

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.

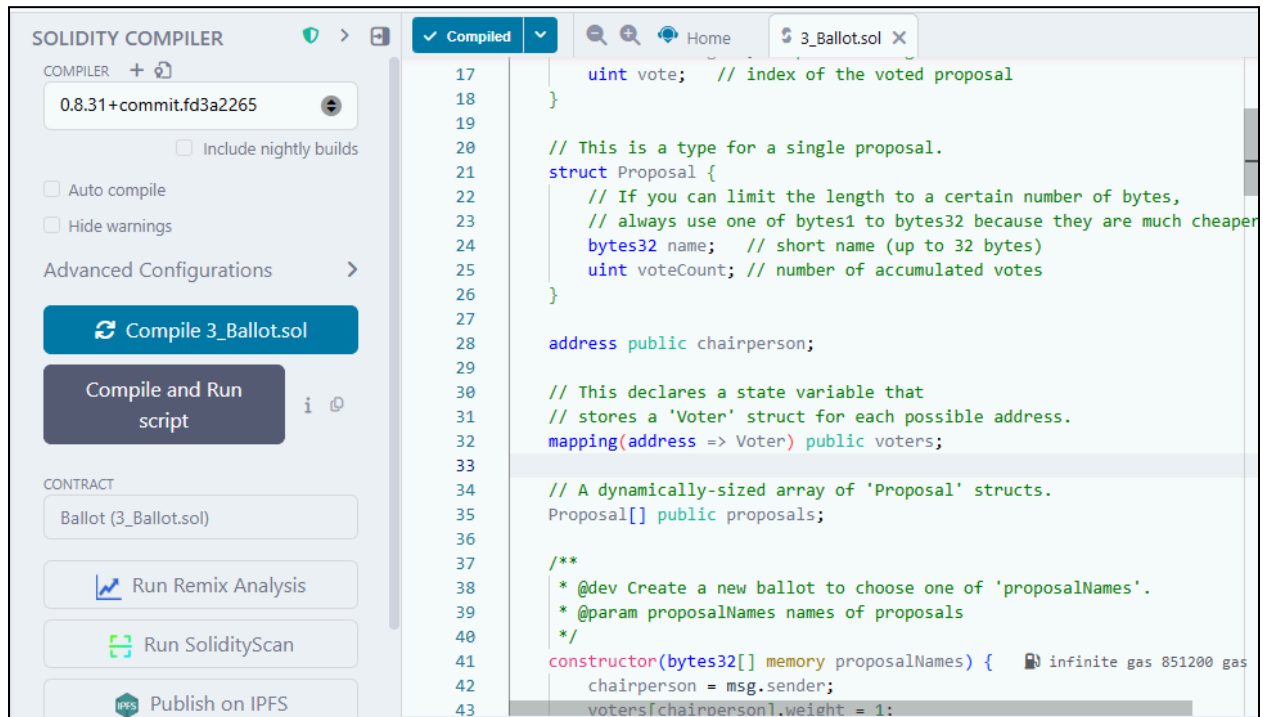
To make the system more user-friendly, modern implementations of the Ballot contract often convert fr

om 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.

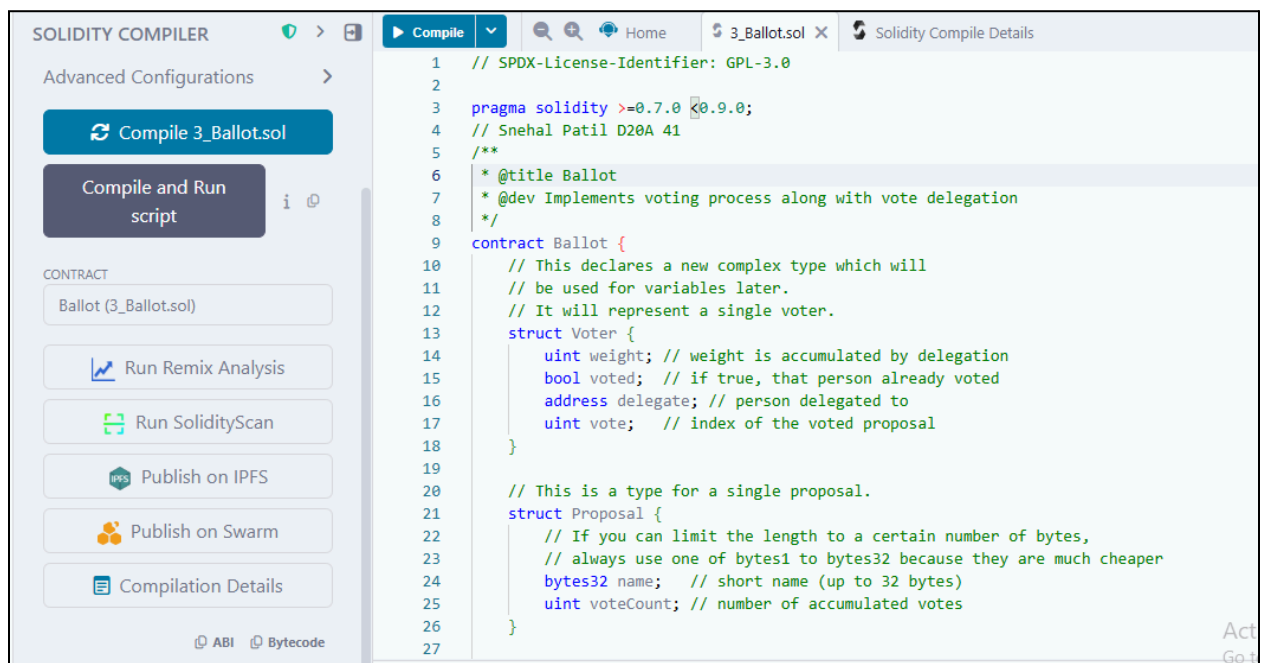
Code:

