BLOCKCHAIN EXPERIMENT - 05
AAYUSH TALREJA (D17C/56)

AIM
Deploying a Voting/Ballot Smart Contract

THEORY
Why bytes32 instead of string?
In the context of blockchain and smart contracts, `bytes32` is often used instead of `string` for several important reasons:

- **Efficiency**: `bytes32` is a fixed-size data type, whereas strings can be of variable length. Fixed-size data types are more efficient to work with in a blockchain environment because they allow for predictable storage and gas costs. In contrast, variable-length strings would require additional gas for storage and processing to determine their length and handle potential resizing.
- **Deterministic Length**: Each `bytes32` variable is always 32 bytes in size, which means that the storage requirements for a `bytes32` variable are known in advance. This predictability is crucial in smart contract development, where precise control over gas costs and storage utilization is essential.
- **Gas Costs**: In Ethereum and other blockchain platforms, every operation performed in a smart contract consumes gas. Using `bytes32` reduces the gas costs compared to using strings because you don't need to perform length calculations or dynamic memory allocation, which can be expensive operations in a blockchain context.
- **Compatibility**: Some blockchain platforms, like Ethereum, have limited support for complex data types like strings. Using `bytes32` ensures compatibility with the platform's data storage and execution model.
- **Immutability**: Data stored in a `bytes32` variable is immutable, which means it cannot be modified once it's set. This property aligns with the blockchain's immutability principle, where once data is recorded on the blockchain, it should remain unchanged

However, it's important to note that using `bytes32` comes with limitations:
- **Encoding**: When using `bytes32` to store text, you need to consider encoding and decoding the text properly, as it is stored as raw bytes. This can add complexity to your smart contract code

CONCLUSION
In summary, the choice between `bytes32` and `string` depends on the specific requirements and constraints of your smart contract. If you need efficiency, determinism, and predictability in terms of gas costs and storage, `bytes32` is a suitable choice. However, if you require more flexibility and dynamic text operations, you might consider using `string`, understanding that it may come with higher gas costs and complexity.

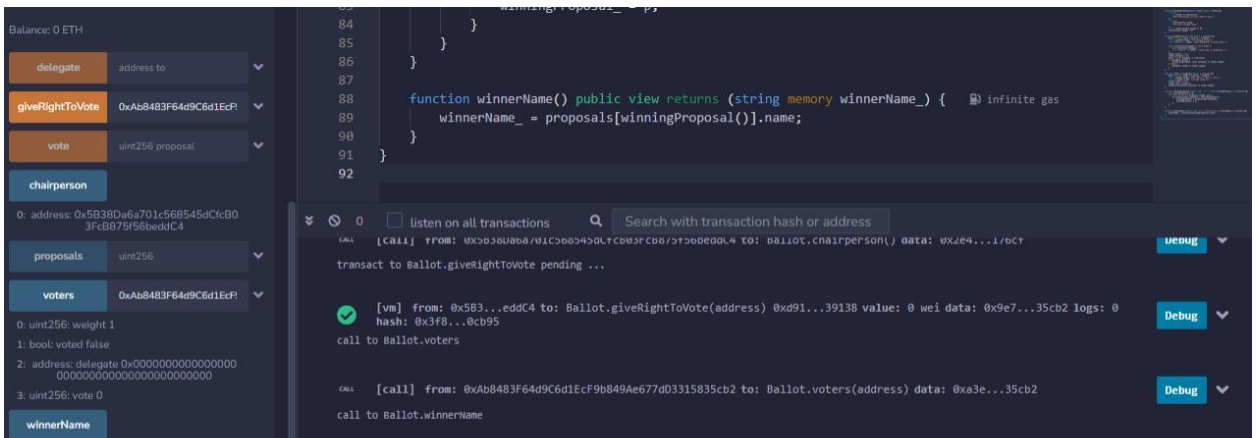
PROGRAM

```
//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;
    }

    struct Proposal {
        string name;
        uint voteCount;
    }

    address public chairperson;

    mapping(address => Voter) public voters;

    Proposal[] public 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 giveRightToVote(address voter) public {
    require(
        msg.sender == chairperson,
        "Only chairperson can give right to vote."
    );
    require(
        !voters[voter].voted,
        "The voter already voted."
    );
    require(voters[voter].weight == 0);
    voters[voter].weight = 1;
}

function delegate(address to) public {
    Voter storage sender = voters[msg.sender];
    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.");
    }
    sender.voted = true;
    sender.delegate = to;
    Voter storage delegate_ = voters[to];
    if (delegate_.voted) {
```

```
            proposals[delegate_.vote].voteCount += sender.weight;
        } else {
            delegate_.weight += sender.weight;
    }

    function vote(uint proposal) public {

        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;

            }

        }

    }

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

OUTPUT



```
```