B3...eddC4 to: Ballot.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xfa3...57297 transact to Ballot.vote pending ...
- [vm] from: 0x5B3...eddC4 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0 hash: 0x593...c1667 call to Ballot.winnerName
- CALL [call] from: 0x5B380a6a701c568545dCfcB03Fc8875f56beddC4 to: Ballot.winnerName() data: 0xe2b...a53f0 transact to Ballot.giveRightToVote pending ...
- [vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...35cb2 logs: 0 hash: 0x697...70233 transact to Ballot.vote pending ...

Each transaction step has a "Debug" button next to it. The bottom right corner of the UI says "Activate Windows Go to Settings to activate Windows."

Voting casted by the candidate

This screenshot is similar to the previous one, but the `vote` function is now selected in the sidebar. The Solidity code for the `vote` function is identical to the `giveRightToVote` function shown above.

```
55 while (voters[to].delegate != address(0)) {
56     to = voters[to].delegate;
57     require(to != msg.sender, "Delegation loop detected");
58 }
```

The "Explain contract" panel shows the same execution steps, starting with the creation of the Ballot contract and the casting of a vote. The bottom right corner of the UI says "Activate Windows Go to Settings to activate Windows."

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with tabs for 'TRANSACTIONS' (0 Wei), 'CONTRACT' (Ballot - contracts/3_Ballot.sol, evm version: osaka), and 'DEPLOY' (proposalNames: ["candidate1","candidate2"], Deploy button). Below these are buttons for 'Calldata', 'Parameters', and 'transact'. A dropdown menu shows 'At Address' and 'Load contract from Address'. Under 'Transactions recorded', there's a checkbox for 'Run transactions using the latest compilation result' and buttons for 'Save' and 'Run'. A 'Deployed Contracts' section shows a single entry: 'BALLOT AT 0xd91...39138 (ME)' with a balance of 0 ETH. On the right, the main area displays the Remix terminal with the following logs:

```
Welcome to Remix 1.5.1
Your files are stored in indexedDB, 496.39 KB / 503.78 MB used
You can use this terminal to:
• Check transaction 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.executeCurrent()` 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 errored: Error encoding arguments: TypeError: expected array value (argument="", value="", code=INVALID_ARGUMENT, version=6.14.0)
Error encoding arguments: TypeError: expected array value (argument="", value="", code=INVALID_ARGUMENT, version=6.14.0)
creation of Ballot errored: Error encoding arguments: TypeError: invalid BytesLike value (argument="value", value="Candidate1", code=INVALID_ARGUMENT, version=6.14.0)
creation of Ballot errored: Error encoding arguments: TypeError: invalid BytesLike value (argument="value", value="Candidate1", code=INVALID_ARGUMENT, version=6.14.0)
creation of Ballot pending...
[vm] from: 0x5B3...eddC4 to: Ballot.(constructor) value: 0 wei data: 0x600...00000 logs: 0 hash: 0xfa3...57297
```

At the bottom right, there's a 'Debug' button and a message: 'Activate Windows Go to Settings to activate Windows.'

Again try to cast vote: one candidate can cast vote only once

This screenshot shows the Remix IDE interface with the 'BALLOT' contract deployed at address 0xd91...39138. The sidebar includes tabs for 'Compiled' (showing the Solidity source code), 'Home', and '3_Ballot.sol'. It also lists variables: 'delegate' (address to), 'giveRightToVote' (0xaB8483F64d9C6d1EcF9b), 'vote' (1), 'chairperson' (blue button), 'proposals' (uint256), 'voters' (address), 'winnerName' (blue button), and 'winningProposal' (blue button). The 'Low level interactions' section shows a 'revert' message: 'The transaction has been reverted to the initial state. Reason provided by the contract: "Already voted". If the transaction failed for not having enough gas, try increasing the gas limit gently.'

The terminal log on the right shows a successful transaction for giving rights and a failed transaction for voting:

```
Hash: 0xb09.../02cc
transact to Ballot.giveRightToVote pending ...

[vm] from: 0xAb8...35cb2 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0xd21...b78a8
transact to Ballot.vote pending ...

[vm] from: 0xAb8...35cb2 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0xb3a...23ba8
transact to Ballot.vote errored: Error occurred: revert.
```

At the bottom right, there's a 'Debug' button and a message: 'Activate Windows Go to Settings to activate Windows.'

Giving right to vote to another candidate

Deployed Contracts (1)

BALLOT AT 0xd91...39138 (ME)

Balance: 0 ETH

Low level interactions

```

54
55
56
57
58
59
}
require(to != msg.sender, "Self-delegation not allowed");
while (voters[to].delegate != address(0)) {
    to = voters[to].delegate;
    require(to != msg.sender, "Delegation loop detected");
}

```

Explain contract

transact to Ballot.giveRightToVote pending ...

