FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING
Department of Computer Engineering

1. Course , Subject & Experiment Details

Academic Year	**2022-23**	**Estimated Time**	**02 - Hours**
Course & Semester	**B.E. (CMPN)- Sem VII**	**Subject Name & Code**	**BCT - (CSDC7022)**
Chapter No.	**03**	**Chapter Title**	**Programming for Blockchain**

Practical No:	**5**
Title:	Voting Application using Solidity
Date of Performance:	**05/09/2022**
Date of Submission:	**12/09/2022**
Roll No:	**8953**
Name of the Student:	**Brendan Lucas**

Evaluation:

Sr. No	Rubric	Grade
1	On time submission Or completion (2)	
2	Preparedness(2)	
3	Skill (4)	
4	Output (2)	

Signature of the Teacher:

Date:

Code:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

/**
 * @title Ballot
 * @dev Implements voting process along with vote delegation
 */
contract Ballot {

    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted;  // if true, that person already voted
        address delegate; // person delegated to
        uint vote;   // index of the voted proposal
    }

    struct Proposal {
        // If you can limit the length to a certain number of bytes,
        // always use one of bytes1 to bytes32 because they are much cheaper
        bytes32 name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }

    address public chairperson;

    mapping(address => Voter) public voters;

    Proposal[] public proposals;

    /**
     * @dev Create a new ballot to choose one of 'proposalNames'.
     * @param proposalNames names of proposals
     */
    constructor(bytes32[] memory proposalNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

        for (uint i = 0; i < proposalNames.length; i++) {
            // 'Proposal({...})' creates a temporary
```

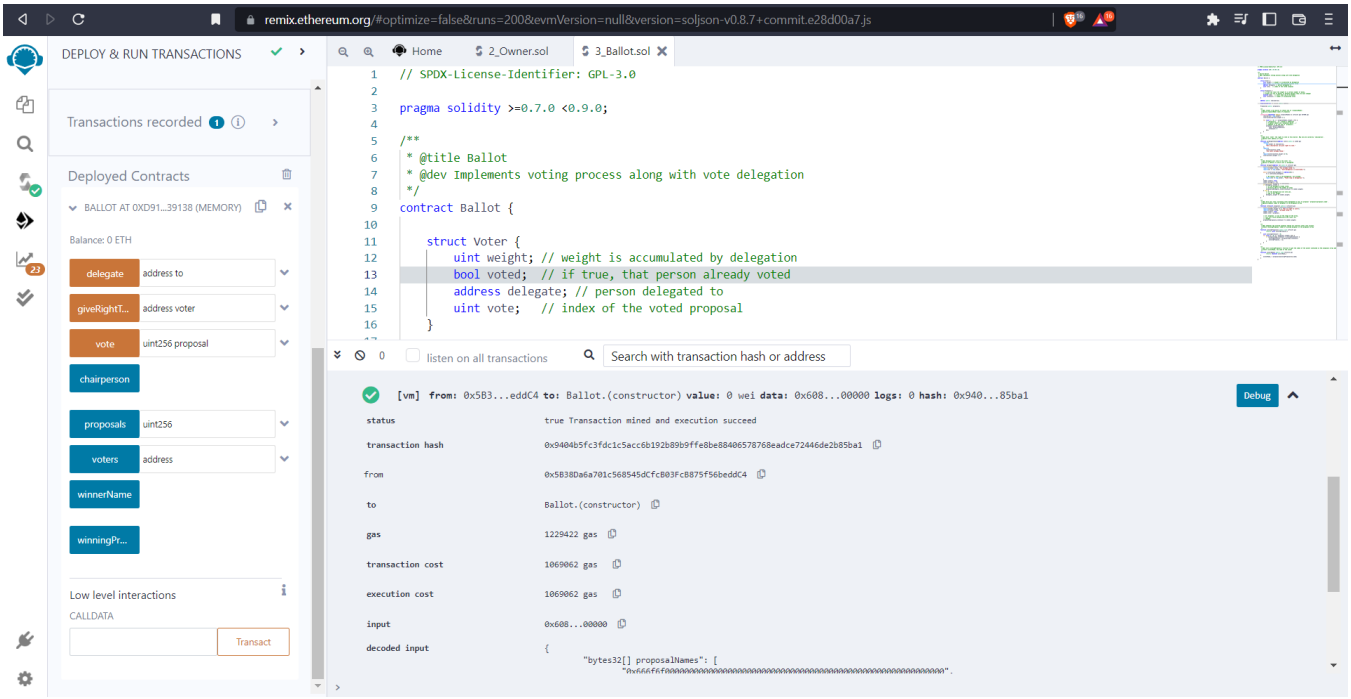
```
            // Proposal object and 'proposals.push(...)'
            // appends it to the end of 'proposals'.
            proposals.push(Proposal({
                name: proposalNames[i],
                voteCount: 0
            }));
        }
    }

