

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**.

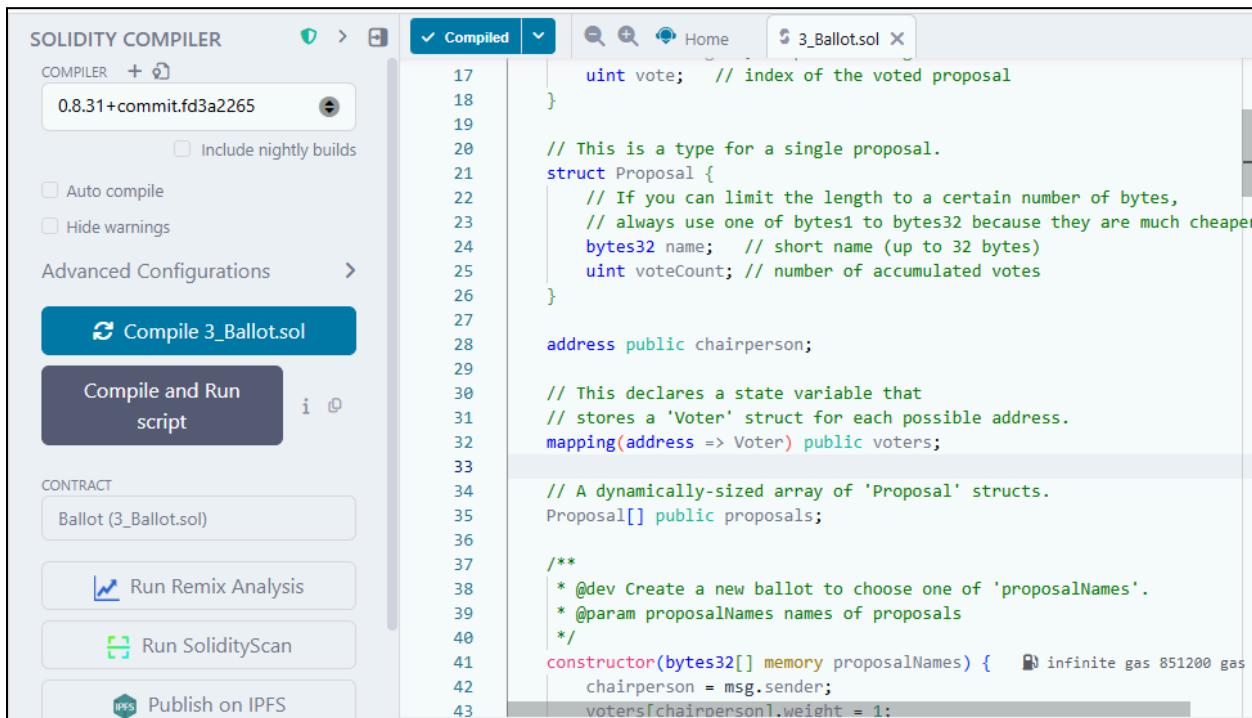
- **bytes32** is a **fixed-size type**, meaning it always stores exactly 32 bytes of data. This makes storage simple, comparison operations faster, and gas costs lower. However, it limits proposal names to 32 characters, which is not very flexible for user-friendly names.
- **string** is a **dynamically 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.

