pragma solidity ^0.5.16;

//Project agreed solidity version

/\*

This script will allow the power providers andcustomers to make an agreement and provide power to customers

\*/

contract Power{

    address payable customer;

    address payable provider;

    uint public no\_of\_agreements = 0; // Number of agreements made between customers and providers

    uint public no\_of\_customers = 0;

    uint public no\_of\_payments = 0; // Number of payments being made

    uint public no\_of\_providers = 0; // Number of power providers that customer and choose from

    uint public no\_of\_slots = 0;

     uint public no\_of\_receipts;

    struct Customer{

        uint customerid; // Customer identification number

        uint agreementid; // Power agreement between customer and power provider

        string customername; // Name of customer

        string customerAdress; // Customer identification

        uint monthlyPayment; // Payment for electrical power

        uint providerid; // Provider's identification number

        uint timestamp;

        address payable currentCustomer; // Payment sent from customer's account

        address payable powerProvider; // Payment recieved by power provider

    }

    mapping(uint => Customer) public Customer\_by\_No; // Customer Array

    struct Providers{

        uint providerid; // Provider identification number

        uint agreementid; // Power agreement between customer and power provider

        string providerbrand; // Provider brand name

        string providerAddress; // provider location

        uint customerid;

        uint monthlyPayment;

        uint securitydeposit;

        uint timestamp;

        address payable currentCustomer; // Payment sent from custome's account

        address payable powerProvider; // Payment recieved by power provider

    }

    mapping(uint => Providers) public Providers\_by\_No;

    struct Slots{

        uint providerid;

        string providerbrand;

        string providerAddress;

        uint customerid;

        string customerAddress;

        uint agreementid;

        uint monthlyPayment;

        uint timestamp;

        address payable currentCustomer;

        address payable powerProvider;

    }

    mapping(uint => Slots) public Slots\_by\_No;

    struct Payment{

        uint slotname;

        uint slotid;

        uint agreementid;

        string slotAddress;

        uint monthlyPayment;

        uint timestamp;

        address payable currentCustomer;

        address payable powerProvider;

    }

    mapping(uint => Payment) public Payment\_by\_No;

    struct PowerAgreement{

        uint agreementid;

        uint customerid;

        string customername;

        uint providerid;

        string providerbrand;

        string provideraddress;

        uint monthlyPayment;

        uint securitydeposit;

        uint timestamp;

        address payable currentCustomer; // Payment sent from custome's account

        address payable powerProvider; // Payment recieved by power provider

    }

    mapping(uint => PowerAgreement) public PowerAgreement\_by\_No;

    struct Receipt{

        uint receiptid;

        string reason\_for\_payment;

        uint monthlyPayment;

        uint timestamp;

        address payable currentCustomer;

        address payable powerProvider;

    }

    mapping(uint => Receipt) public receipt\_by\_number;

    function addcustomer(

        uint \_index,

        address payable currentCustomer,

        address payable powerProvider,

        uint \_customerid,

        string memory \_customername,

        string memory \_customeraddress,

        uint \_payment,

        uint \_monthlyPayment,

        uint \_securitydeposit)

        public{

            require(msg.sender != address(0));

            no\_of\_customers ++;

            Customer\_by\_No[no\_of\_customers] = Customer(no\_of\_customers,0,\_customerid,\_customername,\_payment,\_monthlyPayment,\_securitydeposit,msg.sender,address(0));

        }

    function signAgreement(

        uint \_index,

        address payable currentCustomer,

        address payable powerProvider,

        uint \_providersid,

        string memory providerbrand,

        string memory provideraddress,

        string memory slotname,

        uint payment,

        uint monthlyPayment,

        uint securitydeposit)

        public{

            require(msg.sender != address(0));

            address payable \_powerProvider = Slots\_by\_No[\_index].powerProvider;

            uint totalfee = monthlyPayment + securitydeposit;

            powerProvider.transfer(totalfee);

            no\_of\_agreements;

            Slots\_by\_No[\_index].currentCustomer = msg.sender;

            Slots\_by\_No[\_index].timestamp = block.timestamp;

            Slots\_by\_No[\_index].agreementid = no\_of\_agreements;

            PowerAgreement\_by\_No[no\_of\_agreements] = PowerAgreement(\_index,no\_of\_agreements,Slots\_by\_No[\_index].slotname,Slots\_by\_No[\_index].slotAddress,Slots\_by\_No[\_index].monthlyPayment,Slots\_by\_No[\_index].securitydeposit,block.timestamp,msg.sender,\_powerProvider);

            no\_of\_payments ++;

            Payment\_by\_No[no\_of\_payments] = Payment(no\_of\_payments,\_index,no\_of\_agreements,Slots\_by\_No[\_index].slotname,Slots\_by\_No[\_index].slotAddress,Slots\_by\_No[\_index].monthlyPayment,now,msg.sender,\_powerProvider);

        }

    function payProvider(

        uint \_index,

        address payable currentCustomer,

        address payable powerProvider,

        uint \_providersid,

        string memory \_providerbrand,

        string memory \_provideraddress,

        uint \_payment,

        uint \_monthlyPayment,

        uint \_securitydeposit)

        public{

            require(msg.sender != address (0));

            address payable \_powerProviders = Slots\_by\_No[\_index].powerProvider;

            uint payment = Slots\_by\_No[\_index].monthlyPayment;

            \_powerProviders.transfer(\_payment);

            no\_of\_providers ++;

            Slots\_by\_No[no\_of\_slots] = Slots(no\_of\_slots,\_index,Slots\_by\_No[\_index].agreementid,Slots\_by\_No[\_index].slotname,Slots\_by\_No[\_index].slotAddress,\_payment,now,msg.sender,Slots\_by\_No[\_index].powerProvider);

        }

    function make\_receipt(address payable currentCustomer, address payable powerProvider, string memory reason, uint monthlyPayment)

        private {

            require(powerProvider != address(0));

            require(currentCustomer != address(0));

            no\_of\_receipts++;

            powerProvider.transfer(monthlyPayment);

            receipt\_by\_number[no\_of\_receipts] = Receipt(no\_of\_receipts, reason,monthlyPayment,now,currentCustomer,powerProvider);

        }

    function agreementCompleted(uint \_index) public payable

    {

        require(msg.sender !=(0));

        address payable \_powerProvider = Slots\_by\_No[\_index].currentCustomer;

        uint securitydeposit = Slots\_by\_No[\_index].securitydeposit;

        customer.transfer(securitydeposit);

    }

    function agreementTerminated(uint \_index)

        public{

            require(msg.sender != address (0));

            Slots\_by\_No[\_index].vacant = true;

        }

}