    /**
     * @dev Give 'voter' the right to vote on this ballot. May only be called by 'chairperson'.
     * @param voter address of voter
     */
    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;
    }

    /**
     * @dev Delegate your vote to the voter 'to'.
     * @param to address to which vote is delegated
     */
    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;

            // We found a loop in the delegation, not allowed.
            require(to != msg.sender, "Found loop in delegation.");
```

```
        }
        sender.voted = true;
        sender.delegate = to;
        Voter storage delegate_ = voters[to];
        if (delegate_.voted) {
            // If the delegate already voted,
            // directly add to the number of votes
            proposals[delegate_.vote].voteCount += sender.weight;
        } else {
            // If the delegate did not vote yet,
            // add to her weight.
            delegate_.weight += sender.weight;
        }
    }

    /**
     * @dev Give your vote (including votes delegated to you) to proposal
'proposals[proposal].name'.
     * @param proposal index of proposal in the proposals array
     */
    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;

        // If 'proposal' is out of the range of the array,
        // this will throw automatically and revert all
        // changes.
        proposals[proposal].voteCount += sender.weight;
    }

    /**
     * @dev Computes the winning proposal taking all previous votes into
account.
     * @return winningProposal_ index of winning proposal in the proposals
array
     */
    function winningProposal() public view
            returns (uint winningProposal_)
```

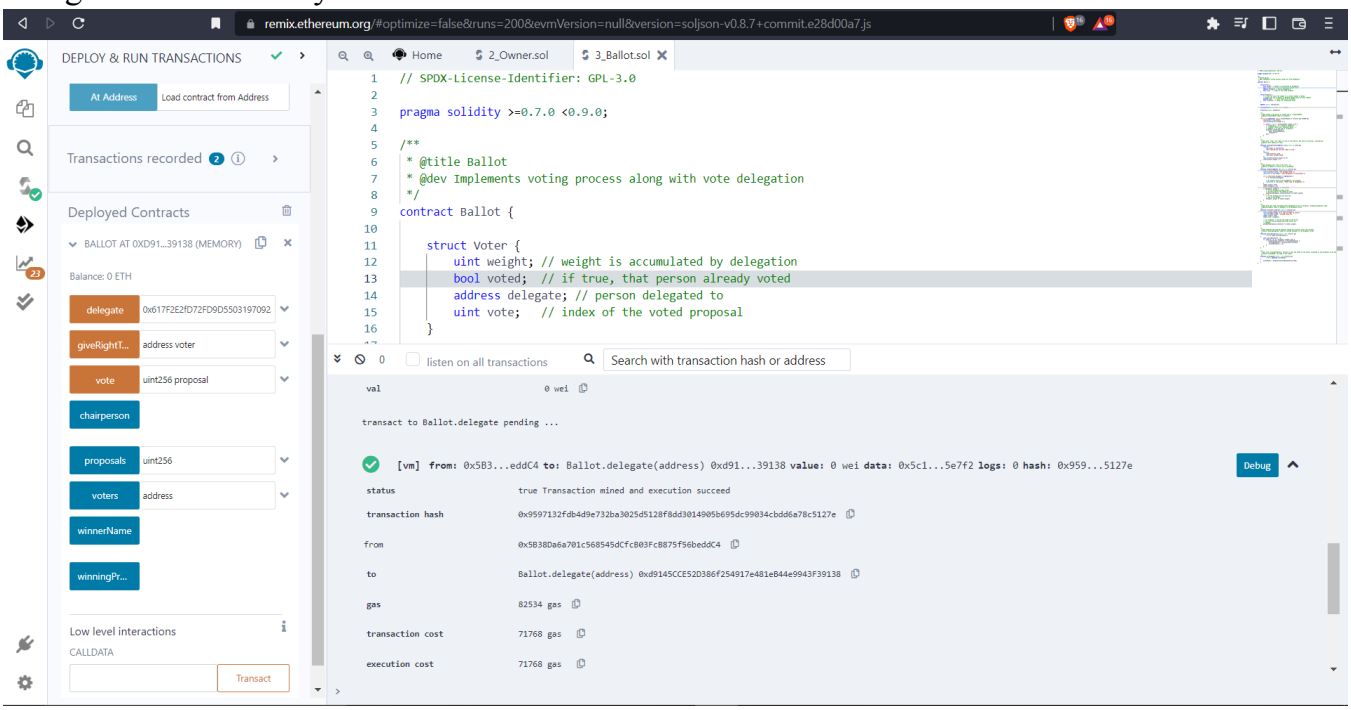
```
    {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
    }

