// SPDX-Licence-Identifier : MIT

pragma solidity >=0.7.0<0.8.0;

contract ballot{

struct vote{

address voteraddress;

bool choice;}

struct voter{

string votername;

bool voted;}

uint private countResult=0;

uint public finalResult=0;

uint public totalVoter=0;

uint public totalVote=0;

address public ballotOfficialAddress;

string public ballotOfficialName;

string public proposal;

mapping(uint->vote) private votes;

mapping(address->voter) public voterRegister;

enum State{Created,Voting,Ended}

State public state;

}

modifier condition(bool\_condition){

require(\_condition);

\_;}

modifier onlyOfficial(){

require(msg.sender==ballotOfficialAddress);

\_;}

modifier inState(State \_state){

require(state== \_state);

\_;}

constructor (string memory \_ballotofficialname,string memory \_proposal) public{

ballotOfficialAddress=msg.sender;

ballotOfficialName= \_ballotofficialname;

proposal = \_proposal;

state = State.Created;

}

function addVoter(address \_voterAdress,

string memory \_voterName) public

inState(State.Created)

onlyOfficial

{

voter memory v;

v.voted=false;

voterRegister[\_voterAdress]=v;

totalVoter++;}

function startVote()

public

inState(State.Created)

onlyOfficial{

state=State.Voting}

function doVote(bool \_choice)

public

inState(State.Voting)

returns (bool voted){

bool isFound=false;

if(byte(voterRegister[msg.sender].voterName).length !=0 && voterRegister[msg.sender].voted==false{

voterRegister[msg.sender].voted= true;

vote memory v;

v.voterAdress=msg.sender;

v.choice= \_choice;

if(\_choice){

countResult++;}

votes[totalVote]=v;

totalVoter++;

isFound=true;}

return isFound;}

function endVote()

public

inState(State.Voting)

onlyOfficial{

state=State.Ended;

finalResult=countResult;

}}