SCSA2410:ETHEREUM AND SOLIDITY PROGRAMMING LANGUAGE LAB

EXP.01. SIMPLE COUNTER SMART CONTRACT

AIM:

To write a program to implement simple counter smart contract using solidity programming language.

ALGORITHM:

STEP1: Start.

STEP2: Declare a contract named counter.

STEP3: Declare a variable count and initiate its values to 0.

STEP4:Declare a function incrementcounter()

STEP4.1:Increase the value of count by 1.

STEP4.2: Return the value of count.

STEP5: Declare a function decrementcounter()

STEP5.1: Decrease the value of count by1.

STEP5.2: Return the value of count.

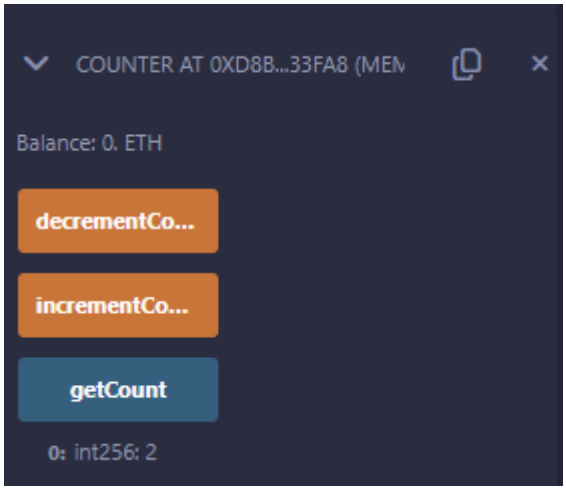
STEP6: Print the final value of count.

STEP7: Stop.

PROGRAM:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;
contract counter {
    int private count=0;
    function incrementCounter() public{    infinite gas
        count+=1;
        }
    function decrementCounter() public{    infinite gas
        count-=1;

    }
    function getCount() public view returns(int){    2437 gas
        return count;
    }
}
```

OUTPUT:



RESULT:

Thus the program to implement simple counter smart contract using solidity programming language has been written and executed successfully.

EXP.02. VARIABLES,DATATYPES AN STRUCTS IN SMART CONTRACT

AIM:

To write a solidity program to implement variables,datatypes and structs in smart contract.

ALGORITHM:

STEP1: Start.

STEP2: Define a contract named types.

STEP3: Declare a variable num1 and initialise its value.

STEP4: Declare a function getresult**.**

STEP5:Declare a structure called book with variables name,writer,id,available.

STEP6: Create two structure objects ‘book1 and ‘book2’ for the ‘Book’ structure.

STEP7: Initialise the values for the variables name,write,id,available for structure object ‘book2’.

STEP8: Declare the function called set_book_details() and initialise the value of variables**.**

STEP9: Declare two functions book_info and get_details for printing the values of the ‘book1’ structure

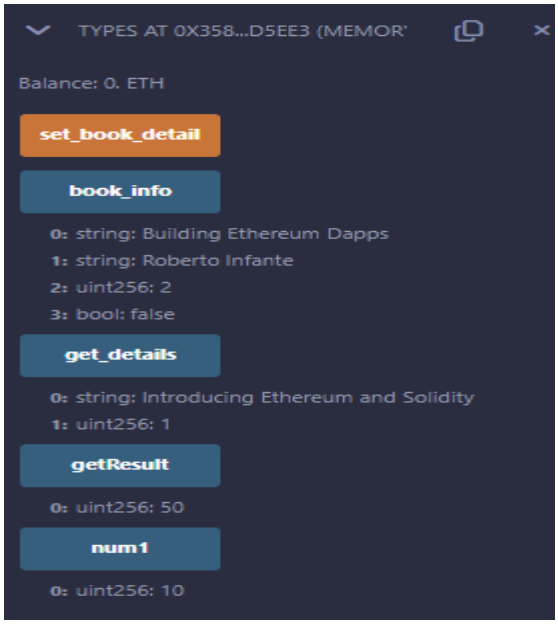
STEP10: Stop

PROGRAM:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// creating a contract
contract Types{
    uint public num1;
    constructor() {    infinite gas 459200 gas
        num1=10 ;
    }

    function getResult() public pure returns(uint)    infinite gas
    {
        uint num2=20;
        uint num3=30;
        uint result=num2+num3;
        return result;
    }

    struct Book{
        string name;
        string writer;
        uint id;
        bool available;
    }

    // declaring structure object
    Book book1;

    // assigning values to the fields for the fields for structure book1
    Book book2 = Book("building Ethereum Dapps","Roberto Infante",2,false);
```

```

    // defining a function to set values for the field for structure book1
    function Set_Book_detail() public{    infinite gas
        book1 = Book("Introducing Ethereum and solidity","hrs Danrel",1,true);
    }

    // defining function to print book2 details
    function book_info() public view returns(string memory,string memory,uint,bool){    infinite gas
        return (book2.name,book2.writer,book2.id,book2.available);
    }

    // defining function to print book1 details
    function get_details() public view returns(string memory,uint){    infinite gas
        return(book1.name,book1.id);
    }
}
```

