PROGRAM:

```
var Election = artifacts.require("./Election.sol");


contract("Election", function(accounts) {
  var electionInstance;


  it("initializes with two candidates", function() {
    return Election.deployed().then(function(instance) {
      return instance.candidatesCount();
    }).then(function(count) {
      assert.equal(count, 2);
    });
  });


  it("it initializes the candidates with the correct values",
function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      return electionInstance.candidates(1);
    }).then(function(candidate) {
```

```
    assert.equal(candidate[0], 1, "contains the correct id");
    assert.equal(candidate[1], "Candidate 1", "contains the
correct name");
    assert.equal(candidate[2], 0, "contains the correct votes
count");
    return electionInstance.candidates(2);
  }).then(function(candidate) {
    assert.equal(candidate[0], 2, "contains the correct id");
    assert.equal(candidate[1], "Candidate 2", "contains the
correct name");
    assert.equal(candidate[2], 0, "contains the correct votes
count");
  });
});


it("allows a voter to cast a vote", function() {
  return Election.deployed().then(function(instance) {
    electionInstance = instance;
    candidateId = 1;
    return electionInstance.vote(candidateId, { from:
accounts[0] });
  }).then(function(receipt) {
```

```
    assert.equal(receipt.logs.length, 1, "an event was
triggered");
    assert.equal(receipt.logs[0].event, "votedEvent", "the event
type is correct");
    assert.equal(receipt.logs[0].args._candidateId.toNumber(),
candidateId, "the candidate id is correct");
    return electionInstance.voters(accounts[0]);
  }).then(function(voted) {
    assert(voted, "the voter was marked as voted");
    return electionInstance.candidates(candidateId);
  }).then(function(candidate) {
    var voteCount = candidate[2];
    assert.equal(voteCount, 1, "increments the candidate's vote
count");
  })
 });


 it("throws an exception for invalid candiates", function() {
  return Election.deployed().then(function(instance) {
    electionInstance = instance;
    return electionInstance.vote(99, { from: accounts[1] })
```

```
  }).then(assert.fail).catch(function(error) {
    assert(error.message.indexOf('revert') >= 0, "error message must contain revert");
    return electionInstance.candidates(1);
  }).then(function(candidate1) {
    var voteCount = candidate1[2];
    assert.equal(voteCount, 1, "candidate 1 did not receive any votes");
    return electionInstance.candidates(2);
  }).then(function(candidate2) {
    var voteCount = candidate2[2];
    assert.equal(voteCount, 0, "candidate 2 did not receive any votes");
  });
 });


 it("throws an exception for double voting", function() {
  return Election.deployed().then(function(instance) {
    electionInstance = instance;
    candidateId = 2;
    electionInstance.vote(candidateId, { from: accounts[1] });
```

```
    return electionInstance.candidates(candidateId);
  }).then(function(candidate) {
    var voteCount = candidate[2];
    assert.equal(voteCount, 1, "accepts first vote");
    // Try to vote again
    return electionInstance.vote(candidateId, { from:
accounts[1] });
  }).then(assert.fail).catch(function(error) {
    assert(error.message.indexOf('revert') >= 0, "error message
must contain revert");
    return electionInstance.candidates(1);
  }).then(function(candidate1) {
    var voteCount = candidate1[2];
    assert.equal(voteCount, 1, "candidate 1 did not receive any
votes");
    return electionInstance.candidates(2);
  }).then(function(candidate2) {
    var voteCount = candidate2[2];
    assert.equal(voteCount, 1, "candidate 2 did not receive any
votes");
  });
```

```
  });
});
```