ASSIGNMENT – 1

(Blockchain Technology)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted to: Amit Kumar VIshwakarma

Name: Ishit Singh

Class: 3NC1

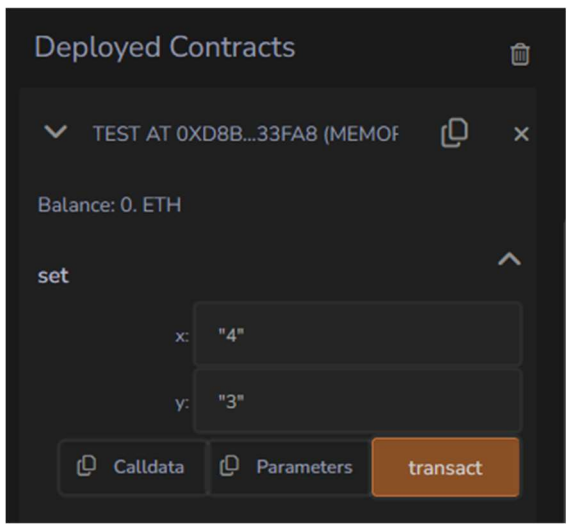
Roll No.: 102115023

Semester: Jan'24-May'24

Assignment - 1

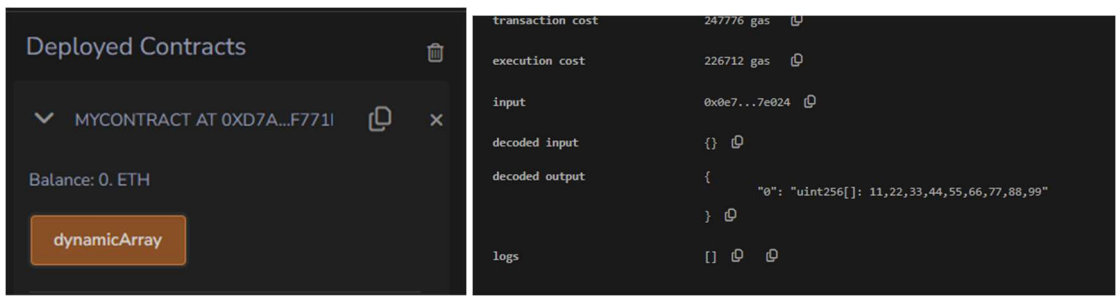
1. Create a file in solidity to declare variables of different data types and arrays (fixed dynamic) and use a function to get their values.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.4.16 < 0.9.0;

contract Test
{
    uint public var1;
    uint public var2;
    uint public sum;

    function set(uint x, uint y) public
    {
        var1 = x;
        var2 = y;
        sum = var1 + var2;
    }

    function get() public view returns(uint)
    {
        return sum;
    }
}
```




```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.13;

contract MyContract{
    /*      1. Static Array [size predefined]
    uint[5] numbers;
```

```
    function staticArray() public returns(uint[5] memory){
        numbers = [uint(10),20,30,40,50];
        return numbers;
    }
    */
    uint[] num;
    function dynamicArray() public returns(uint[] memory){
        num = [uint(11),22,33,44,55,66,77,88,99];
        return num;
    }
}
```



2. Create a file in solidity to declare functions and experiment with its scope as (public/private, pure/view and returns/no-returns.

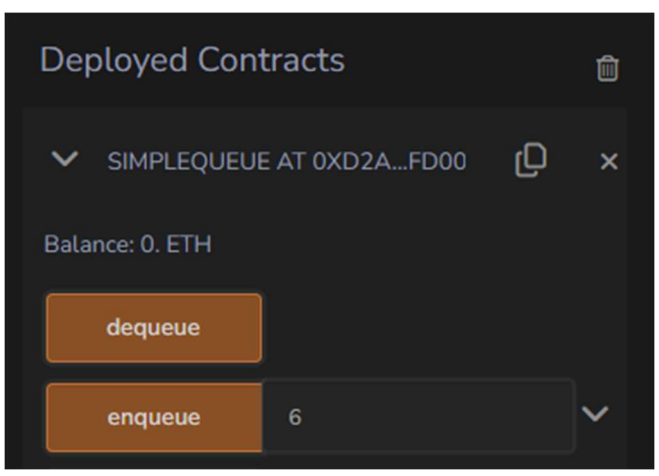
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract NFTcounter{
    uint public numberOfNFT;

    function checkTotalNFT() public view returns(uint){
    return numberOfNFT;
    }

    function addNFT() public{
        numberOfNFT += 1;
    }

    function decreaseNFT() public{
        numberOfNFT -= 1;
    }
}
```