OUTPUT:



RESULT:

Thus the program to implement variables,datatypes and struct in smart contract using solidity programming language has been written and executed successfully.

EXP.03.a. ARRAY DATA STRUCTURE

AIM:

To write a solidity program to implement array data structure inside smart contract.

ALGORITHM:

STEP1:Create a contract named Dynamic Array.

STEP2:Delare an array arr.

STEP3:Declare a function addData() to push values into the arr array.

STEP4:Declare a function getData() to return the entire array arr.

STEP5:Declare a function getLength() to return the length of the array.

STEP6:Declare a afunction getSum() to add the elements of arr and return sum.

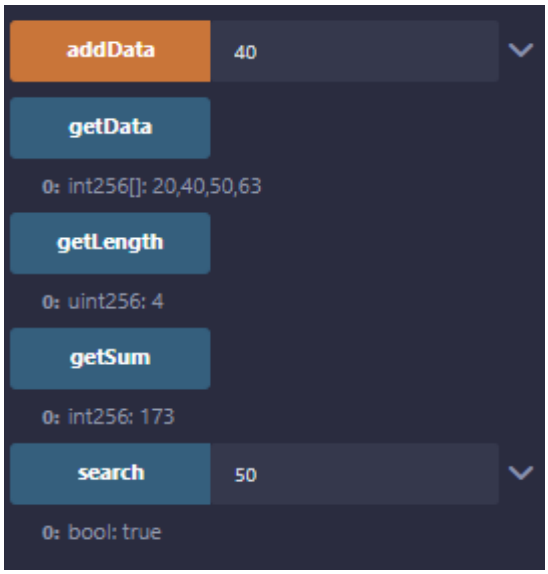
STEP7:Declare a function search() to search the array for a specific value.

STEP8:Stop.

PROGRAM:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Dynamicarray{
    int[] private arr;
    function addData(int num) public{    46807 gas
        arr.push(num);
    }
    function getData() public view returns(int[] memory){    infinite gas
        return arr;
    }
    function getLength() public view returns(uint) {    2489 gas
        return arr.length;
    }
    function search(int num) public view returns(bool){    infinite gas
        uint i;
        for(i=0;i<arr.length;i++){
            if (arr[i]==num) {
                return true;
            }
        }
        if(i>=arr.length){
            return false;
        }
    }
    function getSum() public view returns(int){    infinite gas
        uint i;
        int sum=0;
        for(i=0;i<arr.length;i++){
            sum=sum+arr[i];}
            return sum;

    }
}
```

OUTPUT:



RESULT:

Thus the program to write a solidity program to implement array data structure inside smart contract has been written and executed successfully.

EXP.03.b.MAPPING DATA STRUCTURE

AIM:

To write a solidity program to implement the mapping data structure inside smart contract.

ALGORITHM:

STEP1: Declare a contract named mapping_example.

STEP2: Declare a structure named student with elements name,subject and marks.

STEP3: Declare a mapping named result with key as address and value as student.

STEP4: Declare a function adding_values() to add values to the mapping.

STEP5: Declare a function get_student_result() to return the values of the result mapping.

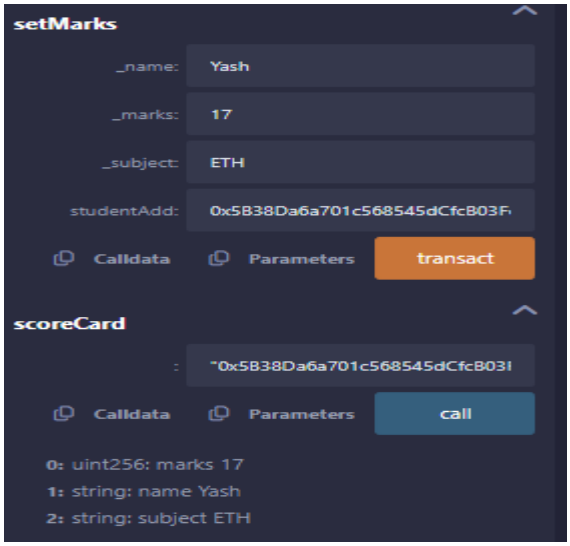
STEP6: Declare a function count_students() to count the number of values in the result mapping and return the count.

STEP7: Stop.

PROGRAM:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract mapping_example{
    struct studentItems {
        uint256 marks;
        string name;
        string subject;

    }

    mapping (address => studentItems) public scoreCard;

    function setMarks( string memory _name, uint256 _marks, string memory _subject, address studentAdd) public {    infinite gas
        scoreCard[studentAdd] = studentItems({
            marks : _marks,
            name : _name,
            subject : _subject
        });

    }

}
```

