



139 lines - 119 Removals

```
1 // SPDX-License-Identifier: GPL
  -3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 /**
6  * @title Ballot
7  * @dev Implements voting process along with vote delegation
8  */
9 contract Ballot {
10
11     address public chairperson;
12
13     struct Voter {
14         uint weight; // weight
15         // is accumulated by delegation
16         bool voted; // if true, that person already voted
17         // address delegate; // person delegated to
18         uint vote; // index of the voted proposals
19     }
20
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
```

116 lines + 96 Additions

```
1 // SPDX-License-Identifier: GPL
  -3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 import "hardhat/console.sol";
6
7 contract Ballot {
8
9     // Authority address (who deploys the contract)
10     address authority;
11
12     struct Voter {
13         string name; // Voter name
14         bool voted; // if true, that person already voted
15         uint vote; // index of the voted proposal
16         bool rightGranted; // if true, voter can use verification2
17     }
18
19     struct Proposal {
```

```

23     string name; // short
    name (up to 32 bytes)
24     uint voteCount; // numb
    er of accumulated votes
25 }
26
27     mapping(address => Voter) p
    ublic voters;
28
29     Proposal[] public proposal
    s;
30
31     /**
32     * @dev Create a new ballot
    to choose one of 'proposalName
    s'.
33     * @param proposalNames nam
    es of proposals
34     */
35     constructor(string[] memory
    proposalNames) {
36         chairperson = msg.sende
    r;
37         voters[chairperson].wei
    ght = 1;
38
39         for (uint i = 0; i < pr
    oposalNames.length; i++) {
40             // 'Proposal
    ({...})' creates a temporary
41             // Proposal object
    and 'proposals.push(...)'
42             // appends it to th
    e end of 'proposals'.
43             proposals.push(Prop
    osal({
44                 name: proposalN
    ames[i],
45                 voteCount: 0
46             }));
47         }
48     }
49

```

```

20     string name; // short
    name (up to 32 bytes)
21     uint voteCount; // numb
    er of accumulated votes
22 }
23
24     mapping(address => Voter) v
    oters;
25
26     Proposal[] public proposal
    s;
27
28     constructor() {
29         authority = msg.sender;
30
31         //Appends two proposals
    to Proposal
32         proposals.push(Proposal
    ({ name: "Joe Biden", voteCoun
    t: 0 }));
33         proposals.push(Proposal
    ({ name: "Donald Trump", voteCo
    unt: 0 }));
34     }
35

```

```

50     /**
51     * @dev Give 'voter' the right to vote on this ballot. May only be called by 'chairperson'.
52     * @param voter address of voter
53     */
54     function giveRightToVote(address voter) public {
55         require(
56             msg.sender == chairperson,
57             "Only chairperson can give right to vote."
58         );
59         require(
60             !voters[voter].voted,
61             "The voter already voted."
62         );
63         require(voters[voter].weight == 0);
64         voters[voter].weight = 1;
65     }
66
67     /**
68     * @dev Delegate your vote to the voter 'to'.
69     * @param to address to which vote is delegated
70

```

```

36     //Function to change name of voter
37     function changeName(string memory _name, address _address) public{
38         //Requirements
39         require(msg.sender != authority, "Authority can not vote");
40         require(msg.sender == _address, "Has no right to change name");
41
42         //Change name
43         voters[_address].name = _name;
44     }
45
46     //Function to give right to use function verification2
47     function giveRightToVerification2(address _address) public {
48         //Requirements
49         require( msg.sender == authority, "Only authority can give right to watch" );
50
51         //Grant permission
52         voters[_address].rightGranted = true;
53     }
54
55     //Function to log out
56     function logOut() public{
57         voters[msg.sender].rightGranted = false;
58     }

```

```

71     function delegate(address to) public {
72         Voter storage sender =
voters[msg.sender];
73         require(!sender.voted,
"You already voted.");
74         require(to != msg.sender, "Self-delegation is disallowed.");
75
76         while (voters[to].delegate != address(0)) {
77             to = voters[to].delegate;
78
79             // We found a loop
in the delegation, not allowed.
80             require(to != msg.sender, "Found loop in delegation.");
81         }
82         sender.voted = true;
83         sender.delegate = to;
84         Voter storage delegate_
= voters[to];
85         if (delegate_.voted) {
86             // If the delegate
already voted,