Compiled ballot



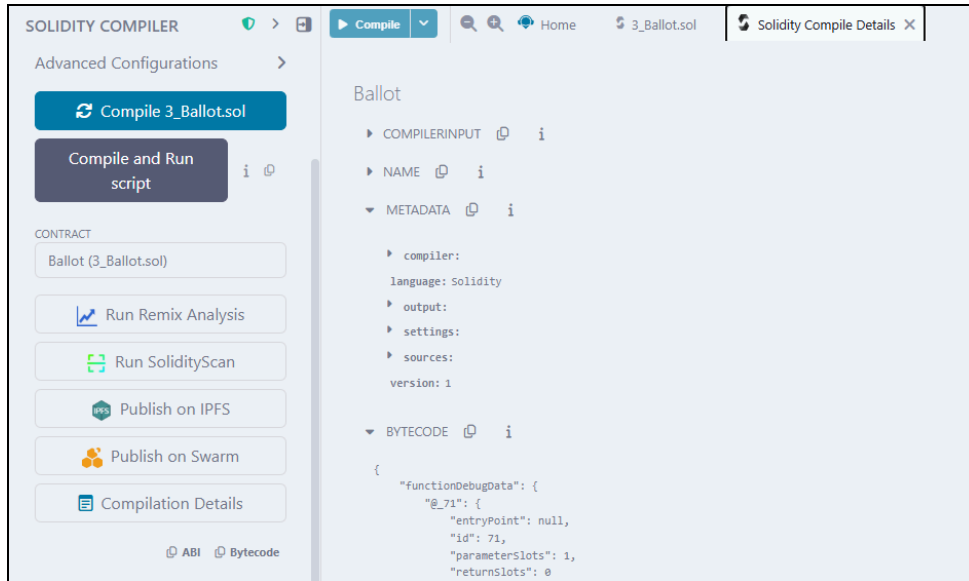
The screenshot shows the Solidity Compiler interface with the source code for `3_Ballot.sol`. The left sidebar contains the compiler version `0.8.31+commit.fd3a2265` and various configuration options. The main editor displays the following code:

```
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;
```

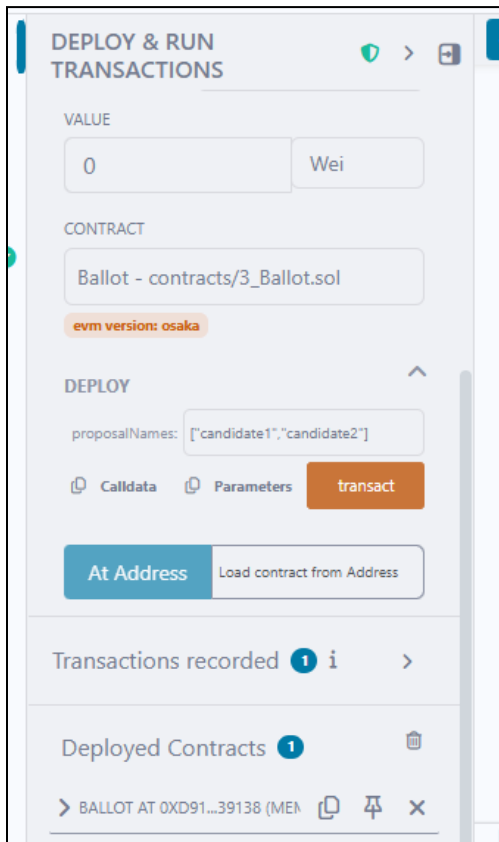


The screenshot shows the Solidity Compiler interface with the compiled code for `3_Ballot.sol`. The left sidebar contains the compiler version `0.8.31+commit.fd3a2265` and various configuration options. The main editor displays the following code:

```
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     }
27 }
```