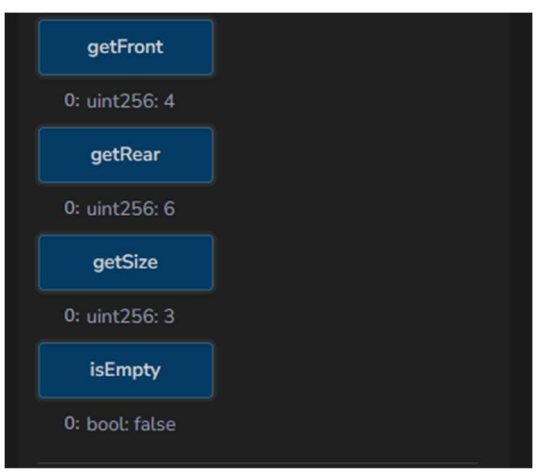
3. Write Smart contracts to perform STACK and QUEUE operations in solidity.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

// STACK Implementation Contract
contract solidityStack{
    int[] private stack;
    uint private capacity;

    constructor(){
        capacity = 5;
    }
    function push(int num) public returns(string memory){
        if(stack.length == capacity){
            return "Stack is Full";
        }
        stack.push(num);
        return "Element Added";
    }

    function pop() public returns(string memory){
        if(stack.length == 0){
            return "Stack is Empty";
        }
        stack.pop();
        return "Element Removed";
    }

    function getStack() public returns(int[] memory){
        return stack;
    }
}
```

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

contract SimpleQueue {
    uint256[] private queue;

    // Enqueue: Add an element to the end of the queue
    function enqueue(uint256 data) external {
        queue.push(data);
    }

    // Dequeue: Remove the element from the front of the queue
    function dequeue() external returns (uint256) {
        require(!isEmpty(), "Queue is empty");
```

```
        uint256 data = queue[0];
        // Shift all elements to the left to remove the front element
        for (uint256 i = 0; i < queue.length - 1; i++) {
            queue[i] = queue[i + 1];
        }
        // Remove the last element (duplicated due to shifting)
        queue.pop();
        return data;
    }

    // Get the current size of the queue
    function getSize() external view returns (uint256) {
        return queue.length;
    }

    // Check if the queue is empty
    function isEmpty() public view returns (bool) {
        return queue.length == 0;
    }

    // Get the front element of the queue
    function getFront() external view returns (uint256) {
        require(!isEmpty(), "Queue is empty");
        return queue[0];
    }

    // Get the rear element of the queue
    function getRear() external view returns (uint256) {
        require(!isEmpty(), "Queue is empty");
        return queue[queue.length - 1];
    }
}
```





4. Write different contracts in a single file with different functions to perform multidimensional array operations.

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

contract ArrayOperations {
    // Function to get the sum of elements in a 1D array
    function sumArray(int256[] memory arr) external pure returns (int256) {
        int256 sum = 0;
        for (uint256 i = 0; i < arr.length; i++) {
            sum += arr[i];
        }
        return sum;
    }
}

contract MatrixOperations {
    // Function to get the sum of elements in a 2D matrix
    function sumMatrix(int256[][] memory matrix) external pure returns (int256) {
        int256 sum = 0;
        for (uint256 i = 0; i < matrix.length; i++) {
            for (uint256 j = 0; j < matrix[i].length; j++) {
                sum += matrix[i][j];
            }
        }
        return sum;
    }
}

contract CubeOperations {
    // Function to get the sum of elements in a 3D cube
    function sumCube(int256[][][] memory cube) external pure returns (int256) {
        int256 sum = 0;
        for (uint256 i = 0; i < cube.length; i++) {
            for (uint256 j = 0; j < cube[i].length; j++) {
                for (uint256 k = 0; k < cube[i][j].length; k++) {
                    sum += cube[i][j][k];
                }
            }
        }
        return sum;
    }
}
```

5. Write smart contracts in solidity and call a function from contract1 to contract2 to give input for solving quadratic equation and the computation need to done in function declared in different contract.

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

contract QuadraticEquationSolver {
    // Function to solve a quadratic equation ax^2 + bx + c = 0
    function solveQuadraticEquation(int256 a, int256 b, int256 c) external pure
returns (int256, int256) {
        int256 discriminant = b**2 - 4 * a * c;

        require(discriminant >= 0, "No real roots");

        int256 root1 = (-b + int256(discriminant**0.5)) / (2 * a);
        int256 root2 = (-b - int256(discriminant**0.5)) / (2 * a);

        return (root1, root2);
    }
}

contract CallerContract {
    QuadraticEquationSolver private solverContract;

    constructor(address _solverContract) {
        solverContract = QuadraticEquationSolver(_solverContract);
    }

    // Function to call the quadratic equation solver in QuadraticEquationSolver
contract
    function callQuadraticEquationSolver(int256 a, int256 b, int256 c) external
view returns (int256, int256) {
        return solverContract.solveQuadraticEquation(a, b, c);
    }
}
```