OUTPUT:



RESULT:

Thus the solidity program to implement the mapping data structure inside smart contract has been written and executed successfully.

EXP.04.EVEN OR ODD

AIM:

To write a solidity program to check the given number is even or not using conditional and looping inside smart contract.

ALGORITHM:

STEP1: Start

STEP2: Declare a contract named Even Odd.

STEP3: Declare two state variables num and result.

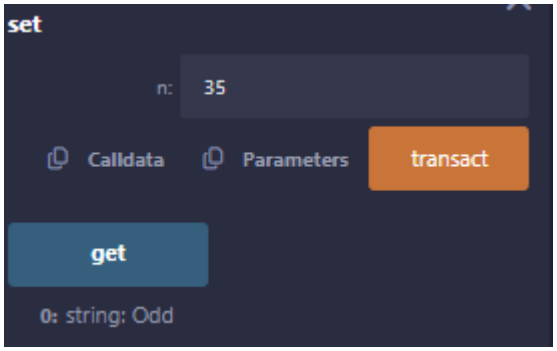
STEP4: Declare a function set that sets result or ‘even’ if the number is divisible by 2 and ‘odd’ even if the number is not divisible by 2.

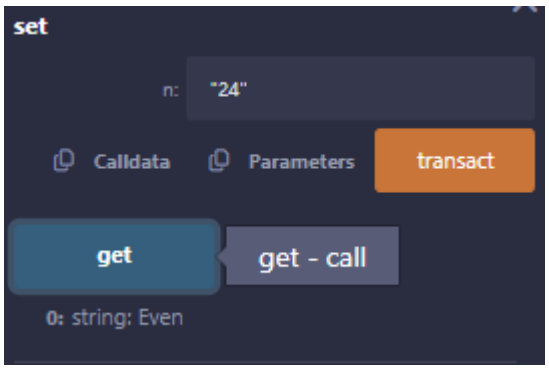
STEP5: Declare a function get() to return the value of result.

PROGRAM:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Evenodd{
    uint num;
    string result;
    function set(uint n) public{    infinite gas
        num = n;
        if(num%2==0){
            result="Even";
        }
        else {
            result="Odd";
        }

    }
    function get() public view returns(string memory){    infinite gas
        return result;
    }
}
```

OUTPUT:





RESULT:

Thus the solidity program to check the given number is even or not using conditional and looping inside smart contract has been written and executed successfully.

EXP.05.CRYPTOCURRENCY PAYMENT IN SMART CONTRACT FOR HOTEL ROOMS

AIM:

To write a smart contract in solidity language to build a code cryptocurrecy payment for hotel rooms.

ALGORITHM:

STEP1: Start

STEP2: Declare a contract named hoel room.

STEP3: Initialize enum statues.

STEP4: Declare the constructor and an object.

STEP5: Declare the global variable msg.sender assign to object.

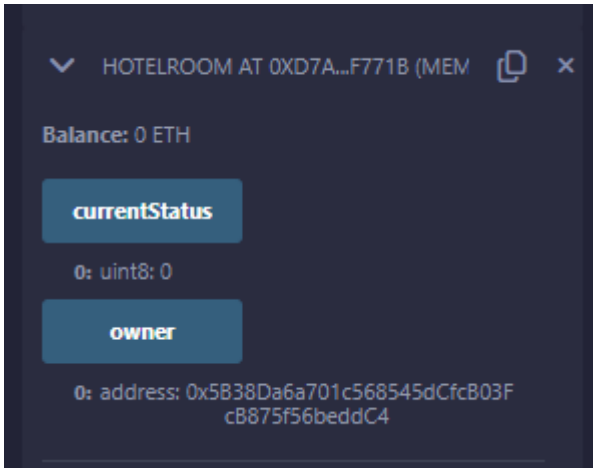
STEP6:Declare the modifiers.

STEP7:Stop.

PROGRAM:

```
contract Hotelroom {
    enum Statuses { vacant, occupied }
    Statuses public currentStatus;
    address payable public owner;
    event Occupy(address _occupant, uint _value);

    constructor() {    292502 gas 243600 gas
        owner = payable(msg.sender);
        currentStatus = Statuses.vacant;
    }

    modifier onlyWhileVacant() {
        require(currentStatus == Statuses.vacant, "Currently occupied");
        _;
    }

    modifier costs(uint _amount) {
        require(msg.value >= _amount, "Not enough Ether provided");
        _;
    }

    receive() external payable onlyWhileVacant costs(1 ether) {    undefined gas
        currentStatus = Statuses.occupied;
        owner.transfer(msg.value);
        emit Occupy(msg.sender, msg.value);
    }
}
```

OUTPUT:



RESULT:

Thus the smart contract in solidity language to build a code cryptocurrecy payment for hotel rooms has been written and executed successfully.

EXP.06.INHERITANCE

AIM:

To write a solidity program to perform the operation in multiple smart contract using inheritance.

ALGORITHM:

STEP1: Start

STEP2:Declare contracts A and B.

STEP3:Declare variables a and b in contracts A and B respectively.

STEP4: Declare functions get A in contract A and get B in contrat B.

STEP5: Declare a contract C which inherits contract A and B.

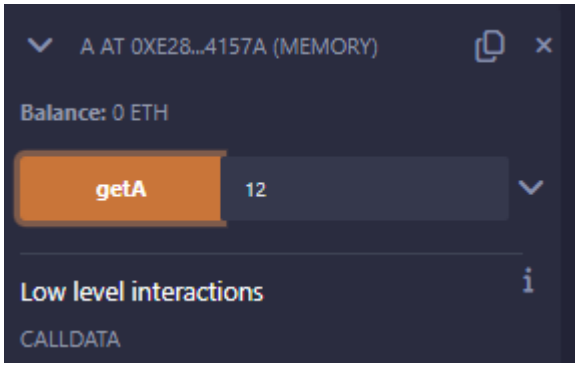
STEP6: Declare a function in C to add a and b return sum.

STEP7: Stop.

PROGRAM:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract A {
    uint internal a;
    function getA(uint _value) external {    22498 gas
        a = _value;
    }
}

contract B {
    uint internal b;
    function getB(uint _value) external {    22498 gas
        b = _value;
    }
}

contract C is A, B {
    function getValueOfSum() external view returns(uint) {    infinite gas
        return a + b;
    }
}

contract Caller {
    C c = new C(); // Corrected contract name

    function testInheritance() public returns (uint) {    infinite gas
        c.getA(10);
        c.getB(20);
        return c.getValueOfSum(); // Corrected contract name
    }
}
```

OUTPUT:



RESULT:

Thus the solidity program to perform the operation in multiple smart contract using inheritance has been written and executed successfully.

EXP.07.SIMPLE MARKETPLACE

AIM:

To write a solidity program to implement smart contract for simple marketplace.

ALGORITHM:

STEP1: Start

STEP2: Declare a contract SimplemarketPlace.

STEP3:Declare struct Item and State variables.

STEP4: Define mapping an dfunctions.

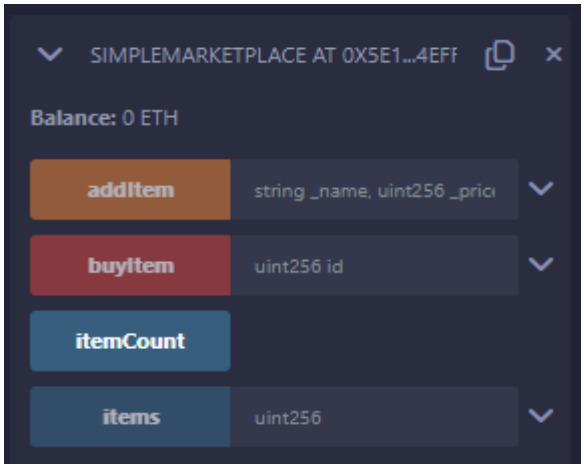
STEP5: Declare an event and exit the event.

STEP6: Stop.