- Loading the Proposal Candidate's Names (string)



Giving right to vote

The screenshot shows the Remix IDE interface. On the left, the 'Deployed Contracts' panel lists the 'BALLOT AT 0xD91...39138 (ME)' with a balance of 0 ETH. The 'giveRightToVote' function is selected, showing its parameters: 'to' (address) and 'value' (uint256). The main editor displays the Solidity code for the Ballot contract, with the 'giveRightToVote' function highlighted. The right panel shows the 'Explain contract' tab, which displays a list of transactions and VM logs. The transactions include the creation of the Ballot contract, a vote transaction, and the 'giveRightToVote' transaction. The logs show the state of the contract after each transaction, including the winner's name and the current proposal.

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

Voting casted by the candidate

The screenshot shows the Remix IDE interface. On the left, the 'Deployed Contracts' panel lists the 'BALLOT AT 0xD91...39138 (ME)' with a balance of 0 ETH. The 'vote' function is selected, showing its parameters: 'proposal' (uint256) and 'value' (uint256). The main editor displays the Solidity code for the Ballot contract, with the 'vote' function highlighted. The right panel shows the 'Explain contract' tab, which displays a list of transactions and VM logs. The transactions include the creation of the Ballot contract, a vote transaction, and the 'giveRightToVote' transaction. The logs show the state of the contract after each transaction, including the winner's name and the current proposal.

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

DEPLOY & RUN TRANSACTIONS

0 Wei

CONTRACT

Ballot - contracts/3_Ballot.sol

evm version: osaka

DEPLOY

proposalNames: ["candidate1","candidate2"]

Calldata Parameters **transact**

At Address Load contract from Address

Transactions recorded 1 i

☐ Run transactions using the latest compilation result

Save Run

Deployed Contracts 1

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 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.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: 0x583...eddC4 to: Ballot.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xfa3...57297 **Debug**

Activate Windows
Go to Settings to activate Windows.

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

DEPLOY & RUN TRANSACTIONS

Deployed Contracts 1

BALLOT AT 0XD91...39138 (ME)

Balance: 0 ETH

delegate address to

giveRightToVote 0xab8483f64d9c6d1ecf9b

vote 1

chairperson

proposals uint256

voters address

winnerName

0: string: winnerName_candidate2

winningProposal

Low level interactions i

Compiled

Home 3_Ballot.sol

```
54 require(to !== msg.sender, "Self-delegation not allowed");
55
56 while (voters[to].delegate !== address(0)) {
57   to = voters[to].delegate;
58   require(to !== msg.sender, "Delegation loop detected");
59 }
60
```

Explain contract

AI copilot

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

hash: 0xd91...b78aa

transact to Ballot.vote pending ...

[vm] from: 0xab8...35cb2 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0 hash: 0xd21...b78aa **Debug**

transact to Ballot.vote pending ...

[vm] from: 0xab8...35cb2 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0 hash: 0x8a3...230a0 **Debug**

transact to Ballot.vote errored: Error occurred: revert.

revert

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.

Activate Windows
Go to Settings to activate Windows.

Giving right to vote to another candidate

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel displays a deployed contract named 'BALLOT AT 0XD91...39138'. The 'vote' button is highlighted with a value of 0. The main editor shows Solidity code with a 'while' loop that checks if the sender is the chairperson before delegating the right to vote. The console on the right shows a transaction to 'Ballot.giveRightToVote' that failed with a revert error: 'Only chairperson can give right to vote'.

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

transaction to Ballot.giveRightToVote pending ...

[vm] from: 0x482...C02db to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x929...2fc43

transaction 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.

transaction to Ballot.giveRightToVote pending ...

[vm] from: 0x583...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x488...ddf3c

Second candidate casted vote to 1

The screenshot shows the Remix IDE interface after a successful transaction. The 'vote' button is now highlighted with a value of 1. The console on the right shows a transaction to 'Ballot.giveRightToVote' that succeeded. The main editor shows the same Solidity code as before.

```
55
56 while (voters[to].delegate != address(0)) {
57     to = voters[to].delegate;
58     require(to != msg.sender, "Delegation loop detected");
59 }
60 0x4B209938c481177ec7E8f571ceCaE8A9e22C02db
61
```

transaction 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.

transaction to Ballot.giveRightToVote pending ...