    /**
     * @dev Calls winningProposal() function to get the index of the winner
contained in the proposals array and then
     * @return winnerName_ the name of the winner
     */
    function winnerName() public view
            returns (bytes32 winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```

Output:

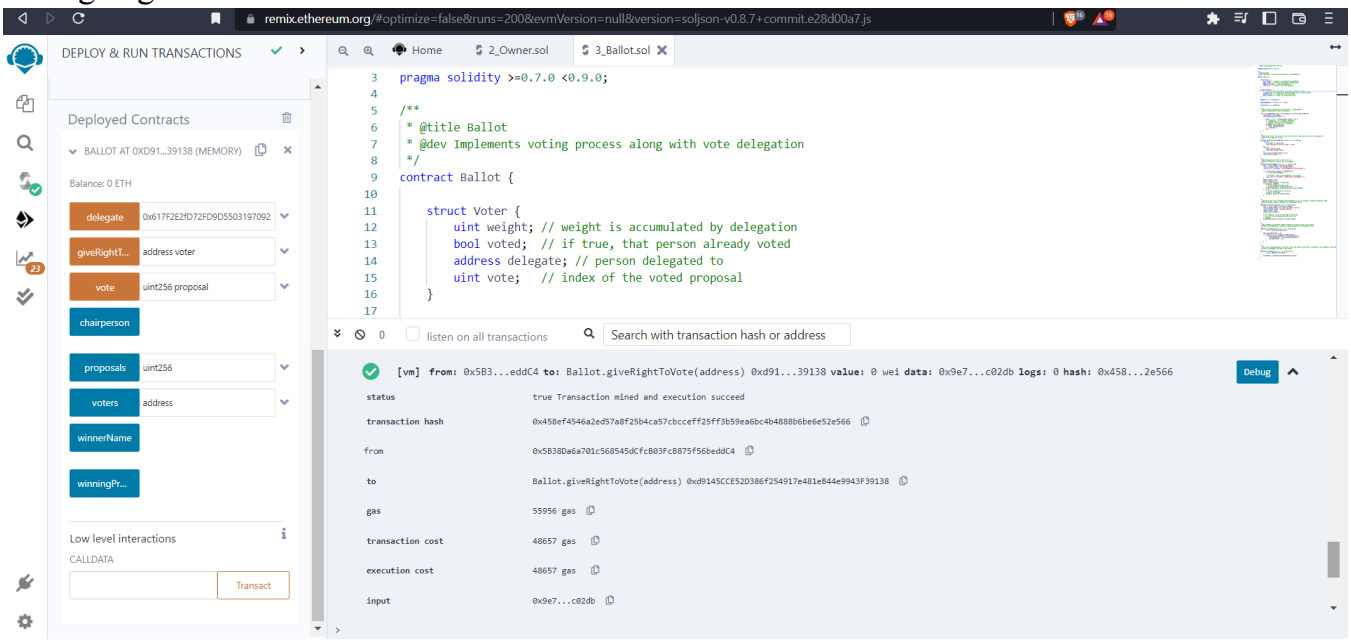
Contract Creation:



