// new project

// SPDX-License-Identifier: UNLICENSED

pragma solidity>0.5.0<0.9.0;

/\*

Task: code a smart contract for crowd funding.

1- manager: manage all the contract

2-participants: they can pay donation with minimum ether limit

3-all fund will be collect in smart contract not in manager account.

4-every transection by manager will decided through voting of donors if more than 50% affirmative.

5-every donor and its donation will be rcorded in a list

6- if collected  fund is less than targeted amount , then every ransection will repay to respective payer after deadline passed.

7- there is time before that targeteted fund must collected.

8- payer will pay direct to smart contract.

variables:----------------------------------

1-manager address

2-dict for payer => ammount

3-time,

4-deadline,

5-minimumContribution,

6-targetSum

--minimum time before fund will get tragret ether,targeted fund,minimum fund u can pay

7- till raised fund

8- noOfParticipants

if manager want to give some charity money for blood donation.

1- adressess: to which adress charity fund will transfer.

2-amount: how much hospital need

3-description: why hospital need, is it truely for blood donation.

terms-----------------------------------

-block.timestamp; when this contract will be deploying, the  recorded time instance of it

-msg.value: person who send message in contract, its value

-msg.sender: address who is trying to contact with contract // first time only manager will send first of first message.

-address payable user1=payable(address) // given address ko paybale banao

-Payable\_User.transfer(No\_of\_ether) // payable user ko transfer karo fund.

-require(conditions)

-modifier mname {}

\*/

contract crowdFunding

{

    // defining variables

    address public manager;

    mapping(address=>uint)public participants; // default is 0=>0.

    uint public deadline; // in seconds since 1971 timestamp.

    uint public minContribution =0 wei;

    uint public targetSum;

    uint public raisedFund;       // default values of uint is zero, unsigned integer.

    uint public noOfParticipants; // to check no of % voting in favour of manager decison transection

    constructor(uint \_deadline,uint \_Mincontribution ,uint \_targetsum)

    // as contract deploy, this will be fist function which will run as construcyor.

    {

        manager=msg.sender;

        deadline=\_deadline +block.timestamp; // current time(in seconds)+ target period

        minContribution=\_Mincontribution;

        targetSum=\_targetsum;

    }

    // function to trransfer ether by participants

    function sendEth() public payable // function with "payable" make it pay+able.

    {

        require(block.timestamp<=deadline, "deadline passed sriesti, u cant pay"); //  accept ether if before deadline of targetted sum.

        require(msg.value>=minContribution ,"hey poor, send some morre ethers.");

        if (participants[msg.sender]==0)

        {

            participants[msg.sender]= msg.value;

            noOfParticipants=noOfParticipants+1;

            raisedFund=raisedFund+msg.value;

        }

        else // if same sender is interested to paying again

        {

            participants[msg.sender]=participants[msg.sender]+ msg.value;

            raisedFund=raisedFund+msg.value;

        }

    }

    // function to check our balance

    function  balance() view public returns(uint)

        {

            return address(this).balance;

        }

    // for refund to participants

    function refund()  public

    {

        require(block.timestamp>deadline && balance()<targetSum,"Sriesti u are not eligible");

        require(participants[msg.sender]!=0); // payer must have pay some fund firstly.

        // make sender  address under payable.

        // syntex: address payable userN=payable(address)

        address payable user=payable(msg.sender);

        // transfer the respective fund to adddresse

        user.transfer(participants[msg.sender]);

        participants[msg.sender]=0;

    }

    // More variables: manager spend charity fund for  welfare

    struct welfareRequest // complex data structure for a request

    {

        string description;

        uint amount;

        address payable welfareAccount; // making welfare account payable to  charity fund from manager

        // about funs transection decison through voting

        bool  pending; // request is pending or not

        uint NoVoters; // total no of votes done to get pecentage

        mapping(address=>bool) voters; // record which voter voted or not// call this while do voting.

    }

    // no of request managers getting for requesting funding

    mapping(uint=>welfareRequest) public getRequests; //getrequest contain no of request it get and metadata about request

    // i.e. : ist request with its description is in NRequesst dict.

    uint public nrequest;  // total no of request yet he receive and about those requests is in complex data strucure welfareRequest.

    // manager interested to use some ethers for  positive , so we need voting and must be >50%.

    modifier modify()

    {

        require( msg.sender==manager,"only manager can see this peding request");

        \_;

    }

    function requestUpdation(string memory \_description,uint \_amount,address payable \_accountAddress) public modify() // by calling modify function, apply  modifier function defined restrictions.

    {

        welfareRequest storage pendingRequest=getRequests[nrequest];

        nrequest=nrequest+1;

        // getRequests={nrequest:pendingRequest} where pendingrequest belongs complex datatype "welfareRequest".

        /\* for understanding

        mapping(uint => string) Example1;

        string storage KeyValue=Example1[9];

        ---- exactly

        # Example1[9] gives value.

        # above assumes that string data type replace welfareRequest complex data type // data structure

        in python syntex will be, e\Example1={9:Keyvalue}.

        \*/

        pendingRequest.description=\_description;

        pendingRequest.amount=\_amount;

        pendingRequest.welfareAccount=\_accountAddress;

        pendingRequest.pending=false;

        pendingRequest.NoVoters=0;

    }

    function voterDecisioning(uint requestNo) public // requetNo told that about which request voting is done.

    {

        require(participants[msg.sender]>0,"you must be pay if interested to vote");

        welfareRequest storage mydecision= getRequests[requestNo];

        // getRequests={requestNo:welfareRequest}

        require(mydecision.voters[msg.sender]=true,"u already voted baby");

        mydecision.voters[msg.sender]=true;

        mydecision.NoVoters= mydecision.NoVoters+1;

    }

    function counting(uint requestNo)  public

    {

        welfareRequest storage dataN= getRequests[requestNo];

        require(raisedFund >targetSum && dataN.amount<raisedFund && dataN.pending==false,"some error, either already considered or fund uissue");

        dataN.pending=true; //requested done

        require(dataN.NoVoters>=noOfParticipants/2,"sriesti u loose faith, transection banned"); // voting must favour of 50%+

        address payable ngo=payable(dataN.welfareAccount);

        ngo.transfer(dataN.amount); // money transfer

    }

}