[vm] from: 0x583...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x488...ddf3c

transaction to Ballot.vote pending ...

[vm] from: 0x482...C02db to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0xcee...2422c

Giving right to vote third candidate

DEPLOY & RUN TRANSACTIONS

At Address Load contract from Address

Transactions recorded 0 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

3_Ballot.sol

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

Explain contract

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: 0x583...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x488...ddf3c Debug

transact to Ballot.vote pending ...

[vm] from: 0x482...C02db to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0xcee...2422c Debug

transact to Ballot.giveRightToVote pending ...

[vm] from: 0x583...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...cabab logs: 0
hash: 0xec5...1f6bb Debug

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 i

CALLDATA Transact

3_Ballot.sol

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

Explain contract

[vm] from: 0x583...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...c02db logs: 0
hash: 0x488...ddf3c Debug

transact to Ballot.vote pending ...

[vm] from: 0x482...C02db to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0
hash: 0xcee...2422c Debug

transact to Ballot.giveRightToVote pending ...

[vm] from: 0x583...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...cabab logs: 0
hash: 0xec5...1f6bb Debug

transact to Ballot.vote pending ...

[vm] from: 0x787...caba8 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
hash: 0x42f...e4973 Debug

Activate Windows
Go to Settings to activate Windows.

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

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel displays the contract state: `delegate` (address to), `giveRightToVote` (0x78731D3Ca6b7E34aC0F8), `vote` (0), `chairperson`, `proposals` (uint256), `voters` (address), `winnerName` (0: string: winnerName_candidate2), and `winningProposal` (0: uint256: winningProposal_1). The 'Transact' button is visible. The top editor shows the Solidity code for `3_Ballot.sol`, including a `while` loop for delegation and a `Voter storage` assignment. The bottom panel, 'Explain contract', shows a list of transactions:

- [vm] from: 0x583...eddC4 to: Ballot.giveRightToVote(address) 0xd91...39138 value: 0 wei data: 0x9e7...cabab logs: 0
- [vm] from: 0x787...caba8 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
- [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cC18A495caba8 to: Ballot.winningProposal() data: 0x609...ff1bd
- [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cC18A495caba8 to: Ballot.winnerName() data: 0xe2b...a53f0

Each transaction has a 'Debug' button. The status bar at the bottom indicates 'Activate Windows'.

The screenshot shows the Remix IDE interface with the same contract state as the first image. The 'Transact' button is visible. The top editor shows the Solidity code for `3_Ballot.sol`. The bottom panel, 'Explain contract', shows a list of transactions:

- [vm] from: 0x787...caba8 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
- [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cC18A495caba8 to: Ballot.winningProposal() data: 0x609...ff1bd
- [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cC18A495caba8 to: Ballot.winnerName() data: 0xe2b...a53f0

Each transaction has a 'Debug' button. The status bar at the bottom indicates 'Activate Windows'. The transaction log shows an error: `call to Ballot.voters errored: Error encoding arguments: TypeError: invalid address (argument="address", value="", code=INVALID_ARGUMENT, version=6.14.0) (argument=`.

- 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

- # DEPLOY & RUN TRANSACTIONS

give weight to vote

0x70731b52c8b07c348c010

vote

0

chairperson

proposals

1

0: string: name candidate2

1: uint256: voteCount 3

voters

0x4820993Bc481177ec7E8f

0: uint256: weight 1

1: bool: voted true

2: address: delegate 0x00

3: uint256: vote 1

winnerName

0: string: winnerName_candidate2

winningProposal

0: uint256: winningProposal_1

DEPLOY & RUN TRANSACTIONS

give weight to vote

0x70731b52c8b07c348c010

vote

0

chairperson

proposals

1

0: string: name candidate2

1: uint256: voteCount 3

voters

0x4820993Bc481177ec7E8f

0: uint256: weight 1

1: bool: voted true

2: address: delegate 0x00

3: uint256: vote 1

winnerName

0: string: winnerName_candidate2

winningProposal

0: uint256: winningProposal_1

- We can view the Voter's Information

The screenshot displays the Remix IDE interface during the execution of a smart contract. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is active, showing a list of transactions. The 'voters' transaction is selected, displaying its details: a uint256 weight of 1, a boolean 'voted true', an address for delegation, and a uint256 vote of 1. The main editor shows the Solidity code for the contract, with a 'while' loop for delegation and a 'Voter storage' assignment. The bottom right pane shows the 'Explain contract' section, which provides a detailed transaction log. The log includes a revert message, a note about gas limits, and call data for 'Ballot.proposals' and 'Ballot.voters'.

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.