[vm] from: 0x4B2...C02db to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x929...2fc43
transact to Ballot.giveRightToVote errored: Error occurred: revert.

revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "Only chairperson can give right to vote".
If the transaction failed for not having enough gas, try increasing the gas limit gently.

transact to Ballot.giveRightToVote pending ...

[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x488...ddf3c

Activate Windows
Go to Settings to activate Windows.

Second candidate casted vote to 1

Deployed Contracts (1)

BALLOT AT 0xd91...39138 (ME)

Balance: 0 ETH

Low level interactions

```

55
56
57
58
59
60
61
while (voters[to].delegate != address(0)) {
    to = voters[to].delegate;
    require(to != msg.sender, "Delegation loop detected");
}
0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db

```

Explain contract

[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x929...2fc43
transact to Ballot.giveRightToVote errored: Error occurred: revert.

revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "Only chairperson can give right to vote".
If the transaction failed for not having enough gas, try increasing the gas limit gently.

transact to Ballot.giveRightToVote pending ...

[vm] from: 0x5B3...eddC4 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0x488...ddf3c
transact to Ballot.vote pending ...

[vm] from: 0x4B2...C02db to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0xcee...2422c

Activate Windows
Go to Settings to activate Windows.

Giving right to vote third candidate

```

DEPLOY & RUN TRANSACTIONS
At Address Load contract from Address
Transactions recorded 9 i
Run transactions using the latest compilation result
Save Run
Deployed Contracts 1
BALLOT AT 0xD91...39138 (ME)
Balance: 0 ETH
delegate address to
giveRightToVote 0x78731D3Ca6b7E34aC0F8
vote 1
chairperson
proposals uint256

```

```

Compile Home 3_Ballot.sol X
55
56 while (voters[to].delegate != address(0)) {
57     to = voters[to].delegate;
58     require(to != msg.sender, "Delegation loop detected");
59 }
60
61
Explain contract
0 Listen on all transactions Filter with transaction hash or ad...
Reason provided by the contract: "Only chairperson can give right to vote".
If the transaction failed for not having enough gas, try increasing the gas limit gently.
transact to Ballot.giveRightToVote pending ...
[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x488...ddf3c
transact to Ballot.vote pending ...
[vm] from: 0x4B2...C02db to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0xcee...2422c
transact to Ballot.giveRightToVote pending ...
[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...cabab logs: 0
hash: 0xec5...1f6bb

```

Activate Windows
Go to Settings to activate Windows.

Third candidate vote to 0

```

DEPLOY & RUN TRANSACTIONS
Balance: 0 ETH
delegate address to
giveRightToVote 0x78731D3Ca6b7E34aC0F8
vote 0
chairperson
proposals uint256
voters address
winnerName
0: string: winnerName_candidate2
winningProposal
Low level interactions
CALLDATA
Transact

```

```

Compile Home 3_Ballot.sol X
56
57 while (voters[to].delegate != address(0)) {
58     to = voters[to].delegate;
59     require(to != msg.sender, "Delegation loop detected");
60 }
61 Voter storage delegate_ = voters[to];
Explain contract
0 Listen on all transactions Filter with transaction hash or ad...
[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x488...ddf3c
transact to Ballot.vote pending ...
[vm] from: 0x4B2...C02db to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0xcee...2422c
transact to Ballot.giveRightToVote pending ...
[vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...cabab logs: 0
hash: 0xec5...1f6bb
transact to Ballot.vote pending ...
[vm] from: 0x787...cabab to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
hash: 0x42f...e4973

```

Activate Windows
Go to Settings to activate Windows.

Winning proposal : candidate2 wins as two candidate voted 1 and one voted 0

DEPLOY & RUN TRANSACTIONS

Balance: 0 ETH

delegate	address to
giveRightToVote	0x78731D3Ca6b7E34aC0F8
vote	0
chairperson	
proposals	uint256
voters	address
winnerName	0: string: winnerName_candidate2
winningProposal	0: uint256: winningProposal_1

Low level interactions
CALldata

[Transact](#)

Compile

```

56   [vm] from: 0x5B3...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...cabab logs: 0
57   hash: 0xec5...1f6bb
58   transact to Ballot.vote pending ...
59
60
61
62   Voter storage delegate_ = voters[to];

```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad... Debug

[vm] from: 0x787...cabaB to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
hash: 0x42f...e4973
call to Ballot.winningProposal

CALL [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cc18A495cabab to: Ballot.winningProposal() data: 0x609...ff1bd
call to Ballot.winnerName

CALL [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cc18A495cabab to: Ballot.winnerName() data: 0xe2b...a53f0
call to Ballot.voters

Activate Windows
Go to Settings to activate Windows.