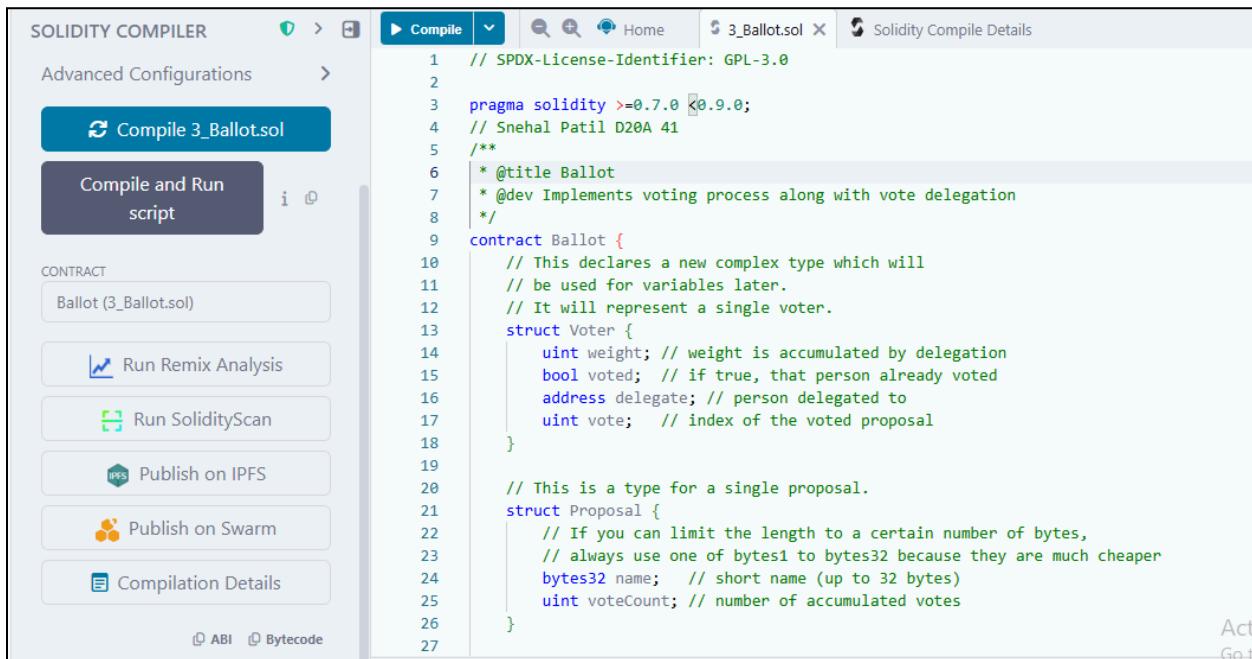
In summary, bytes32 is used when performance and gas efficiency are priorities, while string is preferred for readability and ease of use.

Code:

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 top bar says "DEPLOY & RUN TRANSACTIONS". The "VALUE" field is set to "0 Wei". Under "CONTRACT", "Ballot - contracts/3_Ballot.sol" is selected, with "evm version: osaka" noted. In the "DEPLOY" section, the "proposalNames" field contains the value "[candidate1, \"candidate2\"]". Below it, there are buttons for "Calldata", "Parameters", and a large orange "transact" button. A "At Address" button is also present. At the bottom, it shows "Transactions recorded" with one entry and "Deployed Contracts" with one entry: "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

The screenshot shows the Truffle UI interface for the same deployed contract BALLOT. The sidebar lists the same set of functions. The `vote` 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:

- 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 ...
- [vm] from: 0xAb8...35cb2 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0 hash: 0xd21...b78aa 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."

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), and 'DEPLOY' (proposalNames: [candidate1, candidate2]). 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'. The main area displays the Solidity code for the Ballot contract. On the right, a terminal window shows the output of the transaction, including logs and revert messages. A message at the bottom right says '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 after attempting to cast a vote again. The terminal output indicates a revert error: '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.' This occurs because the voter tried to delegate to themselves again.

Giving right to vote to another candidate

The screenshot shows the Truffle UI interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar lists a deployed contract 'BALLOT AT 0xd91...39138'. The main area displays the Solidity code for the Ballot contract. A transaction is being run with the 'giveRightToVote' function, passing the address '0x4B20993Bc481177ec7E8f' as the delegate. The transaction fails with a revert message: 'Only chairperson can give right to vote'. The UI also shows other functions like 'vote', 'chairperson', 'proposals', 'voters', and 'winnerName'.

```
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 }
```

Second candidate casted vote to 1

This screenshot shows the same setup as the previous one, but the 'vote' function is being called instead. The transaction fails with a revert message: 'Only chairperson can give right to vote'. The UI shows the same contract code and deployment details.

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

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 (IMBALLO) Balance: 0 ETH

delegate address to

giveRightToVote 0x78731D3Ca6b7E34aC0F8

vote 1

chairperson

proposals uint256

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

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.vote 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 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

The screenshot shows the Truffle UI interface. On the left, there's a sidebar titled "DEPLOY & RUN TRANSACTIONS" with fields for "delegate", "giveRightToVote", "vote", "chairperson", "proposals", "voters", "winnerName", and "winningProposal". Below this are sections for "Low level interactions" and "CALLDATA". On the right, the code editor displays the Solidity source code for "3_Ballot.sol". The transaction history pane shows the following sequence:

- [vm] from: 0x5B3...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] [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cc18A495cabab to: Ballot.winningProposal() data: 0x609...ff1bd
- [call] [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cc18A495cabab to: Ballot.winnerName() data: 0xe2b...a53f0

