Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 7**	
Title of Lab Assignment: Simple Experiments using Solidity Program Constructs			
DOP: 08-10-2024		**DOS: 08-10-2024**	
CO Mapped:	**PO Mapped:**		**Signature:**

Practical No. 7

Aim: Simple Experiments using Solidity Program Constructs (if-then, while etc...)

1. Write a program in solidity to create a structure student with Roll no, Name, Class, Department, Course enrolled as variables.
    a. Add information of 5 students.
    b. Search for a student using Roll no.
    c. Display all information

2. Create a structure Consumer with Name, Address, Consumer ID, Units and Amount as members. Write a program in solidity to calculate the total electricity bill according to the given condition.
   For first 50 units Rs. 0.50/unit
   For next 100 units Rs. 0.75/unit
   For next 100 units Rs. 1.20/unit
   For unit above 250 Rs. 1.50/unit
   An additional surcharge of 20% is added to the bill. Display the information of 5 such consumers along with their units consumed and amount.

Theory:

Smart Contracts: Smart contracts are self-executing contracts with the terms of the agreement directly written into code. They run on blockchain platforms, such as Ethereum, allowing for trustless transactions and automation of processes. Smart contracts ensure that conditions are met before executing actions, which eliminates the need for intermediaries.

Data Structures: In Solidity, data structures are essential for organizing and managing complex data efficiently. The two primary types of data structures used in Solidity are:

1. Structs:
    a. Structs are custom data types that group multiple variables under a single name. They are used to define complex entities with various attributes.
    b. For example, the Consumer struct in the ElectricityBillRegistry contract encapsulates all relevant details of an electricity consumer, such as consumer ID, name, address, units consumed, and amount due.

2. Arrays:
    a. Arrays are used to store multiple values of the same type. In the contract, an array of Consumer structs is used to maintain a list of all consumers, facilitating easy access and manipulation of consumer data.

Functionality in Contracts: Smart contracts utilize functions to manipulate data structures:

a. Adding Data: Functions like addConsumer allow users to add new instances of data (consumers) to the contract.
b. Retrieving Data: Functions like getConsumers return the entire list of consumers, providing structured access to data.
c. Calculating Values: Internal functions can perform calculations (e.g., calculateBill) based on the data stored in structs.

Importance of Structs in Smart Contracts:

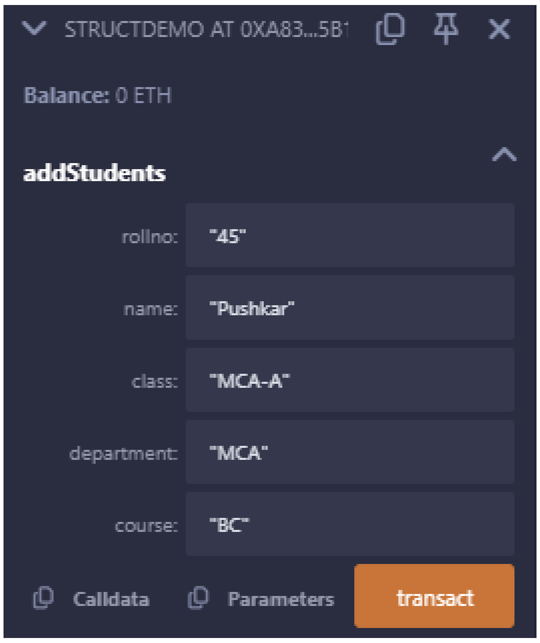
a. Clarity: Structs enhance code readability by grouping related data, making the contract easier to understand.
b. Efficiency: They optimize data management by reducing the complexity of handling multiple variables.
c. Encapsulation: Structs encapsulate data and behavior, allowing for better organization of contract logic.

1. **Student.sol**

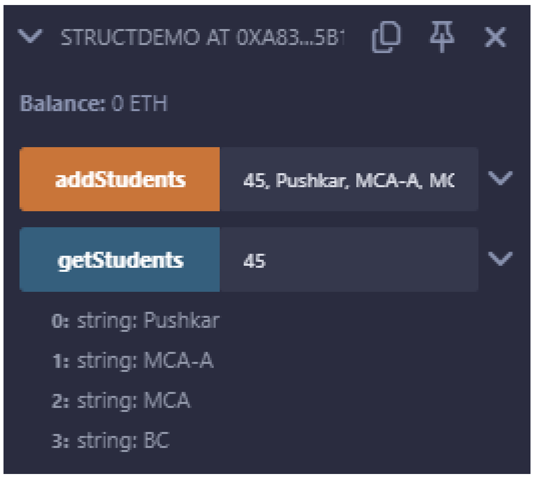
   Code:

```
// SPDX-License-Identifier: GPL-3.0
// Students Details
pragma solidity ^0.8.26;

// Creating a Smart Contract
contract StructDemo{
        // Structure of Students
        struct Students{
                // State variables
                int rollno;
                string name;
                string class;
                string department;
        string course;
        }

        Students []studs;
        // Function to add
        // Students details
        function addStudents(int rollno, string memory name, string memory class,
string memory department, string memory course) public {
                Students memory e = Students(rollno, name, class, department,
course);
                studs.push(e);
        }

        // Function to get
        // details of Students
        function getStudents( int rollno ) public view returns (string memory, string
memory, string memory, string memory){
                uint i;
                for(i=0; i < studs.length; i++) {
                        Students memory e = studs[i];

                        // Looks for a matching
                        // Students id
                        if(e.rollno == rollno) {
                                return(e.name, e.class, e.department, e.course);
                        }
                }

                // If provided Students
                // id is not present
                // it returns Not
                // Found
                return("Not Found", "Not Found", "Not Found", "Not Found");
        }
}
```

