INFORMATION SYSTEMS
ASSIGNMENT 2

Submission by:
Ayça Avcı [S4505972]
Deepshi Garg [S4199456]

**Scripts are provided according to their compatibility with Solidity 0.7.0.

We created a smart contract for voting using the remix.ethereum.org IDE for the following use case:

A company's managing director wants to allow shareholders to make binary decisions (true or false) (s)he will propose to the shareholders. This use case holds the following functionalities:

- Director will be the only one who has the right to upload the contract.
- Director has the ability to upload any number of questions, one at a time.
- Director has the ability to enable shareholders to be able to vote and see results for approved decisions at any point of time.
- Each shareholder can only vote once for each decision.
- Director has the ability to close the voting process for a specific question.
- The majority result for a given question only computed and be seen by all shareholders, who have a see permission, after the voting process is closed for that question.
- Results can be seen as in the form of the winning answer (true, false or neutral) and the winning answer's count (3, 5 etc.).

SOLUTION :

- First, we created a *Shareholder* and *Question* structs and initialized the *Director* using the constructor as below.
- *Shareholder* struct has the *abilityToVote*, *abilityToSeeResult* boolean attributes that can only be changed by the *director*.
    - They enable the ability of shareholders to vote and see results. *mapping(uint => uint) votePerQuestion* is to determine whether the *shareholder* voted for the specific *question* already.

- *Question* struct has *description*, *trueCount*, *falseCount* and *closed* attributes.
    - *Description* is a string that represents the question itself.
    - *trueCount* and *falseCount* are integers that hold the count of true and false answers given by shareholders.
    - *closed* is a boolean attribute that is set by the *director* to determine whether the question is closed for voting.
- *Question[]* is an array that contains all the questions uploaded by the *director*.
- *numOfQuestions* is the integer that holds the number of questions uploaded.
- Inside the *constructor* function, message sender is set to the *director* initially, and its *allowedToVote* and *allowedToSeeResult* attributes are set to true, since the *director* must have the ability to vote and see the results.
- *numOfQuestions* is set to 0 initially, since there is no question uploaded in the initial state.

```
pragma solidity ^0.7.0;

contract Decisions {

    struct Shareholder {
        bool allowedToVote;
        bool allowedToSeeResult;
        mapping(uint => uint) votePerQuestion;
    }

    struct Question {
        string description;
        uint trueCount;
        uint falseCount;
        bool closed;
    }

    address public director;

    mapping(address => Shareholder) public shareholders;

    Question[] public questions;

    uint public numOfQuestions;

    constructor() {
        director = msg.sender;
        shareholders[director].allowedToVote = true;
        shareholders[director].allowedToSeeResult = true;

        numOfQuestions = 0;
    }
```

- Second, we implemented *uploadQuestion*, *vote*, *changeVotePermission*, *changeSeePermission* and *closeVotingProcess* functions.

    - *uploadQuestion* function takes a string *questiondescription* argument which is set to the *description*.
        - We use require, since only the director should be the one who uploads the questions. If "*msg.sender == director*" returns false, the function raises an error and says "*Only director can add questions.*". If it returns true, the question is pushed to the questions array with initial values as below.

```
function uploadQuestion(string memory questionDescription) public {
    require(
        msg.sender == director,
        "Only Director can add questions."
    );

    questions.push(Question({
        description: questionDescription,
        trueCount: 0,
        falseCount: 0,
        closed: false
    }));

    numOfQuestions++;
}
```

    - *vote* function takes 2 arguments. *question* is an integer value which represents the specific question number that is going to be voted, and a boolean value *answer*, which is the vote value : true or false.
        - If there is no such question in the array, it raises an error and says "*Invalid question*". If a question is closed for voting (*closed == true*), again raises an error and says "*Question is closed for voting*". Same applies (error raises) if the voter is not allowed to vote or already voted for that specific question. If the given *answer* is "true", *trueCount* increments by 1, otherwise *falseCount* increments by 1.

```
function vote(uint question, bool answer) public {
    require(question < numOfQuestions,"Invalid question.");
    require(!questions[question].closed, "Question is closed for voting.");

    Shareholder storage voter = shareholders[msg.sender];

    require(voter.allowedToVote, "This shareholder is not allowed to vote.");
    require(voter.votePerQuestion[question] == 0, "This shareholder already voted for the specific question.");

    if (answer) {
        questions[question].trueCount += 1;
        voter.votePerQuestion[question] = 1;
    } else {
        questions[question].falseCount += 1;
        voter.votePerQuestion[question] = 2;
    }
}
```

- In *changeVotePermisson* and *changeSeePermission* functions, given shareholder's *allowedToVote* and *allowedToSeeResult* attribute values can be changed by the *director* only.

```
function changeVotePermission(bool abilityToVote, address shareholder) public {
    require(
        msg.sender == director,
        "Only Director can change permissions to vote."
    );
    shareholders[shareholder].allowedToVote = abilityToVote;
}

function changeSeePermission(bool abilityToSee, address shareholder) public {
    require(
        msg.sender == director,
        "Only Director can change permissions to see results."
    );
    shareholders[shareholder].allowedToSeeResult = abilityToSee;
}
```

- In *closeVotingProcess* function, only the *director* has the ability to change the *closed* attribute value of the given *question* to *false* or *true*.
  - If the given *question* number argument value exceeds the number of questions in the *questions* array, it raises an error.
  - If the given *question* is already closed for voting, it raises an error again as well.

```
function closeVotingProcess(uint question) public {
    require(
        msg.sender == director,
        "Only Director can close the voting process."
    );

    require(question < numOfQuestions, "Invalid question.");
    require(!questions[question].closed, "Question is already closed for voting.");

    questions[question].closed = true;
}
```

- In the *results* function, for the given *question* number, the result can be seen in the form of the *winningAnswer (true, false or neutral),* and the *winningVoteCount*, which is the winning answer's count (0, 3, 8 etc.).
  - For results to be computed, the shareholder who makes the request needs to have the permission to view results.
  - If there is no such question, i.e., the given *question* number is invali or the question is not closed for voting, the function raises an error.

```
    function results(uint question) public view returns (string memory winningAnswer_, uint winningVoteCount_) {
        require(
            shareholders[msg.sender].allowedToSeeResult,
            "This shareholder is not allowed to see the result"
        );

        require(question < numOfQuestions, "Invalid question.");
        require(questions[question].closed, "Question is still open for voting.");

        if (questions[question].trueCount > questions[question].falseCount) {
            winningVoteCount_ = questions[question].trueCount;
            winningAnswer_ = "approved";
        }

        if (questions[question].trueCount < questions[question].falseCount) {
            winningVoteCount_ = questions[question].falseCount;
            winningAnswer_ = "rejected";
        }

        if (questions[question].trueCount == questions[question].falseCount) {
            winningVoteCount_ = 0;
            winningAnswer_ = "neutral";
        }
    }
}
```