PRACTICAL NO. 7

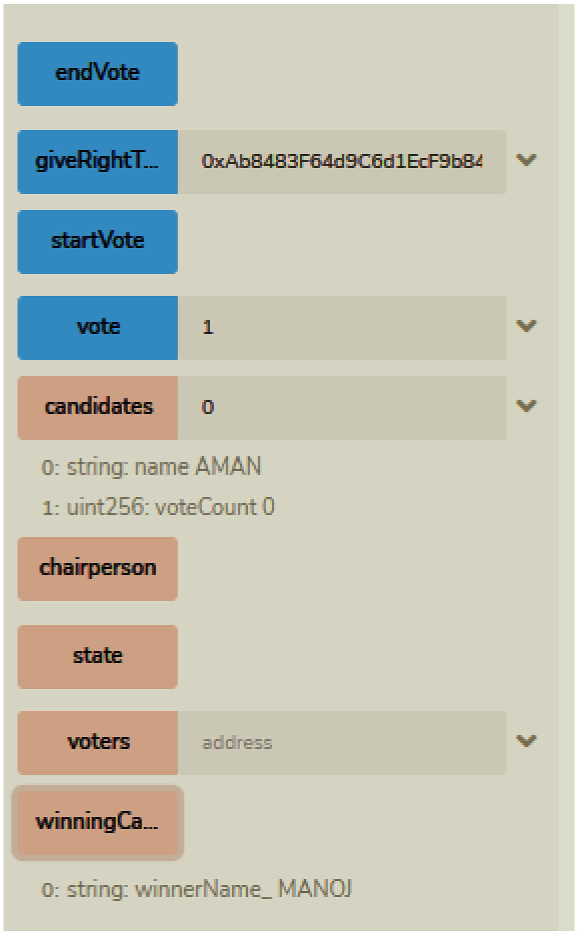
CODE:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.13;

pragma abicoder v2;

contract Ballot {

struct Voter {

uint weight; // weight is accumulated by delegation

bool voted; // if true, that person already voted

uint vote; // index of the voted proposal

}

struct Candidate {

string name; // candidate name

uint voteCount; // number of accumulated votes

}

address public chairperson;

mapping(address => Voter) public voters;

Candidate[] public candidates;

enum State { Created, Voting, Ended } // State of voting period

State public state;

constructor(string[] memory candidateNames) {

chairperson = msg.sender;

voters[chairperson].weight = 1;

state = State.Created;

for (uint i = 0; i < candidateNames.length; i++) {

candidates.push(Candidate({
```

```
name: candidateNames[i],

voteCount: 0

}));

}

}

// MODIFIERS

modifier onlySmartContractOwner() {

require(

msg.sender == chairperson,

"Only chairperson can start and end the voting"

);

_;

}

modifier CreatedState() {

require(state == State.Created, "it must be in Started");

_;

}

modifier VotingState() {

require(state == State.Voting, "it must be in Voting Period");

_;

}

modifier EndedState() {

require(state == State.Ended, "it must be in Ended Period");

_;

}

function startVote() public onlySmartContractOwner CreatedState
```

```
{

state = State.Voting;

}

/*

* to end the voting period

* can only end if the state in Voting period

*/

function endVote() public onlySmartContractOwner VotingState

{

state = State.Ended;

}

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 vote(uint candidate) public VotingState

{

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 = candidate;

// If 'candidate' is out of the range of the array,

// this will throw automatically and revert all

// changes.

candidates[candidate].voteCount += sender.weight;

}

function winningCandidate() public EndedState view returns (string memory winnerName_)

{

uint winningVoteCount = 0;

for (uint p = 0; p < candidates.length; p++) {

if (candidates[p].voteCount > winningVoteCount) {

winningVoteCount = candidates[p].voteCount;

winnerName_ = candidates[p].name;

}

}

}

}
```

OUTPUT:



```
0: string: name AMAN
1: uint256: voteCount 0
```

```
0: string: winnerName_ MANOJ
```