Output:



```
STRUCTDEMO AT 0XA83...5B1
Balance: 0 ETH
addStudents
rollno: "45"
name: "Pushkar"
class: "MCA-A"
department: "MCA"
course: "BC"
Calldata  Parameters  transact
```



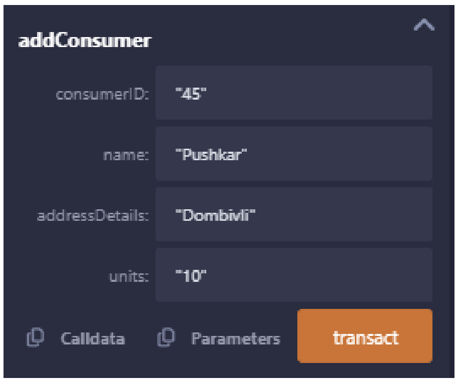
```
STRUCTDEMO AT 0XA83...5B1
Balance: 0 ETH
addStudents  45, Pushkar, MCA-A, MC
getStudents  45
0: string: Pushkar
1: string: MCA-A
2: string: MCA
3: string: BC
```

2. **Consumer.sol**

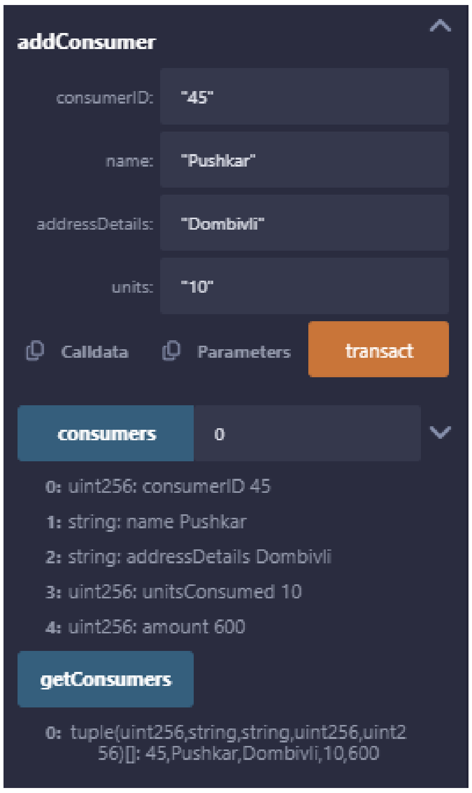
Code:

```
// SPDX-License-Identifier: GPL-3.0
// Solidity program
// to store Consumer Details
pragma solidity ^0.8.26;

// Creating a Smart Contract
contract ElectricityBillRegistry {
    // Structure of consumer
    struct Consumer {
        // State variables
        uint256 consumerID;
        string name;
        string addressDetails;
        uint256 unitsConsumed;
        uint256 amount;
    }

    Consumer[] public consumers;

    // Function to add consumer details
    function addConsumer(uint256 consumerID, string memory name, string memory
addressDetails, uint256 units) public {
        uint256 amount = calculateBill(units);
        Consumer memory c = Consumer(consumerID, name, addressDetails, units,
amount);
        consumers.push(c);
    }

    // Function to calculate the electricity bill
    function calculateBill(uint256 units) private pure returns (uint256) {
        uint256 bill = 0;

        if (units <= 50) {
            bill = units * 50; // Rs. 0.50/unit
        } else if (units <= 150) {
            bill = (50 * 50) + ((units - 50) * 75); // Rs. 0.75/unit for units 51-150
        } else if (units <= 250) {
            bill = (50 * 50) + (100 * 75) + ((units - 150) * 120); // Rs. 1.20/unit for units
151-250
        } else {
            bill = (50 * 50) + (100 * 75) + (100 * 120) + ((units - 250) * 150); // Rs.
1.50/unit for units above 250
        }
        // Apply additional surcharge of 20%
        bill = bill + (bill * 20 / 100);
        return bill;
    }

    // Function to get details of all consumers
    function getConsumers() public view returns (Consumer[] memory) {
        return consumers;
    }
}
```