PROGRAM:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleMarketplace {
    struct Item {
        uint id;
        string name;
        uint price;
        address payable owner;
        bool sold;
    }

    uint public itemCount;
    mapping(uint => Item) public items;

    event ItemAdded(uint id, string name, uint price, address payable owner);
    event ItemSold(uint id, address buyer, uint price);

    function addItem(string memory _name, uint _price) public {    infinite gas
        itemCount++;
        items[itemCount] = Item(itemCount, _name, _price, payable(msg.sender), false);
        emit ItemAdded(itemCount, _name, _price, payable(msg.sender));
    }

    function buyItem(uint id) public payable {    infinite gas
        require(id > 0 && id <= itemCount, "Invalid Item ID");
        require(!items[id].sold, "Item is already sold");
        require(msg.value >= items[id].price, "Not enough ether");

        items[id].owner.transfer(items[id].price);
        items[id].sold = true;
        emit ItemSold(id, msg.sender, items[id].price);
    }
}
```

OUTPUT:



RESULT:

Thus the solidity program to implement smart contract for simple marketplace has been written and executed successfully.

EXP.08.SHIPPING CONTRACT

AIM:

To write a solidity program to create a shipping contract by using the blockchain development kit for blockchain.

ALGORITHM:

STEP1: Start

STEP2: Declare a contract named shipping contract

STEP3: Declare state variables

STEP4: Declare a constructor to initialise state variables

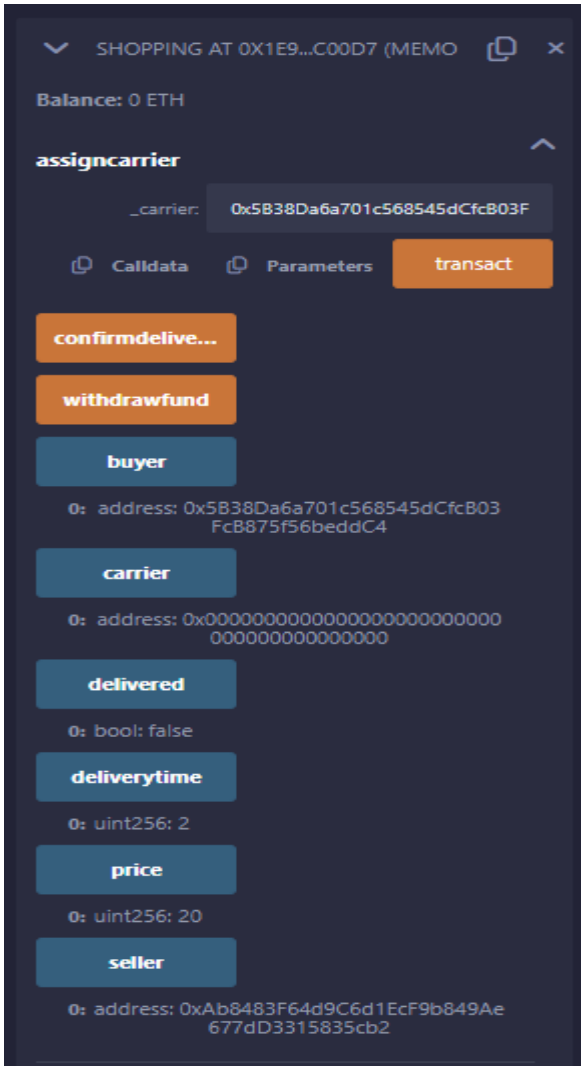
STEP5: Declare functions assign Carrier,confirm delivery,withdraw funds

STEP6: Stop

PROGRAM:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract shopping{
    address public buyer;
    address public seller;
    address public carrier;
    uint public price;
    uint public deliverytime;
    bool public delivered;

    constructor(address _buyer,address _seller,uint _price,uint _deliverytime){    infinite gas 538000 gas
        buyer=_buyer;
        seller=_seller;
        price=_price;
        deliverytime=_deliverytime;
    }

    function assigncarrier(address _carrier)public{    26892 gas
        require(msg.sender==seller,"only the buyer can confirm the delivery");
        carrier=_carrier;
    }
    function confirmdelivery() public{    30962 gas
        require(msg.sender==buyer,"Only the buyer can confirm");
        require(carrier!=address(0),"Carrier must be assigned before delivery can be confirmed");
        require(block.timestamp>=deliverytime,"Delivery time has not yet passed");
        delivered=true;
    }
    function withdrawfund() public {    infinite gas
        require(msg.sender==seller||msg.sender==carrier,"Only the seller or carrier can withdraw funds");
        require(delivered==true,"Delivery must be confirmed before funds can be wihtdrawn");
        uint amount=price;
        price=0;
        payable(msg.sender).transfer(amount);

    }
}
```

OUTPUT:



RESULT:

Thus the solidity program to create a shipping contract by using the blockchain development kit for blockchain has been written and executed successfully.