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

This screenshot is similar to the one above, showing the Truffle UI. The sidebar and code editor are identical. However, the transaction history pane shows a different sequence of events:

- [vm] from: 0x787...caba8 to: Ballot.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00000 logs: 0
- [call] [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cc18A495cabab to: Ballot.winningProposal() data: 0x609...ff1bd
- [call] [call] from: 0x78731D3Ca6b7E34aC0F824c42a7cc18A495cabab to: Ballot.winnerName() data: 0xe2b...a53f0
- call to Ballot.voters errored: Error encoding arguments: typeError: invalid address (argument="address", value="", code=INVALID_ARGUMENT, version=6.14.0) (argume

The last line shows an error message: "call to Ballot.voters errored: Error encoding arguments: typeError: invalid address (argument="address", value="", code=INVALID_ARGUMENT, version=6.14.0) (argume".

- 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

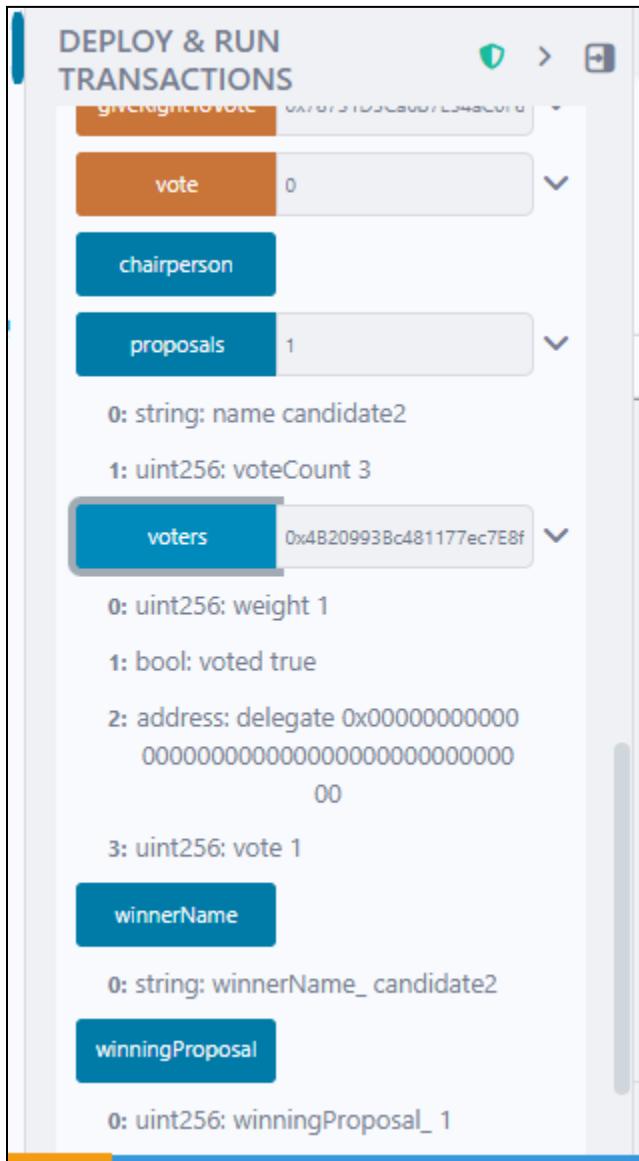
0: uint256: winningProposal_ 1

Low level interactions i

CALldata

Transact

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



- We can view the Voter's Information

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar displays various contract variables and their current values. On the right, the main area shows the Solidity code for the `3_Ballot.sol` contract, which includes a while loop for delegation and a voter storage assignment. Below the code, the 'Explain contract' section provides a detailed explanation of the transaction revert, noting that the transaction has been reverted to the initial state due to a gas limit issue. It also lists several calls made to the contract, such as `Ballot.proposals` and `Ballot.voters`.

```

Deploy & Run Transactions
  vote: 0
  chairperson
  proposals: 1

  voters
    voters - call
      0: uint256: weight 1
      1: bool: voted true
      2: address: delegate 0x0000000000000000000000000000000000
      3: uint256: vote 1

  winnerName
  winningProposal
    0: uint256: winningProposal_1

Contract: 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
    - 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: 0x5B38Da6a701c568545dCfcB03FcB8875f56beddC4 to: Ballot.proposals(uint256) data: 0x01
    call to Ballot.voters
    call [call] from: 0x5B38Da6a701c568545dCfcB03FcB8875f56beddC4 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.

