```
//Q1. Blockchain assignment
pragma solidity >=0.7.0 <0.9.0;
contract Ballot {
  /* mapping field below is equivalent to an associative array or hash.
  The key of the mapping is candidate name stored as type bytes32 and
value is
  an unsigned integer to store the vote count
  */
  struct voter
  {
    bool voted;
  }
  mapping (string => uint256) private votesReceived;
  address public chairperson;

  /* Solidity doesn't let you pass in an array of strings in the
constructor (yet).
  We will use an array of bytes32 instead to store the list of candidates
  */

  string[] private candidateList;

/* This is the constructor which will be called once when you
  deploy the contract to the blockchain. When we deploy the contract,
  we will pass an array of candidates who will be contesting in the
election
  */
  constructor(string[] memory candidateNames) {
      chairperson = msg.sender;
      candidateList = candidateNames;
  }

  /* To check chairperson is calling*/
  modifier onlyChair() {
        require(msg.sender == chairperson, 'Only chairperson');
        _;
    }

    function addcanditate(string memory name) public onlyChair{
    candidateList.push(name);
  }
  function checkvoted(string memory sender)
  {
    Voter storage sender = voters[msg.sender];
        require(!sender.voted, "Already voted.");
        sender.voted = true;
  }

  // This function returns the total votes a candidate has received so far
  function totalVotesFor(string memory candidate) view public onlyChair
returns (uint256)  {

    require(validCandidate(candidate));
    return votesReceived[candidate];
  }

  // This function increments the vote count for the specified candidate.
This
  // is equivalent to casting a vote
  function castVote(string memory candidate) public {
    require(validCandidate(candidate));
    votesReceived[candidate] += 1;
  }

  function validCandidate(string memory candidate) view private returns
(bool) {
    for(uint i = 0; i < candidateList.length; i++) {
      if (keccak256(bytes(candidateList[i])) ==
keccak256(bytes(candidate))) {
        return true;
      }
    }
    return false;
  }
  function stopvote() public onlyChair
  {
    require(voter.voted,"Voting stopped due to repeated attempts")
    require(validCandidate(candidate),"The candidate is not valid. The
voting is stopped")
  }
```

```
}
```

```
//Q2. Blockchain assignment
pragma solidity ^0.5.8;

contract bank
{
     mapping (address => uint) private bal;
     address public owner;
     event LogDepositMade(address indexed accountAddress, uint amount);
     uint8 private cc;

     constructor() public payable {
        require(msg.value == 30 ether, "insufficient funds");
        owner = msg.sender;
        cc = 0;
    }
     function getbal() public view returns (uint)
     {
        return bal[msg.sender];
    }
    function deposit(uint8)
    {
        bal[msg.sender] += msg.value;
        emit LogDepositMade(msg.sender, msg.value);
        return bal[msg.sender];
    }
    function transfer(address receiver, uint256 numTokens) public  returns
(bool)
    {
        require(numTokens <= bal[msg.sender]);
        bal[msg.sender] = bal[msg.sender]-numTokens;
        bal[receiver] = bal[receiver]+numTokens;
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }
    function reg() public returns (uint) {
        if (cc < 3) {
            cc=cc+1;
            bal[msg.sender] = 10 ether;
        }
        return bal[msg.sender];
    }
    function withdraw(uint amt) public returns (uint avlb) {

        if (amt <= balances[msg.sender]) {
            balances[msg.sender] -= amt;
            msg.sender.transfer(amt);
        }
        return bal[msg.sender];
    }


}
```