DEPLOY & RUN TRANSACTIONS

Balance: 0 ETH

delegate	address to
giveRightToVote	0x78731D3Ca6b7E34aC0F8
vote	0
chairperson	
proposals	uint256
voters	address
winnerName	0: string: winnerName_candidate2
winningProposal	0: uint256: winningProposal_1

Low level interactions
CALldata

[Transact](#)

Compile

```

56   [vm] from: 0x787...cabaB to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
57   hash: 0x42f...e4973
58   call to Ballot.winningProposal
59
60
61
62   Voter storage delegate_ = voters[to];

```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad... Debug

[vm] from: 0x787...cabaB to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
hash: 0x42f...e4973
call to Ballot.winningProposal

CALL [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cc18A495cabab to: Ballot.winningProposal() data: 0x609...ff1bd
call to Ballot.winnerName

CALL [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cc18A495cabab to: Ballot.winnerName() data: 0xe2b...a53f0
call to Ballot.voters
call to Ballot.voters errored: Error encoding arguments: TypeError: invalid address (argument="address", value="", code=INVALID_ARGUMENT, version=6.14.0) (argume
transact to Ballot.vote pending ...

- Viewing the details of the Proposal Candidate

DEPLOY & RUN TRANSACTIONS

delegate address to

giveRightToVote 0x78731D3Ca6b7E34aC0F8

vote 0

chairperson

proposals 1

0: string: name candidate2

1: uint256: voteCount 3

voters address

winnerName

0: string: winnerName_ candidate2

winningProposal

0: uint256: winningProposal_ 1

Low level interactions i

CALldata

Transact

This screenshot shows a user interface for interacting with a blockchain contract. The top section is titled 'DEPLOY & RUN TRANSACTIONS'. It contains several input fields and dropdown menus. One dropdown menu is expanded, showing two items: '0: string: name candidate2' and '1: uint256: voteCount 3'. Below this, there are more fields: 'voters address', 'winnerName', and another expanded dropdown showing '0: string: winnerName_ candidate2' and 'winningProposal'. At the bottom, there are buttons for 'Low level interactions' (with an 'i' icon) and 'CALldata', and a large orange 'Transact' button.

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

DEPLOY & RUN TRANSACTIONS

```

{
  "from": "0x4B209938c481177ec7E8f",
  "gas": 240000,
  "gasPrice": 2000000000000000000,
  "maxFeePerGas": 2000000000000000000,
  "maxPriorityFeePerGas": 0,
  "nonce": 0,
  "to": "0x0000000000000000000000000000000000000000",
  "value": 0
}

```

vote 0

chairperson

proposals 1

voters 0x4B209938c481177ec7E8f

0: string: name candidate2

1: uint256: voteCount 3

2: address: delegate 0x00

3: uint256: weight 1

4: bool: voted true

5: address: voter 0x4B209938c481177ec7E8f

6: uint256: proposalIndex 1

7: uint256: proposalWeight 1

winnerName

0: string: winnerName_ candidate2

winningProposal

0: uint256: winningProposal_ 1

- We can view the Voter's Information

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar displays various function calls and their results. On the right, the code editor shows the Solidity source code for the `3_Ballot.sol` contract.

```

DEPLOY & RUN TRANSACTIONS
givedigitovote 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
vote 0
chairperson
proposals 1
0: string: name candidate2
1: uint256: voteCount 3
voters voters - call 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
0: uint256: weight 1
1: bool: voted true
2: address: delegate 0x0000000000000000000000000000000000000000
00
3: uint256: vote 1
winnerName
0: string: winnerName_candidate2
winningProposal
0: uint256: winningProposal_1

3_Ballot.sol
56 while (voters[to].delegate != address(0)) {
57     to = voters[to].delegate;
58     require(to != msg.sender, "Delegation loop detected");
59 }
60
61 Voter storage delegate_ = voters[to];
62
Explain contract
0 Listen on all transactions Filter with
call to Ballot.proposals errored: Error occurred: revert.
revert
The transaction has been reverted to the initial state.
Note: The called function should be payable if you send value and the value you send should be less than your current
If the transaction failed for not having enough gas, try increasing the gas limit gently.

call to Ballot.proposals
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ballot.proposals(uint256) data: 0x01
call to Ballot.voters
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Ballot.voters(address) data: 0xa3e..

```

Conclusion:

In this experiment, a Voting/Ballot smart contract was deployed using Solidity on the Remix IDE. The concepts of require statements, mapping, and data location specifiers like storage and memory were explored to understand their role in ensuring security, efficiency, and correctness in smart contracts. The difference between using bytes32 and string for proposal names was also studied, highlighting the trade-off between gas efficiency and readability. Overall, the experiment provided practical insights into the design and deployment of voting contracts on the blockchain.