Output:



```
consumerID: "45"
name: "Pushkar"
addressDetails: "Dombivli"
units: "10"
```



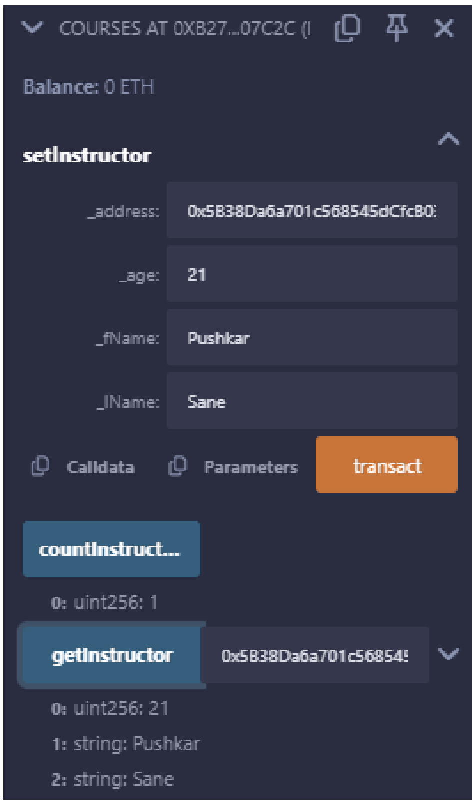
```
consumerID: "45"
name: "Pushkar"
addressDetails: "Dombivli"
units: "10"

consumers 0
0: uint256: consumerID 45
1: string: name Pushkar
2: string: addressDetails Dombivli
3: uint256: unitsConsumed 10
4: uint256: amount 600

getConsumers
0: tuple(uint256,string,string,uint256,uint2
   56)[]: 45,Pushkar,Dombivli,10,600
```

3. **Instructor.sol**

Code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.26;
contract Courses {
   struct Instructor {
      uint age;
      string fName;
      string lName;
   }

   mapping (address => Instructor) instructors;
   address[] public instructorAccts;
   function setInstructor(address _address, uint _age, string memory _fName, string
memory _lName) public {
   //var instructor = instructors[_address];
   instructors[_address].age = _age;
   instructors[_address].fName = _fName;
   instructors[_address].lName = _lName;
      instructorAccts.push(_address);
   }

   function getInstructors() view public returns(address[] memory) {
      return instructorAccts;
   }

   function getInstructor(address _address) view public returns (uint, string memory,
string memory) {
      return (instructors[_address].age, instructors[_address].fName,
instructors[_address].lName);
   }

   function countInstructors() view public returns (uint) {
      return instructorAccts.length;
   }
}
```

Output:



COURSES AT 0XB27...07C2C (I

Balance: 0 ETH

setInstructor

_address: 0x5B38Da6a701c568545dCfcB0:

_age: 21

_fName: Pushkar

_lName: Sane

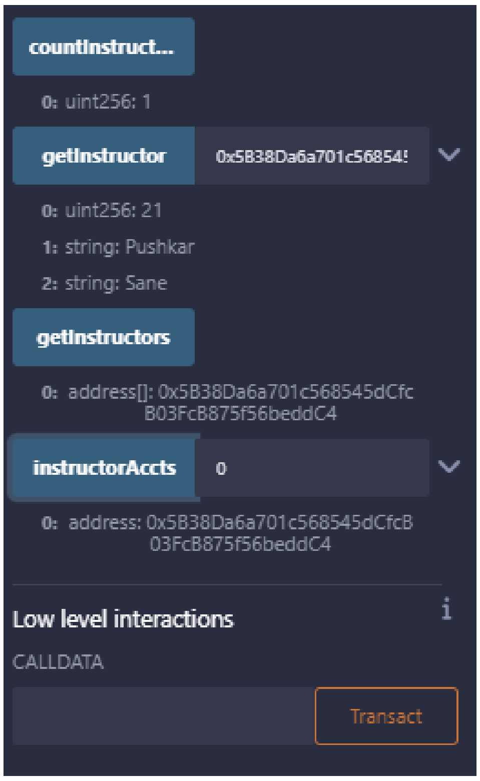
Calldata Parameters transact

countInstruct...

0: uint256: 1

getInstructor 0x5B38Da6a701c56854!

0: uint256: 21
1: string: Pushkar
2: string: Sane



countInstruct...

0: uint256: 1

getInstructor 0x5B38Da6a701c56854!

0: uint256: 21
1: string: Pushkar
2: string: Sane

getInstructors

0: address[]: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

instructorAccts 0

0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

Low level interactions

CALLDATA

Transact

Conclusion:

Successfully demonstrated simple experiments using Solidity Program Constructs.