```

```

59
60     //Function to vote
61     function vote(uint _proposal) public {
62         Voter storage sender =
voters[msg.sender];
63         //Requirements
64         require(msg.sender != authority, "Authority can not vote");
65         require(bytes(sender.name).length != 0, "Please change your name first");
66         require(!sender.voted, "Already voted");
67
68         //Add vote
69         sender.voted = true;
70         sender.vote = _proposal;
71         proposals[_proposal].voteCount += 1;
72     }
73
74     //Function to use Type 1 Verification
75     function verification1(address _address, uint _token) public view returns(bool voted_){
76         //Requirements
77         require(msg.sender == _address, "Has no right to verify");
78         require(_token == 123456, "Invalid token");
79
80         //Shows if user voted
81         voted_ = voters[_address].voted;

```

```

87         // directly add to
the number of votes
88         proposals[delegate
_.vote].voteCount += sender.wei
ght;
89     } else {
90         // If the delegate
did not vote yet,
91         // add to her weigh
t.
92         delegate_.weight +=
sender.weight;
93     }
94 }
95 */
96
97 /**
98  * @dev Give your vote (inc
luding votes delegated to you)
to proposal 'proposals[proposa
l].name'.
99  * @param proposal index of
proposal in the proposals array
100 */
101
102 function vote(uint proposa
l) public {
103     Voter storage sender =
voters[msg.sender];
104
105     require(sender.weight !
= 0, "Has no right to vote");
106     require(!sender.voted,
"Already voted.");
107     sender.voted = true;
108     sender.vote = proposal;
109
110     // If 'proposal' is out
of the range of the array,
111     // this will throw auto
matically and revert all

```

```

82 }
83
84 //Function to use Type 2 Ve
rification
85 function verification2(addr
ess _address, uint _token) publ
ic view returns(string memory n
ame_, bool voted_, string memor
y vote_){
86     //Requirements
87
88     require(msg.sender == _
address, "Has no right to verif
y");
89     require(_token == 12345
6 , "Invalid token");
90     require(voters[_addres
s].rightGranted == true, "Has n
o right granted");
91
92     //Shows data voter
93
94     name_ = voters[_addres
s].name;

```

```

110         // changes.
111         proposals[proposal].voteCount += sender.weight;
112     }
113
114     /**
115      * @dev Computes the winning proposal taking all previous votes into account.
116      * @return winningProposal_ index of winning proposal in the proposals array
117      */
118     function winningProposal() public view
119         returns (uint winningProposal_)
120     {
121         uint winningVoteCount = 0;
122         for (uint p = 0; p < proposals.length; p++) {
123             if (proposals[p].voteCount > winningVoteCount) {
124                 winningVoteCount = proposals[p].voteCount;
125                 winningProposal_ = p;
126             }
127         }

```

```

93         voted_ = voters[_address].voted;
94         if (voted_ != false) {
95             vote_ = proposals[voters[_address].vote].name;
96         }
97     }
98
99     //Function to call winning proposal
100     function winningProposal() public view returns (uint winningProposal_, string memory winnerName_) {
101         //Winning proposal
102         uint winningVoteCount = 0;
103         for (uint p = 0; p < proposals.length; p++) {
104             if (proposals[p].voteCount > winningVoteCount) {
105                 winningVoteCount = proposals[p].voteCount;
106                 winningProposal_ = p;
107             }
108         }
109         if (winningProposal_ == 0) {
110             winnerName_ = "NULL";
111         } else {
112             winnerName_ = proposals[winningProposal_].name;
113         }

```

```
128     }
129
130     /**
131     * @dev Calls winningPropos
al() function to get the index
of the winner contained in the
proposals array and then
132     * @return winnerName_ the
name of the winner
133     */
134     function winnerName() publi
c view
135         returns (string mem
ory winnerName_)
136     {
137         winnerName_ = proposals
[winningProposal()].name;
138     }
139 }
```

```
114     }
115
116 }
```