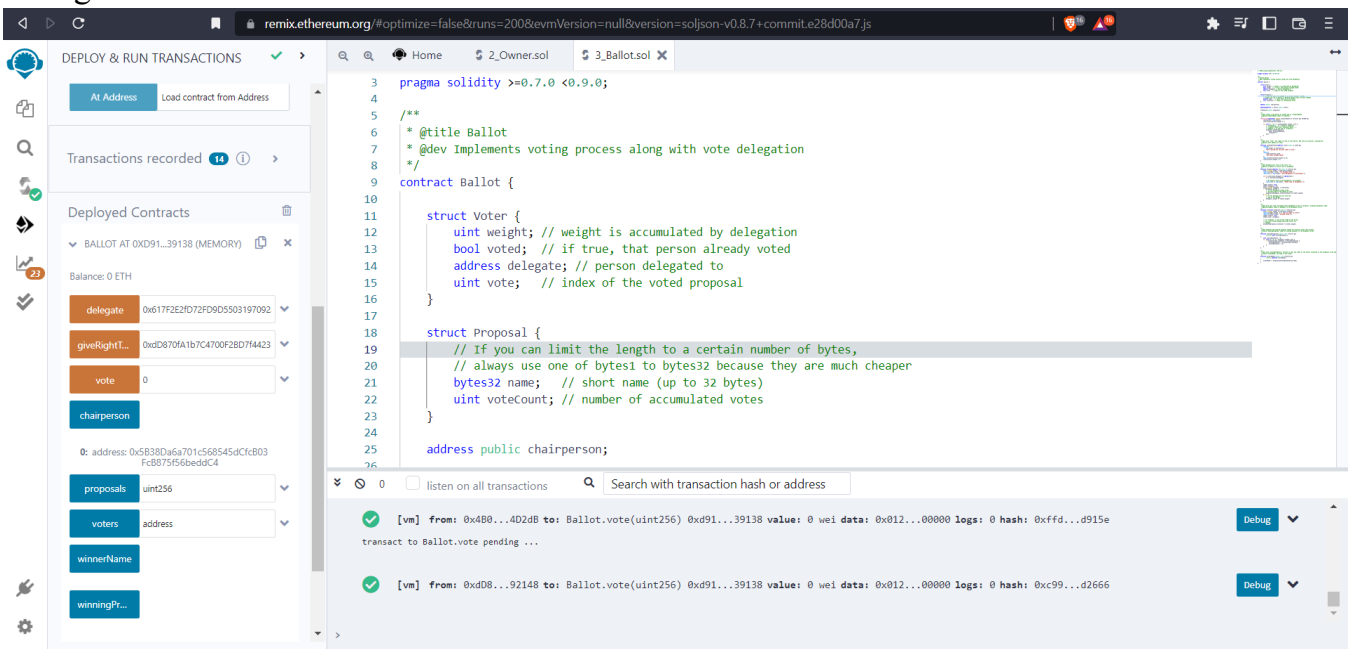
Delegation of Authority:



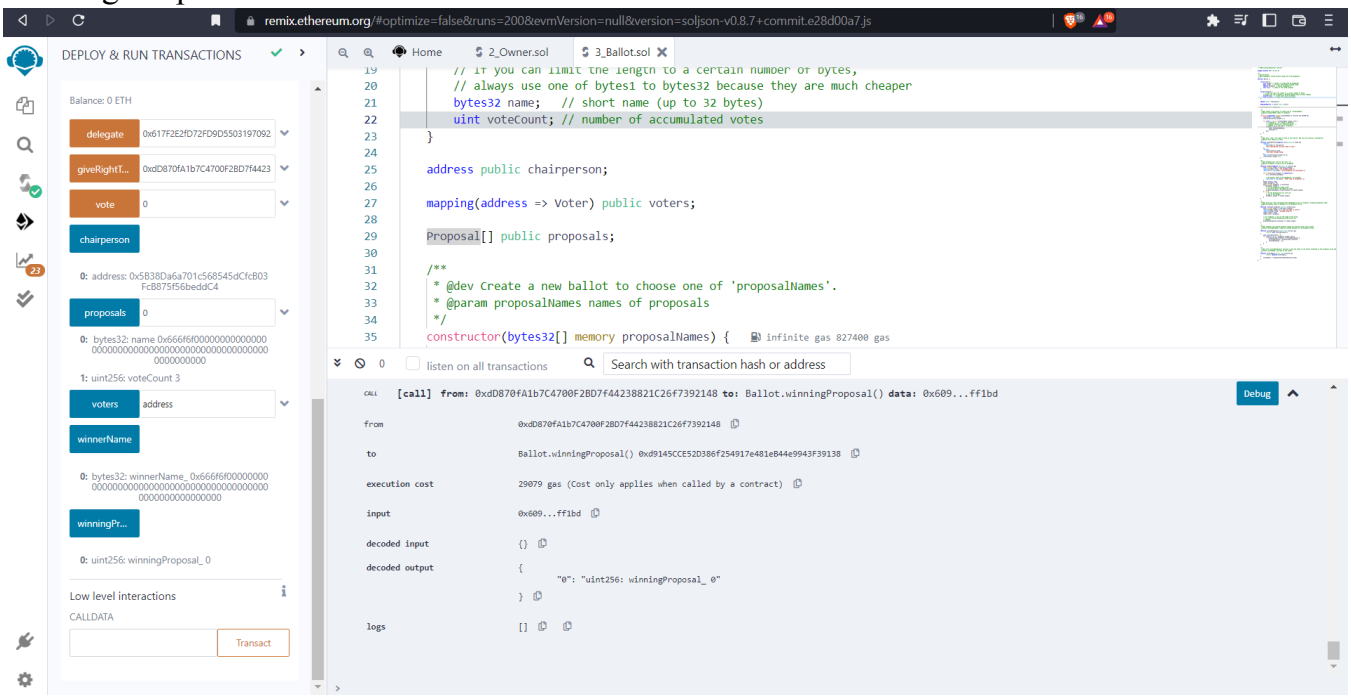
Giving Right to Vote:



Voting:



Winning Proposal:



Winning Proposal Name: