Creating a Blockchain Voting System

Things to ensure :

- Correct assignment of the voting rights
- Automatic vote counting
- Transparent process

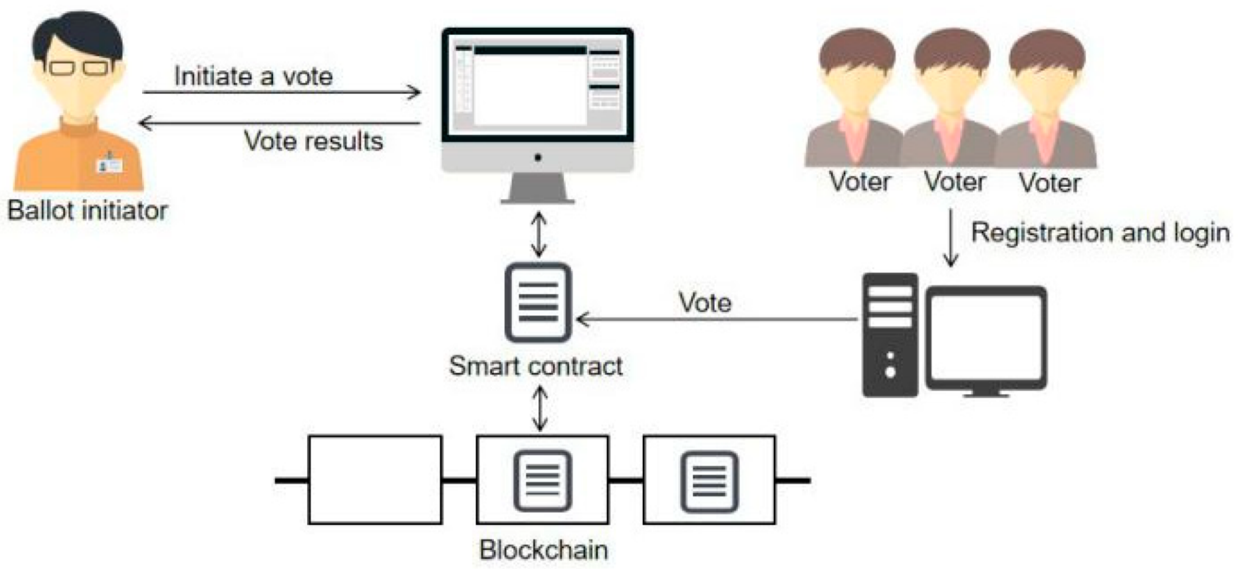
Requirements :

- create smart contract
- contract creator provides voting rights to each address, their owner can use or delegate their votes.
- When the election finishes, the proposal that received the most votes is returned.

Steps in coding :

- Declare the license identifier & version pragma
- Defining participants in voting system - struct voter & struct proposal here
- Defining functions and constructor if needed :
    - Giving the right to a voter to vote - done by chairperson/delegate
    - Ensure vote casting made only once by each voter
    - Maintaining an array of proposals & pushing each proposal names to that array
    - Winning proposal name or winner is found
    - Returning winner proposal name - publishing the winner

CONCEPTUAL FIGURE (**voter login isn't included in our smart contract)



```
//SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.7.0;

contract Voting {
```

---

write the struct to represent a single vote holder. It holds the information whether the person voted, and (if they did) which option they chose.

If they trusted someone else with their ballot, you can see the new voter in the `address` line. Delegation also makes `weight` accumulate:

```
struct Voter {
        uint weight;
        bool if_voted;
        address delegated_to;
        uint vote;
   }
```

---

struct represent a single proposal. Its name can’t be bigger than 32 bytes.

`voteCount` - how many votes the proposal has received:

```
struct Proposal {
        bytes32 name;
        uint voteCount;
    }

    address public chairperson;
```

`Mapping` - we declare a state variable necessary for assigning the blockchain voting rights. It stores the struct we created to represent the voter in each address required:

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

include a dynamically-sized structs array for proposal

```
Proposal[] public proposals;
```

---

create a new ballot to choose one of the proposals. Each new proposal name requires creating a new proposal object and placing it at the end of array:

```
constructor(bytes32[] memory proposalNames) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

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

---

provide the voter with a right to use their ballot. Only the chairperson can call this function.

If the first argument of `require` returns `false`, function stops executing. All the changes it has made are cancelled too.

`Require`- check if the function calls execute as they should. If they don't, you may include an explanation in the second argument:

```
function giveRightToVote(address voter) public {
    require(
        msg.sender == chairperson,
        "Only the chairperson can assign voting rights."
    );
    require(
        !voters[voter].voted,
        "The voter has used their ballot."
    );
    require(voters[voter].weight == 0);
    voters[voter].weight = 1;
}
```

---

write a function to delegate the vote.

`to` represent an address to which the voting right goes.

If they delegate as well, the delegation is forwarded.

add a message to inform about a located loop. Those can cause issues if they run for a long time. In such case, a contract might get stuck, as delegation may need more gas than a block has available:

```
function delegate(address to) public {
        Voter storage sender = voters[msg.sender];
        require(!sender.voted, "You have already voted.");

        require(to != msg.sender, "You can’t delegate to yourself.");

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

`sender` - a reference that affects `voters[msg.sender].voted`.

If the delegate has used their ballot, it adds to the total number of received votes.

If they haven’t, their weight increases:

```
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;
        }
    }
```

`vote` allows giving your vote to a certain proposal, along with any votes you may have been delegated:

```
function vote(uint proposal) public {
        Voter storage sender = voters[msg.sender];
        require(sender.weight != 0, "Cannot vote");
        require(!sender.voted, "Has voted.");
        sender.voted = true;
        sender.vote = proposal;
        proposals[proposal].voteCount += sender.weight;
    }
```

Note: if the proposal you name is not accessible by the array, the function will revert all changes.

`winningProposal()` - count the votes received & choose the winner
`winnerName()` - To get their index and return a name

```
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 (bytes32 winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }}
```

VOTING SMART CONTRACT

```
//SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.7.0;

contract Voting {
struct Voter {
        uint weight;
        bool if_voted;
        address delegated_to;
        uint vote;
    }
struct Proposal {
        bytes32 name;
        uint voteCount;
    }

    address public chairperson;
mapping(address => Voter) public voters;
Proposal[] public proposals;
constructor(bytes32[] memory proposalNames) public {
        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 the chairperson can assign voting rights."
        );
        require(
            !voters[voter].voted,
            "The voter has used their ballot."
        );
        require(voters[voter].weight == 0);
        voters[voter].weight = 1;
    }
function delegate(address to) public {
        Voter storage sender = voters[msg.sender];
        require(!sender.voted, "You have already voted.");

        require(to != msg.sender, "You can’t delegate to yourself.");

        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, "Cannot vote");
        require(!sender.voted, "Has 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 (bytes32 winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```